RiSK

Design Specification

Team 4

Team 4

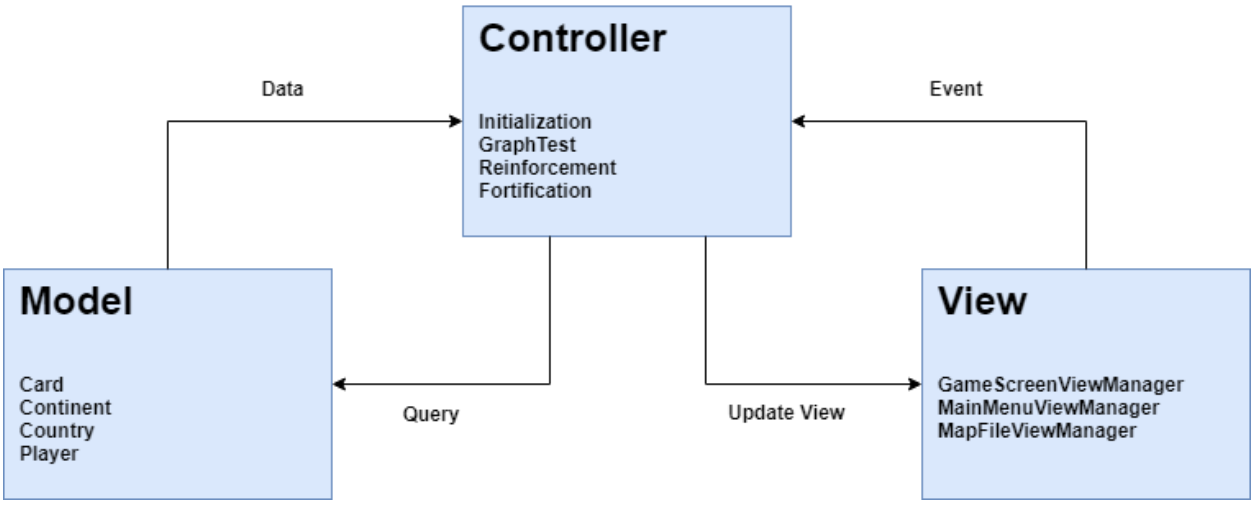
Shivani Panwar, 40092376
Charan Simha Reddy Gangeyedula, 40092878
Sourabh Rajeev Badagandi, 40098471
Aravind Ashoka Reddy, 40103248
Arvind Korchibettu Adiga, 40105178

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 Model, View, Controller architecture and is divided into the following packages:

1. **Model** - 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.

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.

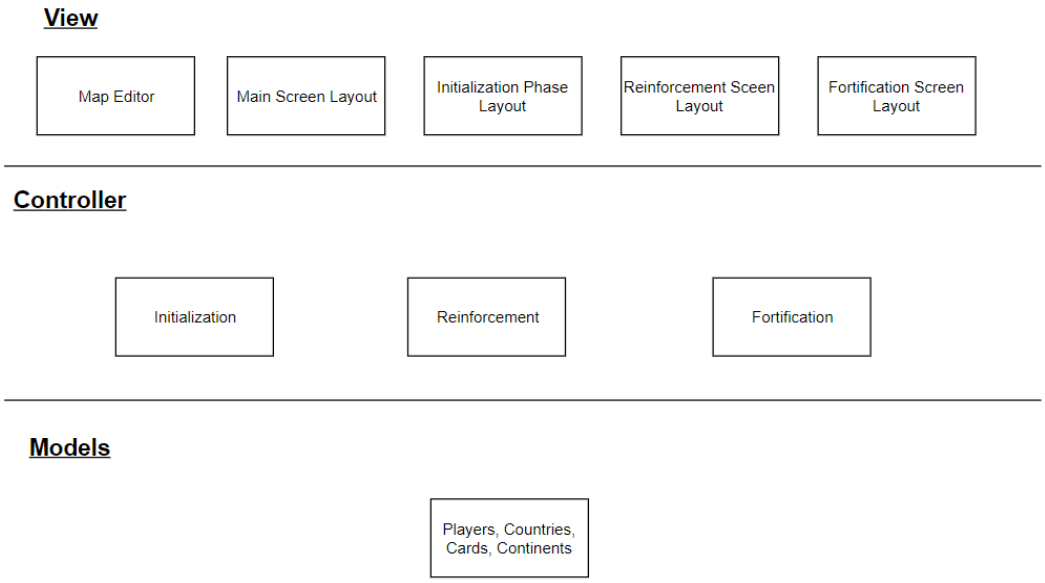
3. **Controller** - The package of the project that contains all the classes pertaining to the phases of the game. The package contains the following packages:
   i. ***Initialization***: This package contains the class with all the methods that are required to initialize a player before the game starts. Actions like random allocation of countries, initial allocation of armies are done in this class.
   ii. ***Reinforcement***: This package contains the class with all the methods for calculating the number of armies that the player will get for the reinforcement phase and also the method that performs the reinforcement.
   iii. ***Fortification***: This package contains the class that is used for the fortification phase of the game. It contains the method that allows the player to move armies from one country to another.

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

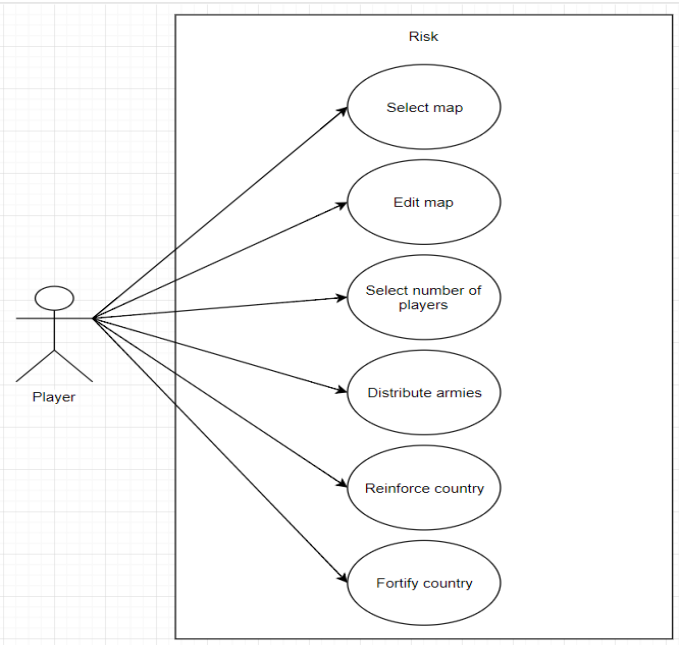
5. **Utilities** - This package contains all the methods that will be used by the other packages frequently to perform certain actions.

6. **Constants** - This package contains all the constant values for game phases and log levels that will be used throughout the project.
7. **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 High Level Diagram

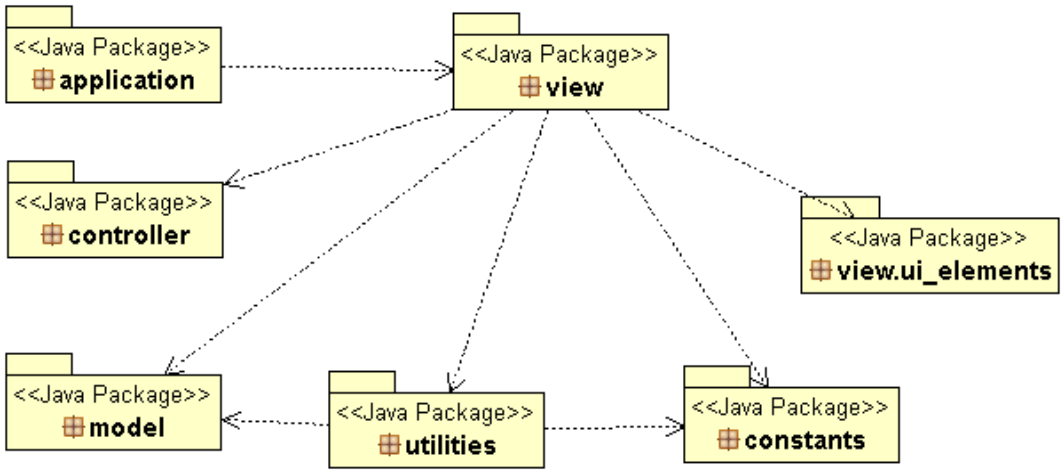


2.2 Use case diagram



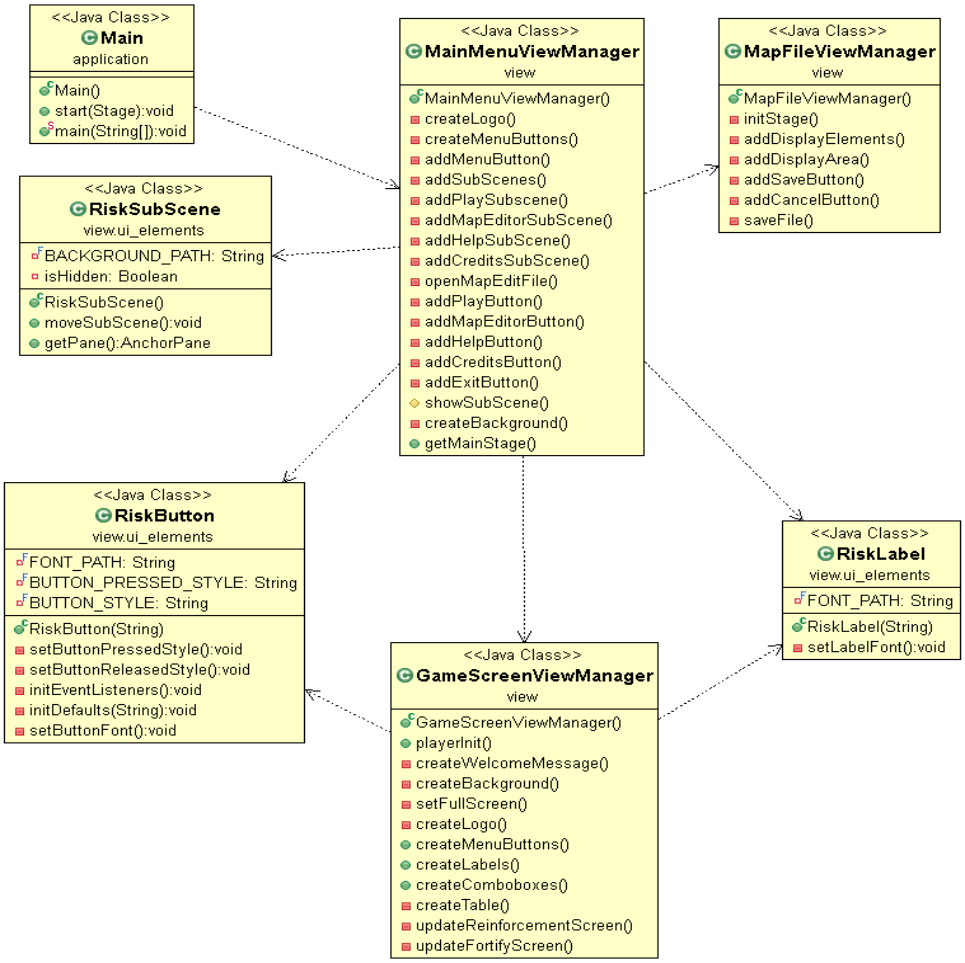
2.3 UML diagram

2.3.1 Package Level Diagram:

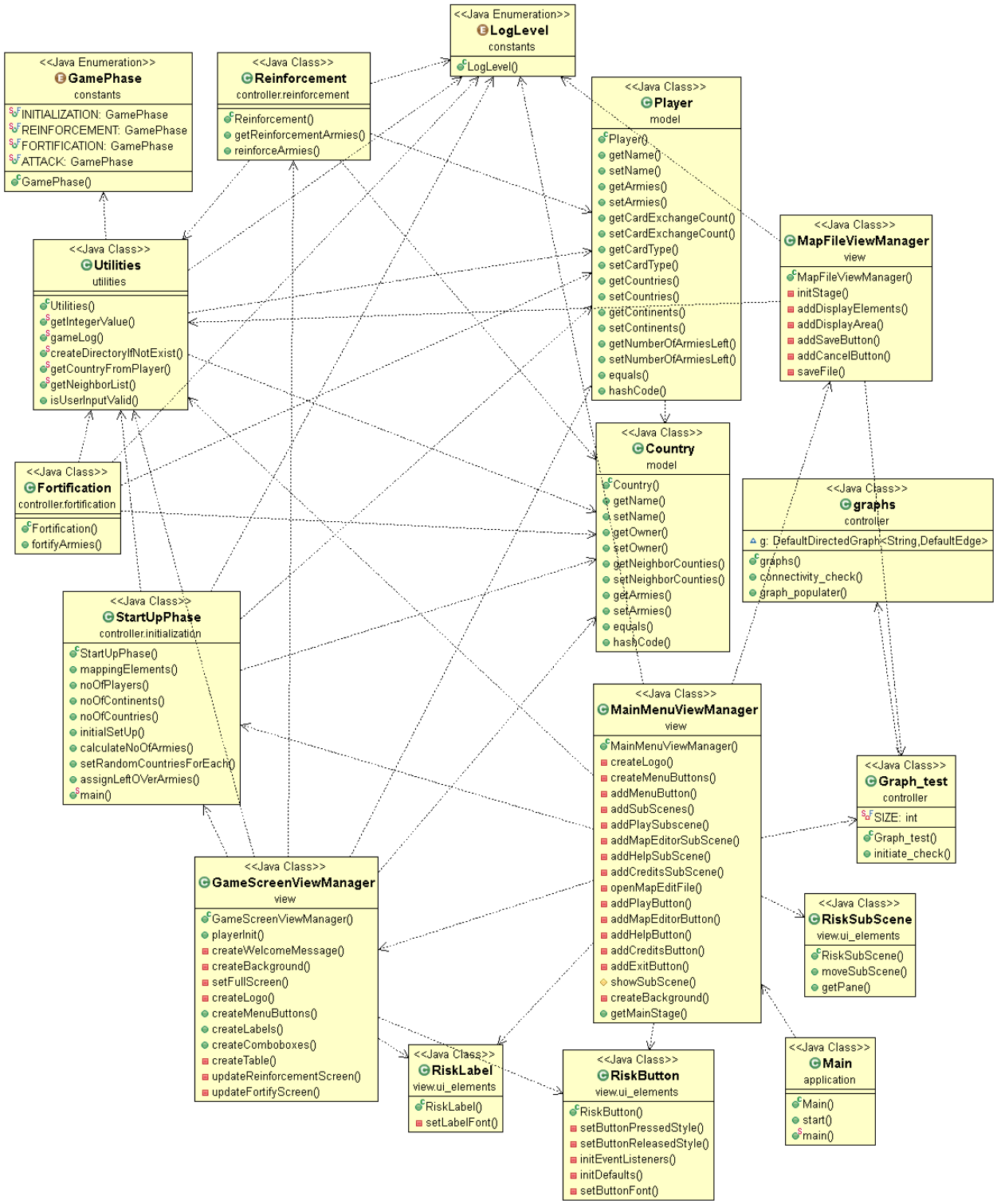


2.3.2 Class Level Diagram

View:

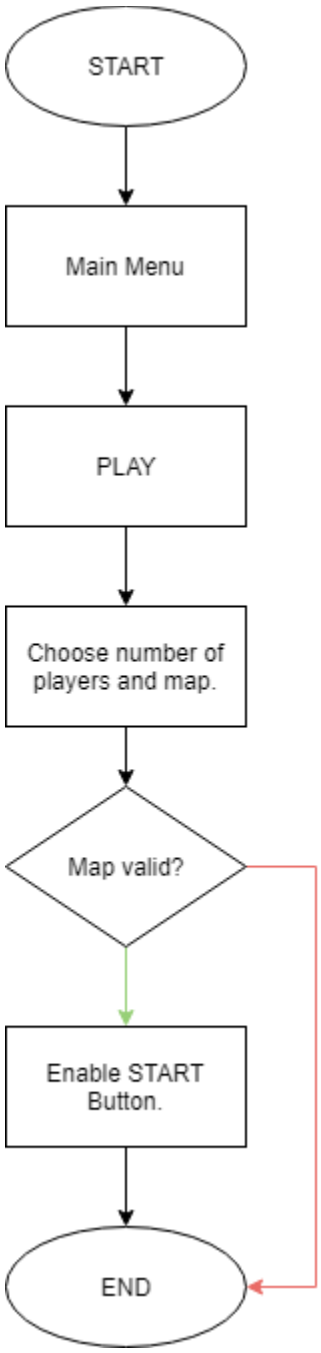


All-Classes:

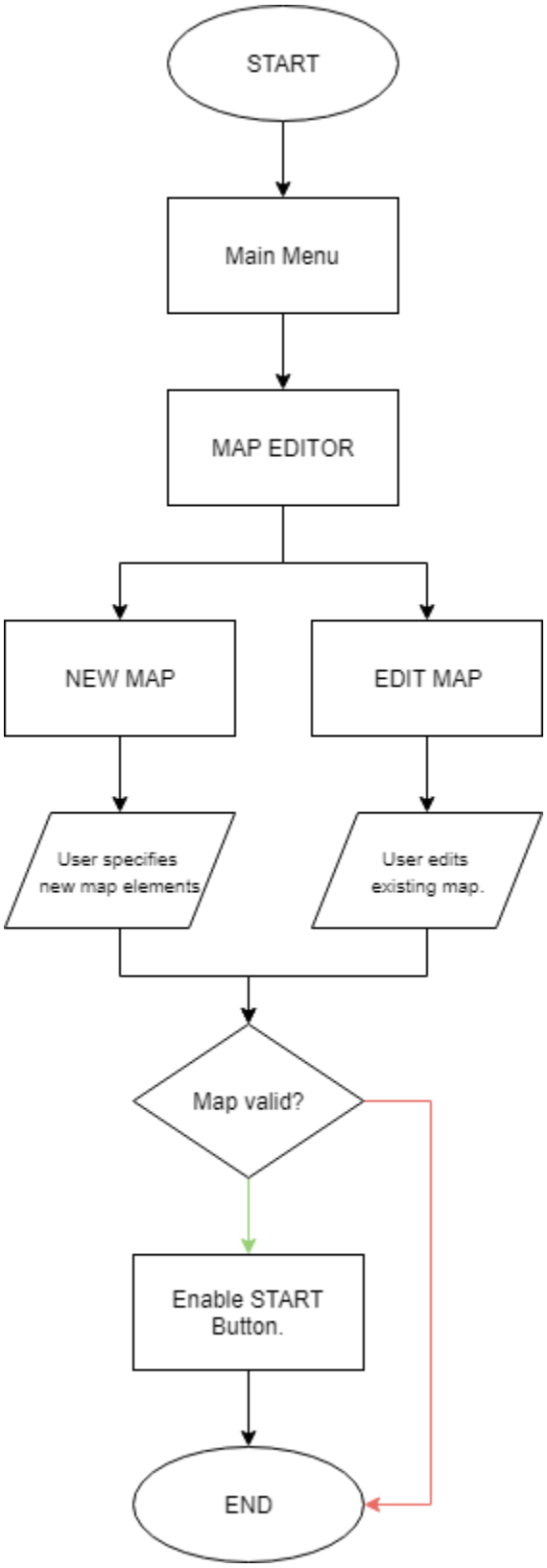

<<Java Enumeration>>
LogLevel
constants
LogLevel()
<<Java Enumeration>>
GamePhase
constants
INITIALIZATION: GamePhase
REINFORCEMENT: GamePhase
FORTIFICATION: GamePhase
ATTACK: GamePhase
GamePhase()
<<Java Class>>
Reinforcement
controller.reinforcement
Reinforcement()
getReinforcementArmies()
reinforceArmies()
<<Java Class>>
Player
model
Player()
getName()
setName()
getArmies()
setArmies()
getCardExchangeCount()
setCardExchangeCount()
getCardType()
setCardType()
getCountries()
setCountries()
getContinents()
setContinents()
getNumberOfArmiesLeft()
setNumberOfArmiesLeft()
equals()
hashCode()
<<Java Class>>
MapFileViewManager
view
MapFileViewManager()
initStage()
addDisplayElements()
addDisplayArea()
addSaveButton()
addCancelButton()
saveFile()
<<Java Class>>
Utilities
utilities
Utilities()
getIntegerValue()
gameLog()
createDirectoryIfNotExist()
getCountryFromPlayer()
getNeighborList()
isUserInputValid()
<<Java Class>>
Country
model
Country()
getName()
setName()
getOwner()
setOwner()
getNeighborCounties()
setNeighborCounties()
getArmies()
setArmies()
equals()
hashCode()
<<Java Class>>
Fortification
controller.fortification
Fortification()
fortifyArmies()
<<Java Class>>
graphs
controller
g: DefaultDirectedGraph<String,DefaultEdge>
graphs()
connectivity_check()
graph_populater()
<<Java Class>>
StartUpPhase
controller.initialization
StartUpPhase()
mappingElements()
noOfPlayers()
noOfContinents()
noOfCountries()
initialSetUp()
calculateNoOfArmies()
setRandomCountriesForEach()
assignLeftOVerArmies()
main()
<<Java Class>>
MainMenuViewManager
view
MainMenuViewManager()
createLogo()
createMenuButtons()
addMenuButton()
addSubScenes()
addPlaySubscene()
addMapEditorSubScene()
addHelpSubScene()
addCreditsSubScene()
openMapEditFile()
addPlayButton()
addMapEditorButton()
addHelpButton()
addCreditsButton()
addExitButton()
showSubScene()
createBackground()
getMainStage()
<<Java Class>>
Graph_test
controller
SIZE: int
Graph_test()
initiate_check()
<<Java Class>>
GameScreenViewManager
view
GameScreenViewManager()
playerInit()
createWelcomeMessage()
createBackground()
setFullScreen()
createLogo()
createMenuButtons()
createLabels()
createComboboxes()
createTable()
updateReinforcementScreen()
updateFortifyScreen()
<<Java Class>>
RiskSubScene
view.ui_elements
RiskSubScene()
moveSubScene()
getPane()
<<Java Class>>
RiskLabel
view.ui_elements
RiskLabel()
setLabelFont()
<<Java Class>>
RiskButton
view.ui_elements
RiskButton()
setButtonPressedStyle()
setButtonReleasedStyle()
initEventListeners()
initDefaults()
setButtonFont()
<<Java Class>>
Main
application
Main()
start()
main()

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. 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.