SOEN 6441 (Advance Programming Practices)

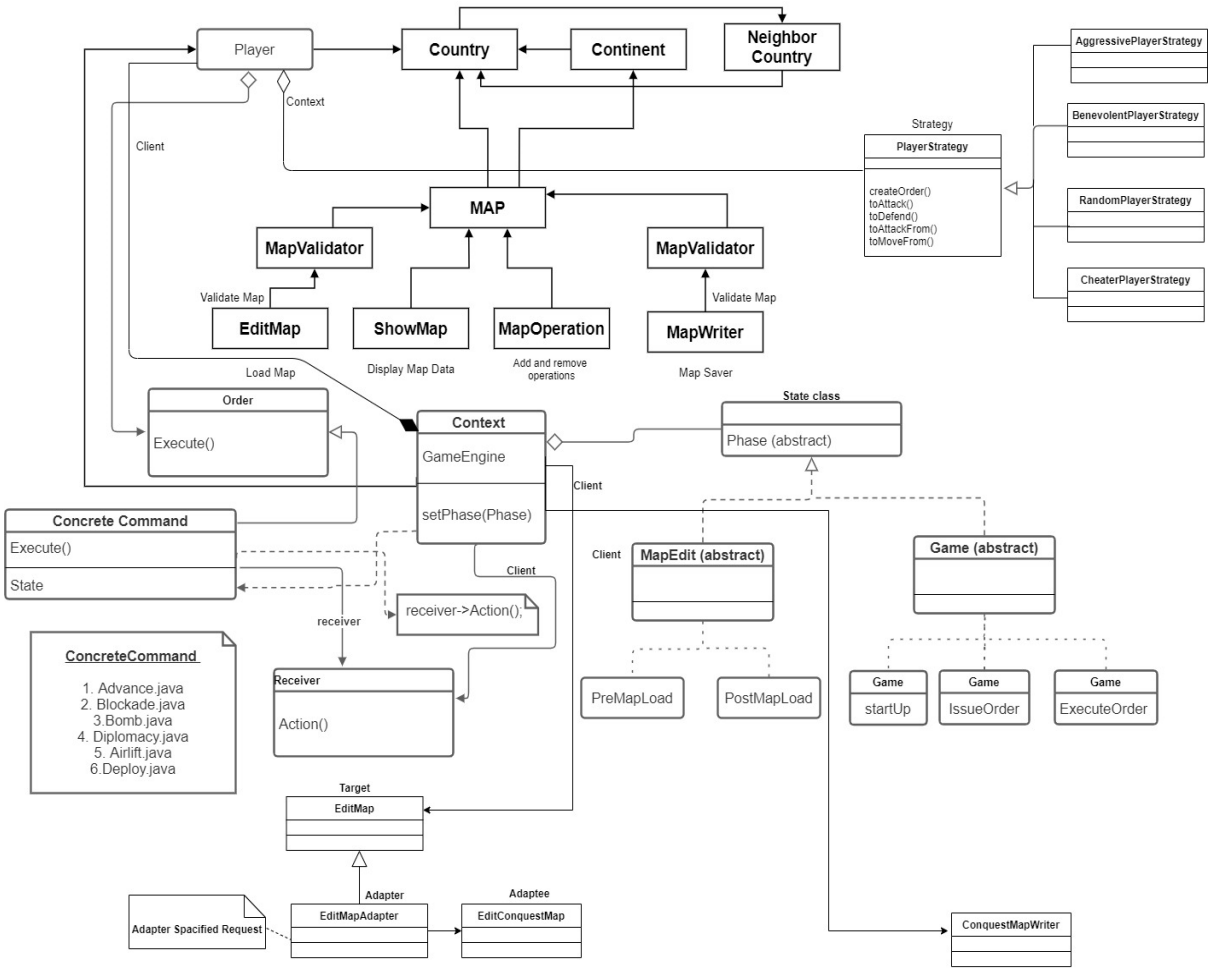
Build - 3

Professor: Dr. Joey Paquet

Team 19

Chirag Patel	**40160656**
Harsh Patel	**40165709**
Harshil Patel	**40163431**
Parth Navsariwala	**40178800**
Rishabh Patel	**40170811**

❖ Architectural Diagram:

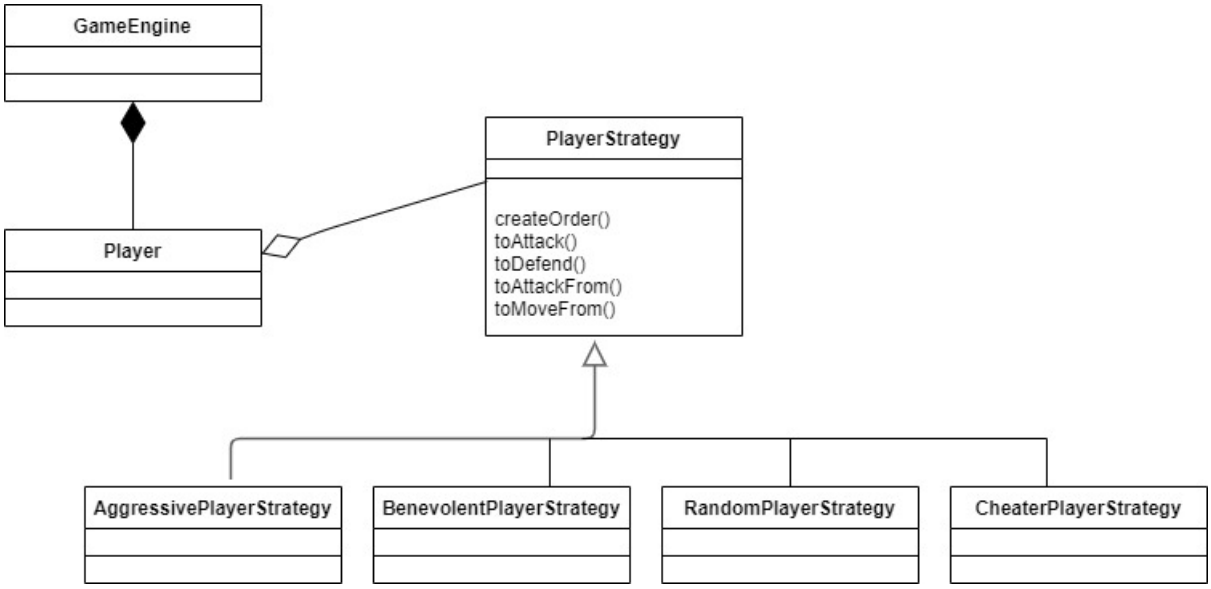


Architectural Diagram

- In build 3 we have main **5 major changes** in our existing code which are as below: -

1. **Implementation of Strategy Pattern**
2. **Implementation of Adapter Pattern**
3. **Implementation of Single mode & Tournament Mode**
4. **Implementation of Game save/load**
5. **Refactoring existing code**

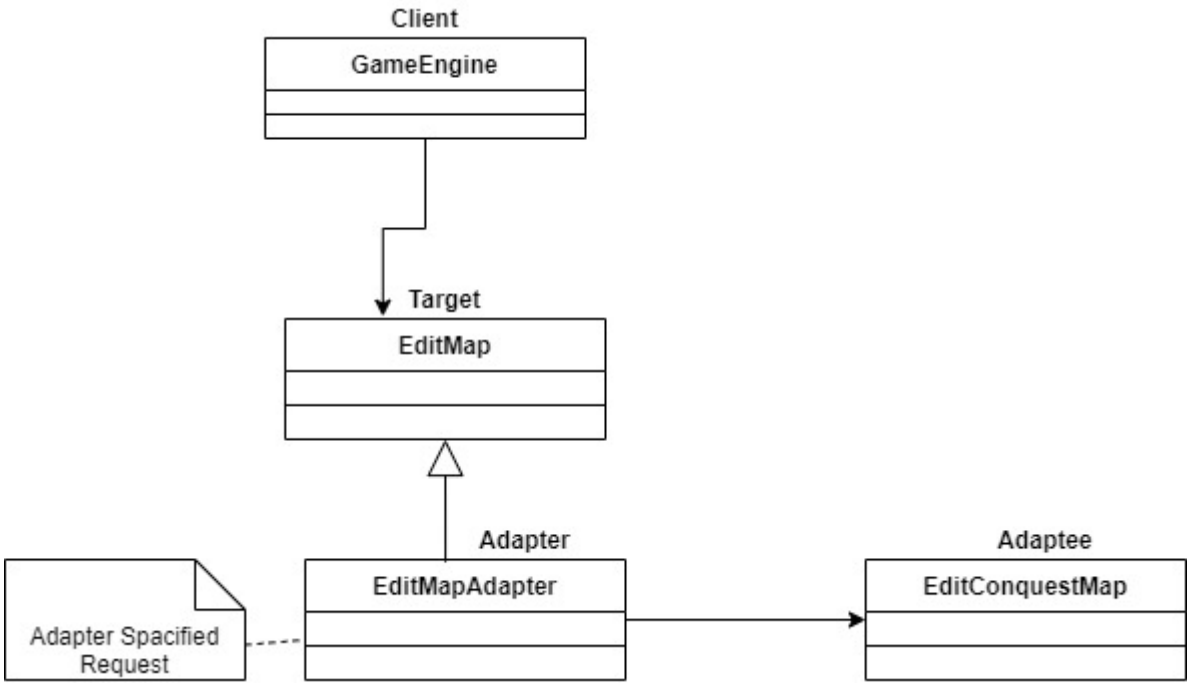
❖ Strategy Pattern:



Strategy Pattern

- Here GameEngine works as a client.
- Player class works as a context.
- PlayerStrategy works as a strategy class and it contain mainly 5 methods
    - CreateOrder()
    - toAttack()
    - toDefend()
    - toAttackFrom()
    - toMoveFrom()
- There are mainly four concrete strategy which are as below:
    - AggressivePlayerStrategy,java
    - BenevolentPlayerStrategy,java
    - RandomPlayerStrategy.java
    - CheaterPlayerStrategy.java

❖ Adapter Pattern



Adapter Pattern

- Here GameEngine class work as a client.
- Here EditMap class work as a target.
- Here EditMapAdapter class work as an adapter.
- Here EditConquestMap class work as an adaptee.

Possible Refactoring Targets:

Listed below are 15 potential refactoring targets:

1. Implement Strategy Pattern

2. Implement Adapter Pattern

3. Modify gameplayer command to pass strategy as an argument

4. Modify savemap command to pass Map type (Conquest or Domination) to save map

5. Refactored editConquestMap

6. Remove ContinentID from Country Class since BelongsToContinent is already present. [Country.java]

7. Remove getPlayerFromPlayerID since it is not used anywhere. [Player.java]

8. Rename getCountryFromCountryName() to getCountry(). [Country.java]

9. Rename getContinentFromContinentName() to getContinent(). [Continent.java]

10. Merge displayEditorMap() and displayGameMap() into one. [ShowMap.java]

11. Rename d_PlayerList to Players to store all the players of the game. [Main.java]

12. Rename removeContinentFromContinentList() to removeContinent(). [Map.java]

13. Rename addContinentToContinentList() to addContinent(). [Map.java]

14. Rename removeCountryFromCountryList() to removeCountry(). [Continent.java]

15. Rename addCountryToCountryList() to addCountry(). [Continent.java]

Refactoring: -

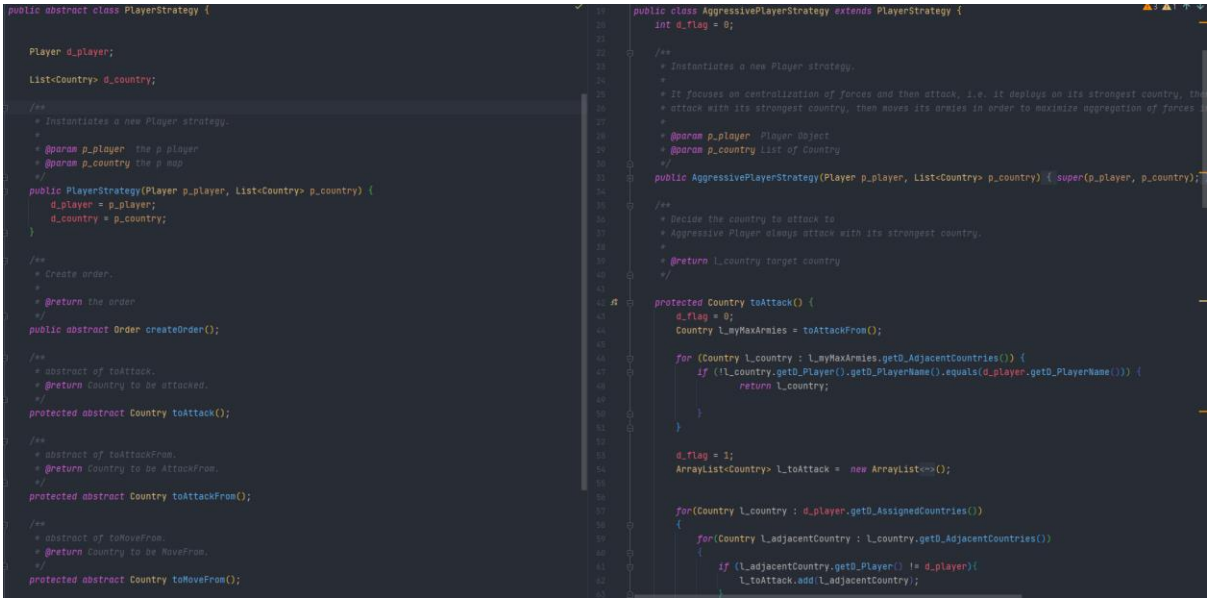
1. Implemented Strategy Pattern

   Refactored our current user-driven player code so that the implementation of the Player's issueOrder() method's behavior is using the Strategy pattern. Then, during the main development phase, implemented different computer player behaviors using the Strategy pattern, where the strategies provide varying behavior that support the Player class to expose varying behavior when executing the issueOrders() method

```java
/**
 * Instantiates a new Player.
 *
 * @param p_PlayerName Name of the player
 */
public Player(String p_PlayerName) {
    super();
    d_playerName = p_PlayerName;
    d_armies = 0;
    d_assignedCountries = new ArrayList<Country>();
    d_orderList = new ArrayList<>();
    testOrderList = new ArrayList<>();
    d_diplomacyPlayerList = new ArrayList<>();
    d_cardList = new ArrayList<>();
}
```

```java
public Player(String p_PlayerName, String p_playerStrategyType) {
    super();
    d_playerName = p_PlayerName;
    d_armies = 0;
    d_assignedCountries = new ArrayList<Country>();
    d_orderList = new ArrayList<>();
    testOrderList = new ArrayList<>();
    d_diplomacyPlayerList = new ArrayList<>();
    d_cardList = new ArrayList<>();
    d_playerStrategyType = p_playerStrategyType;

    if(d_playerStrategyType.equals("human"))
    {
        d_isHuman = true;
    }
    else {
        d_isHuman = false;
    }
    d_isHuman = false;

    switch (p_playerStrategyType){
        case "human":
            d_isHuman = true;
            break;

        case "aggressive":
            setD_playerStrategy(new AggressivePlayerStrategy( p_player: this, d_Map.getCountryListOfMap() ));
            break;

        case "benevolent":
            setD_playerStrategy(new BenevolentPlayerStrategy( p_player: this, d_Map.getCountryListOfMap() ));
            break;

        case "cheater":
            setD_playerStrategy(new CheaterPlayerStrategy( p_player: this, d_Map.getCountryListOfMap() ));
            break;

        case "random":
            setD_playerStrategy(new RandomPlayerStrategy( p_player: this, d_Map.getCountryListOfMap() ));
            break;

    }
```

1.1 Constructor added in Player Class to store strategy type (Before and After)



1.2 Abstract Strategy Class and Concrete Aggressive Strategy Class

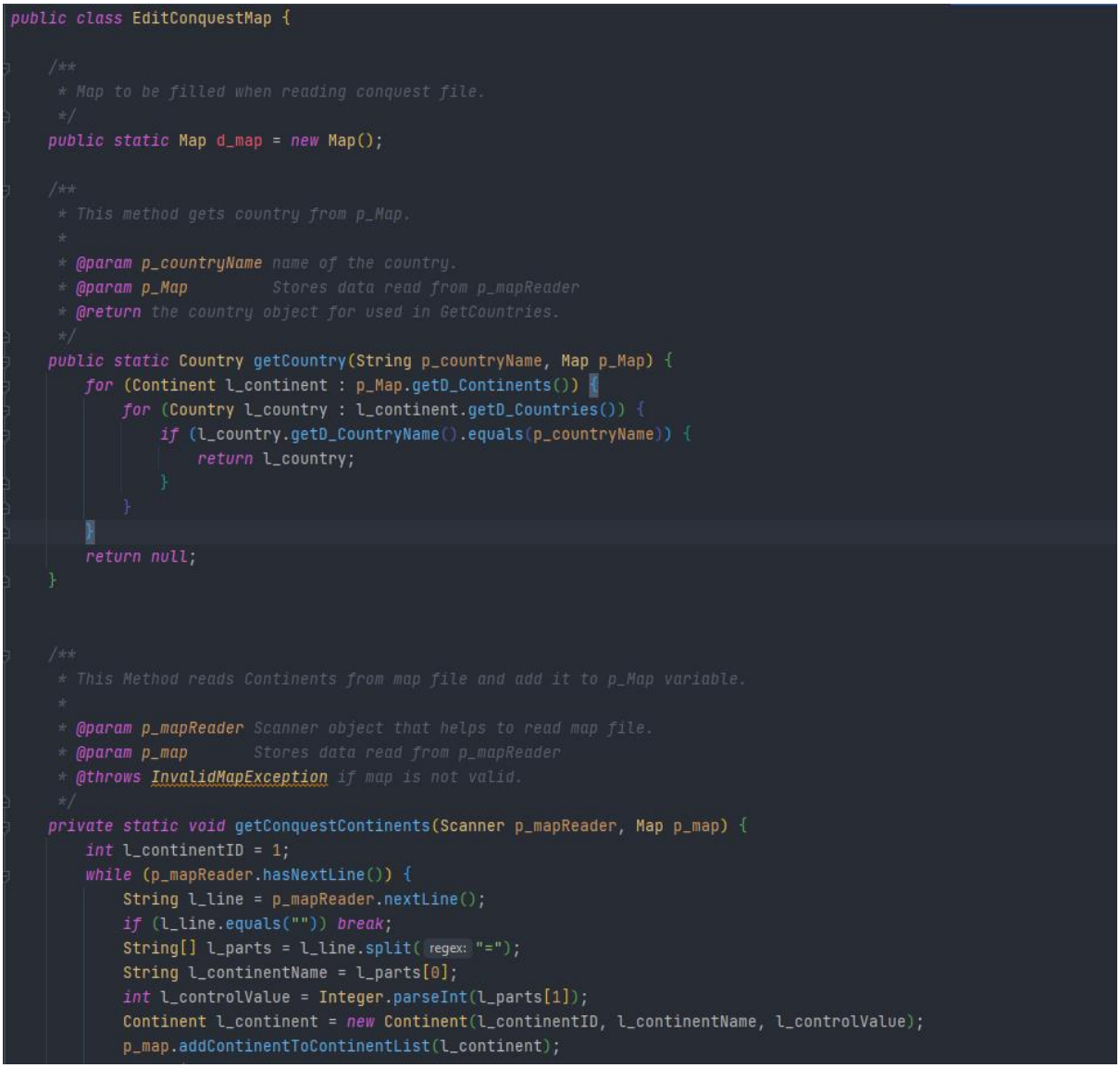
Tests : Created tests to check the validity of all kinds of strategies and their order creation.

2. Implemented Adapter Pattern

Refactored our code to use the Adapter pattern to enable the application to read/write from/to a file using the “conquest” game map format. The Game is be able to decide to use either the original “domination” file reader or the “conquest” file reader adapter when a file is opened, depending on the file type. When a map file is saved, the user is given the option as to which file format to use as output.

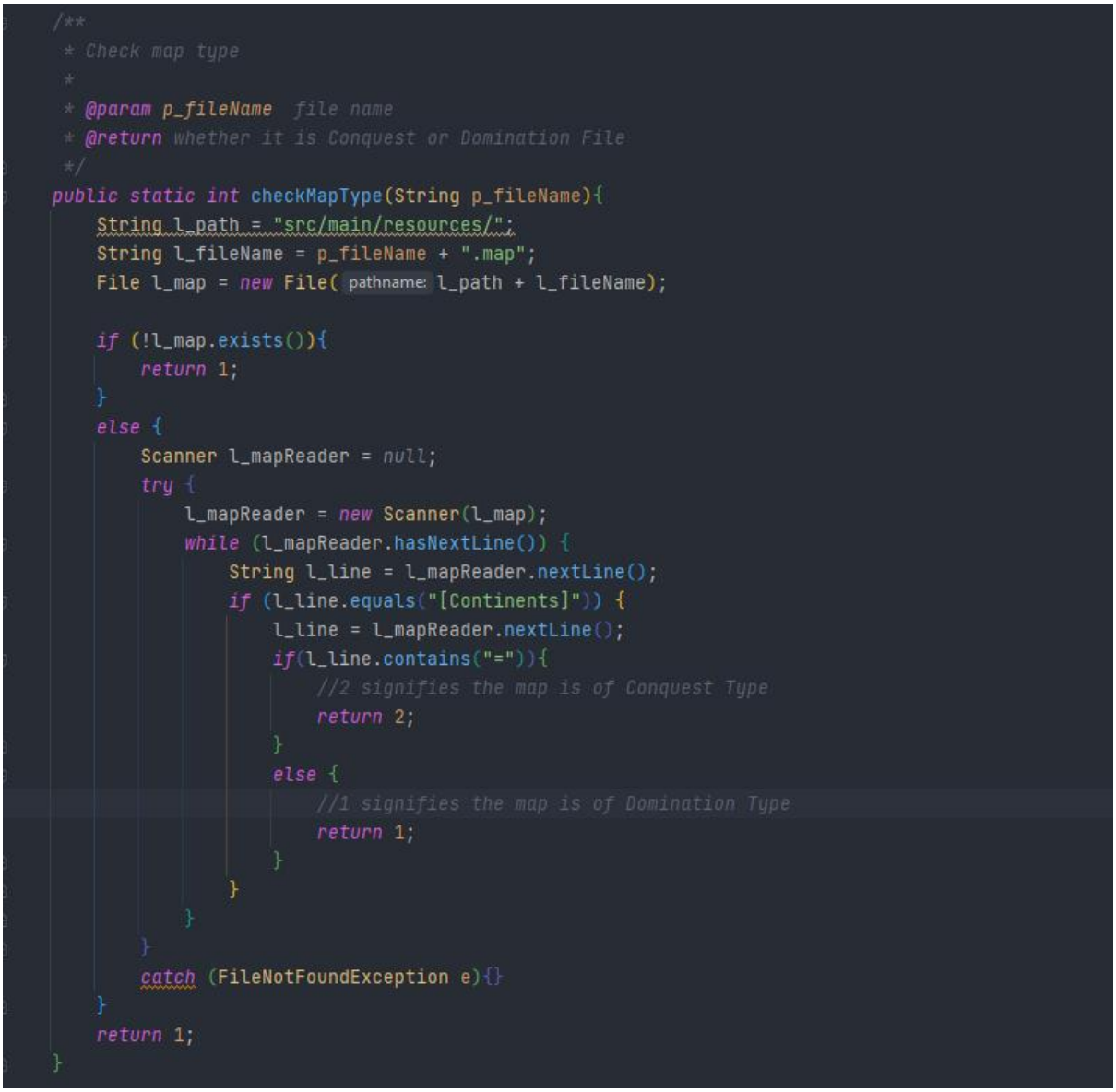
```java
public class EditMapAdapter extends EditMap {

    private static EditConquestMap d_editConquestMap;

    /**
     * Parameterized constructor for EditMapAdapter.
     *
     * @param p_editConquestMap object of LoadConquestMap
     */
    public EditMapAdapter(EditConquestMap p_editConquestMap) {
        this.d_editConquestMap = p_editConquestMap;
    }

    /**
     * Overridden editMap method of EditMap class
     *
     * @param p_fileName File name of the conquest file
     * @return Map object to be used in the game
     * @throws InvalidMapException if Map is invalid.
     */
    public static Map editMap(String p_fileName) throws InvalidMapException {

        String l_path = "src/main/resources/";
        String l_fileName = p_fileName + ".map";
        File l_map = new File( pathname: l_path + l_fileName);
        if (l_map.exists())
            return d_editConquestMap.loadConquestMap(p_fileName);
        return d_editConquestMap.createConquestMap(p_fileName);
    }
}
```

2.1 EditMapAdapter Class



```java
public class EditConquestMap {

    /**
     * Map to be filled when reading conquest file.
     */
    public static Map d_map = new Map();

    /**
     * This method gets country from p_Map.
     *
     * @param p_countryName name of the country.
     * @param p_Map         Stores data read from p_mapReader
     * @return the country object for used in GetCountries.
     */
    public static Country getCountry(String p_countryName, Map p_Map) {
        for (Continent l_continent : p_Map.getD_Continents()) {
            for (Country l_country : l_continent.getD_Countries()) {
                if (l_country.getD_CountryName().equals(p_countryName)) {
                    return l_country;
                }
            }
        }
        return null;
    }


    /**
     * This Method reads Continents from map file and add it to p_Map variable.
     *
     * @param p_mapReader Scanner object that helps to read map file.
     * @param p_map       Stores data read from p_mapReader
     * @throws InvalidMapException if map is not valid.
     */
    private static void getConquestContinents(Scanner p_mapReader, Map p_map) {
        int l_continentID = 1;
        while (p_mapReader.hasNextLine()) {
            String l_line = p_mapReader.nextLine();
            if (l_line.equals("")) break;
            String[] l_parts = l_line.split( regex: "=");
            String l_continentName = l_parts[0];
            int l_controlValue = Integer.parseInt(l_parts[1]);
            Continent l_continent = new Continent(l_continentID, l_continentName, l_controlValue);
            p_map.addContinentToContinentList(l_continent);
```

2.2 Snippet of newer ConquestMapReader class



```java
/**
 * Check map type
 *
 * @param p_fileName  file name
 * @return whether it is Conquest or Domination File
 */
public static int checkMapType(String p_fileName){
    String l_path = "src/main/resources/";
    String l_fileName = p_fileName + ".map";
    File l_map = new File( pathname: l_path + l_fileName);

    if (!l_map.exists()){
        return 1;
    }
    else {
        Scanner l_mapReader = null;
        try {
            l_mapReader = new Scanner(l_map);
            while (l_mapReader.hasNextLine()) {
                String l_line = l_mapReader.nextLine();
                if (l_line.equals("[Continents]")) {
                    l_line = l_mapReader.nextLine();
                    if(l_line.contains("=")){
                        //2 signifies the map is of Conquest Type
                        return 2;
                    }
                    else {
                        //1 signifies the map is of Domination Type
                        return 1;
                    }
                }
            }
        }
        catch (FileNotFoundException e){}
    }
    return 1;
}
```

2.3 Checking format of Map and reading the map accordingly

Tests: Created Tests to load both Conquest and Domination Map files

3. Modify gameplayer command to pass strategy as an argument.

Previously gameplayer command only had one argument for both add and remove options. Since Strategy pattern is introduced, gameplayer command has been modified to take two arguments for add option. The second argument is used to set the strategy type of the player added.

```java
public static void gamePlayerCommand(List<String> p_argumentTokens) {

    String l_playerName;
    int l_flag = 0;

    for (int i = 0; i < p_argumentTokens.size(); i++) {
        if (p_argumentTokens.get(i).equals("-add")) {
            try {
                l_playerName = p_argumentTokens.get(++i);

                for (Player l_player : d_PlayerList) {
                    if (l_player.getD_PlayerName().equals(l_playerName)) {
                        System.out.println("Player with player name " + l_playerName + " already exists. Try again with different name.");
                        d_Log.notify( p_data: "Player with player name " + l_playerName + " already exists. Try again with different name.");
                        l_flag = 1;
                        break;
                    }
                }

                if (l_flag == 1) {
                    l_flag = 0;
                    continue;
                }

                System.out.println("Player Name: " + l_playerName + " has joined the Game");

                d_Log.notify( p_data: "Player Name: " + l_playerName + " has joined the Game");
                Player l_player = new Player(l_playerName);
                d_PlayerList.add(l_player);


            } catch (Exception e) {
                System.out.println("Wrong number of Arguments provided. add option has 1 arguments");
                d_Log.notify( p_data: "Wrong number of Arguments provided. add option has 1 arguments");
                return;
            }
```

```java
public static void gamePlayerCommand(List<String> p_argumentTokens) {

    String l_playerName, l_playerStrategy;
    int l_flag, l_flag1;

    for (int i = 0; i < p_argumentTokens.size(); i++) {
        l_flag = 0;
        l_flag1 = 0;

        if (p_argumentTokens.get(i).equals("-add")) {
            try {
                l_playerName = p_argumentTokens.get(++i);
                l_playerStrategy = p_argumentTokens.get(++i);

                for (Player l_player : d_PlayerList) {
                    if (l_player.getD_PlayerName().equals(l_playerName)) {
                        System.out.println("Player with player name " + l_playerName + " already exists. Try again with different name.");
                        d_Log.notify( p_data: "Player with player name " + l_playerName + " already exists. Try again with different name.");
                        l_flag = 1;
                        break;
                    }
                }

                if (l_flag == 1) {
                    l_flag = 0;
                    continue;
                }

                for(d_playerStrategyType j : d_playerStrategyType.values())
                {
                    if(j.name().equals(l_playerStrategy)){
                        l_flag1 = 1;
                        break;
                    }
                }

                if(l_flag1 == 0){
                    System.out.println("Invalid Strategy Type, Player " + l_playerName + " not added. Allowed Types of Strategy are :" +
                            "human, aggressive, benevolent, cheater, random");
                    d_Log.notify( p_data: "Invalid Strategy Type, Player " + l_playerName + " not added. Allowed Types of Strategy are :" +
                            "human, aggressive, benevolent, cheater, random");
                    break;
                }
```

3.1 Gameplayer method (Before and After)

Tests : Create players method in testcases now have an additional argument of strategy type

4. Modify savemap command to pass Map type (Conquest or Domination) to save the map

Previously savemap command only had one argument that is file name to save the map as. Since Adapter pattern is introduced, savemap command has been modified to take two arguments. The second argument is used to set the format in which map is to be saved. The two formats are Conquest and Domination.

```
public static void saveMapCommand(List<String> p_argumentTokens) {
    if (p_argumentTokens.stream().count() != 1) {
        System.out.println("Wrong Number of Arguments provided. savemap command has only one argument.");
        d_Log.notify( p_data: "Wrong Number of Arguments provided. savemap command has only one argument.");
        return;
    }

    File l_file = new File( pathname: "src/main/resources/" + p_argumentTokens.get(0) + ".map");
    try {
        MapValidator.validateMap(d_Map);
    } catch (Exception e) {
        System.out.println("Map Validation Failed");
        d_Log.notify( p_data: "Map Validation Failed");
        System.out.println(e.getMessage());
        d_Log.notify(e.getMessage());
    }
    if (MapValidator.d_isValid == false) {
        System.out.println("Map cannot be saved");
        d_Log.notify( p_data: "Map cannot be saved");
    } else {
        new MapWriter().writeMapFile(d_Map, l_file);
    }


}
```

```
public static void saveMapCommand(List<String> p_argumentTokens) {
    if (p_argumentTokens.stream().count() != 2) {
        System.out.println("Wrong Number of Arguments provided. savemap command has only one argument.");
        d_Log.notify( p_data: "Wrong Number of Arguments provided. savemap command has only one argument.");
        return;
    }

    File l_file = new File( pathname: "src/main/resources/" + p_argumentTokens.get(0) + ".map");

    try {
        MapValidator.validateMap(d_Map);
    } catch (Exception e) {
        System.out.println("Map Validation Failed");
        d_Log.notify( p_data: "Map Validation Failed");
        System.out.println(e.getMessage());
        d_Log.notify(e.getMessage());
    }
    if (MapValidator.d_isValid == false) {
        System.out.println("Map cannot be saved");
        d_Log.notify( p_data: "Map cannot be saved");
    }
    else {
        if(p_argumentTokens.get(1).equals("domination")) {
            new MapWriter().writeMapFile(d_Map, l_file);
            System.out.println("Map has been saved successfully");
            d_Log.notify( p_data: "Map has been saved successfully");

        }
        else if(p_argumentTokens.get(1).equals("conquest")){
            new ConquestMapWriter().writeConquestMapFile(d_Map, l_file);
            System.out.println("Map has been saved successfully");
            d_Log.notify( p_data: "Map has been saved successfully");
        }
```

4.1 Savemap method Before and After

Test: Tested saving map in Conquest map format after loading a Domination Map

5. Refactored editConquestMap.

Refactored editConquestMap method to loadConquestMap and createConquestMap according to their functionalities and requirements.

Before

```
public static Map editMap(String p_fileName) throws InvalidMapException {
    return d_editConquestMap.editConquestMap(p_fileName);
}
```

After

```
public static Map editMap(String p_fileName) throws InvalidMapException {

    String l_path = "src/main/resources/";
    String l_fileName = p_fileName + ".map";
    File l_map = new File( pathname: l_path + l_fileName);
    if (l_map.exists())
        return d_editConquestMap.loadConquestMap(p_fileName);
    return d_editConquestMap.createConquestMap(p_fileName);
}
```