DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE

ENGINEERING

SOEN 6441, Fall 2019

RISK Game (Build-2)

Coding Standards

Submitted To: JABABO KHALED

Submitted By: **Team E**

Git URL: https://github.com/Surya64/APP_SOEN-6441_TeamE

Sr. No.	Name	Student ID
1	Surya Prakash Govindaraju	40085527
2	Shruthi Kondapura Venkataiah	40091427
3	Sahana Anantha	40085533
4	Sai Charan Teja Doddi	40076338
5	Dolly Modha	40084358

Introduction:

Coding conventions are a set of prescriptive rules that pertain to how code is to be written. It defines the Programming style. The main advantage of coding conventions are maintainability, compatibility and readability. Coding convention makes it easier for the distinct teams to interface and read the code of other teams.

Coding Conventions and Standards adopted in project:

- Naming Conventions:

We have used CamelCase naming convention. All package names are in lowercase. UpperCamelCase is used for naming the classes and lowerCamelCase for naming the methods. All the local variables, method parameters follow lowerCamelCase convention. All the constants are in Uppercase.

- Layout:

To make the code more understandable and shorter we have placed the open braces on the same line where the method starts.

- Indentation:

To convey the proper program structure, we have used the formatter present in Eclipse tool. Depending on the code's environment the while loops, for loops and if conditions are indented by providing single tab space. Horizontal whitespace and Line wrapping are done using the available formatter.

```
159         if (choice.equalsIgnoreCase("Yes")) {
160             while (playersList.get(0).getNoOfArmies() > 0) {
161                 for (int round = 1; round <= playersList.size(); round++) {
162                     gameplayer = roundRobin.nextTurn();
163                     boolean placeArmyFlag = true;
164                     System.out.println("Name: " + gameplayer.getPlayerName());
165                     System.out.println("No of Armies remaining: " + gameplayer.getNoOfArmies());
166                     do {
167                         placeArmyFlag = false;
168                         boolean middlePlace = false;
169                         System.out.println("Enter Command to place Army to Country");
170                         String input = br.readLine().trim();
171                         if (input.equalsIgnoreCase("placeall")) {
172                             placeallAmry();
173                             middlePlace = true;
174                             placeArmyFlag = false;
175                             round = playerNames.size();
176                         }
177                         if (!middlePlace) {
178                             String[] data = input.split(" ");
179                             Pattern commandName = Pattern.compile("placearmy");
180                             Matcher commandMatch = commandName.matcher(data[0]);
181                             if (!commandMatch.matches() || input.isEmpty()) {
182                                 System.out.println("\nIncorrect Command");
183                                 placeArmyFlag = true;
184                             }
185                             if (!placeArmyFlag) {
186                                 boolean ownCountryFlag = false;
187                                 ArrayList<Country> playerCountries = gameplayer.getPlayerCountries();
188                                 for (int i = 0; i < playerCountries.size(); i++) {
189                                     if (playerCountries.get(i).getCountryName().equalsIgnoreCase(data[1])) {
190                                         if (gameplayer.getNoOfArmies() > 0) {
191                                             playerCountries.get(i)
192                                                     .setNoOfArmies(playerCountries.get(i).getNoOfArmies() + 1);
193                                             gameplayer.setNoOfArmies(gameplayer.getNoOfArmies() - 1);
194                                             System.out.println(
195                                                     "One Army is placed in " + playerCountries.get(i).getCountryName
196                                             ownCountryFlag = true;
197                                         } else {
198                                             System.out.println("All armies are placed.\n");
199                                             ownCountryFlag = true;
200                                             placeArmyFlag = false;
```

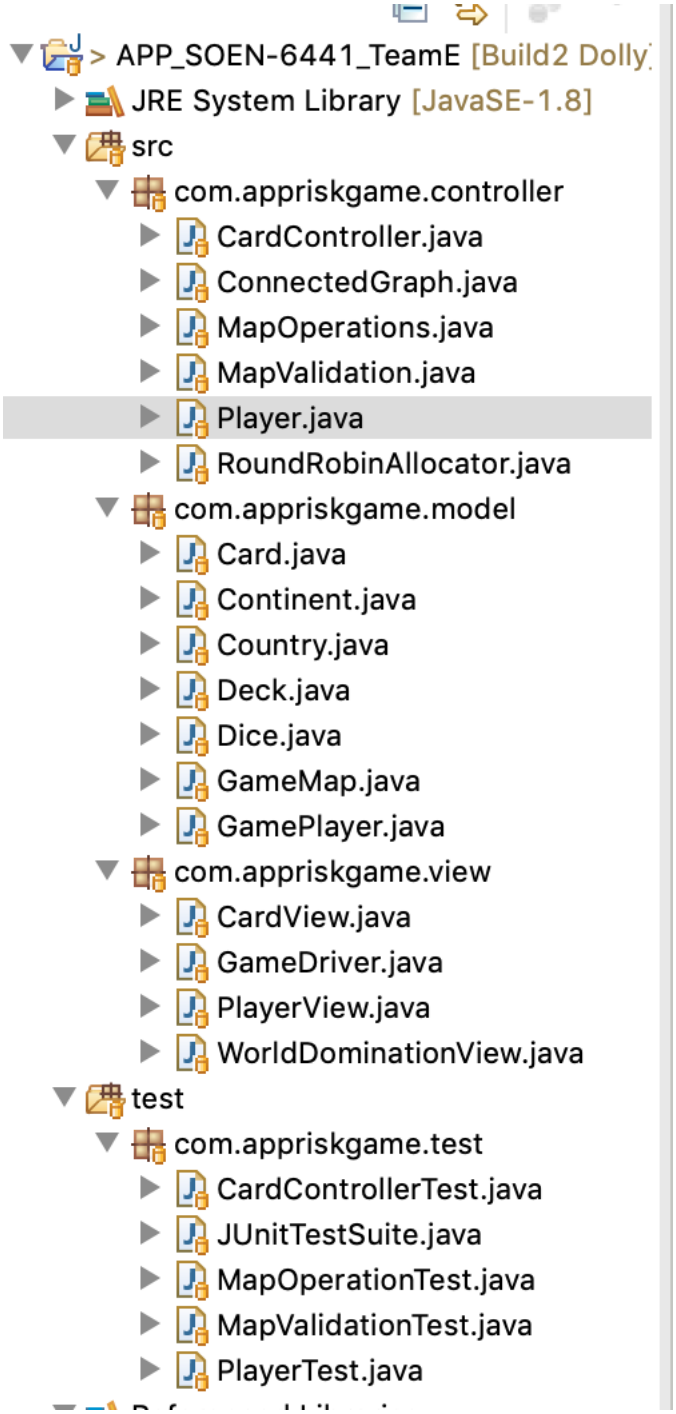
❖ Commenting:

We have provided comments for the methods, variables, classes to maximize readability and understandability. Comments are placed at the beginning of the classes and methods. Comments give a brief description about what that method does, the parameters used in it and return types. In few cases, we have included the comments inside the methods for more detailed explanation. Javadoc comments are highly used in our projects which consists of special tags on classes, methods and member variables such as @param, @return.

```
677
678  /**
679   *
680   * This method check the army count entered by the user and if it is less than
681   * the available, it assigned to the mentioned country
682   *
683   * @param country     - the country given to players
684   * @param armiesCount - the count of the armies player has
685   * @param player      - Current player object
686   */
687  public void userAssignedArmiesToCountries(Country country, int armiesCount, GamePlayer player) {
688      if (player.getPlayerCountries().contains(country)) {
689          if ((player.getNoOfArmies()) > 0 && player.getNoOfArmies() >= armiesCount) {
690              country.setNoOfArmies(country.getNoOfArmies() + armiesCount);
691              player.setNoOfArmies(player.getNoOfArmies() - armiesCount);
692          } else {
693              System.out.println("Insufficient number of armies.\n");
694          }
695      } else {
696          System.out.println("This country is not owned by you!\n");
697      }
698  }
699
700  /**
701   * Based on Reinforcement conditions the player will be allocated with some
702   * armies to assign to countries
703   *
704   * @param player    - The player to whom armies will be allocated to
705   * @param continent - Continent to which the player belongs to
706   * @return armies to be assigned to any country of players choice
707   */
708  public int assignReinforcedArmies(GamePlayer player, Continent continent) {
709      int contriesPlyerOwns = player.getPlayerCountries().size();
710      int reinformentArmiesAssigned;
711      if (contriesPlyerOwns >= MINIMUM_NUM_OF_PLAYERS_COUNTRY) {
712          reinformentArmiesAssigned = (int) Math.floor(contriesPlyerOwns / 3);
713      } else {
714          reinformentArmiesAssigned = MINIMUM_REINFORCEMENT_ARMY;
715      }
716      for (int i = 0; i < listOfPlayerContinents.size(); i++) {
717          if (doesPlayerOwnAContinent(player, listOfPlayerContinents.get(i).getListOfCountries()))
718              reinformentArmiesAssigned = reinformentArmiesAssigned
719                      + listOfPlayerContinents.get(i).getContinentControlValue();
720      }
```

- File Naming and Organization:

We have given the relatable names for files based on their functionality and placed all the related files in the corresponding package.



- Exception Handling:

We have used try & catch block in our project. If there is any exception, we have used meaningful print statements which helps programmer to identify and fix the bug.

```
977     if (choice.equalsIgnoreCase("Yes")) {
978         System.out.println("Enter the command to save the Map File");
979         String command = br.readLine().trim();
980         String[] cmdDetails = command.split(" ");
981         String cmdType = cmdDetails[0];
982         if (cmdType.equals("savemap")) {
983             if (cmdDetails.length == 2) {
984                 String fileName = cmdDetails[1];
985                 String ouputGameMapName = mapLocation + fileName + ".map";
986                 try {
987                     writeGameMap(ouputGameMapName, fileName);
988                 } catch (IOException e) {
989                     System.out.println("File Not found Exception");
990                 }
991                 MapValidation validate = new MapValidation();
992                 boolean uploadSuccessful = false;
993                 try {
994                     uploadSuccessful = validate.validateMap(ouputGameMapName);
995                 } catch (IOException e) {
996                     System.out.println("File Not found Exception");
997                 }
998 
999                 if (isContinentCountrySatisfied()) {
1000 
1001                 } else {
1002                     uploadSuccessful = false;
1003                 }
1004                 if (uploadSuccessful) {
1005                     System.out.println("Successfully Saved");
1006                 } else {
1007                     File file = new File(ouputGameMapName);
1008                     file.delete();
1009                     System.out.print(isContinentCountrySatisfiedError());
1010                     System.out.println(MapValidation.getError());
1011                     System.out.println("\nPlease rectify all the above mentioned issues");
1012                     flag = true;
1013                 }
1014             } else {
1015                 System.out.println("Incorrect command");
1016                 flag = true;
1017             }
```