CONCORDIA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

SOEN-6441

Risk game - build 3

Application Design document

Team# 31

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

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: an abstraction of player type hosting some default methods and other abstract methods which will be defined in concrete types with various strategies for reinforcement, attack and fortification operations.
   vii. Several concrete implementations of player type : HumanPlayer, AggressivePlayer, BenevolentPlayer, RandomPlayer and CheaterPlayer. Each representing different playing strategy, the strategy is injected at runtime based on the type of player been selected to play.
   viii. Players: holder for all players playing the game.

B. Utility package: provides utility classes as below
   i. Constants: representing default number of armies each player should have in startup phase based on number of players playing the game.
   ii. Backup / save game and load game: utility classes used to save the game and reloaded using java serialization.

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. main drivers (MainDriver, GameMainDriver, TournamentMainDriver): MainDriver an abstraction which will have two concrete implementations one for single game mode (GameMainDriver) and another one for tournament mode (TournamentMainDriver). provides the game main follow, according to risk game rules and grading sheet.
    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 and player's strategy, where for every turn all reinforcement armies are placed on the player owned countries.
        2. Attack phase according to game rules and player's strategy with all-out option for attack, till end of attack round.
        3. Fortification phase, according to risk game rules and player's strategy with validation prior moving armies (valid path between the two countries in the player graph).
    vi. Views (based on observer pattern)
        1. Players world domination view
        2. Phase view (including startup phase)
        3. Card exchange view
        4. Country view to reflect changed in ownership and number of armies.

(*) Refer to first appendix for complete UML representation

III. Used Technology and references

- Standard Java library (JDK 1.8) including java doc tool
- Junit 4
- Brown university coding standard - java style
- Course material and reference used for design patterns (Singleton, Observer, Strategy)
- 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.

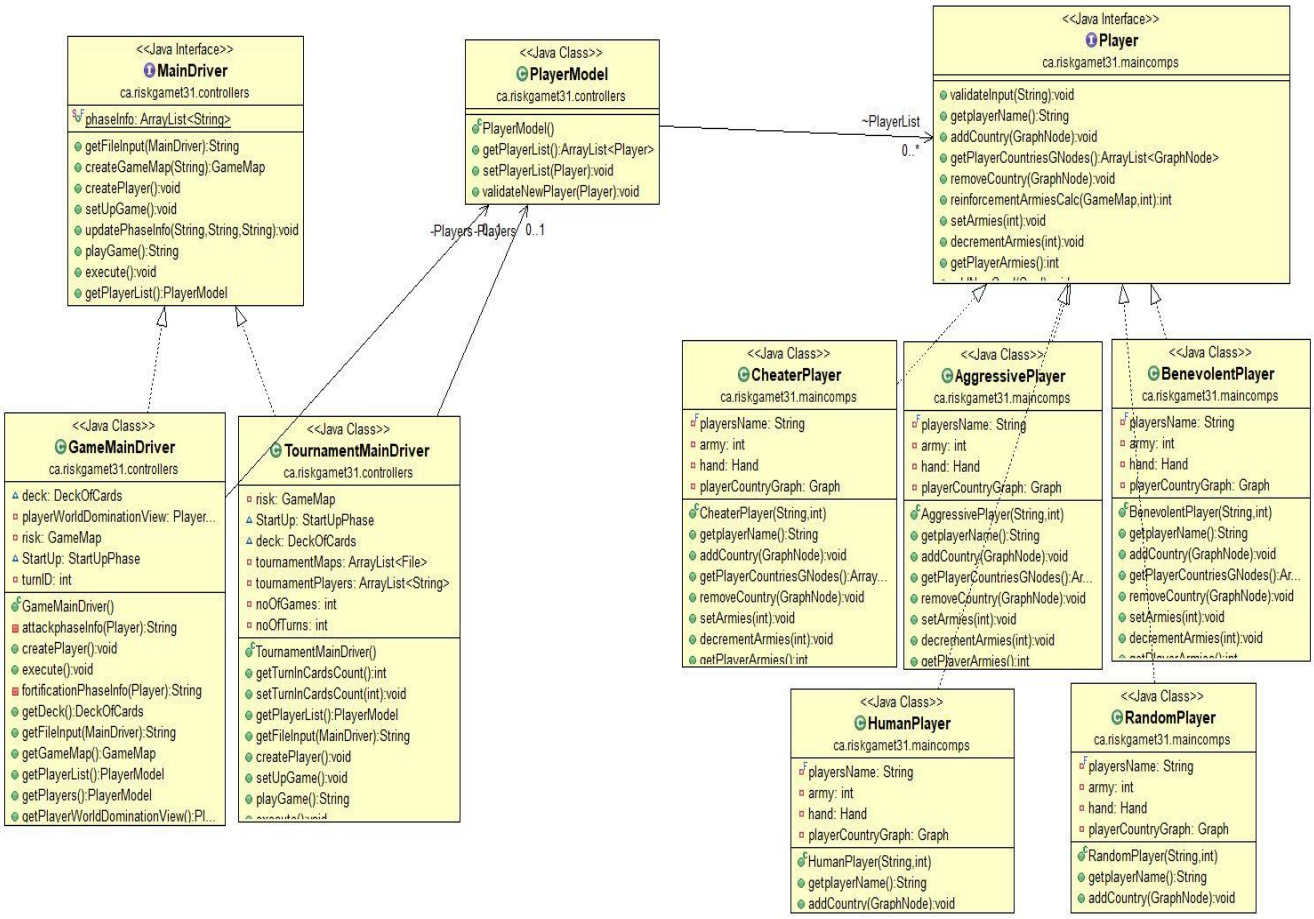
B. Packages structure

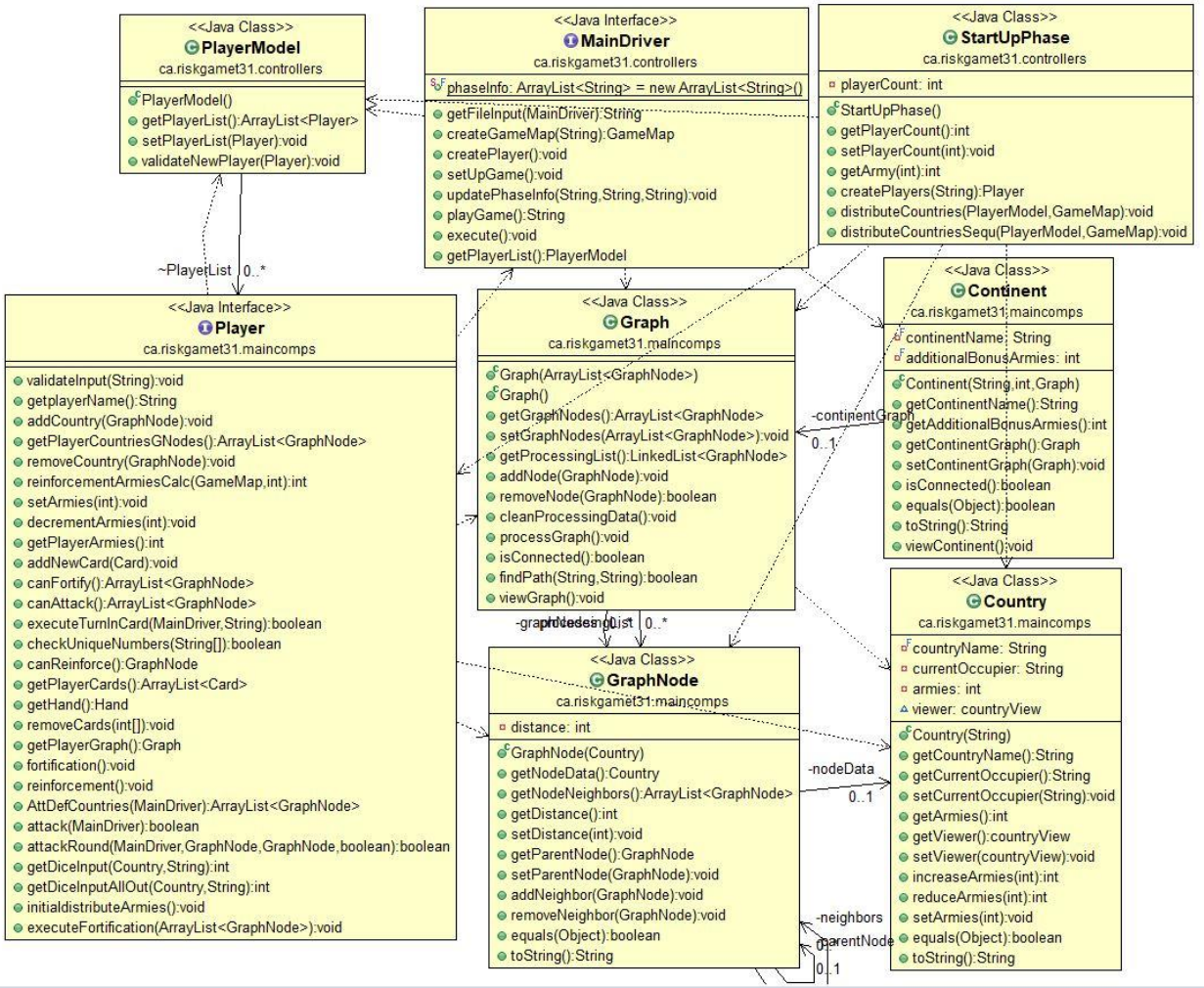
i. ca.riskgamet31.utilitiy: utility classes(constants …etc)
ii. ca.riskgamet31.controllers: controllers classes
iii. ca.riskgamet31.exceptions: customized exception classes
iv. ca.riskgamet31.maincomps: game main model classes
v. ca.riskgamet31.mapdata: map related classes
vi. ca.riskgamet31.views: views classes

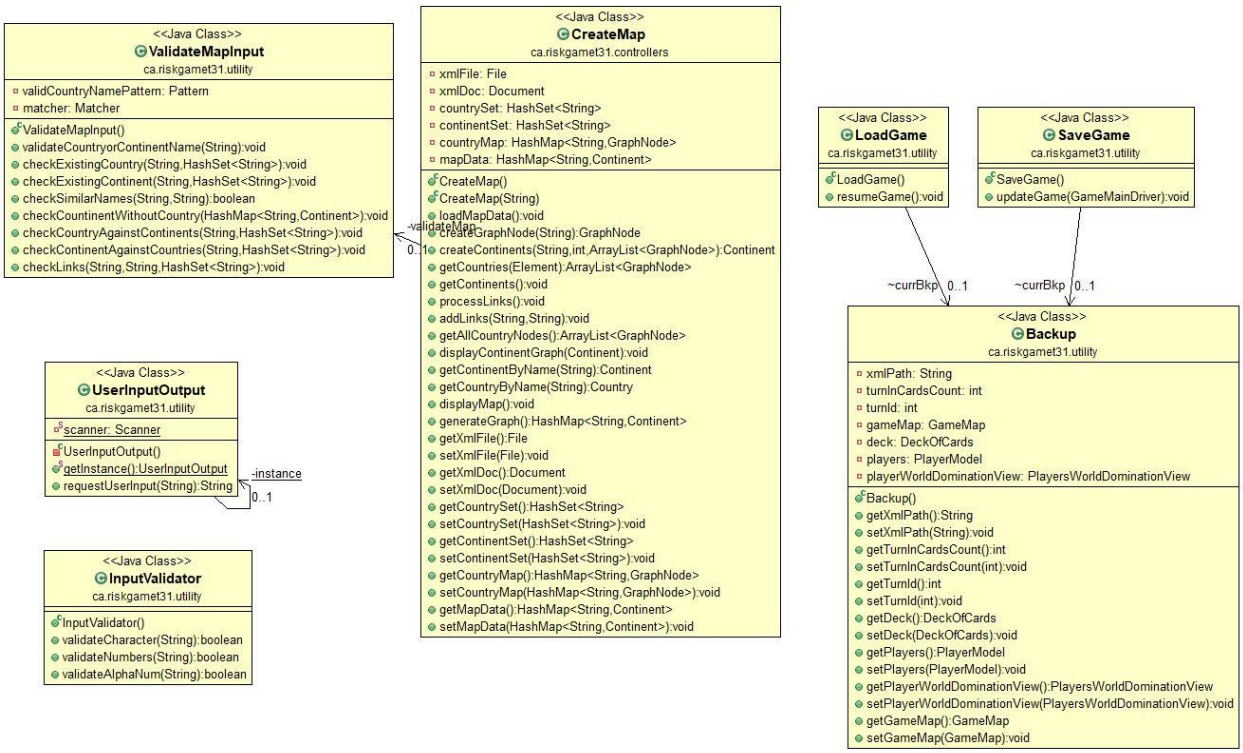
below contains joint test classes and suites for various packages

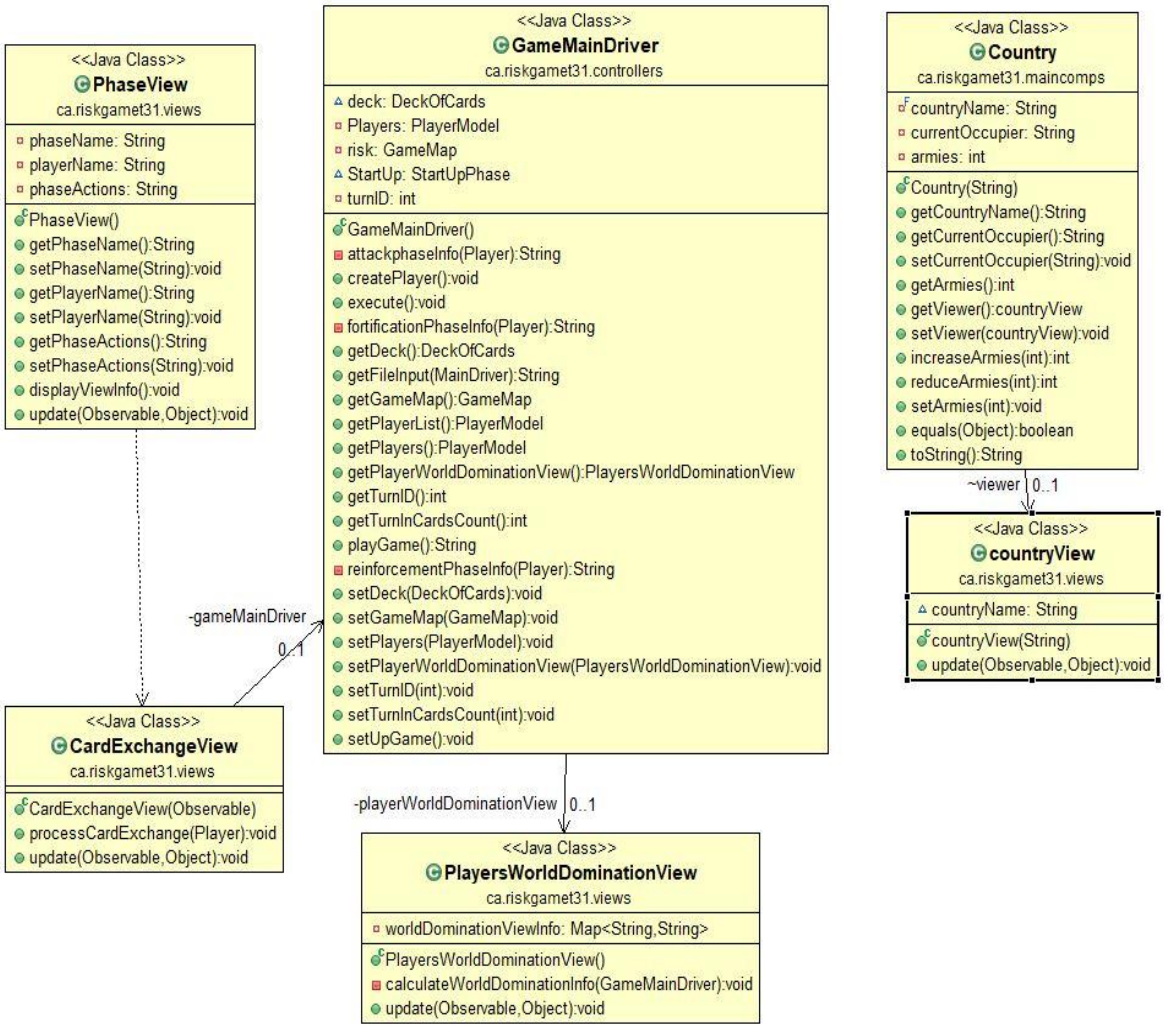
vii. ca.riskgamet31test.maincomps
viii. ca.riskgamet31test.controllers
ix. ca.riskgamet31test.mapdata
x. ca.riskgamet31test.utility

v. Appendix I – UML




<<Java Class>>
PlayerModel
ca.riskgamet31.controllers
PlayerModel()
getPlayerList():ArrayList<Player>
setPlayerList(Player):void
validateNewPlayer(Player):void
<<Java Interface>>
MainDriver
ca.riskgamet31.controllers
phaseInfo: ArrayList<String> = new ArrayList<String>()
getFileInput(MainDriver):String
createGameMap(String):GameMap
createPlayer():void
setUpGame():void
updatePhaseInfo(String,String,String):void
playGame():String
execute():void
getPlayerList():PlayerModel
<<Java Class>>
StartUpPhase
ca.riskgamet31.controllers
playerCount: int
StartUpPhase()
getPlayerCount():int
setPlayerCount(int):void
getArmy(int):int
createPlayers(String):Player
distributeCountries(PlayerModel,GameMap):void
distributeCountriesSequ(PlayerModel,GameMap):void
~PlayerList
0..*
<<Java Interface>>
Player
ca.riskgamet31.maincomps
validateInput(String):void
getplayerName():String
addCountry(GraphNode):void
getPlayerCountriesGNodes():ArrayList<GraphNode>
removeCountry(GraphNode):void
reinforcementArmiesCalc(GameMap,int):int
setArmies(int):void
decrementArmies(int):void
getPlayerArmies():int
addNewCard(Card):void
canFortify():ArrayList<GraphNode>
canAttack():ArrayList<GraphNode>
executeTurnInCard(MainDriver,String):boolean
checkUniqueNumbers(String[]):boolean
canReinforce():GraphNode
getPlayerCards():ArrayList<Card>
getHand():Hand
removeCards(int[]):void
getPlayerGraph():Graph
fortification():void
reinforcement():void
AttDefCountries(MainDriver):ArrayList<GraphNode>
attack(MainDriver):boolean
attackRound(MainDriver,GraphNode,GraphNode,boolean):boolean
getDiceInput(Country,String):int
getDiceInputAllOut(Country,String):int
initialdistributeArmies():void
executeFortification(ArrayList<GraphNode>):void
<<Java Class>>
Graph
ca.riskgamet31.maincomps
Graph(ArrayList<GraphNode>)
Graph()
getGraphNodes():ArrayList<GraphNode>
setGraphNodes(ArrayList<GraphNode>):void
getProcessingList():LinkedList<GraphNode>
addNode(GraphNode):void
removeNode(GraphNode):boolean
cleanProcessingData():void
processGraph():void
isConnected():boolean
findPath(String,String):boolean
viewGraph():void
-graphNodes
0..*
<<Java Class>>
Continent
ca.riskgamet31.maincomps
continentName: String
additionalBonusArmies: int
Continent(String,int,Graph)
getContinentName():String
getAdditionalBonusArmies():int
getContinentGraph():Graph
setContinentGraph(Graph):void
isConnected():boolean
equals(Object):boolean
toString():String
viewContinent():void
-continentGraph
0..1
<<Java Class>>
GraphNode
ca.riskgamet31.maincomps
distance: int
GraphNode(Country)
getNodeData():Country
getNodeNeighbors():ArrayList<GraphNode>
getDistance():int
setDistance(int):void
getParentNode():GraphNode
setParentNode(GraphNode):void
addNeighbor(GraphNode):void
removeNeighbor(GraphNode):void
equals(Object):boolean
toString():String
-nodeData
0..1
-neighbors
0..1
-parentNode
0..1
<<Java Class>>
Country
ca.riskgamet31.maincomps
countryName: String
currentOccupier: String
armies: int
viewer: countryView
Country(String)
getCountryName():String
getCurrentOccupier():String
setCurrentOccupier(String):void
getArmies():int
getViewer():countryView
setViewer(countryView):void
increaseArmies(int):int
reduceArmies(int):int
setArmies(int):void
equals(Object):boolean
toString():String


<<Java Class>>
ValidateMapInput
ca.riskgamet31.utility
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
<<Java Class>>
CreateMap
ca.riskgamet31.controllers
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
<<Java Class>>
LoadGame
ca.riskgamet31.utility
LoadGame()
resumeGame():void
<<Java Class>>
SaveGame
ca.riskgamet31.utility
SaveGame()
updateGame(GameMainDriver):void
~currBkp
0..1
~currBkp
0..1
<<Java Class>>
Backup
ca.riskgamet31.utility
xmlPath: String
turnInCardsCount: int
turnId: int
gameMap: GameMap
deck: DeckOfCards
players: PlayerModel
playerWorldDominationView: PlayersWorldDominationView
Backup()
getXmlPath():String
setXmlPath(String):void
getTurnInCardsCount():int
setTurnInCardsCount(int):void
getTurnId():int
setTurnId(int):void
getDeck():DeckOfCards
setDeck(DeckOfCards):void
getPlayers():PlayerModel
setPlayers(PlayerModel):void
getPlayerWorldDominationView():PlayersWorldDominationView
setPlayerWorldDominationView(PlayersWorldDominationView):void
getGameMap():GameMap
setGameMap(GameMap):void
<<Java Class>>
UserInputOutput
ca.riskgamet31.utility
scanner: Scanner
UserInputOutput()
getInstance():UserInputOutput
requestUserInput(String):String
-instance
0..1
<<Java Class>>
InputValidator
ca.riskgamet31.utility
InputValidator()
validateCharacter(String):boolean
validateNumbers(String):boolean
validateAlphaNum(String):boolean


<<Java Class>>
PhaseView
ca.riskgamet31.views
phaseName: String
playerName: String
phaseActions: String
PhaseView()
getPhaseName():String
setPhaseName(String):void
getPlayerName():String
setPlayerName(String):void
getPhaseActions():String
setPhaseActions(String):void
displayViewInfo():void
update(Observable,Object):void
<<Java Class>>
GameMainDriver
ca.riskgamet31.controllers
deck: DeckOfCards
Players: PlayerModel
risk: GameMap
StartUp: StartUpPhase
turnID: int
GameMainDriver()
attackphaseInfo(Player):String
createPlayer():void
execute():void
fortificationPhaseInfo(Player):String
getDeck():DeckOfCards
getFileInput(MainDriver):String
getGameMap():GameMap
getPlayerList():PlayerModel
getPlayers():PlayerModel
getPlayerWorldDominationView():PlayersWorldDominationView
getTurnID():int
getTurnInCardsCount():int
playGame():String
reinforcementPhaseInfo(Player):String
setDeck(DeckOfCards):void
setGameMap(GameMap):void
setPlayers(PlayerModel):void
setPlayerWorldDominationView(PlayersWorldDominationView):void
setTurnID(int):void
setTurnInCardsCount(int):void
setUpGame():void
<<Java Class>>
Country
ca.riskgamet31.maincomps
countryName: String
currentOccupier: String
armies: int
Country(String)
getCountryName():String
getCurrentOccupier():String
setCurrentOccupier(String):void
getArmies():int
getViewer():countryView
setViewer(countryView):void
increaseArmies(int):int
reduceArmies(int):int
setArmies(int):void
equals(Object):boolean
toString():String
~viewer
0..1
<<Java Class>>
countryView
ca.riskgamet31.views
countryName: String
countryView(String)
update(Observable,Object):void
-gameMainDriver
0..1
<<Java Class>>
CardExchangeView
ca.riskgamet31.views
CardExchangeView(Observable)
processCardExchange(Player):void
update(Observable,Object):void
-playerWorldDominationView
0..1
<<Java Class>>
PlayersWorldDominationView
ca.riskgamet31.views
worldDominationViewInfo: Map<String,String>
PlayersWorldDominationView()
calculateWorldDominationInfo(GameMainDriver):void
update(Observable,Object):void

VI. Appendix II – Refactoring list

Class	Method/code part	Refactor description	Priority/status
Player	Changed to abstraction	For strategy pattern implementation	1
HumanPlayer	Concrete implementation of player	For strategy pattern implementation	2
MainDriver	Abstraction for tournament/single game modes	for bridge implementation	3
GameMainDriver	Concrete implementation of Main driver for single game mode	For single game mode	4
turnInCards	Moved to player	Moved to player as no views are needed in tournament mode	5