REFACTORING DOCUMENT

1.Introduced Strategy Pattern

Earlier the Game Play was happening directly and only between the human players. Now the assign countries phase , reinforcement phase and gameplay phase all are happening automatically. New strategies are also implemented such as Random Player Strategy, Aggressive Player Strategy, Benevolent Player Strategy , Cheater Player Strategy.

ASSOCIATED TEST CASES.

1.**public void gameloop()**

2.**public void gameloop1()**

3.**public void checkStartUpPhase()**

4.**public void EndGametest()**

GameEngine.java | Player.java | TournamentGameEngine.java | TournamentMain.java | GameEngine1.java

```java
        while (end) {
            boolean an_order = true;

            do {
                for (String l_player : PLAYERS_LIST.keySet()) {
                    Player p = PLAYERS_LIST.get(l_player);
                    an_order = p.issue_order(order);
                    if (!an_order)
                        break;
                }
                for(String l_player: PLAYERS_LIST.keySet())
                {
                    if (PLAYERS_LIST.get(l_player).d_owned.size()==0)
                    {
                        PLAYERS_LIST.remove(l_player);
                        d_message = l_player+" you have lost the game. So you're out of the game!!!";
                        System.out.println(d_message);
                        l_observable.setMsg(d_message);
                    }
                }
                if(PLAYERS_LIST.size()==1)
                {
                    for(String player: PLAYERS_LIST.keySet()) {
                        d_message = player + " is the winner of the game!!!!!!!!";
                        System.out.println(d_message);
                        l_observable.setMsg(d_message);
                        end = false;
                        break;
                    }
                    break;
                }
                sMap.check();
```

```java
public void executeAllOrders() {
    System.out.println("===============BEGIN EXECUTING ALL ORDERS================");
    Order order;
    boolean still_more_orders = false;
    do {
        still_more_orders = false;
        for (String l_s : PLAYERS_LIST.keySet()) {
            Player p = PLAYERS_LIST.get(l_s);
            order = p.next_order();
            if (order != null) {
                still_more_orders = true;
                order.Execute(p);
            }
        }
    } while (still_more_orders);
    System.out.println("===============END EXECUTING ALL ORDERS==================");
}
```

2.Introduced Adapter Pattern

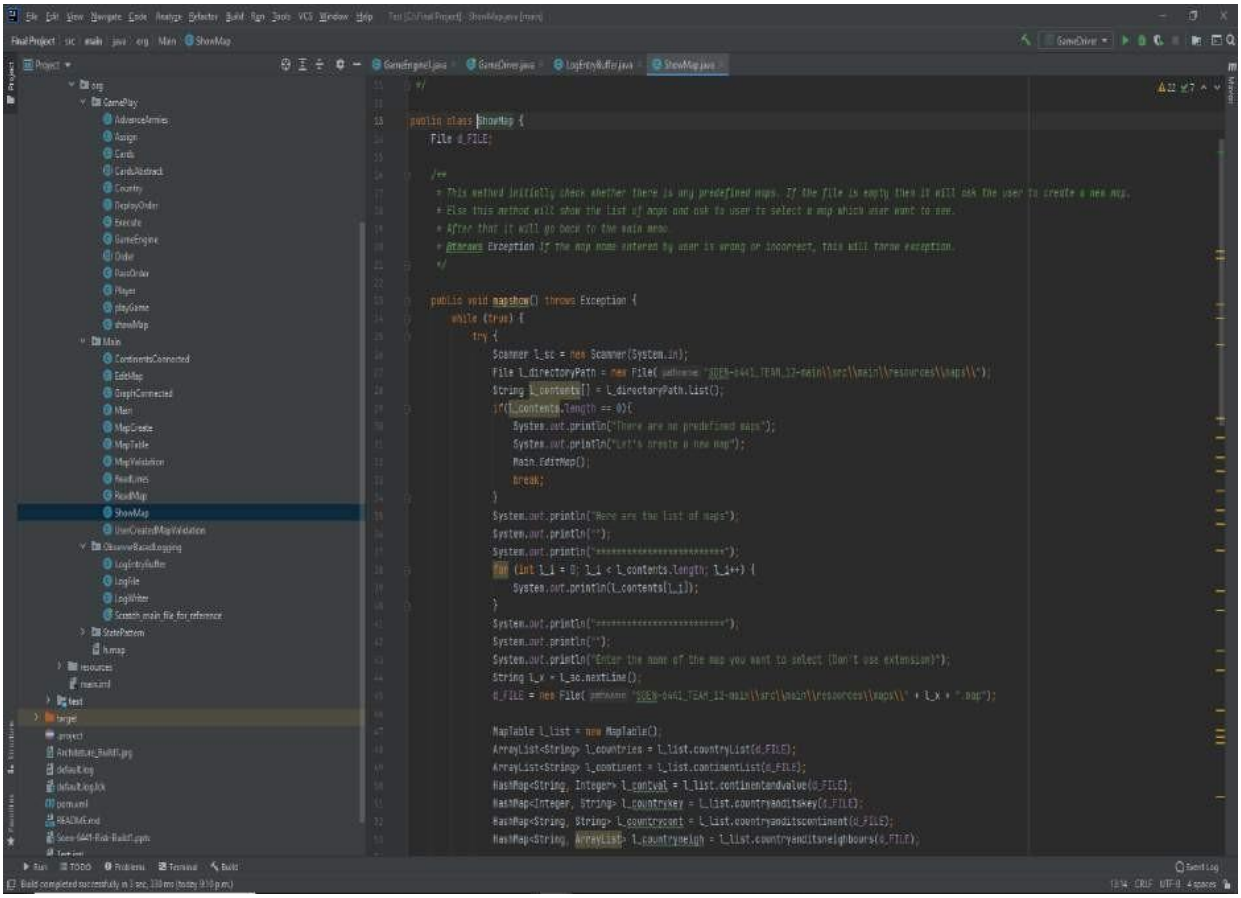
Previously, there was no adapter pattern, due to which loading of map was getting done in the ShowMap class only. But now we have introduced adapter pattern because of which the loading of map file is done separately in the Main class only.

Associated Test Cases.

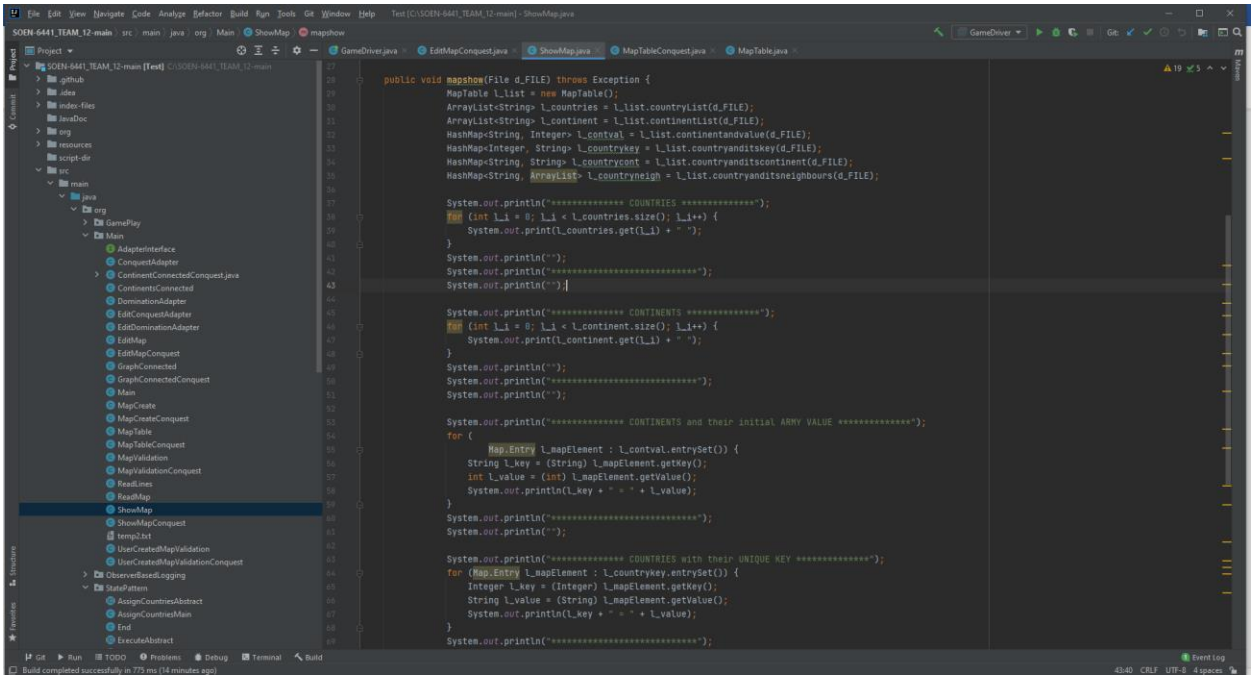
1.ConquestGraphConnected()

2.ConquestGraphNotConnected()

Before:-



After:-



```java
public void mapshow(File d_FILE) throws Exception {
        MapTable l_list = new MapTable();
        ArrayList<String> l_countries = l_list.countryList(d_FILE);
        ArrayList<String> l_continent = l_list.continentList(d_FILE);
        HashMap<String, Integer> l_contval = l_list.continentandvalue(d_FILE);
        HashMap<Integer, String> l_countrykey = l_list.countryanditskey(d_FILE);
        HashMap<String, String> l_countrycont = l_list.countryanditscontinent(d_FILE);
        HashMap<String, ArrayList> l_countryneigh = l_list.countryanditsneighbours(d_FILE);

        System.out.println("************* COUNTRIES *************");
        for (int l_i = 0; l_i < l_countries.size(); l_i++) {
            System.out.print(l_countries.get(l_i) + " ");
        }
        System.out.println("");
        System.out.println("***************************");
        System.out.println("");

        System.out.println("************* CONTINENTS *************");
        for (int l_i = 0; l_i < l_continent.size(); l_i++) {
            System.out.print(l_continent.get(l_i) + " ");
        }
        System.out.println("");
        System.out.println("***************************");
        System.out.println("");

        System.out.println("************* CONTINENTS and their initial ARMY VALUE *************");
        for (
                Map.Entry l_mapElement : l_contval.entrySet()) {
            String l_key = (String) l_mapElement.getKey();
            int l_value = (int) l_mapElement.getValue();
            System.out.println(l_key + " = " + l_value);
        }
        System.out.println("***************************");
        System.out.println("");

        System.out.println("************* COUNTRIES with their UNIQUE KEY *************");
        for (Map.Entry l_mapElement : l_countrykey.entrySet()) {
            Integer l_key = (Integer) l_mapElement.getKey();
            String l_value = (String) l_mapElement.getValue();
            System.out.println(l_key + " = " + l_value);
        }
        System.out.println("***************************");
```

3.Introduced Tournament Mode

Earlier only single Game mode was present ,now in Build 3 we have introduced Tournament Mode in this one can play multiple number of games on multiple numbers of maps.

Associated Test Cases

1. **runtestCompletefail()**
2. **runMapTest()**
3. **runTestPlayer()**
4. **runTestGame()**
5. **runTestTurns()**

GameEngine.java | Player.java | TournamentGameEngine.java | TournamentMain.java | GameEngine1.java

```java
            return "Wrong command";
        }

        for (int i = 0; i < listOfMapFiles.length; i++) {
            System.out.println("Map" + listOfMapFiles[i] + " : ");
            d_file = new File( pathname: "src/main/resources/maps/" + listOfMapFiles[i]);


            for (int j = 0; j < numberofgames; j++) {
                for(int t = 0 ; t<d_playersName.length -1 ; t++)
                {
                    PLAYERS_LIST.put(d_playersName[t], new Player(d_playersName[t]));
                }

                d_assign = new Assign(PLAYERS_LIST,d_country);
                d_assign.assignCountries(d_file);
                COUNTRIES = d_assign.d_country.COUNTRIESLIST;
                d_country = d_assign.d_country;

                play = new playGame(PLAYERS_LIST,d_country);
                play.playGameLoop();
                int k = j+1;
                gameState = gameEngine.gamePlay(d_file,maxnumberofturns,PLAYERS_LIST,d_country,listOfMapFiles[i],gameState);
                winners.add("Winner of game " + k +" on map " + listOfMapFiles[i] + " is " +gameState.get(listOfMapFiles[i]) +"\n");
            }

            System.out.println(winners + "\n");
        }
        return "Tournament Over";
    }
```

4.Introduced New Class to Check map validation of conquest maps.

In this build other than the domination worlds map we also have to play on conquest type maps and in order to play on them we also need to validate them so instead of using the old GraphConnected.java we have introduced new GraphConnectedConquest.java in order to handle the map validation of Conquest maps.

Associated Test Cases.

1.ConquestGraphConnected()

2.ConquestGraphNotConnected()

ContinentConnectedConquest.java

```java
     * if both are equal the continents are connected and if not the not connected.
     * @return boolean value whether a particulars maps continents are connected or not.
     * @throws Exception
     */
    public boolean ifContinentsConnected() throws Exception {

        MapTableConquest l_obj = new MapTableConquest();
        ArrayList<Integer> l_ContinentAndKey = l_obj.continentsKeyConquest(d_file);
        ArrayList<Integer> l_ContId = l_obj.CountriesContinentConquest(d_file);
        Collections.sort(l_ContId);
        GraphConnectedConquest l_check = new GraphConnectedConquest(d_file);

        if (l_check.ifGraphConnected() == true) {

            if (l_ContinentAndKey.equals(l_ContId)==true){
                return true;
            }


        }
        return false;
    }

    public HashMap<Integer,ArrayList> ifContinentCountriesConnected() throws Exception {
        MapTableConquest l_obj = new MapTableConquest();

        ArrayList<Integer> copy = new ArrayList<>();
        HashMap<Integer,ArrayList> nodes = new HashMap<>();
        HashMap<Integer,Integer> l_connect = l_obj.contConquest(d_file);
```

GraphConnectedConquest.java

```java
    /**
     * This method finally checks the visited boolean array and sees if any value is false in it.
     * If yes that means our graph is not connected and if no our graph is connected.
     * @return boolean variable boolean which is simple true or false.
     */
    public boolean ifGraphConnected() {
        int l_key_1 = d_countryNeigh.keySet().stream().findFirst().get();
        int p_start = l_key_1;
        dfs(p_start);
        d_visited[0] = true;
        for (int l_i = 0; l_i < d_realNodes.size(); l_i++) {
            int l_check = d_realNodes.get(l_i);
            boolean l_comp = d_visited[l_check];
            if(l_comp == false){
                return false;
            }
        }
        return true;
    }


    /**
     * This function checks if each country is a connected sub graph.
     * @return boolean whether the individual continent is connected or not.
     * @throws Exception as we are using ConnectedContinent class Functions
     */
    public boolean ContinentsCheck() throws Exception {
        ContinentsConnectedConquest d_obj = new ContinentsConnectedConquest(FILE);
        HashMap<Integer,ArrayList> l_list = d_obj.ifContinentCountriesConnected();
        for(Map.Entry<Integer, ArrayList> l_entry: l_list.entrySet()) {
            HashMap<Integer,ArrayList> l_check = d_borders.CountryIDConquest(FILE,l_entry.getValue());
```

5.Reduced the number of phases as now the game is running automatically.

Earlier it was asked from the user to enter commands on the console to assigncountries and also for the reinforment phase now all that is happening automatically since the introduction of adapter pattern.

ASSOCIATED TEST CASES.

1.**public void gameloop()**

2.**public void gameloop1()**

3.**public void checkStartUpPhase()**

4.**public void EndGametest()**

Before:-

```
            System.out.println("Bye!");
            return;
        }
        int l_i=0;
        l_keyboard.nextLine();
        do {
            System.out.println(" ==================================================");
            System.out.println("| #   PHASE                    : command           |");
            System.out.println(" ==================================================");
            System.out.println("| 1.  Show Map                 : showmap           |");
            System.out.println("| 2.  Edit Map                 : editmap           |");
            System.out.println("| 3.  GamePlay : load map      : loadmap           |");
            System.out.println("| 4.  GamePlay: set Player     : gameplayer        |");
            System.out.println("| 5.  Play:PlaySetup           : assign countries  |");
            System.out.println("| 6.  Play:Issue Order         : order             |");
            System.out.println("| 7.  Play:Execute Order       : execute           |");
            System.out.println("| 7.  Play:mapshow             : mapshow           |");
            System.out.println("| 9.  Any Phase                : end game          |");
            System.out.println("| 10. Any Phase                : next phase        |");
            System.out.println(" ==================================================");
```

After:-

GraphConnectedConquest.java | GameEngine1.java

```java
        }
        int l_i=0;
        l_keyboard.nextLine();
        do {
            System.out.println(" ==============================================");
            System.out.println("| #   PHASE                  : command          |");
            System.out.println(" ==============================================");
            System.out.println("| 1.  Show Map               : showmap          |");
            System.out.println("| 2.  Edit Map               : editmap          |");
            System.out.println("| 3.  GamePlay : load map    : loadmap          |");
            System.out.println("| 4.  GamePlay: set Player   : gameplayer       |");
            System.out.println("| 6.  Game Play: execution   : gameplay         |");
            System.out.println("| 5.  Any Phase              : end game         |");
            System.out.println("| 6.  Any Phase              : next phase       |");
            System.out.println(" ==============================================");
            System.out.println("enter a " + d_gamePhase.getClass().getSimpleName() + " phase command: ");
            MYCOMMAND = l_keyboard.nextLine();
            System.out.println(" ==============================================");

            switch (MYCOMMAND) {
                case ("showmap"):
                    d_gamePhase.ShowMaps();
                    break;
                case ("editmap"):
                    d_gamePhase.EditMap();
                    break;
                case ("exit"):
                    l_i=1;
```

6.Code Repitition in Editmap class

- There is validation call part in the EditMap class which is called thrice in the same class.
- We could have refactored this one too, but this is not an important part which we need to do so.
- Also, this is making the understandability of code much easier.

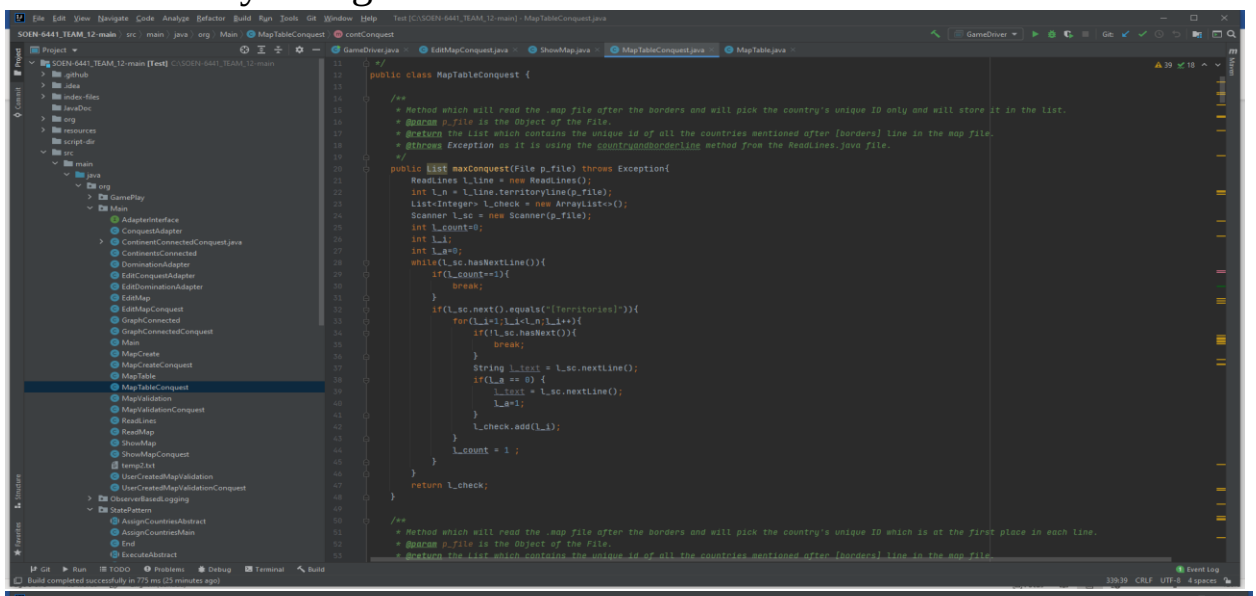
```
File l_file2 = new File(l_address);
l_flag_error=0;
while(true) {
    try {
        System.out.println("Enter validatemap command");
        String l_reply = d_SC.nextLine();
        if (l_reply.equalsIgnoreCase( anotherString: "validatemap")) {
            MapValidation l_validate = new MapValidation();
            l_validate.mapValidate(l_file2, l_countries, l_continent, l_cont_val, l_country_key, l_country_cont
            break;
        }
        else{
            System.out.println("Enter valid command");
        }
    } catch (Exception p_e) {
        p_e.printStackTrace();
    }
```

```
File l_file3 = new File(l_address);
l_flag_error = 0;

while(true) {
    try {
        System.out.println("Enter validatemap command");
        String l_reply = d_SC.nextLine();
        if (l_reply.equalsIgnoreCase( anotherString: "validatemap")) {
            MapValidation l_validate = new MapValidation();
            l_validate.mapValidate(l_file3, l_countries, l_continent, l_cont_val, l_country_key, l_country_cont
            break;
        }
        else{
            System.out.println("Enter valid command");
        }
    } catch (Exception p_e) {
        p_e.printStackTrace();
```

7.We could have used single MapTable.java class.

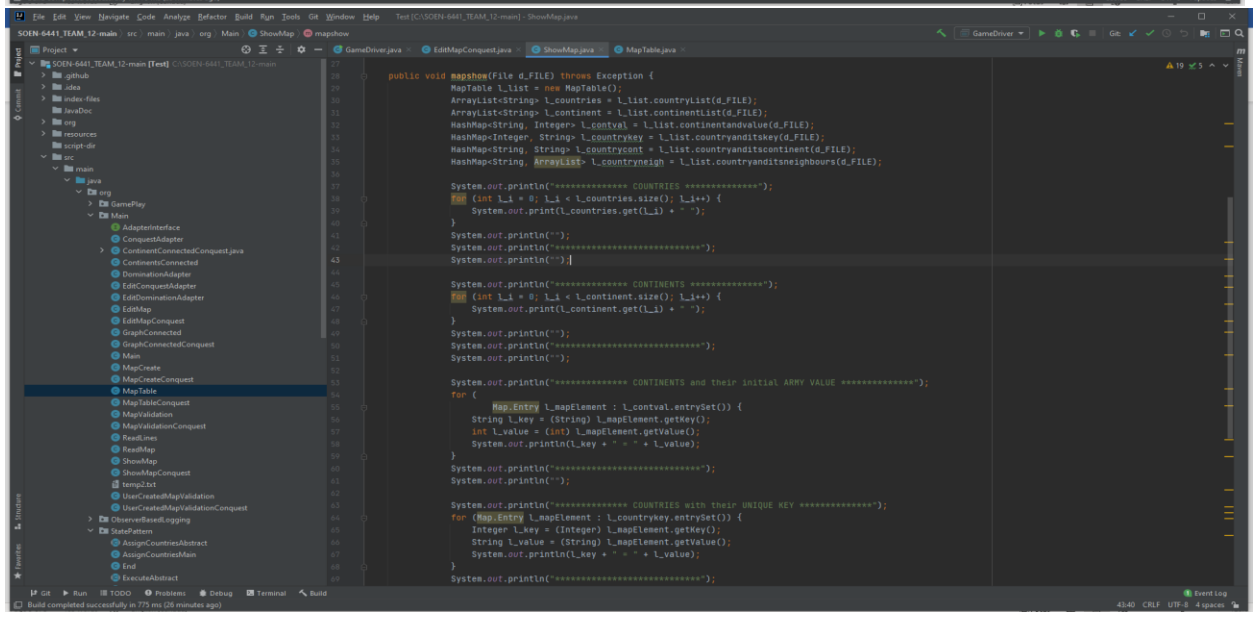
- In MapEditor phase we are using a class named MapTable to fetch information about the map file like the countries list, continents list.
- Previously, there was only one class but due to Adapter pattern now there are two map files i.e., domination map file and conquest map file. So, we have introduced two classes for that one is MapTable class and the other one is MapTableConquest. We could have used only a single class.



```
public class MapTableConquest {

    /**
     * Method which will read the .map file after the borders and will pick the country's unique ID only and will store it in the list.
     * @param p_file is the Object of the File.
     * @return the list which contains the unique id of all the countries mentioned after [borders] line in the map file.
     * @throws Exception as it is using the countryandborderline method from the ReadLines.java file.
     */
    public list maxConquest(File p_file) throws Exception{
        ReadLines l_line = new ReadLines();
        int l_n = l_line.territoryline(p_file);
        List<Integer> l_check = new ArrayList<>();
        Scanner l_sc = new Scanner(p_file);
        int l_count=0;
        int l_i;
        int l_a=0;
        while(l_sc.hasNextLine()){
            if(l_count==1){
                break;
            }
            if(l_sc.next().equals("[Territories]")){
                for(l_i=1;l_i<l_n;l_i++){
                    if(!l_sc.hasNext()){
                        break;
                    }
                    String l_text = l_sc.nextLine();
                    if(l_a == 0) {
                        l_text = l_sc.nextLine();
                        l_a=1;
                    }
                    l_check.add(l_i);
                }
                l_count = 1 ;
            }
        }
        return l_check;
    }

    /**
     * Method which will read the .map file after the borders and will pick the country's unique ID which is at the first place in each line.
     * @param p_file is the Object of the File.
     * @return the list which contains the unique id of all the countries mentioned after [borders] line in the map file.
```



```
    public void mapshow(File d_FILE) throws Exception {
            MapTable l_list = new MapTable();
            ArrayList<String> l_countries = l_list.countryList(d_FILE);
            ArrayList<String> l_continent = l_list.continentList(d_FILE);
            HashMap<String, Integer> l_contval = l_list.continentandvalue(d_FILE);
            HashMap<Integer, String> l_countrykey = l_list.countryanditskey(d_FILE);
            HashMap<String, String> l_countrycont = l_list.countryanditscontinent(d_FILE);
            HashMap<String, ArrayList> l_countryneigh = l_list.countryanditsneighbours(d_FILE);

            System.out.println("************* COUNTRIES *************");
            for (int l_i = 0; l_i < l_countries.size(); l_i++) {
                System.out.print(l_countries.get(l_i) + " ");
            }
            System.out.println("");
            System.out.println("***************************");
            System.out.println("");

            System.out.println("************* CONTINENTS *************");
            for (int l_i = 0; l_i < l_continent.size(); l_i++) {
                System.out.print(l_continent.get(l_i) + " ");
            }
            System.out.println("");
            System.out.println("***************************");
            System.out.println("");

            System.out.println("************* CONTINENTS and their initial ARMY VALUE *************");
            for (
                    Map.Entry l_mapElement : l_contval.entrySet()) {
                String l_key = (String) l_mapElement.getKey();
                int l_value = (int) l_mapElement.getValue();
                System.out.println(l_key + " = " + l_value);
            }
            System.out.println("***************************");
            System.out.println("");

            System.out.println("************* COUNTRIES with their UNIQUE KEY *************");
            for (Map.Entry l_mapElement : l_countrykey.entrySet()) {
                Integer l_key = (Integer) l_mapElement.getKey();
                String l_value = (String) l_mapElement.getValue();
                System.out.println(l_key + " = " + l_value);
            }
            System.out.println("***************************");
```

8. Introduced new GamePlay phase in the game.

Since the introduction of the strategy pattern all the game has become automatic. So to do that we have introduced the new phase called gameplay phase once the player names has been selected then the user will time the gameplay command and the game will automatically run and at the end give us the winner.

GamePlayMain.java

```java
package org.StatePattern;

import org.GamePlay.GameEngine;

public class GamePlayMain extends GamePlayAbstract{
    GameEngine d_ge;
    GameEngine1 d_ge1;

    /**
     * Constructor to initialize the values.
     * @param p_ge is the Object of the class GameEngine1.
     * @param p_ge1 is the Object of the class GameEngine
     */
    public GamePlayMain(GameEngine1 p_ge,GameEngine p_ge1) {
        super(p_ge);
        this.d_ge1 = p_ge;
        this.d_ge = p_ge1;
    }

    /**
     * Method to start the GameEngine.
     */
    @Override
    public void gamePlay() { d_ge.gamePlay(d_ge.FILE); }

    /**
     * Method to change the Phase. If the size of the
     */
```

9. Player class is modified.

Earlier in issue order functions we were simply issueing orders from the console now in issue order function players of respective strategies will give the order accordingly.

GamePlayMain.java Player.java

```java
/**
 *Player Class containing all the stats of the Player Class
 */
public class Player implements Serializable {
    private static final long serialVersionUID= 3710154222911855669L;
    public ArrayList<Country> d_owned = new ArrayList<~>();
    public Queue<Order> d_orders = new LinkedList<~>();// Queue only
    public Integer d_armiesNum;
    public String d_name;
    public int d_continentValue; // Continent Control Value ;
    public ArrayList<String> d_cards = new ArrayList<>();
    public ArrayList<String> d_negotiate = new ArrayList<>();
    PlayerStrategy strategy;


    /**
     * Parameterised Constructor to initilize Player Object by name .
     * @param p_name  Player Name
     */
    public Player(String p_name) { this.d_name = p_name; }

    public void setStrategy(PlayerStrategy p_strat) { strategy = p_strat; }

    /**
```

```java
/**
 *function to issue order .
 * @param order issue order object
 */
public boolean issue_order(Order order)
{
    order = strategy.createOrder();
    if (order != null) {
        d_orders.add(order);
        return true;
    }
    return false;
}
```

10. playGame.java is Modified

Earlier in maingameloop() function we were simply adding the orders to the orders list now here we are returning orders to the player class and then respective player is saving the orders in their list.

```
System.out.println("**Deploy**\n**Advance**\n**Cards**\n    -Bomb\n    -Blockade\n    -Airlift\n    -Negotiate\n**Pass**\n**Sho
System.out.println("*************************");
Scanner l_sc = new Scanner(System.in);
String l_command = l_sc.nextLine();
String l_commandSplit[] = l_command.split( regex: " ");
if (l_commandSplit[0].equalsIgnoreCase( anotherString: "deploy")) {
    if (l_commandSplit.length == 3) {
        String l_countryId = l_commandSplit[1];
        if (d_country.COUNTRIESLIST.containsKey(l_countryId)) {
            String l_regex = "\\d+";
            if (l_commandSplit[2].matches(l_regex)) {
                int l_armiesToPlace = Integer.parseInt(l_commandSplit[2]);
                if (l_armiesToPlace > 0) {
                    l_flag_1 = false;
                    return new DeployOrder(l_countryId, l_armiesToPlace, d_country);
                } else {
                    System.out.println("Negative Army count not allowed");
                    continue;
                }
            } else {
                System.out.println("The army number should be an integer");
                continue;
            }
        } else {
            System.out.println("The country that you entered doesn't exist in the Map");
            continue;
        }
    } else {
        System.out.println("Your deploy command should be like 'deploy countryName armyToPlace'\n");
```

11.In Cards we can encapsulate each card functionality into a function and simply call it.

- We have simple just implemented the functionality of each card directly in the respective switch cases.
- We can create functions for each card and then simply call it under the switch case.

```
case "blockade":
    if(p_p.d_cards.contains("BLOCKADE") && p_p.d_owned.contains(d_country.COUNTRIESLIST.get(d_CountryID))) {
        d_country.COUNTRIESLIST.get(d_CountryID).d_numOfArmiesPlaced *= 3;
        p_p.d_owned.remove(d_country.COUNTRIESLIST.get(d_CountryID));
        d_country.COUNTRIESLIST.get(d_CountryID).d_owner = "Neutral";
        l_message = d_CountryID + " has been neutralized , now nobody owns this country";
        System.out.println(l_message);
        l_observable.setMsg(l_message);
        int l_indexCardName=-1;
        for(String l_cardName: p_p.d_cards)
        {
            l_indexCardName+=1;
            if(l_cardName.equalsIgnoreCase( anotherString: "BLOCKADE"))
            {
                p_p.d_cards.remove(l_indexCardName);
                break;
            }
        }
    }
    else{
        l_message = "You don't have the BLOCKADE card.";
        System.out.println(l_message);
        l_observable.setMsg(p_p+" "+l_message);
    }
    break;
case "airlift":
    if(p_p.d_cards.contains("AIRLIFT")) {
        if (!p_p.d_negotiate.contains(d_country.COUNTRIESLIST.get(d_CountryTo).d_owner)) {
            if (d_country.COUNTRIESLIST.get(d_CountryFrom).d_numOfArmiesPlaced >= d_armiesToAirlift) {
                d_obj = new AdvanceArmies(d_CountryFrom, d_CountryTo, d_armiesToAirlift, d_country, d_card);
                d_obj.Execute(p_p);
```

12.Invalid Command Handling can be encapsulated inside a function in playGame class.

- No when the user types command ,to validate this command all the handling is done using if-else statements.

We can do this handling inside a function and simply call it at the appropriate place

```
} else if (l_commandSplit[0].equalsIgnoreCase( anotherString: "bomb")) {
    if (l_commandSplit.length == 2) {
        String l_countryId = l_commandSplit[1];
        if (d_country.COUNTRIESLIST.containsKey(l_countryId)) {
            if (!l_p.d_owned.contains(d_country.COUNTRIESLIST.get(l_countryId))) {
                l_flag_1 = false;
                Cards l_card = new Cards(l_countryId, l_commandSplit[0], d_country);
                l_p.issue_order(l_card);
            } else {
                System.out.println("Cannot Bomb its own country");
                continue;
            }
        } else {
            System.out.println("Not a Valid Country");
            continue;
        }
    } else {
        System.out.println("Not a Valid Command");
        continue;
    }
} else if (l_commandSplit[0].equalsIgnoreCase( anotherString: "advance")) {
    if (l_commandSplit.length == 4) {
        String l_country_from = l_commandSplit[1];
        String l_country_to = l_commandSplit[2];
        String l_armies = l_commandSplit[3];
        String l_card = "empty";
        int l_armiesToAdvance = Integer.parseInt(l_armies);
        if (d_country.COUNTRIESLIST.containsKey(l_country_from) && d_country.COUNTRIESLIST.containsKey(l_country
            if (l_p.d_owned.contains(d_country.COUNTRIESLIST.get(l_country_from))) {
                if (d_country.COUNTRIESLIST.get(l_country_from).d_neighbours.contains(l_country_to)) {
                    if (l_p.d_armiesNum >= l_armiesToAdvance) {
```

13.Code repetition in continent connectivity check

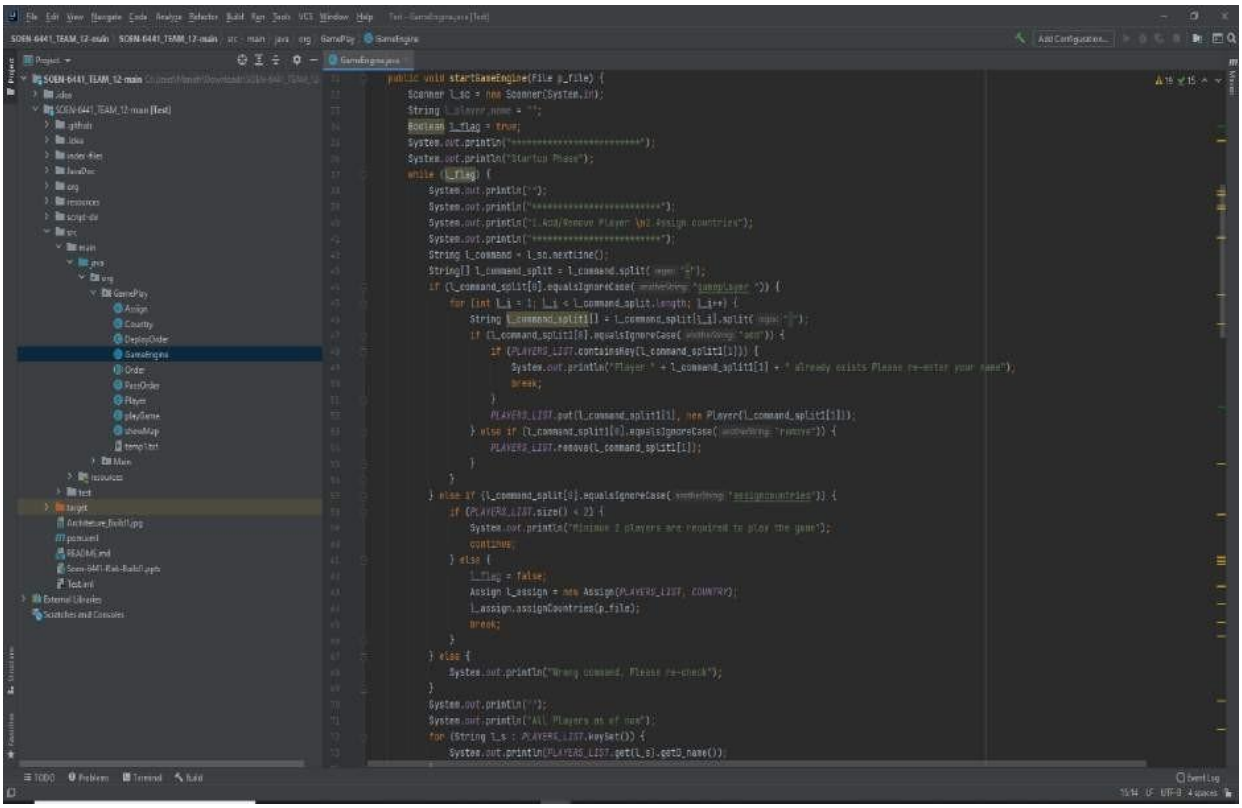
- In Graph Connected class for countries connectivity there is dfs() function and finally checking the connectivity ifGraphConnected() function is there.
- In ContinentCheck both above functions can be used but instead whole code is written from scratch.
- This can be avoided by making dfs() and ifGraphConnected() function generalized.

GraphConnected.java

```java
    public boolean ContinentsCheck() throws Exception {
        ContinentsConnected d_obj = new ContinentsConnected(FILE);
        HashMap<Integer,ArrayList> l_list = d_obj.ifContinentCountriesConnected();
        for(Map.Entry<Integer, ArrayList> l_entry: l_list.entrySet()) {
            HashMap<Integer,ArrayList> l_check = d_borders.CountryNeighbours(FILE,l_entry.getValue());
            List<Integer> l_nodes = l_entry.getValue();
            ArrayList<Integer> l_empty = new ArrayList<>();
            if(l_nodes.size() == 1){
                l_check.put(l_nodes.get(0),l_empty);
            }
            int l_max = Collections.max(l_nodes);
            d_scan = new boolean[l_max + 1];
            boolean [] l_traversed = new boolean[l_max +1];
            int l_key_1 = l_check.keySet().stream().findFirst().get();
            int p_start = l_key_1;
            d_stack.push(p_start);
            l_traversed[p_start] = true;
            d_country_index = l_nodes;
            d_scan[0] = true;
            for (int l_i = 0; l_i<d_country_index.size(); l_i++){
                d_scan[d_country_index.get(l_i)]= true;
            }
            for (int l_j = 0; l_j<l_nodes.size(); l_j++){
                if(l_check.containsKey(l_nodes.get(l_j))==false){
                    int l_note = l_nodes.get(l_j);
                    l_check.put(l_note,l_empty);
                }
            }
            int l_count = 0;
            while (true) {
                if (d_stack.isEmpty() == true) {
                    break;
```

14.Changes in the GameEngine class

- We have broken the continuity of the game to introduce the state pattern in the game.
- Previously, there was a loop to add player and assign countries together but now we have now broken this continuity so as to create different phases for every part.



15.Add Card Function is never used in Player Class

- add_card() function inside player class is never used.
- Instead Arraylist d_cards is directly accessed in various parts of our program.
- Either we can remove this function or we can use this function in order to access d_cards arraylist where it is accessed in our code.

```
/**
 * Method to add the cards to the list.
 * @param cards is the Name of the card that player has won.
 */
public void add_card(String cards) { this.d_cards.add(cards); }
```