RiSK

Design Specification

Team 4

Team 4

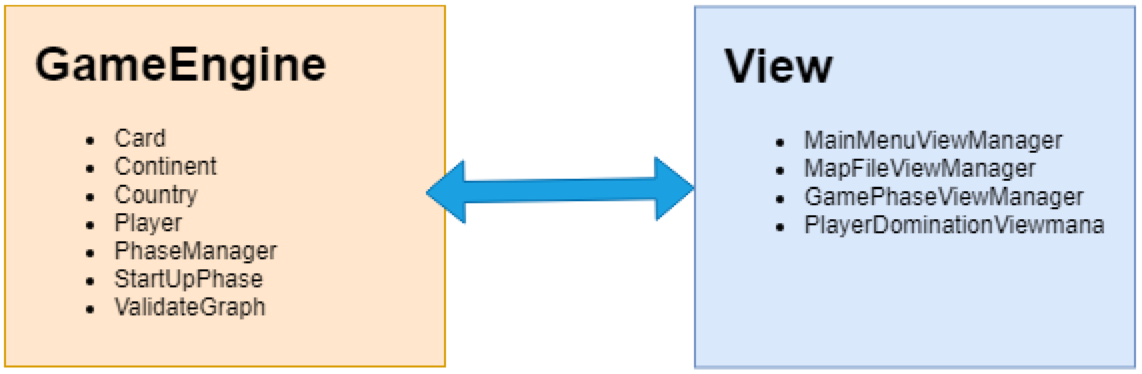
Shivani Panwar, 40092376
Charan Simha Reddy Gangeyedula, 40092878
Sourabh Rajeev Badagandi, 40098471
Aravind Ashoka Reddy, 40103248
Arvind Korchibettu Adiga, 40105178

Contents

1. SCOPE

The document highlights the basic architecture and the high-level design used for the implementation of RiSK Java-based game.

2. ARCHITECTURE



The project follows Observer Pattern and is divided into the following packages:

1. **GameEngine** - This package contains the structure of player, countries, continents and cards, and helps in organizing the game and manipulating the data in an orderly fashion using the PhaseManager. The PhaseManager acts as the Observable and notifies changes to GamePhaseViewManager Observer.

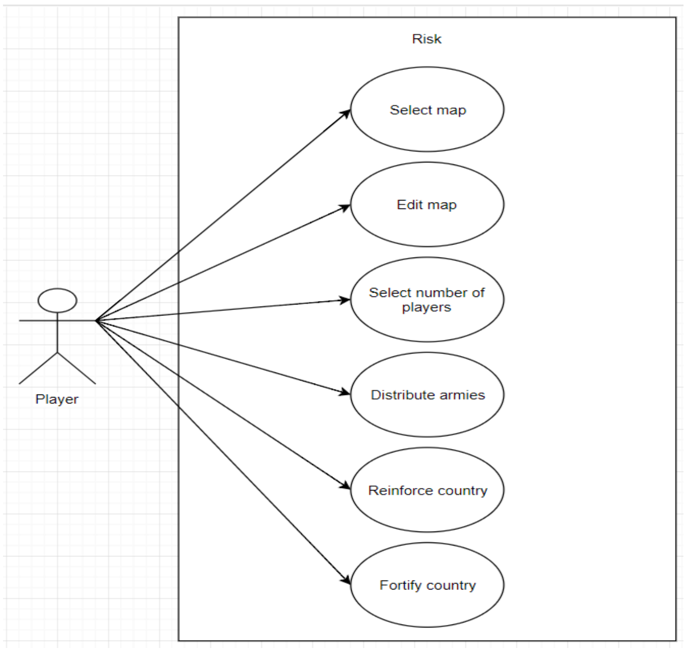
2. **View** - The package is responsible for providing an interface to the user so that he can play and interact with the application. The view consists of options to view, modify or create a map file, and play the game with his custom map. The GamePhaseViewManager is an Observer to the PhaseManager Observable.

3. **Application** - This package contains the main method which is used to start the application by calling the main menu layout method.

4. **Utilities** - This package contains all the methods that will be used by the other packages frequently to perform certain actions.

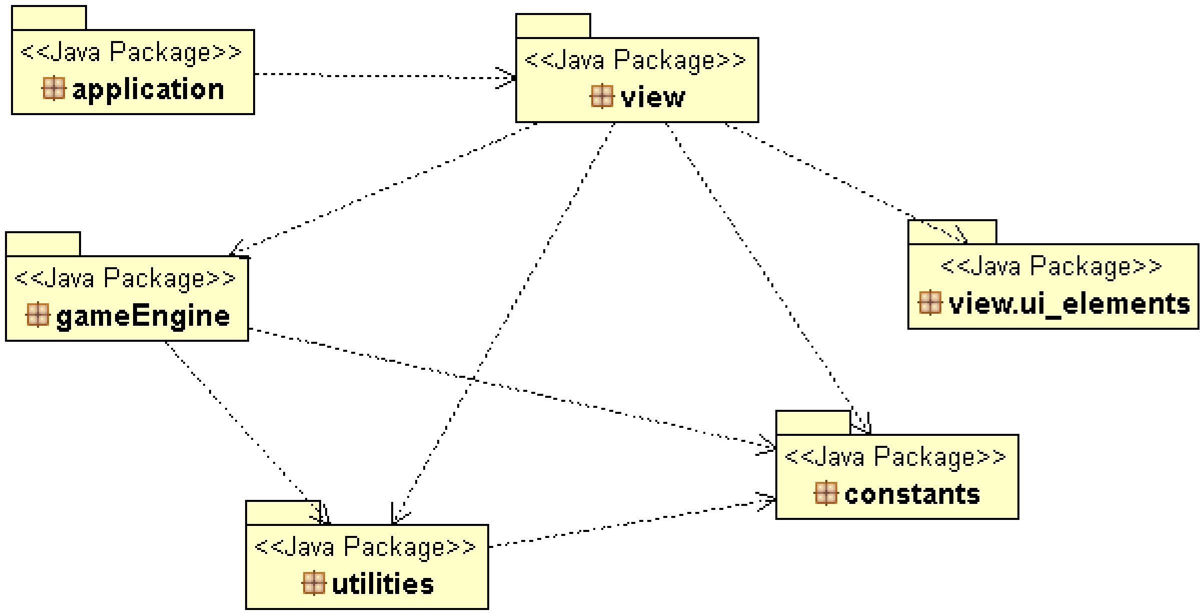
5. **Constants** - This package contains all the constant values for game phases and log levels that will be used throughout the project.
6. **Test** - This package contains all the Junit test cases for individual modules except for the GUI of the project which has been tested manually for correctness.

2.1 Use case diagram



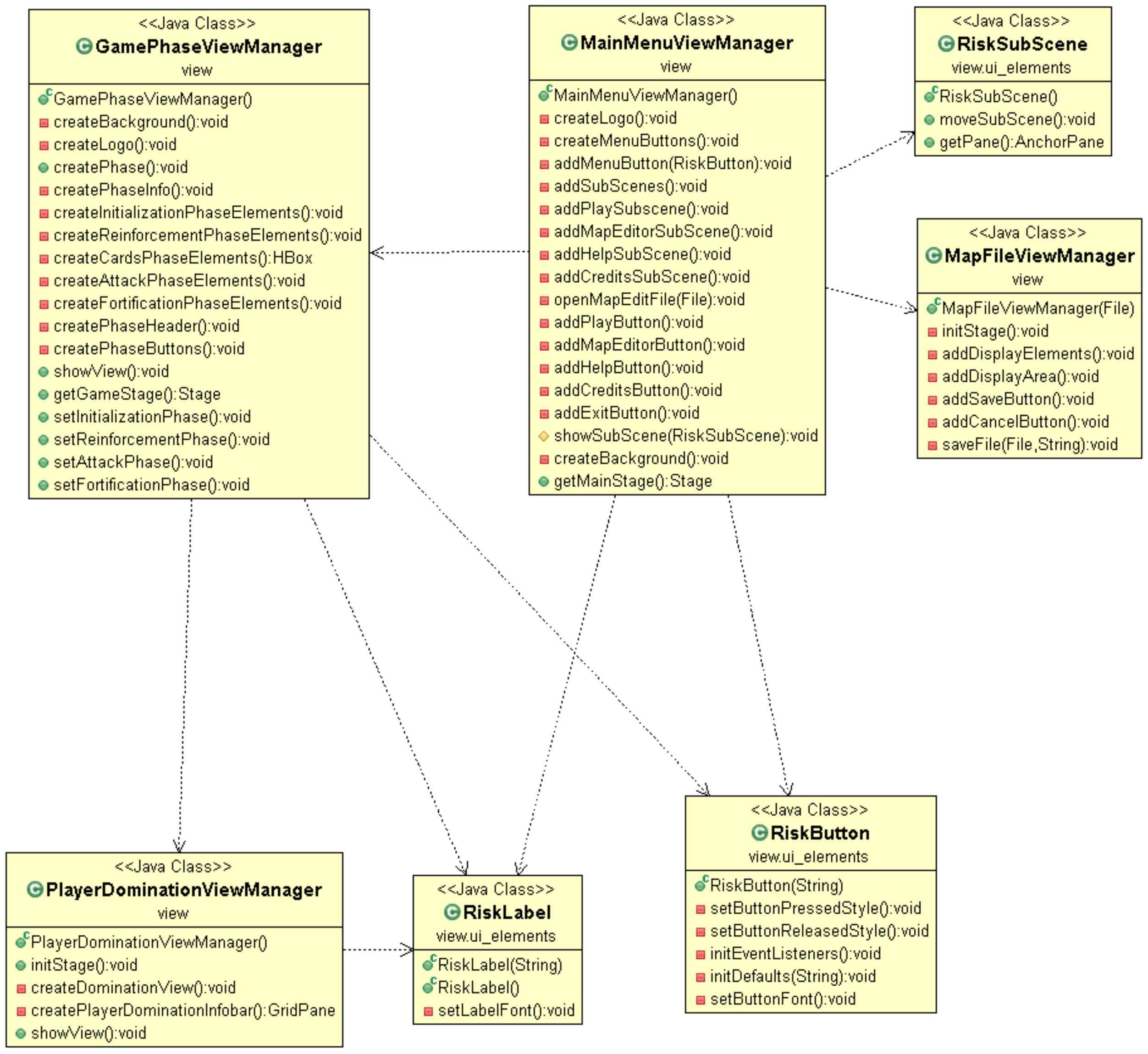
2.2 UML diagram

2.2.1 Package Level Diagram:

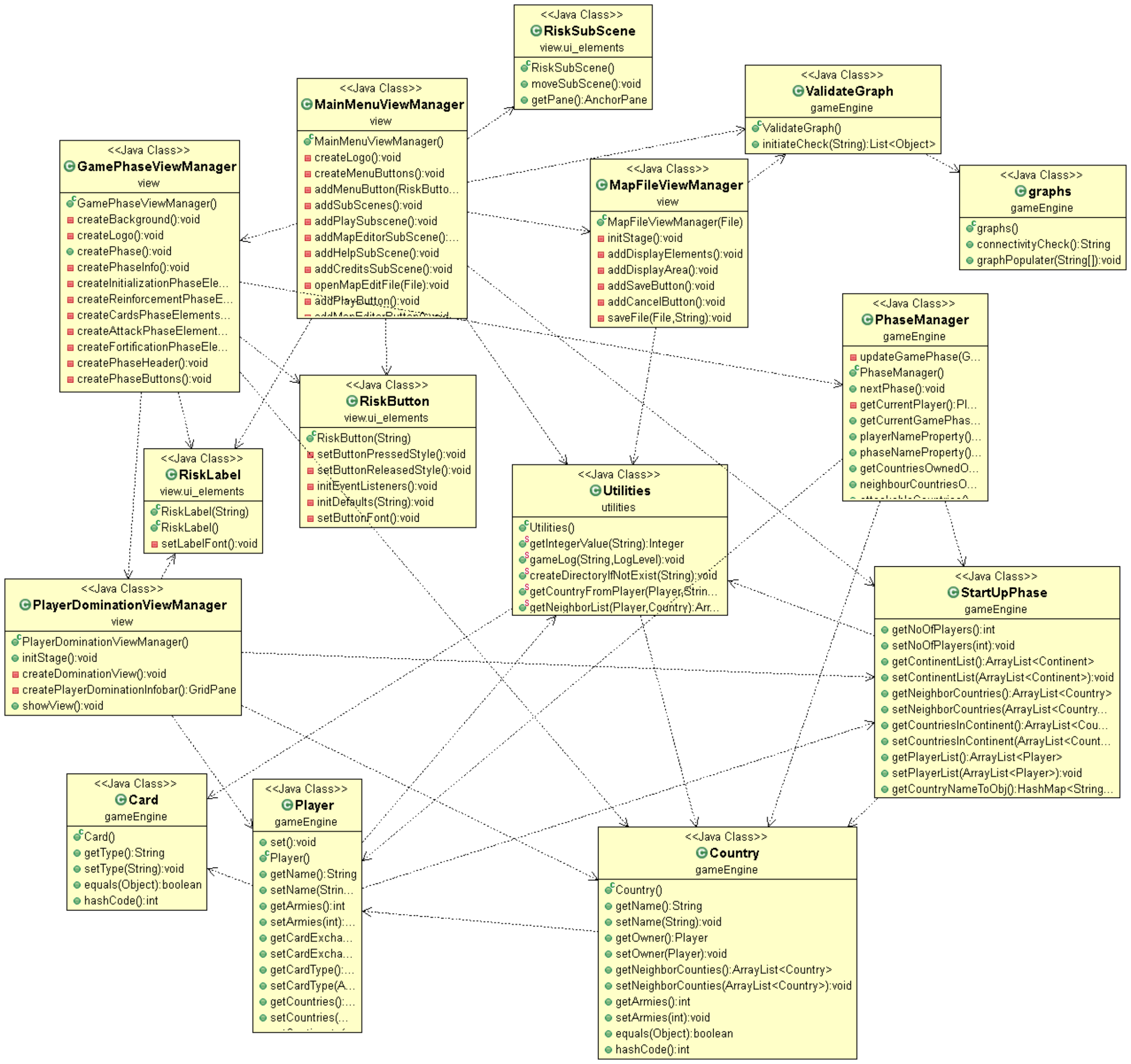

<<Java Package>> application
<<Java Package>> view
<<Java Package>> gameEngine
<<Java Package>> view.ui_elements
<<Java Package>> constants
<<Java Package>> utilities

2.2.2 Class Level Diagram

View:

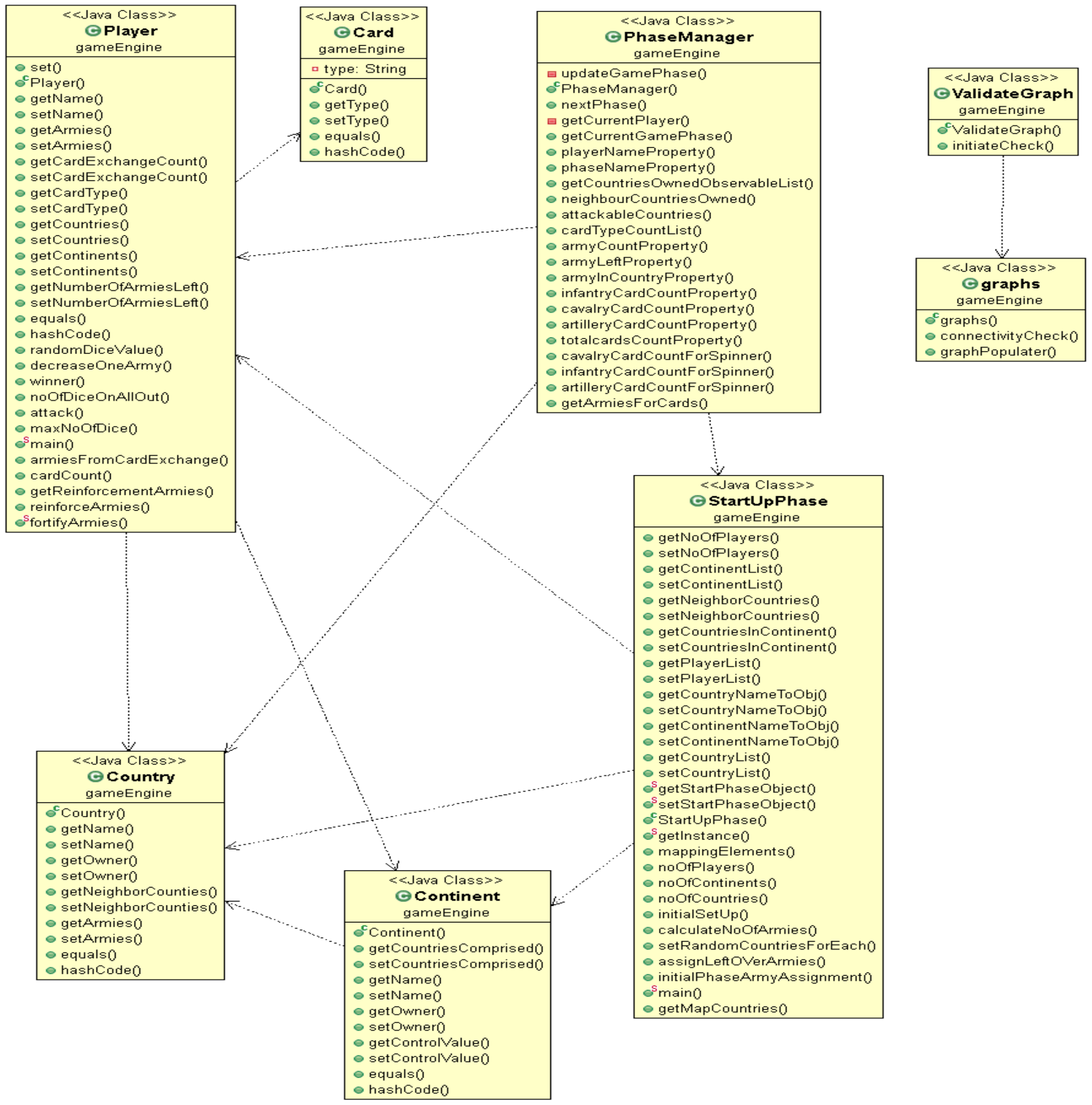


All-Classes:



<<Java Class>>
RiskSubScene
view.ui_elements
RiskSubScene()
moveSubScene():void
getPane():AnchorPane
<<Java Class>>
MainMenuViewManager
view
MainMenuViewManager()
createLogo():void
createMenuButtons():void
addMenuButton(RiskButto...
addSubScenes():void
addPlaySubscene():void
addMapEditorSubScene():...
addHelpSubScene():void
addCreditsSubScene():void
openMapEditFile(File):void
addPlayButton():void
<<Java Class>>
ValidateGraph
gameEngine
ValidateGraph()
initiateCheck(String):List<Object>
<<Java Class>>
graphs
gameEngine
graphs()
connectivityCheck():String
graphPopulater(String[]):void
<<Java Class>>
GamePhaseViewManager
view
GamePhaseViewManager()
createBackground():void
createLogo():void
createPhase():void
createPhaseInfo():void
createInitializationPhaseEle...
createReinforcementPhaseE...
createCardsPhaseElements...
createAttackPhaseElement...
createFortificationPhaseEle...
createPhaseHeader():void
createPhaseButtons():void
<<Java Class>>
MapFileViewManager
view
MapFileViewManager(File)
initStage():void
addDisplayElements():void
addDisplayArea():void
addSaveButton():void
addCancelButton():void
saveFile(File,String):void
<<Java Class>>
PhaseManager
gameEngine
updateGamePhase(G...
PhaseManager()
nextPhase():void
getCurrentPlayer():Pl...
getCurrentGamePhas...
playerNameProperty()...
phaseNameProperty()...
getCountriesOwnedO...
neighbourCountriesO...
<<Java Class>>
RiskButton
view.ui_elements
RiskButton(String)
setButtonPressedStyle():void
setButtonReleasedStyle():void
initEventListeners():void
initDefaults(String):void
setButtonFont():void
<<Java Class>>
RiskLabel
view.ui_elements
RiskLabel(String)
RiskLabel()
setLabelFont():void
<<Java Class>>
Utilities
utilities
Utilities()
getIntegerValue(String):Integer
gameLog(String,LogLevel):void
createDirectoryIfNotExist(String):void
getCountryFromPlayer(Player,Strin...
getNeighborList(Player,Country):Arr...
<<Java Class>>
StartUpPhase
gameEngine
getNoOfPlayers():int
setNoOfPlayers(int):void
getContinentList():ArrayList<Continent>
setContinentList(ArrayList<Continent>):void
getNeighborCountries():ArrayList<Country>
setNeighborCountries(ArrayList<Country...
getCountriesInContinent():ArrayList<Cou...
setCountriesInContinent(ArrayList<Count...
getPlayerList():ArrayList<Player>
setPlayerList(ArrayList<Player>):void
getCountryNameToObj():HashMap<String...
<<Java Class>>
PlayerDominationViewManager
view
PlayerDominationViewManager()
initStage():void
createDominationView():void
createPlayerDominationInfobar():GridPane
showView():void
<<Java Class>>
Card
gameEngine
Card()
getType():String
setType(String):void
equals(Object):boolean
hashCode():int
<<Java Class>>
Player
gameEngine
set():void
Player()
getName():String
setName(Strin...
getArmies():int
setArmies(int):...
getCardExcha...
setCardExcha...
getCardType():...
setCardType(A...
getCountries():...
setCountries(...
<<Java Class>>
Country
gameEngine
Country()
getName():String
setName(String):void
getOwner():Player
setOwner(Player):void
getNeighborCounties():ArrayList<Country>
setNeighborCounties(ArrayList<Country>):void
getArmies():int
setArmies(int):void
equals(Object):boolean
hashCode():int

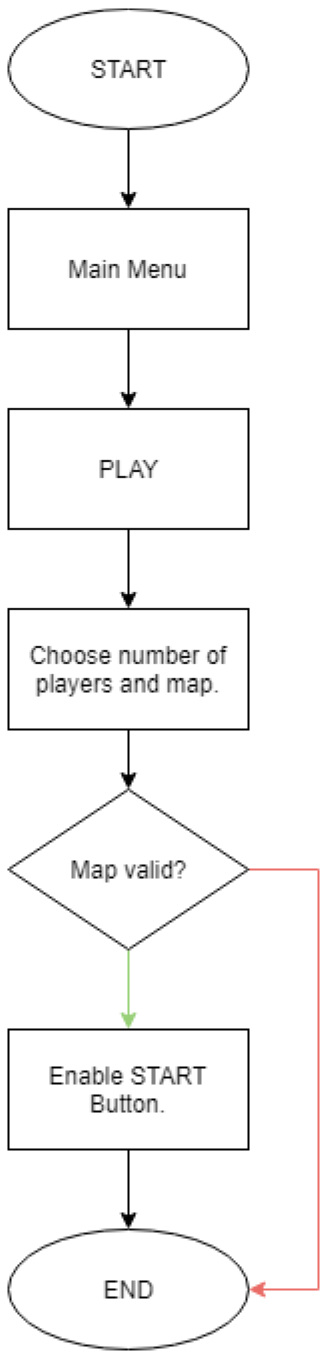
Game Engine



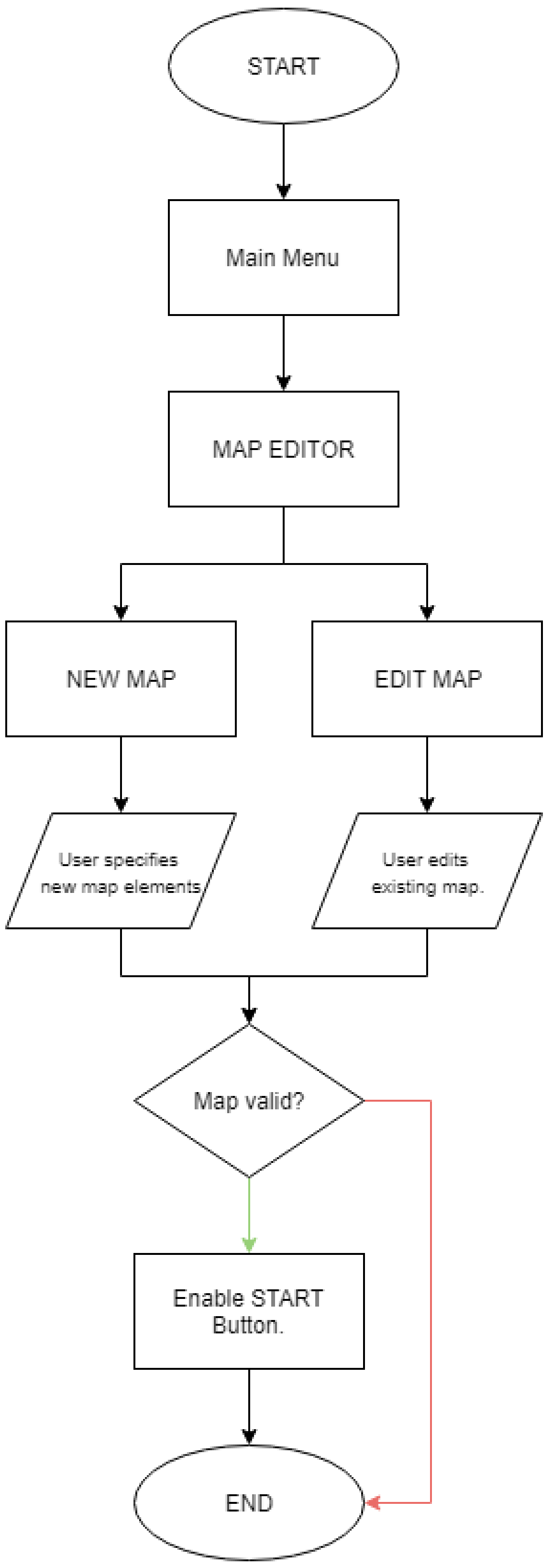
<<Java Class>>
Player
gameEngine
set()
Player()
getName()
setName()
getArmies()
setArmies()
getCardExchangeCount()
setCardExchangeCount()
getCardType()
setCardType()
getCountries()
setCountries()
getContinents()
setContinents()
getNumberOfArmiesLeft()
setNumberOfArmiesLeft()
equals()
hashCode()
randomDiceValue()
decreaseOneArmy()
winner()
noOfDiceOnAllOut()
attack()
maxNoOfDice()
main()
armiesFromCardExchange()
cardCount()
getReinforcementArmies()
reinforceArmies()
fortifyArmies()
<<Java Class>>
Card
gameEngine
type: String
Card()
getType()
setType()
equals()
hashCode()
<<Java Class>>
PhaseManager
gameEngine
updateGamePhase()
PhaseManager()
nextPhase()
getCurrentPlayer()
getCurrentGamePhase()
playerNameProperty()
phaseNameProperty()
getCountriesOwnedObservableList()
neighbourCountriesOwned()
attackableCountries()
cardTypeCountList()
armyCountProperty()
armyLeftProperty()
armyInCountryProperty()
infantryCardCountProperty()
cavalryCardCountProperty()
artilleryCardCountProperty()
totalcardsCountProperty()
cavalryCardCountForSpinner()
infantryCardCountForSpinner()
artilleryCardCountForSpinner()
getArmiesForCards()
<<Java Class>>
ValidateGraph
gameEngine
ValidateGraph()
initiateCheck()
<<Java Class>>
graphs
gameEngine
graphs()
connectivityCheck()
graphPopulater()
<<Java Class>>
StartUpPhase
gameEngine
getNoOfPlayers()
setNoOfPlayers()
getContinentList()
setContinentList()
getNeighborCountries()
setNeighborCountries()
getCountriesInContinent()
setCountriesInContinent()
getPlayerList()
setPlayerList()
getCountryNameToObj()
setCountryNameToObj()
getContinentNameToObj()
setContinentNameToObj()
getCountryList()
setCountryList()
getStartPhaseObject()
setStartPhaseObject()
StartUpPhase()
getInstance()
mappingElements()
noOfPlayers()
noOfContinents()
noOfCountries()
initialSetUp()
calculateNoOfArmies()
setRandomCountriesForEach()
assignLeftOVerArmies()
initialPhaseArmyAssignment()
main()
getMapCountries()
<<Java Class>>
Country
gameEngine
Country()
getName()
setName()
getOwner()
setOwner()
getNeighborCounties()
setNeighborCounties()
getArmies()
setArmies()
equals()
hashCode()
<<Java Class>>
Continent
gameEngine
Continent()
getCountriesComprised()
setCountriesComprised()
getName()
setName()
getOwner()
setOwner()
getControlValue()
setControlValue()
equals()
hashCode()

2.3 Basic Flow Diagram

2.3.1 Main Flow Diagram



2.3.2 Map Editor Flow Diagram



3. CODING CONVENTIONS

The project abides by the coding standards used in java. Following are the standards being used:

Naming Conventions:

1. All the methods have logical and self-explanatory names and camel case is used to denote the methods.
2. All the local variables use camel case.
3. Constants used across the project are in a separate package and are denoted by capital letters.

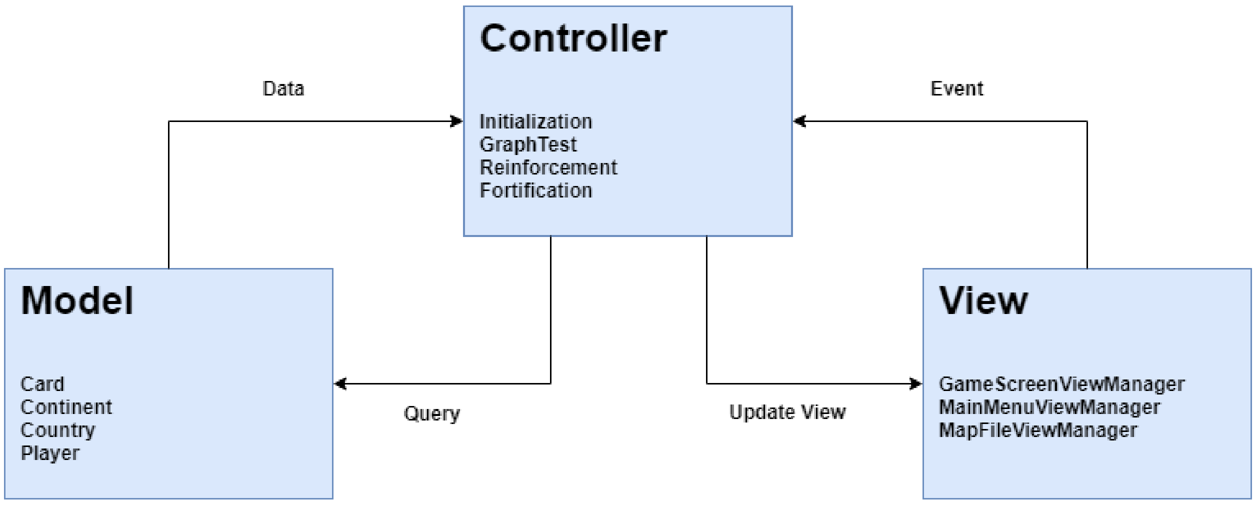
Comments:
Javadoc comments are used before all the classes and the methods to describe the business logic behind it. Blocks of code that are not self explanatory have comments describing their functioning.

Indentation and Layout:
The code is indented using the standard tab i.e., 8 spaces. Different sections in the same class have been separated by blank lines. The braces for loops, methods and classes are kept in the same line.

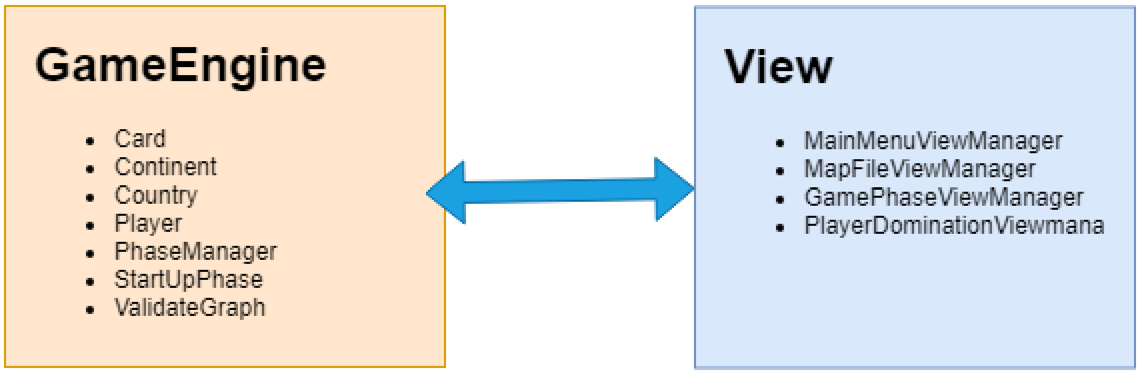
Exception handling:
Logs are being maintained in a text file to capture both successful and unsuccessful moves. All exceptions are being handled and they can be tracked using the log files.

4. REFACTORING TARGETS

Previous Architecture



Current Architecture



1. Controller Package
   a. Move Reinforcement Operations into Player Class.
   b. Move Fortification Operations into Player Class.
   c. Move StartUpPhase operation into GameEngine Package.
2. GameEngine Package
   a. Move Classes from Model package to GameEngine package.

5. TOOLS AND RESOURCES USED

1. Eclipse IDE for development.
2. JavaFX (e(fx)clipse, version: 3.0.0) is used for the GUI of the project.
3. JGraphT library is used to validate whether the map is a connected graph or not.
4. Junit4 for Unit Testing.
5. UMl Lab Class Diagram Editor (Version 1.13.0) is used to generate all the UML diagrams involving the packages and classes.
6. Javadoc is used to generate all the API documents for all the methods and classes.
7. Kenney (www.kenney.nl) game assets for fonts and background images.