# CODING CONVENTIONS

## 1. Code Layout:

- **Indentation**

  The syntactical elements of the code are indented according to the nesting level.

```
if(this.territoriesHeld.size()/3 < 3) {
    reinforcementArmies = 3;
} else {
    reinforcementArmies = this.territoriesHeld.size()/3;
}
if(!this.continentHeld.isEmpty()) {
    System.out.println("[ReinforcementPhase][Checking for continent list of the player]");
    for(String key : this.continentHeld.keySet()) {
        Continent continent = this.continentHeld.get(key);
        reinforcementArmies = reinforcementArmies + continent.getContinentScore();
    }
}

System.out.println("[Reinforcement amries are calculated]" +" [" +reinforcementArmies + "]");
increaseArmyCountByValue(reinforcementArmies);
```

- **Method Indentation**

  The body of the methods are indented with respect to its function header with a space after declaration.

```
     */
    public void addTerritory(Territories territory) {

        territoriesHeld.put(territory.getTerritorieName(),territory);
        setChanged();
        notifyObservers(territoriesHeld);
    }

}
```

- **Statement Format**

  The open curly braces are appended to the statement that proceeds it.

```
if(sourceCountry.getAdjacentTerritories().contains(attackedCountry)) {
    return true;
}
return false;
```

- **Blank Lines**

  Blank Lines are inserted between method definitions, class declarations and sections of long and complicated code to enhance readability.

```java
import java.util.*;
import java.util.Map;

/**
 * This class maintains the player details and implements methods related to the player.
 * @author Team43
 */
public class Player extends Observable {

    private String playerId;
    private String playerName;
    private int playerArmies;
    private Boolean isKnocked;
    private String playerCharacter;
    private Color playerColor;
    private int cardsHolded;
    private HashMap<String, Territories> territoriesHeld;
    private HashMap<String, Continent> continentHeld;
    private List<Card> cardsHeld;

    /**
     * Constructor with parameters.
     *
     * @param playerName
     * @param playerArmies
```

## 2. Naming Conventions

- **Class Names**

  Class names follow camel case. The class names begin with capital letter and if the names contain multiple words, each word is separated by capital letter at the beginning of each word.

```java
public class NewMapControllerTest1 {
```

- **Method Names**

  Method Names begin with lower case letter and if there're multiple words, upper case letters are used to separate them.

```java
public boolean doFortification(int noOfArmiesToMove,String sourceCountry,String destinationCountry) {

    Territories sourceTerritory = territoriesHeld.get(sourceCountry);
    sourceTerritory.setArmiesHeld(sourceTerritory.getArmiesHeld()-noOfArmiesToMove);
```

- **Constants**

  The constants will be denoted with upper case letters.

```
private static final String ATTACKPHASE = "ATTACKPHASE";

private static final String FORTIFICATIONPHASE = "FORTIFICATIONPHASE";

private static final String WINNER = "WINNER";

private static final String LOSER = "LOSER";
```

- **Local Variables**

The local variables begin with capital letter and if the names contain multiple words, each word is separated by capital letter at the beginning of each word.

```
/
public void IncreaseTheArmyCount() {
    int noOfArmies = findCorrectSetofArmies(playerList.get(currentPlayer).getCardTurn());
```

### 3. Commenting Conventions

- Comments precede all class declaration that describes the purpose of the class. These comments are used to generate Javadoc.

```
/**
 * This class maintains the player details and implements methods related to the player.
 * @author Team43
 */
public class Player extends Observable {
```

- All the methods have comments describing their role and also their parameters names.

```
/**
 * This method performs attack by rolling the dice and declares the winner.
 * @param attackerDice
 * @param attackedDice
 * @param attackerCountry
 * @param attackedCountry
 * @param defender
 * @return attack result
 */
public String doAttack(int attackerDice,int attackedDice,String attackerCountry,String attackedCountry,Player defender) {

    // declare arraylist for dice.
```

## 4.Exception Handling

Format of try and catch statements is as seen in the picture below:

```
try {
    BufferedWriter writer = new BufferedWriter(new FileWriter(file));

    writer.write(newmapTextarea.getText());
    writer.flush();
    writer.close();

    boolean result = meController.ismapValid(file,"NEWMAP");

    if(result) {
        // Check if the map name is empty. If empty then give some default name.
        if(mapnameTextfield.getText().trim().isEmpty()) {
            fileName = "NewMap"+count;
            count++;
        } else {
            fileName = mapnameTextfield.getText().trim();
        }

        // Create file and write the map contents to the file and store.
        Path path = Paths.get(filePath+fileName+".map");
        try {
            writer = Files.newBufferedWriter(path);
            writer.write(newmapTextarea.getText());
            writer.flush();

            System.out.println("Map is valid. Successfully saved the map.\nMap saved location foldername ="
                               " Modifiedmaps");
        } finally {
            if(writer != null) {
                writer.close();
                cancelButtonAction();
            }
```