Refactoring Document

Refactoring Targets

No.	Refactoring Targets
1.	Executed Adapter Pattern
2.	Implemented Strategy Pattern
3.	Refactored the game for single and tournament mode
4.	Updated gameplayer command to adopt the player strategy
5.	Move logic of issuing commands from PlayerController to individual strategies classes to generate Order commands automatically according to Build description
6.	Added strategy field in Player class
7.	Modified the game termination logic
8.	Added a feature to save and load game
9.	Shifted the logic to enter manual commands in human player strategy class and also validating the commands in same class
10.	Changed the playerIssueOrder() logic for player to incorporate game strategy
11.	Remove deadcode
12.	IssueOrder() method signature change
13.	Added a winnerPlayer (d_winner) field in PlayerController
14.	Change logic in advanceOrder()
15.	Altered the logic to end each turn

1. Adapter Pattern

The game required the capability to load and save map files for Conquest scenarios, in addition to the already existing Domination maps. The refactoring process utilized the Adapter pattern, with the goal of making minimal modifications to the existing Domination processing code while adapting to the distinct format of Conquest maps.

Before:

```
 */
@Override
public String loadMap(String p_command) {
    String l_response = "";

    l_response = d_GameEngine.getD_MapController().loadMap(p_command);
    String l_ValidResult = d_GameEngine.getD_MapController().validateMap();
    System.out.println(l_ValidResult);
    if (l_ValidResult.equalsIgnoreCase(" Map is not valid!!")) {
        l_response = "Reseting loaded map!! please load a valid map";
        d_GameEngine.getD_MapController().resetMap();
        return l_response;
    }
    d_GameEngine.setD_GamePhase(new Startup(d_GameEngine));
    d_LEB.setResult(l_response);
    return l_response;

}
```

```
@Override
public String saveMap(String p_command) {
    String l_response = "";
    String l_ValidResult = d_GameEngine.getD_MapController().validateMap();
    System.out.println(l_ValidResult);
    if (l_ValidResult.equalsIgnoreCase(" Map is not valid!!")) {
        l_response = "Map cannot be saved!!";
    } else {
        l_response = d_GameEngine.getD_MapController().saveMap(p_command);
    }
    d_LEB.setResult(l_response);
    return l_response;
}
```

After:

```
1 Continents
2 c1 10
3 c2 5
4
5 Countries
6 a 1
7 b 1
8 c 2
9 d 2
10
11 Neighbors
12 1 2 3
13 2 1
14 3 1 4
15 4 3
16
17
```

com.soen6441.risk_game_u14.adapterpattern
- Adaptee.java
- Adapter.java
- Target.java

```
1
2 [Continents]
3 c1a=10
4 c2=10
5
6 [Territories]
7 a,0,0,c1 a,b,c
8 b,0,0,c1 a,a
9 c,0,0,c2,a,d
10 d,0,0,c2,c
11
12
```

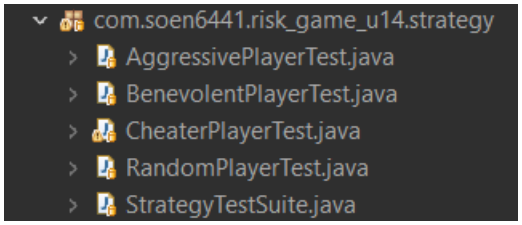
```
71      public String loadMap(String p_command) {
72          String l_response = "";
73
74          boolean l_Flag = false;
75          String l_AckMsg;
76          try {
77              String l_Path = "saved maps\\";
78              File l_File = new File(l_Path + p_command.split(" ")[1]);
79              Scanner l_Sc = new Scanner(l_File);
80              while (l_Sc.hasNextLine()) {
81
82                  String l_Line = l_Sc.nextLine();
83                  if (l_Line.contains("Territories")) {
84                      l_Flag = true;
85                      break;
86                  }
87              }
88              l_Sc.close();
89              if (l_Flag) {
90                  Target l_TargetObject = new Adapter(new Adaptee(), d_GameEngine);
91                  l_AckMsg = l_TargetObject.loadMap(p_command.split(" ")[1]);
92              } else {
93                  l_response = d_GameEngine.getD_MapController().loadMap(p_command);
94              }
95
96          } catch (Exception p_Exception) {
97              l_response = "Reseting loaded map!! please load a valid map";
98              d_LEB.setResult(l_response);
99              d_GameEngine.setD_GamePhase(new Edit(d_GameEngine));
00              return l_response;
01          }
02          String l_ValidResult = d_GameEngine.getD_MapController().validateMap();
03          System.out.println(l_ValidResult);
04          if (l_ValidResult.equalsIgnoreCase(" Map is not valid!!")) {
05              l_response = "Reseting loaded map!! please load a valid map";
06              d_GameEngine.getD_MapController().resetMap();
07              return l_response;
08          }
09          d_GameEngine.setD_GamePhase(new Startup(d_GameEngine));
10
11          d_LEB.setResult(l_response);
12          return l_response;
13      }
```

```
@Override
public String saveMap(String p_command) {
    String l_response = "";
    String l_ValidResult = d_GameEngine.getD_MapController().validateMap();
    System.out.println(l_ValidResult);
    if (l_ValidResult.equalsIgnoreCase(" Map is not valid!!")) {
        l_response = "Map cannot be saved!!";
    } else {
        Scanner s = new Scanner(System.in);
        try {
            while (true) {
                System.out.println("Enter 1 to save map in Conquest Format   2 to save in Domination format\n");
                int choice = s.nextInt();

                if (choice == 1) {

                    Target l_TargetObject = new Adapter(new Adaptee(), d_GameEngine);
                    l_response = l_TargetObject.saveMap(p_command.split(" ")[1]);
                    break;
                } else if (choice == 2) {
                    Target l_TargetObject = new Target(d_GameEngine);
                    l_response = l_TargetObject.saveMap(p_command);
                    break;
                } else {
                    System.out.println("Please Enter your choice 1 or 2");
                }
            }
        } catch (Exception p_Exception) {
            l_response = p_Exception.getMessage();
        }

    }
    d_LEB.setResult(l_response);

    return l_response;
}
```

2. Updated gameplayer command to adopt the player strategy

Originally, the player strategy was not a specified requirement. However, in build 3, it was introduced as a part of the game player command, which was initially in the form "gameplayer -add p1 -add p2." To accommodate this change, the command was modified to "gameplayer -add p1 Strategy Name," and a separate strategy class was constructed to handle each strategy

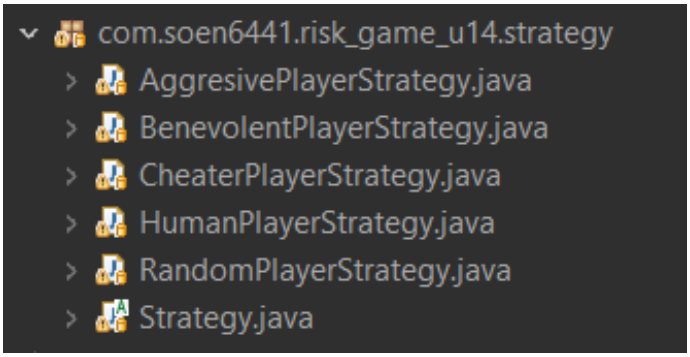
Test Class:



Before:

```
*/
public void addPlayers(String p_PlayerName) throws Exception {
    if ((d_Players.size() >= d_Map.getD_CountryObjects().size())) {
        throw new Exception("Reached max number of players can be added to the game");
    }
    if (duplicatePlayerExist(p_PlayerName)) {
        throw new Exception("Please enter a different player name as this name already exists")
    }
    Player l_TempPlayer = new Player(p_PlayerName, this);
    d_Players.add(l_TempPlayer);
}
```

After:

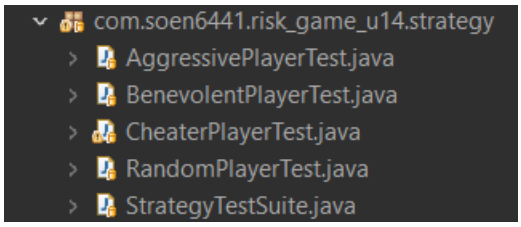


```
*/
public void addPlayers(String p_PlayerName,String p_Strategy) throws Exception {
    if ((d_Players.size() >= d_Map.getD_CountryObjects().size())) {
        throw new Exception("Reached max number of players can be added to the game");
    }
    if (duplicatePlayerExist(p_PlayerName)) {
        throw new Exception("Please enter a different player name as this name already exists");
    }
    Player l_TempPlayer = new Player(p_PlayerName, this);

    switch(p_Strategy) {
    case "aggressive" :
        l_TempPlayer.setD_PlayerStrategy(new AggresivePlayerStrategy(l_TempPlayer,this));
        break;

    case "human" :
        l_TempPlayer.setD_PlayerStrategy(new HumanPlayerStrategy(l_TempPlayer,this));
        break;

    case "benevolent" :
        l_TempPlayer.setD_PlayerStrategy(new BenevolentPlayerStrategy(l_TempPlayer,this));
        break;

    case "random":
        l_TempPlayer.setD_PlayerStrategy(new RandomPlayerStrategy(l_TempPlayer,this));
        break;
    case "cheater":

        l_TempPlayer.setD_PlayerStrategy(new CheaterPlayerStrategy(l_TempPlayer,this));

        break;

    default:
        System.out.println("Invalid Command. Please try again.\n");
        break;
    }

    d_Players.add(l_TempPlayer);
```

3. Move logic of issuing commands from PlayerController to individual strategies classes to generate Order commands automatically according to Build description

Transferred the responsibility of issuing commands from the `PlayerController` to individual strategy classes as part of the software refactoring process. This adjustment enables the automatic generation of Order commands based on the specifications outlined in the build description, utilizing the strategy pattern.

Test Class:



Before:

```
public void playerIssueOrder() {

    List<Player> l_PlayerList = d_GameModel.getD_Players();

    Map<Player, Boolean> l_IsPlayerExit = new HashMap<>();
    for (Player l_Player : l_PlayerList) {
        l_Player.setD_SkipCommands(false);
        l_IsPlayerExit.put(l_Player, false);
    }

    boolean l_AllPlayerDone = false;

    while (!l_AllPlayerDone) {

        l_AllPlayerDone = true;
        for (Player l_Player : l_PlayerList) {

            if (!l_Player.getD_PlayerName().equalsIgnoreCase("Neutral Player")) {

                if (l_IsPlayerExit.get(l_Player) == false) {

                    l_AllPlayerDone = false;
                    System.out.print(l_Player.getD_PlayerName() + "'s turn: ");

                    int l_PlayerArmies = l_Player.getD_ArmiesCount();
                    boolean l_CorrectCommand = false;

                    // if correct command then dongt ask again
                    String l_InputCommand = "";
                    Scanner sc = new Scanner(System.in);

                    while (!l_CorrectCommand) {

                        String l_EnteredComamnd = sc.nextLine();

                        if (isCommandValid(l_EnteredComamnd)) {
                            l_CorrectCommand = true;
                            l_InputCommand = l_EnteredComamnd;
                        } else {
                            System.out.println("Invalid Command!!");
                        }
                    }
                    if (l_InputCommand.equalsIgnoreCase("exit")) {
                        l_IsPlayerExit.put(l_Player, true);
                    } else {
                        d_LEB.setResult(l_InputCommand);
                        l_Player.issueOrder(l_InputCommand);
                    }
```

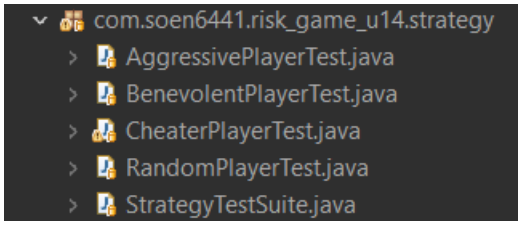
After :

```
public void playerIssueOrder() {

    for(Player l_player:d_GameModel.getD_Players()) {
        if(!l_player.getD_PlayerName().equalsIgnoreCase("Neutral Player"))
            l_player.setIsExit(false);
            l_player.setD_SkipCommands(false);
    }

    for(Player l_player:d_GameModel.getD_Players()) {
        if(!l_player.getD_PlayerName().equalsIgnoreCase("Neutral Player") && l_player.getIsExit()==false)
        l_player.issueOrder();
    }

}
```

4. Added strategy field in Player class

Originally, there was no requirement for player strategy. However, in build 3, the necessity for player strategy emerged. Consequently, the player class underwent modification to include a new field called d_Strategy. This field serves as a means to map the strategy associated with each player.

Test Class:



Before:

```
public class Player {
    private static int D_PlayerCount = 0;
    private int d_Playerid;
    private String d_PlayerName;
    private int d_ArmiesCount;
    private Order d_CurrentOrder;
    private String d_Result;
    private List<Country> d_PlayerOwnedCountries;
    private List<Continent> d_PlayerOwnedContinent;
    private Queue<Order> d_PlayerOrderQueue;
    private Boolean d_SkipCommands;
    private GameModel d_GameModel;
    private ArrayList<Player> d_NegotiatedPlayers;
    private boolean d_AtleastOneBattleWon;
    private ArrayList<String> d_Cards;
```

After:

```
public class Player implements Serializable{
    private static int D_PlayerCount = 0;
    private int d_Playerid;
    private String d_PlayerName;
    private int d_ArmiesCount;
    private Order d_CurrentOrder;
    private String d_Result;
    private List<Country> d_PlayerOwnedCountries;
    private List<Continent> d_PlayerOwnedContinent;
    private Queue<Order> d_PlayerOrderQueue;
    private Boolean d_SkipCommands;
    private GameModel d_GameModel;
    private ArrayList<Player> d_NegotiatedPlayers;
    private boolean d_AtleastOneBattleWon;
    private ArrayList<String> d_Cards;
    private Strategy d_PlayerStrategy;
    private Boolean isExit;
    /**
     * Boolean to check if a game should be saved or not
     */
```

5. Implemented new logic for game termination

Introduced a novel game termination logic in build 3. Originally, with no automatic players, participants were required to manually conclude their turns until a single player gained ownership of all countries, declaring them the winner. However, in the latest update, as outlined in build 3, the inclusion of automatic players has led to the establishment of a maximum limit of 50 turns per game.

Before:

```
public ExecuteOrder(GameEngine p_Ge) {
    super(p_Ge);
    try {
        d_LEB = new LogEntryBuffer();
        d_LEB.setResult("In Execute Order Phase");
        p_Ge.getD_PlayerController().playerExecuteOrder();
        p_Ge.getD_PlayerController().show();
        if (p_Ge.getD_PlayerController().checkTheWinner() == 1) {
            Scanner s = new Scanner(System.in);
            System.out.println("Do you want to continue y = Yes n = No");
            String inpString = s.nextLine();
            if (inpString.equalsIgnoreCase("y")) {
                p_Ge.setD_GamePhase(new Reinforcement(p_Ge));
            } else {
                System.out.println("Byee");
            }
        } else {
            p_Ge.setD_GamePhase(new Gameover(p_Ge));

        }
    } catch (Exception p_E) {
        d_LEB.setResult(p_E.getMessage());
    }
}
```

After:

```
public ExecuteOrder(GameEngine p_Ge) {
    super(p_Ge);
    counter++;
    try {
        d_LEB = new LogEntryBuffer();
        d_LEB.setResult("In Execute Order Phase");
        p_Ge.getD_PlayerController().playerExecuteOrder();
        p_Ge.getD_PlayerController().show();
        int checkWinner = p_Ge.getD_PlayerController().checkTheWinner();
        if (checkWinner == 1) {
            if (counter > 50) {
                System.out.println("Number Of Rounds Exhausted");
                System.out.println("Its a Draw!!!");
                p_Ge.setD_GamePhase(new Gameover(p_Ge));
            }
            p_Ge.setD_GamePhase(new Reinforcement(p_Ge));
        } else {
            p_Ge.setD_GamePhase(new Gameover(p_Ge));
        }
    } catch (Exception p_E) {
        d_LEB.setResult(p_E.getMessage());
    }
}
```

6. IssueOrder() method signature change

In Build 2, orders were initially issued from the `PlayerController` based on user input. The `issueOrder` method in the `Player` class originally accepted string order arguments to accommodate different order types. However, with the introduction of several new strategies in Build 2 that require automatic order generation without human input, the string order arguments were deemed unnecessary. As a result, each player now independently generates orders within its own class.

Before:

```
public void issueOrder(String p_Orders) {


    String l_InputCommandSplit[] = p_Orders.split(" ");
    String l_command = l_InputCommandSplit[0];
    if (l_command.equalsIgnoreCase("deploy")) {
        Country l_TargetCountryObject = checkCountryBelongstoPlayer(l_InputCommandSplit[1]);
        getD_PlayerOrderQueue()
                .add(new Deploy(this, l_TargetCountryObject, Integer.parseInt(l_InputCommandSplit[2])));
    } else if (l_command.equalsIgnoreCase("advance")) {
        Country l_SourceCountry = checkCountryBelongstoPlayer(l_InputCommandSplit[1]);
        Country l_TargetCountry = d_GameModel.getD_Map().findCountryByName(l_InputCommandSplit[2]);
        int l_NumArmies1 = Integer.parseInt(l_InputCommandSplit[3]);
        getD_PlayerOrderQueue().add(new Advance(this, l_SourceCountry, l_TargetCountry, l_NumArmies1));
    } else if (l_command.equalsIgnoreCase("bomb")) {
        Country l_TargetCountry = d_GameModel.getD_Map().findCountryByName(l_InputCommandSplit[1]);
        getD_PlayerOrderQueue().add(new Bomb(this, l_TargetCountry));
    } else if (l_command.equalsIgnoreCase("blockade")) {
        Country l_SourceCountry = d_GameModel.getD_Map().findCountryByName(l_InputCommandSplit[1]);
        getD_PlayerOrderQueue().add(new Blockade(this, l_SourceCountry));
    } else if (l_command.equalsIgnoreCase("airlift")) {
        Country l_SourceCountry = checkCountryBelongstoPlayer(l_InputCommandSplit[1]);
        Country l_TargetCountry = checkCountryBelongstoPlayer(l_InputCommandSplit[2]);
        int l_NumArmies1 = Integer.parseInt(l_InputCommandSplit[3]);
        getD_PlayerOrderQueue().add(new Airlift(this, l_SourceCountry, l_TargetCountry, l_NumArmies1));
    } else if (l_command.equalsIgnoreCase("negotiate")) {
        Player l_TempPlayer = findPlayerByName(l_InputCommandSplit[1]);
        getD_PlayerOrderQueue().add(new Negotiate(this, l_TempPlayer));
    }
}
```

After:

```
public void issueOrder() {

    String playerGeneratedOrder = d_PlayerStrategy.createOrder();
    if(playerGeneratedOrder==null)
        return;
    System.out.println(playerGeneratedOrder);
    String l_InputCommandSplit[] = playerGeneratedOrder.split(" "); //p_Orders.split(" ");
    String l_command = l_InputCommandSplit[0];
    if (l_command.equalsIgnoreCase("deploy")) {
        Country l_TargetCountryObject = checkCountryBelongstoPlayer(l_InputCommandSplit[1]);
        getD_PlayerOrderQueue()
                .add(new Deploy(this, l_TargetCountryObject, Integer.parseInt(l_InputCommandSplit[2])));
    } else if (l_command.equalsIgnoreCase("advance")) {
        Country l_SourceCountry = checkCountryBelongstoPlayer(l_InputCommandSplit[1]);
        Country l_TargetCountry = d_GameModel.getD_Map().findCountryByName(l_InputCommandSplit[2]);
        int l_NumArmies1 = Integer.parseInt(l_InputCommandSplit[3]);
        getD_PlayerOrderQueue().add(new Advance(this, l_SourceCountry, l_TargetCountry, l_NumArmies1));
    } else if (l_command.equalsIgnoreCase("bomb")) {
        Country l_TargetCountry = d_GameModel.getD_Map().findCountryByName(l_InputCommandSplit[1]);
        getD_PlayerOrderQueue().add(new Bomb(this, l_TargetCountry));
    } else if (l_command.equalsIgnoreCase("blockade")) {
        Country l_SourceCountry = d_GameModel.getD_Map().findCountryByName(l_InputCommandSplit[1]);
        getD_PlayerOrderQueue().add(new Blockade(this, l_SourceCountry));
    } else if (l_command.equalsIgnoreCase("airlift")) {
        Country l_SourceCountry = checkCountryBelongstoPlayer(l_InputCommandSplit[1]);
        Country l_TargetCountry = checkCountryBelongstoPlayer(l_InputCommandSplit[2]);
        int l_NumArmies1 = Integer.parseInt(l_InputCommandSplit[3]);
        getD_PlayerOrderQueue().add(new Airlift(this, l_SourceCountry, l_TargetCountry, l_NumArmies1));
    } else if (l_command.equalsIgnoreCase("negotiate")) {
        Player l_TempPlayer = findPlayerByName(l_InputCommandSplit[1]);
        getD_PlayerOrderQueue().add(new Negotiate(this, l_TempPlayer));
    }
```