Coding Conventions

1.Code Layout:

- **Indentation**

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

```
try {
    BufferedWriter writer = new BufferedWriter(new FileWriter(file));
    writer.write(textArea.getText());
    writer.flush();
    writer.close();

    boolean result = ismapValid(file, "EDITMAP");
    if(result) {
        System.out.println("Map is Valid and successfully saved.");
        Path path = Paths.get(filePath+selectedfile.getName());
        try {
            writer = Files.newBufferedWriter(path);
            writer.write(textArea.getText());
            writer.flush();
        } finally {
            if(writer != null) writer.close();
            cancelAction();
        }

    } else {
        System.out.println("Invalid Map. Remodify the map.");

        // Clear the data.
        GameDetails.getGamedetails().clearData();

        // Closes the window
        cancelAction();
    }

} catch(Exception e) {
    System.out.println("Cannot write contents to the file.");
    e.printStackTrace();
    return;
```

- **Method Indentation**

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

```
public void cancelButtonAction() {

    Stage stage = (Stage) newmapCancelbutton.getScene().getWindow();
    stage.close();
}
```

- **Statement Format**

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

```
} else {
    System.out.println("Map is not valid. Please check the continents and territories and try again.");

    // Clear the data.
    GameDetails.getGamedetails().clearData();

    // Close the window.
    cancelButtonAction();
}
```

- **Blank Lines**

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

```
import javafx.fxml.FXML;

public class Controller {

    @FXML
    private MenuBar menubarId;

    @FXML
    private BorderPane MainBorderPane;

    @FXML
    private Canvas canvasId;

    @FXML
    public void initialize() {

        GraphicsContext graphicsContext = canvasId.getGraphicsContext2D();
        Image image = new Image("resources/risk.png");
        graphicsContext.drawImage(image,0,0,canvasId.getWidth(),canvasId.getHeight());
    }
```

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.

```
public class GameDetails {
```

- **Method Names**

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

```
*/
public void distributeTerritories(int index) {

    territoriesList = gamedetails.get(index).getTerritoriesList();
    playersList = gamedetails.get(index).getPlayersList();
```

- **Constants**

The constants will be denoted with upper case letters. In build one we have not used constants

- **Local Variables**

Local variables are written in lowercase letters without any delimiter.

3. Commenting Conventions

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

```
/**
 * This class is used to store the initial game parameters.
 */
public class GameDetails {
```

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

```
/**
 * This function distributes the player armies to their corresponding territories.
 * First retrieves the no of players.
 * For each player it retrives the territories and for each territory it assigns one army.
 * After the loop if the player left any armies then it retrives the random territory and increaments
 * the count by 1. We do this untill player has no armies left.
 * @param index
 */
public void distributeArmiestoTerritories(int index) {
```