CONCORDIA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

SOEN-6441

Risk game - build 1

Application Design document

Team# 31

Divyaprabha Rajendran	40089282
Chitra Gunasekaran	40107070
Ishpreet Singh	40093666
Yash Pragneshkumar Doshi	40105936
Fareed Tayar	40102053

I.   Assignment Objective

To design and build a simple Risk game, A Risk game consists of a connected graph map representing a world map, where each node is a country and each edge represents adjacency between countries. Three to six players can play by placing armies on countries they own, from which they can attack adjacent countries to conquer them. The objective of the game is to conquer all countries on the map.

To deliver an operational version demonstrating a subset of the capacity of your system. This is about demonstrating that the code build is effectively aimed at solving specific project problems or completely implementing specific system features.

II.   High level system architecture*

The application consists of below main components and functionalities

   A. Game main model components
        i.   Graph node: consists of graph data (the country) and links to neighboring graph nodes
        ii. Graph: an abstraction of a graph, used in player, continent and game map types. It provides verification methods for graph connectivity and finding path between two graph nodes. As well as view method for graph nodes.
        iii. Country: encapsulates data related to the country along with needed methods to modify its data. Every country has an observer (using Observer pattern) which is triggered whenever country ownership and/or number of armies exists changed.
        iv. Continent: holder of continent data like name, value of bonus armies, continent graph. As well as view continent method and a utility method to verify if a single user owns all countries belongs to this continent.
        v.   Game map: holder for played game map data (continents, reference to map xml file…etc.)
        vi. Player: the player main type, holds players related data like its name, hand (hosting cards), player's graph along with needed methods to modify players related data and calculate number of reinforcement armies for this player
        vii. Players: holder to all players playing the game.

   B. Configuration constants: constants representing default number of armies each player should have in startup phase based on number of players playing the game.
   C. Exceptions: customized exceptions representing invalid runtime faults related to graph, continent, country, naming conventions, player … etc.
   D. Map creation and validation: to read xml map file, build continent and game map objects and validate its data.
   E. Game main driver: provides the game main follow, according to risk game rules
        i.   start with start-up phase (creation, edit and loading of map xml files), then
        ii. creation of game model objects (continents, game map, players… etc.)
        iii. initial distribution of counties (randomly)
        iv. initial distribution of players armies

v.   Play game flow (provides in players turn)
1.   Reinforcement phase in accordance to risk game rules , where for every turn all reinforcement armies are placed on the player owned countries
2.   Attack (will be concluded by second build)
3.   Fortification phase (optional based on user input), according to risk game rules with valid move of armies (valid path between the two countries in the player graph)

(*) Refer to first appendix for complete UML representation

(*) the document illustrates assignment 1 requirements, hence other functionalities are not presented in this document (card, deck Of Cards, hand… etc.)

III.   Used Technology and references

- Standard Java library (JDK 1.8) including java doc tool
- Junit 5
- Brown university coding standard - java style
- Course material in reference to design patterns (Observer Model)
- Bitbucket GIT repository
- Eclipse IDE

IV.   Folders and packages structure

A.   Folders structure
i.   src: contains application source code files
1.   /ca
a.   /riskgamet31: contains application source code packages and files
b.   /riskgamet31test: contains Junit test package, suite and classes
ii.   ProjDocs: contains project reference files (i.e. coding style reference, assignment 1 documentation and presentation.
iii. Risk_JDocs: contains the automatically generated java documentation using jdoc tool.
iv. Risk_MapData: contains several valid and invalid maps source xml files.

B.   Packages structure
i.   ca.riskgamet31.configuration: configuration types (constants …etc)
ii.   ca.riskgamet31.controllers: controllers classes
iii. ca.riskgamet31.exceptions: customized exception classes
iv. ca.riskgamet31.maincomps: game main model classes
v.   ca.riskgamet31.mapdata: map related classes
vi. ca.riskgamet31. views: view classes

below contains joint test classes and suites for various packages
vii. ca.riskgamet31test.maincomps
viii. ca.riskgamet31test.controllers
ix. ca.riskgamet31test.mapdata

v.   Appendix I – UML

<<Java Class>>
CreateMap
ca.riskgamet31.mapdata

- ◻ xmlFile: File
- ◻ xmlDoc: Document
- ◻ countrySet: HashSet<String>
- ◻ continentSet: HashSet<String>
- ◻ countryMap: HashMap<String,GraphNode>
- ◻ mapData: HashMap<String,Continent>

- CreateMap()
- CreateMap(String)
- loadMapData():void
- createGraphNode(String):GraphNode
- createContinents(String,int,ArrayList<GraphNode>):Continent
- getCountries(Element):ArrayList<GraphNode>
- getContinents():void
- processLinks():void
- addLinks(String,String):void
- getAllCountryNodes():ArrayList<GraphNode>
- displayContinentGraph(Continent):void
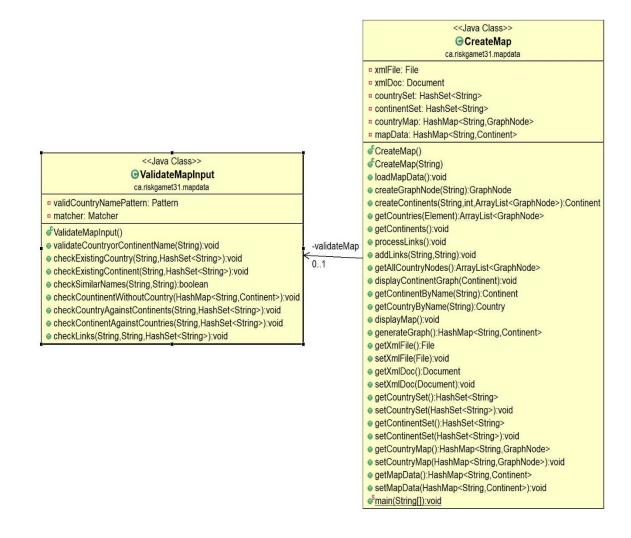- getContinentByName(String):Continent
- getCountryByName(String):Country
- displayMap():void
- generateGraph():HashMap<String,Continent>
- getXmlFile():File
- setXmlFile(File):void
- getXmlDoc():Document
- setXmlDoc(Document):void
- getCountrySet():HashSet<String>
- setCountrySet(HashSet<String>):void
- getContinentSet():HashSet<String>
- setContinentSet(HashSet<String>):void
- getCountryMap():HashMap<String,GraphNode>
- setCountryMap(HashMap<String,GraphNode>):void
- getMapData():HashMap<String,Continent>
- setMapData(HashMap<String,Continent>):void
- main(String[]):void

<<Java Class>>
ValidateMapInput
ca.riskgamet31.mapdata

- ◻ validCountryNamePattern: Pattern
- ◻ matcher: Matcher

- ValidateMapInput()
- validateCountryorContinentName(String):void
- checkExistingCountry(String,HashSet<String>):void
- checkExistingContinent(String,HashSet<String>):void
- checkSimilarNames(String,String):boolean
- checkCountinentWithoutCountry(HashMap<String,Continent>):void
- checkCountryAgainstContinents(String,HashSet<String>):void
- checkContinentAgainstCountries(String,HashSet<String>):void
- checkLinks(String,String,HashSet<String>):void

-validateMap
0..1