Actual refactoring code				
	Previous	Later	Description/Need of refactoring	Testing class

1	private boolean executeOrders() {
	while (!d_PlayerOrderList.isEmpty()) {
	Order l_PlayerOrder = Player.next_order();
	if (!l_PlayerOrder.execute()) {
	return false;
	}
	}
	return true;
	}
		private void executeOrders() {
		int l_Counter = 0;
		while (l_Counter < d_GameMap.getPlayers().size()) {
		l_Counter = 0;
		for (Player l_Player : d_GameMap.getPlayers().values()) {
		Order l_Order = l_Player.nextOrder();
		if (l_Order == null) {
		l_Counter++;
		} else {
		if (l_Order.execute()) {
		l_Order.printOrderCommand();
		}
		}
		}
		}
		}
			Increased Clarity and Readability: The refactored version appears to be more explicit in its intent. It's clearer how the orders are being processed and by whom. The loop structure separates the handling of orders for each player, making it easier to understand the flow of execution.
			
			Better Handling of Players: Instead of relying on a specific list of orders (d_PlayerOrderList), the refactored version appears to iterate over all players in the game (d_GameMap.getPlayers().values()). This makes the code more flexible, as it can accommodate varying numbers of players without modification.
			
			Printing Order Commands: The refactored version introduces a call to l_Order.printOrderCommand() after executing an order. This implies that the code now logs or prints information about the executed orders, which could be useful for debugging or logging purposes.
			
			Elimination of the isEmpty() Check: In the original code, the loop condition relies on checking whether d_PlayerOrderList is empty. In the refactored version, this check is eliminated. Instead, the loop continues until l_Counter equals the number of players, which effectively means that all players' orders have been processed.
			
			Consistency in Method Naming: In the original code, Player.next_order() is called to get the next order for a player, while in the refactored version, it's called l_Player.nextOrder(). This change may aim to enforce consistency in method naming conventions.	

2	public static Order next_order() {
	Order l_firstOrder = d_PlayerOrderList.getFirst();
	d_PlayerOrderList.removeFirst();
	return l_firstOrder;
	}
		public Order nextOrder() {
		return d_CurrentOrders.poll();
		}
			Improving Encapsulation: In the original code, next_order() is a static method, implying that it's a method of the class itself rather than an instance method. However, it operates on an instance variable d_PlayerOrderList. By refactoring it into a non-static method nextOrder(), the method now belongs to an instance of the class. This change aligns better with object-oriented principles and encapsulation, as the method operates directly on the object's state (d_CurrentOrders) rather than on a static list.
			
			Enhancing Readability and Maintainability: By renaming the method to follow Java naming conventions (camelCase for method names), the code becomes more readable and consistent with standard Java practices. Developers familiar with Java codebases will find it easier to understand and work with.
			
			Using a More Appropriate Data Structure: In the refactored version, d_CurrentOrders is a Queue (likely LinkedList or similar) obtained from d_PlayerOrderList. This change suggests that the refactored version is utilizing a more appropriate data structure (Queue) for managing orders. Queues are well-suited for scenarios where elements need to be processed in a FIFO (First-In-First-Out) manner, which aligns with the behavior of processing orders.
			
			Simplifying the Code: The refactored version is more concise and expressive. It directly returns the order at the front of the queue (poll() removes and returns the head of the queue), making the code simpler and easier to understand.
				AdvanceOrderTest.java
				AirliftOrderTest.java
				BlockadeOrderTest.java
				BombOrderTest.java
				DeployOrderTest.java
				NegotiateOrderTest.java

3	private String getCommandFromPlayer(Player p_CurrentPlayer) {
	String l_DeployCommand = "";
	System.out.println(Constants.ISSUE_COMMAND_MESSAGE);
	System.out.println(Constants.DEPLOY_COMMAND_MESSAGE);
	while (!l_DeployCommand.equals(Constants.EXIT)) {
	l_DeployCommand = d_Scanner.nextLine();
	if (Constants.DEPLOY_COMMAND.equalsIgnoreCase(l_DeployCommand.split(" ")[0])) {
	if (checkIfCommandIsContainsDeploy(l_DeployCommand.toLowerCase(),p_CurrentPlayer)) {
	// Split the string based on consecutive whitespaces
	String[] l_StringParts = l_DeployCommand.trim().split("\\s+");
	return String.join(" ", l_StringParts);
	}
	} else {
	System.out.println(Constants.DEPLOY_COMMAND_MESSAGE);
	}
	}
	return l_DeployCommand;
	}
		private String getCommandFromPlayer() {
		String l_Command = "";
		System.out.println(Constants.ISSUE_COMMAND_MESSAGE);
		Constants.showIssueOrderCommand();
		l_Command = d_Scanner.nextLine();
		if(Objects.equals(l_Command.split(" ")[0], Constants.SHOW_MAP)){
		new ShowMapController(d_GameMap).show();
		return getCommandFromPlayer();
		}
		return l_Command;
		}
			In build 1 only deploy order was there so only deploy order was having the validation but as in build 2 each and every command has their validation so getCommandFromPlayer only handles showmap and getting data from user.	IssueOrderControllerTest.java

4	private boolean checkIfCommandIsContainsDeploy(String p_Command,Player p_CurrentPlayer) {
	boolean l_CapturedCountry = false;
	String[] l_CommandList;
	String commandString = p_Command.trim();
	
	// Split the string based on consecutive whitespaces
	l_CommandList = commandString.split("\\s+");
	
	if (l_CommandList.length == 3) {
	try {
	int l_Number = Integer.parseInt(l_CommandList[2].trim());
	if (l_Number <= 0) {
	System.out.println(Constants.ARMIES_NON_ZERO);
	return false;
	}
	} catch (NumberFormatException e) {
	System.out.println(Constants.ARMIES_NON_ZERO);
	return false;
	}
	for (Country l_Country : p_CurrentPlayer.getCapturedCountries()){
	if(Objects.equals(l_CommandList[1].trim(),l_Country.get_countryId().toLowerCase())){
	l_CapturedCountry=true;
	break;
	}
	}
	if(!l_CapturedCountry){
	System.out.println(Constants.COUNTRIES_DOES_NOT_BELONG);
	}
	return (l_CommandList[0].equals(Constants.DEPLOY_COMMAND) && l_CapturedCountry);
	} else
	return false;
	}
		public boolean validateCommand() {
		Player l_Player = getOrderInfo().getPlayer();
		Country l_Destination = getOrderInfo().getDestination();
		int l_Reinforcements = getOrderInfo().getNumberOfArmy();
		if (l_Player == null || l_Destination == null) {
		System.out.println(Constants.INVALID_COMMAND);
		return false;
		}
		if (!l_Player.isCaptured(l_Destination)) {
		System.out.println(Constants.COUNTRIES_DOES_NOT_BELON
		return false;
		}
		if (!l_Player.deployReinforcementArmiesFromPlayer(l_Reinforcemen
		System.out.println(Constants.NOT_ENOUGH_REINFORCEMEN
		return false;
		}
		return true;
		}
			The function name is changed and location of that function has been changed from IssueOrderController to DeployOrder.java	DeployOrderTest.java

5	get_continentId() set_continentId() get_controlValue() set_controlValue() get_countries() set_Countries() get_continentFileIndex() set_continentFileIndex() get_mapName() set_mapName() get_isValid() set_isValid() get_continents() set_continents() get_countries()	getContinentId() getContinentId() getControlValue() setControlValue() getCountries() setCountries() getContinentFileIndex() setContinentFileIndex() getMapName() setMapName() getIsValid() setIsValid() getContinents() setContinents() getCountries()	Changed function name to adhere coding conventions	CountryTest.java ContinentTest.java GameMapTest.java

6	while((l_lineString=l_fileReader.readLine())!=null){
	switch (l_lineString){
	case "[continents]":
	l_fileReader = processContinents (l_fileReader);
	break;
	case "[countries]":
	l_fileReader = processCountries (l_fileReader);
	break;
	case "[borders]":
	l_fileReader = processBorders (l_fileReader);
	break;
	}
	}
		while((l_lineString=l_fileReader.readLine())!=null){
		switch (l_lineString) {
		case "[continents]" -> processContinents(l_fileReader);
		case "[countries]" -> processCountries(l_fileReader);
		case "[borders]" -> processBorders(l_fileReader);
		default -> {
		}
		}
		}
			Removed switch statement with enchanced switch statement using lambdas to improve code readability
			
			Also removed extra assigned filereader which was unnecessary	

7	"src/main/resources/maps/"	MAP_FILE_DIRECTORY	Added constant in hardcoded string's place	

8	public void clearMap() {
	this.d_mapName = null;
	this.d_continents = null;
	this.d_countries = null;
	this.d_isValid = false;
	}
		public void clearMap() {
		this.setMapName("");
		this.setPlayers(new HashMap<String, Player>());
		this.setCountries(new HashMap<String, Country>());
		this.setContinents(new HashMap<String, Continent>());
		this.setIsValid(false);
		this.setPlayer(new Player());
		}
			Refactored Clear map function to put valid empty data to the map attributes	Tested in many test case's tearDown() function.

9	public boolean validateCommand() {
	Player l_Player = getOrderInfo().getPlayer();
	Country l_Country = getOrderInfo().getTargetCountry();
	if(l_Player == null){
	System.err.println(Constants.INVALID_PLAYER);
	return false;
	}
	if(l_Country.getPlayer() != l_Player){
	System.err.println(Constants.TARGET_COUNTRY_DOES_NOT_BELONG);
	return false;
	}
	if(!l_Player.checkIfCardAvailable(CardType.BLOCKADE)){
	System.err.println(Constants.NO_BLOCKADE_CARD);
	return false;
	}
	return true;
	}
		public boolean validateCommand() {
		Player l_Player = getOrderInfo().getPlayer();
		Country l_Country = getOrderInfo().getTargetCountry();
		if(l_Player == null){
		printValidationOfValidateCommand(Constants.INVALID_PLAYER);
		return false;
		}
		if(l_Country.getPlayer() != l_Player){
		printValidationOfValidateCommand(Constants.TARGET_COUNTRY_DOES_NOT_BELONG);
		return false;
		}
		if(!l_Player.checkIfCardAvailable(CardType.BLOCKADE)){
		printValidationOfValidateCommand(Constants.NO_BLOCKADE_CARD);
		return false;
		}
		return true;
		}
			In build 1 we used System.out.pritln after every validation command in validateCommand method so to reduce the use of System.out.println, we did
			redudancy by creating a common method in Constants file which will get called to print the validation.	

10	if(l_country.get_Neighbours().isEmpty()) {
	if(l_isContinentPrinting && l_isCountryPrinting) {
	System.out.printf("\n%-30s%-30s%-10s%-10s%-10s%-50s%n\n", l_continent.get_continentId(), l_country.get_countryId(), l_country.getPlayer().getName(), l_country.getArmies(), "");
	l_isContinentPrinting = false;
	l_isCountryPrinting = false;
	}
	else if(l_isCountryPrinting) {
	System.out.printf("\n%-30s%-30s%-10s%-10s%-10s%-50s%n\n", "", l_country.get_countryId(), l_country.getPlayer().getName(), l_country.getArmies(), "");
	l_isCountryPrinting = false;
	}
	}
	for(Country l_neighbor : l_country.get_Neighbours().values()) {
	if(l_isContinentPrinting && l_isCountryPrinting) {
	System.out.printf("\n%-30s%-30s%-10s%-10s%-10s%-50s%n\n", "", l_continent.get_continentId(), l_country.get_countryId(), l_country.getPlayer().getName(), l_country.getArmies(), l_neighbor.get_countryId());
	l_isContinentPrinting = false;
	l_isCountryPrinting = false;
	}
	else if(l_isCountryPrinting) {
	System.out.printf("\n%-30s%-30s%-10s%-10s%-10s%-50s%n\n", "", l_country.get_countryId(), l_country.getPlayer().getName(), l_country.getArmies(), l_neighbor.get_countryId());
	l_isCountryPrinting = false;
	}
	else {
	System.out.printf("%-30s%-30s%-10s%-10s%-10s%-50s%n\n", "", "", "", "", "", l_neighbor.get_countryId());
	}
	}
		if(l_country.getNeighbours().isEmpty()) {
		if (l_IsCountryPrinting) {
		String continentId = l_IsContinentPrinting ? l_continent.getContinentId() : "i";
		System.out.printf(Constants.SPACE_FORMATTER, continentId, l_country.getCountryId(), l_country.getPlayer().getName(), l_country.getArmies(), "");
		l_IsCountryPrinting = false;
		l_IsContinentPrinting = false;
		}
		}
		for(Country l_Neighbor : l_country.getNeighbours().values()) {
		if (l_IsCountryPrinting) {
		String continentId = l_IsContinentPrinting ? l_continent.getContinentId() : "";
		System.out.printf(Constants.SPACE_FORMATTER, continentId, l_country.getCountryId(), l_country.getPlayer().getName(), l_country.getArmies(), l_Neighbor.getCountryId());
		l_IsCountryPrinting = false;
		l_IsContinentPrinting = false;
		}
		else {
		System.out.printf(Constants.SPACE_FORMATTER, "", "", "", "", l_Neighbor.getCountryId());
		}
		}
			This part of conditional code, to show the countries and continents on the map. There
			were repetition of the condition which was unnecessary so we refactored by changing the
			condition which helped to reduce the lines of code, redundancy was decreased and without
			affecting the codes readability.	

11	if (l_Player.getNeutralPlayers().contains(l_To.
	getPlayer())) {
	System.out.printf("Truce between %s and %
	s\n", l_Player.getName(), l_To.getPlayer().getName());
	l_Player.getNeutralPlayers().remove(l_To.
	getPlayer());
	l_To.getPlayer().getNeutralPlayers().remove
	(l_Player);
	return true;
	}
	if (l_Player.isCaptured(l_To) || Objects.isNull
	(l_To.getPlayer())) {
	l_From.deleteArmies(l_Armies);
	l_To.deployArmies(l_Armies);
	if (!l_Player.getCapturedCountries().contains
	(l_To)) {
	l_Player.getCapturedCountries().add
	(l_To);
	}
	l_To.setPlayer(l_Player);
	System.out.println("Advanced/Moved " +
	l_Armies + " from " + l_From.getCountryId() + " to " +
	l_To.getCountryId());
	d_Leb.logEvent("Advanced/Moved " +
	l_Armies + " from " + l_From.getCountryId() + " to " +
	l_To.getCountryId());
	return true;
	} else if (d_GameStrategy.attack(l_Player,
	l_From, l_To, l_Armies)) {
	return true;
	}
		if (l_Player.getNeutralPlayers().contains(l_To.getPlayer())) {
		System.out.printf("Truce between %s and %s\n",
		l_Player.getName(), l_To.getPlayer().getName());
		d_GameEventLogger.logEvent("Truce between
		"+l_Player.getName()+" "+l_To.getPlayer().getName());
		l_Player.getNeutralPlayers().remove(l_To.getPlayer());
		l_To.getPlayer().getNeutralPlayers().remove(l_Player);
		} else if (l_Player.isCaptured(l_To) || Objects.isNull(l_To.
		getPlayer())) {
		l_From.deployArmies(l_Armies);
		l_To.deployArmies(l_Armies);
		if (!l_Player.getCapturedCountries().contains(l_To)) {
		l_Player.getCapturedCountries().add(l_To);
		}
		l_To.setPlayer(l_Player);
		System.out.println("Advanced/Moved " + l_Armies + "
		from " + l_From.getCountryId() + " to " + l_To.getCountryId());
		d_GameEventLogger.logEvent("Advanced/Moved " +
		l_Armies + " from " + l_From.getCountryId() + " to " + l_To.
		getCountryId());
		
		} else if (d_GameStrategy.attack(l_Player, l_From, l_To,
		l_Armies)) {
		}
		return true;
			In this code we removed the repetition of return statement after every condition by making a
			common return statement after the whole condition check and also changed the multiple if
			by making another if condition to else if condition.
				AdvanceOrderTest.java

Refactoring targets	
1	Remove hardcoded strings for better readability, improved reusability, ease of change, enhanced testing.
2	Make all functions name in camel case.
3	Implement state pattern for GamePhase.
4	Enhance the functions for more readability, performance and code reusability.
5	Implement command pattern for All the orders and also include validation and showing what executed by commands in Models rather than in controllers.
6	Removing deadcode, add more understandable comments, change variable name so all variable names after "_" should be in capital. (e.g. d_logger changed to d_Logger)
7	Implement data abstraction and encapsulation with Order model so all the orders such as Deploy, AirLift, Bomb, Negotiate etc can implement all the methods of Order.
8	Add Javadoc for private data members.
9	Refactoring test cases with Suit and also using singleton for map logic.
10	IssueOrderController was waiting for all the players to deploy their army but now it will be available until all the countries get captured.
11	Reduce if else statements and add more switch statements with ENUMS and use those ENUMS across the code
12	All commands have two layer of validation. One validation will be checked in IssueOrderController and other will be inside the respective models.
13	Change data structures from Array to ArrayDeque for more improvised logic of Queue
14	Rather than handling everything with Player class. We will improve with Current GameMap's game player. So we will reverse dependancy from Player->Player inside map to GameMap's player -> Player
15	Change functions of nextOrder and executeOrder.

Refactoring code

Previous	Later	Description/Need of refactoring

private boolean executeOrders() {
while (!d_PlayerOrderList.isEmpty()) {
Order l_PlayerOrder = Player.next_order();
if (!l_PlayerOrder.execute()) {
return false;
}
}
return true;
}
	private void executeOrders() {
	int l_Counter = 0;
	while (l_Counter < d_GameMap.getPlayers().size()) {
	l_Counter = 0;
	for (Player l_Player : d_GameMap.getPlayers().values()) {
	Order l_Order = l_Player.nextOrder();
	if (l_Order == null) {
	l_Counter++;
	} else {
	if (l_Order.execute()) {
	l_Order.printOrderCommand();
	}
	}
	}
	}
	}
		In build 1 We were having player's list in execute to keep track of howmany players have added/removed. But now we are doing with help of Game map's players. On top of that we are keeping track of howmany orders have been executed or remaining.

public static Order next_order() {
Order l_firstOrder = d_PlayerOrderList.getFirst();
d_PlayerOrderList.removeFirst();
return l_firstOrder;
}
	public Order nextOrder() {
	return d_CurrentOrders.poll();
	}
		First next order was being done with for loop but now we are using poll method for improved efficiency and improved readability.

private String getCommandFromPlayer(Player p_CurrentPlayer) {
String l_DeployCommand = "";
System.out.println(Constants.ISSUE_COMMAND_MESSAGE);
System.out.println(Constants.DEPLOY_COMMAND_MESSAGE);
while (!l_DeployCommand.equals(Constants.EXIT)) {
l_DeployCommand = d_Scanner.nextLine();
if (Constants.DEPLOY_COMMAND.equalsIgnoreCase(l_DeployCommand.split(" ")[0])) {
if (checkIfCommandIsContainsDeploy(l_DeployCommand.toLowerCase(),p_CurrentPlayer)) {
// Split the string based on consecutive whitespaces
String[] l_StringParts = l_DeployCommand.trim().split("\\s+");
return String.join(" ", l_StringParts);
}
} else {
System.out.println(Constants.DEPLOY_COMMAND_MESSAGE);
}
}
return l_DeployCommand;
}
	private String getCommandFromPlayer() {
	String l_Command = "";
	System.out.println(Constants.ISSUE_COMMAND_MESSAGE);
	Constants.showIssueOrderCommand();
	l_Command = d_Scanner.nextLine();
	if(Objects.equals(l_Command.split(" ")[0], Constants.SHOW_MAP)){
	new ShowMapController(d_GameMap).show();
	return getCommandFromPlayer();
	}
	return l_Command;
	}
		In build 1 only deploy order was there so only deploy order was having the validation but as in build 2 each and every command has their validation so getCommandFromPlayer only handles showmap and getting data from user.

private boolean checkIfCommandIsContainsDeploy(String p_Command,Player p_CurrentPlayer) {
boolean l_CapturedCountry = false;
String[] l_CommandList;
String commandString = p_Command.trim();
 	
// Split the string based on consecutive whitespaces
l_CommandList = commandString.split("\\s+");
 	
if (l_CommandList.length == 3) {
try {
int l_Number = Integer.parseInt(l_CommandList[2].trim());
if (l_Number <= 0) {
System.out.println(Constants.ARMIES_NON_ZERO);
return false;
}
} catch (NumberFormatException e) {
System.out.println(Constants.ARMIES_NON_ZERO);
return false;
}
for (Country l_Country : p_CurrentPlayer.getCapturedCountries()){
if(Objects.equals(l_CommandList[1].trim(),l_Country.get_countryId().toLowerCase())){
l_CapturedCountry=true;
break;
}
}
if(!l_CapturedCountry){
System.out.println(Constants.COUNTRIES_DOES_NOT_BELONG);
}
return (l_CommandList[0].equals(Constants.DEPLOY_COMMAND) && l_CapturedCountry);
} else
return false;
}
	public boolean validateCommand() {
	Player l_Player = getOrderInfo().getPlayer();
	Country l_Destination = getOrderInfo().getDestination();
	int l_Reinforcements = getOrderInfo().getNumberOfArmy();
	if (l_Player == null || l_Destination == null) {
	System.out.println(Constants.INVALID_COMMAND);
	return false;
	}
	if (!l_Player.isCaptured(l_Destination)) {
	System.out.println(Constants.COUNTRIES_DOES_NOT_BELONG);
	return false;
	}
	if (!l_Player.deployReinforcementArmiesFromPlayer(l_Reinforcements)) {
	System.out.println(Constants.NOT_ENOUGH_REINFORCEMENTS);
	return false;
	}
	return true;
	}
		The function name is changed and location of that function has been changed from IssueOrderController to DeployOrder.java

Refactoring code		
Previous	Later	Description/Need of refactoring

Refactoring code		
Previous	Later	Description/Need of refactoring

Refactoring code

Previous	Later	Description/Need of refactoring

public boolean validateCommand() {
Player l_Player = getOrderInfo().getPlayer();
Country l_Country = getOrderInfo().getTargetCountry();
if(l_Player == null){
System.err.println(Constants.INVALID_PLAYER);
return false;
}
if(l_Country.getPlayer() != l_Player){
System.err.println(Constants.TARGET_COUNTRY_DOES_NOT_BELONG);
return false;
}
if(!l_Player.checkIfCardAvailable(CardType.BLOCKADE)){
System.err.println(Constants.NO_BLOCKADE_CARD);
return false;
}
return true;
}
	public boolean validateCommand() {
	Player l_Player = getOrderInfo().getPlayer();
	Country l_Country = getOrderInfo().getTargetCountry();
	if(l_Player == null){
	printValidationOfValidateCommand(Constants.INVALID_PLAYER);
	return false;
	}
	if(l_Country.getPlayer() != l_Player){
	printValidationOfValidateCommand(Constants.TARGET_COUNTRY_DOES_NOT_BELONG);
	return false;
	}
	if(!l_Player.checkIfCardAvailable(CardType.BLOCKADE)){
	printValidationOfValidateCommand(Constants.NO_BLOCKADE_CARD);
	return false;
	}
	return true;
	}
		In build 1 we used System.out.println after every validation command in validateCommand method so to reduce the use of System.out.println, we did reductancy by creating a common method in Constants file which will get called to print the validation.

if(l_country.get_Neighbours().isEmpty()) {
if(l_isContinentPrinting && !_isCountryPrinting) {
System.out.printf("\n%-30s%-30s%-10s%-10s%-50s%n\n", t_continent.get_continentId(), l_country.get_countryId(), l_country.getPlayer().getName(), l_country.getArmies(), "");
l_isContinentPrinting = false;
l_isCountryPrinting = false;
}
else if(!_isCountryPrinting) {
System.out.printf("\n%-30s%-30s%-10s%-10s%-50s%n\n", "", l_country.get_countryId(), l_country.getPlayer().getName(), l_country.getArmies(), "");
l_isCountryPrinting = false;
}
}
for(Country l_neighbor : l_country.get_Neighbours().values()) {
if(l_isContinentPrinting && !_isCountryPrinting) {
System.out.printf("\n%-30s%-30s%-10s%-10s%-50s%n\n", t_continent.get_continentId(), l_country.get_countryId(), l_country.getPlayer().getName(), l_country.getArmies(), l_neighbor.get_countryId());
l_isContinentPrinting = false;
l_isCountryPrinting = false;
}
else if(!_isCountryPrinting) {
System.out.printf("\n%-30s%-30s%-10s%-10s%-50s%n\n", "", l_country.get_countryId(), l_country.getPlayer().getName(), l_country.getArmies(), l_neighbor.get_countryId());
l_isCountryPrinting = false;
}
else {
System.out.printf("%-30s%-30s%-10s%-10s%-50s%n\n", "", "", "", "", l_neighbor.get_countryId());
}
}
}
	if(l_country.getNeighbours().isEmpty()) {
	if (l_isCountryPrinting) {
	String continentId = l_isContinentPrinting ? l_continent.getContinentId() : "\";
	System.out.printf(Constants.SPACE_FORMATTER, continentId, l_country.getCountryId(), l_country.getPlayer().getName(), l_country.getArmies(), "");
	l_isCountryPrinting = false;
	l_isContinentPrinting = false;
	}
	}
	for(Country l_Neighbor : l_country.getNeighbours().values()) {
	if (l_isCountryPrinting) {
	String continentId = l_isContinentPrinting ? l_continent.getContinentId() : "";
	System.out.printf(Constants.SPACE_FORMATTER, continentId, l_country.getCountryId(), l_country.getPlayer().getName(), l_country.getArmies(), l_Neighbor.getCountryId());
	l_isCountryPrinting = false;
	l_isContinentPrinting = false;
	}
	else {
	System.out.printf(Constants.SPACE_FORMATTER, "", "", "", "", l_Neighbor.getCountryId());
	}
	}
	}
		In this part of conditional code to show the countries and continents on the map there were repetition of the condition which was unnecessary so we refactored by changing the condition which helped to reduce the lines of code, redundancy was decreased and without affecting the codes readability.

if(l_Player.getReinforcementArmies() != 0){
if (l_Player.getReinforcementArmies() <= 0 && !(l_ZeroReinforcementPlayers.contains(l_Player.getName()))) {
l_ZeroReinforcementPlayers.add(l_Player.getName());
l_PlayerCounts++;
continue;
}
if (l_PlayerCounts == d_GameMap.getPlayers().size()) {
System.out.println(Constants.ARMY_DEPLETED);
System.out.println(Constants.SEPERATER);
return p_CurrentGamePhase.nextState(d_UpcomingGamePhase);
}
System.out.println(Constants.SEPERATER);
System.out.println("Player: " + l_Player.getName() + ", armies assigned are: " + l_Player.getReinforcementArmies());
System.out.println(Constants.ELIGIBLE_NATIONS_ARMY);
for (Country l_CapturedCountry : l_Player.getCapturedCountries()) {
System.out.println(l_CapturedCountry.get_countryId() + " ");
}
System.out.println(Constants.SEPERATER);
String l_DeployCommands = getCommandFromPlayer(l_Player);
l_Player.issue_order(l_DeployCommands);
}else{
if (l_Player.getReinforcementArmies() <= 0 && !(l_ZeroReinforcementPlayers.contains(l_Player.getName()))) {
l_ZeroReinforcementPlayers.add(l_Player.getName());
l_PlayerCounts++;
continue;
}
if (l_PlayerCounts == d_GameMap.getPlayers().size()) {
System.out.println(Constants.ARMY_DEPLETED);
System.out.println(Constants.SEPERATER);
return p_CurrentGamePhase.nextState(d_UpcomingGamePhase);
}
}
	if (l_Player.getReinforcementArmies() <= 0 && !(l_ZeroReinforcementPlayers.contains(l_Player.getName()))) {
	l_ZeroReinforcementPlayers.add(l_Player.getName());
	l_PlayerCounts++;
	continue;
	}
	if (l_PlayerCounts == d_GameMap.getPlayers().size()) {
	System.out.println(Constants.ARMY_DEPLETED);
	System.out.println(Constants.SEPERATER);
	return p_CurrentGamePhase.nextState(d_UpcomingGamePhase);
	}
	if (l_Player.getReinforcementArmies() != 0) {
	System.out.println(Constants.SEPERATER);
	System.out.println("Player: " + l_Player.getName() + ", armies assigned are: " + l_Player.getReinforcementArmies());
	System.out.println(Constants.ELIGIBLE_NATIONS_ARMY);
	for (Country l_CapturedCountry : l_Player.getCapturedCountries()) {
	System.out.println(l_CapturedCountry.getCountryId() + " ");
	}
	System.out.println(Constants.SEPERATER);
	String l_Commands = getCommandFromPlayer(l_Player);
	l_Player.issueOrder(l_Commands);
	}	

if (l_Player.getNeutralPlayers().contains(l_To.getPlayer())) {
System.out.println("Truce between %s and %s\n", l_Player.getName(), l_To.getPlayer().getName());
l_Player.getNeutralPlayers().remove(l_To.getPlayer());
l_To.getPlayer().getNeutralPlayers().remove(l_Player);
return true;
}
if (l_Player.isCaptured(l_To) || Objects.isNull(l_To.getPlayer())) {
l_From.deleteArmies(l_Armies);
l_To.deployArmies(l_Armies);
if (!l_Player.getCapturedCountries().contains(l_To)) {
l_Player.getCapturedCountries().add(l_To);
}
l_To.setPlayer(l_Player);
System.out.println("Advanced/Moved " + l_Armies + " from " + l_From.getCountryId() + " to " + l_To.getCountryId());
d_Lab.logEvent("Advanced/Moved " + l_Armies + " from " + l_From.getCountryId() + " to " + l_To.getCountryId());
return true;
} else if (d_GameStrategy.attack(l_Player, l_From, l_To, l_Armies)) {
return true;
}
	if (l_Player.getNeutralPlayers().contains(l_To.getPlayer())) {
	System.out.printf("Truce between %s and %s\n" , l_Player.getName(), l_To.getPlayer().getName());
	d_GameEventLogger.logEvent("Truce between "+l_Player.getName()+" "+l_To.getPlayer().getName());
	l_Player.getNeutralPlayers().remove(l_To.getPlayer());
	l_To.getPlayer().getNeutralPlayers().remove(l_Player);
	} else if (l_Player.isCaptured(l_To) || Objects.isNull(l_To.getPlayer())) {
	l_From.deployArmies(l_Armies);
	l_To.deployArmies(l_Armies);
	if (!l_Player.getCapturedCountries().contains(l_To)) {
	l_Player.getCapturedCountries().add(l_To);
	}
	l_To.setPlayer(l_Player);
	System.out.println("Advanced/Moved " + l_Armies + " from " + l_From.getCountryId() + " to " + l_To.getCountryId());
	d_GameEventLogger.logEvent("Advanced/Moved " + l_Armies + " from " + l_From.getCountryId() + " to " + l_To.getCountryId());
	
	} else if (d_GameStrategy.attack(l_Player, l_From, l_To, l_Armies)) {
	}
	return true;
		In this code we removed the repetition of return statement after every condition by making a common return statement after the whole condition check and also changed the multiple if by making another if condition to else if condition.

Refactoring code

Previous	Later	Description/Need of refactoring	
Continent Model			
get_continentId()	getContinentId()	Changed function name to adhere coding conventions	
set_continentId()	getContinentId()	Changed function name to adhere coding conventions	
get_controlValue()	getControlValue()	Changed function name to adhere coding conventions	
set_controlValue()	setControlValue()	Changed function name to adhere coding conventions	
get_countries()	getCountries()	Changed function name to adhere coding conventions	
set_Countries()	setCountries()	Changed function name to adhere coding conventions	
get_continentFileIndex()	getContinentFileIndex()	Changed function name to adhere coding conventions	
set_continentFileIndex()	setContinentFileIndex()	Changed function name to adhere coding conventions	
GameMap Model			
get_mapName()	getMapName()	Changed function name to adhere coding conventions	
set_mapName()	setMapName()	Changed function name to adhere coding conventions	
get_isValid()	getIsValid()	Changed function name to adhere coding conventions	
set_isValid()	setIsValid()	Changed function name to adhere coding conventions	
get_continents()	getContinents()	Removed redundant getContinents function also	
set_continents()	setContinents()	Changed function name to adhere coding conventions	
get_countries()	getCountries()	Removed redundant getCountries function too	

public void clearMap() {
this.d_mapName = null;
this.d_continents = null;
this.d_countries = null;
this.d_isValid = false;
}
	public void clearMap() {
	this.setMapName("");
	this.setPlayers(new HashMap<String, Player>());
	this.setCountries(new HashMap<String, Country>());
	this.setContinents(new HashMap<String, Continent>());
	this.setIsValid(false);
	this.setPlayer(new Player());
	}
		Refactored Clear map function to put valid empty data to the map attributes	

Country Model			
	setNeighbours()	Changed function name to adhere coding conventions	
	setXCoordinate()	Changed function name to adhere coding conventions	
	setYCoordinate()	Changed function name to adhere coding conventions	
get_countryFileIndex()	getCountryFileIndex()	Changed function name to adhere coding conventions	
set_countryFileIndex()	setCountryFileIndex()	Changed function name to adhere coding conventions	
get_parentContinent()	getParentContinent()	Changed function name to adhere coding conventions	
set_parentContinent()	setParentContinent()	Changed function name to adhere coding conventions	
get_countryId()	getCountryId()	Changed function name to adhere coding conventions	
set_countryId()	setCountryId()	Changed function name to adhere coding conventions	
get_Neighbours()	getNeighbours()	Changed function name to adhere coding conventions	
get_numberOfArmies()	getNumberOfArmies()	Changed function name to adhere coding conventions	
set_numberOfArmies()	setNumberOfArmies()	Changed function name to adhere coding conventions	
Map Load File			

while((l_lineString=l_fileReader.readLine())!=null){
switch (l_lineString){
case "[continents]":
l_fileReader = processContinents(l_fileReader);
break;
case "[countries]":
l_fileReader = processCountries(l_fileReader);
break;
case "[borders]":
l_fileReader = processBorders(l_fileReader);
break;
}
}
	while((l_lineString=l_fileReader.readLine())!=null){
	switch (l_lineString) {
	case "[continents]" -> processContinents(l_fileReader);
	case "[countries]" -> processCountries(l_fileReader);
	case "[borders]" -> processBorders(l_fileReader);
	default -> {
	}
	}
	}
		Removed switch statement with enchanced switch statement using lambdas to improve code readability
		
		Also removed extra assigned filereader which was unnecessary	

"src/main/resources/maps/"	MAP_FILE_DIRECTORY	Added constant in hardcoded string's place	
Order Owner			

public static Order issueOrder(String[] p_commands, Player player) {
String l_Type = p_commands[0].toLowerCase();
Order l_Order = null;
if (l_Type.equals(Constants.DEPLOY_COMMAND)) {
l_Order = new Deploy();
l_Order.setOrderInfo(createDeployOrderInfo(p_commands, player));
}
return l_Order;
}
	public static Order issueOrder(String[] p_commands, Player player) {
	String l_OrderType = p_commands[0].toLowerCase();
	Order l_Order;
	
	switch (l_OrderType) {
	case "deploy":
	l_Order = new Deploy();
	l_Order.setOrderInfo(generateDeployInfo(p_commands, player));
	break;
	case "airlift":
	l_Order = new AirliftOrder();
	l_Order.setOrderInfo(generateAirliftOrderInfo(p_commands, player));
	
	break;
	case "bomb":
	l_Order = new BombOrder();
	l_Order.setOrderInfo(generateBombOrderInfo(p_commands, player));
	
	break;
	default:
	System.out.println("\nFailed to create an order due to invalid arguments");
	return null;
	}
	
	return l_Order;
	}
		Added switch statements in issue order to handle multiple cases other wise had to hard code every case	

Refactoring code		
Previous	Later	Description/Need of refactoring