CONCORDIA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

SOEN-6441

Risk game - build 3

Application Design document

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

Team# 31

1. 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.

1. High level system architecture\*

The application consists of below main components and functionalities

* 1. Game main model components
     1. Graph node: consists of graph data (the country) and links to neighboring graph nodes
     2. 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.
     3. 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.
     4. 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.

* + 1. Game map: holder for played game map data (continents, reference to map xml file…etc.)
    2. 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.
    3. 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.
    4. Players: holder for all players playing the game.
  1. Utility package: provides utility classes as below
     1. Constants: representing default number of armies each player should have in startup phase based on number of players playing the game.
     2. Backup / save game and load game: utility classes used to save the game and reloaded using java serialization.
  2. Exceptions: customized exceptions representing invalid runtime faults related to graph, continent, country, naming conventions, player … etc.
  3. Map creation and validation: to read xml map file, build continent and game map objects and validate its data.
  4. 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.
     1. start with start-up phase (creation, edit and loading of map xml files), then
     2. creation of game model objects (continents, game map, players… etc.)
     3. initial distribution of counties (randomly)
     4. initial distribution of players armies
     5. 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).
     6. 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

1. Used Technology and references

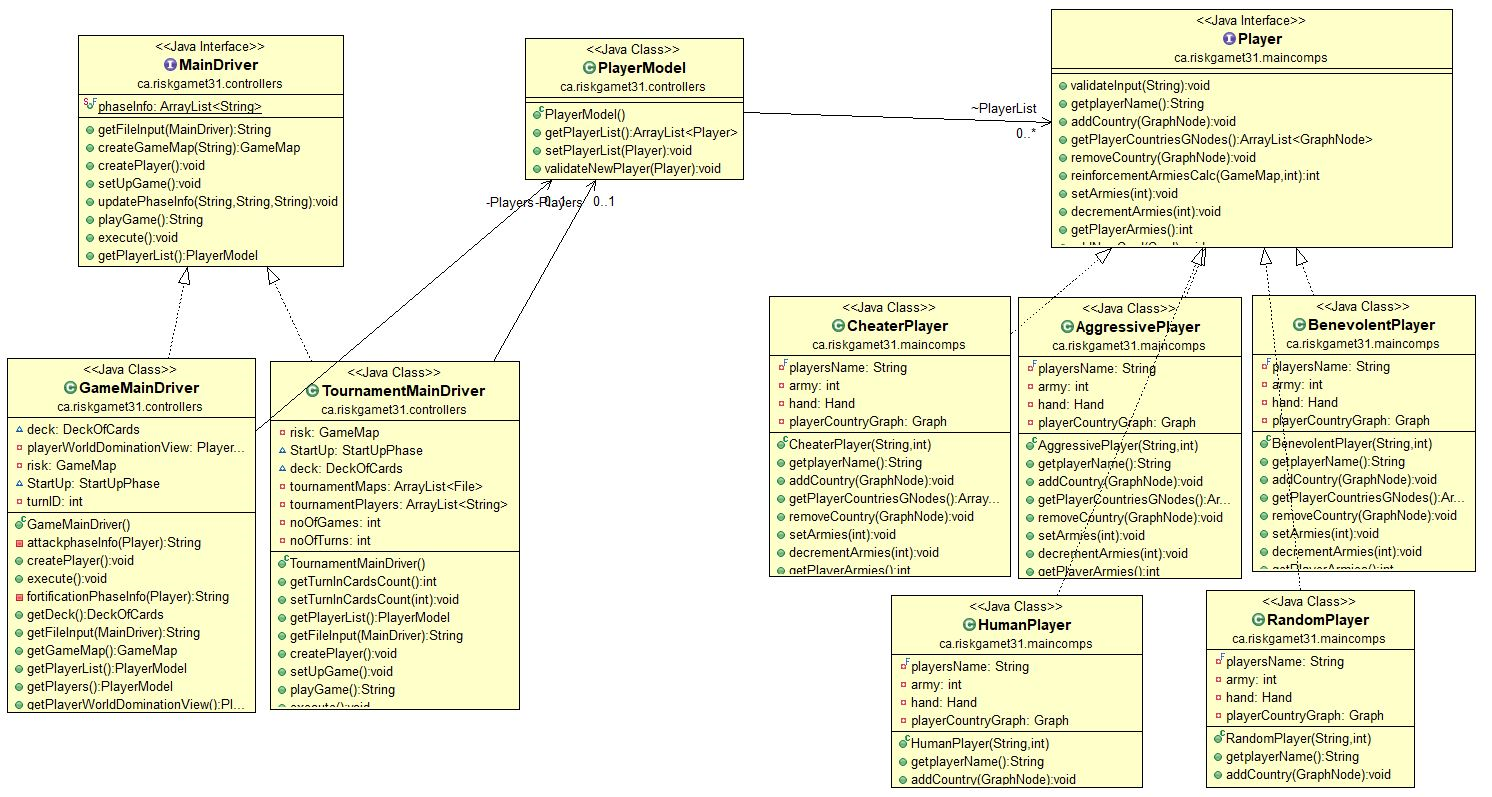
* Standard Java library (JDK 1.8) including java doc tool
* Junit 4
* Brown university coding standard - java style
* Course material and reference used for design patterns (Singleton, Observer, Strategy)
* Bitbucket GIT repository
* Eclipse IDE

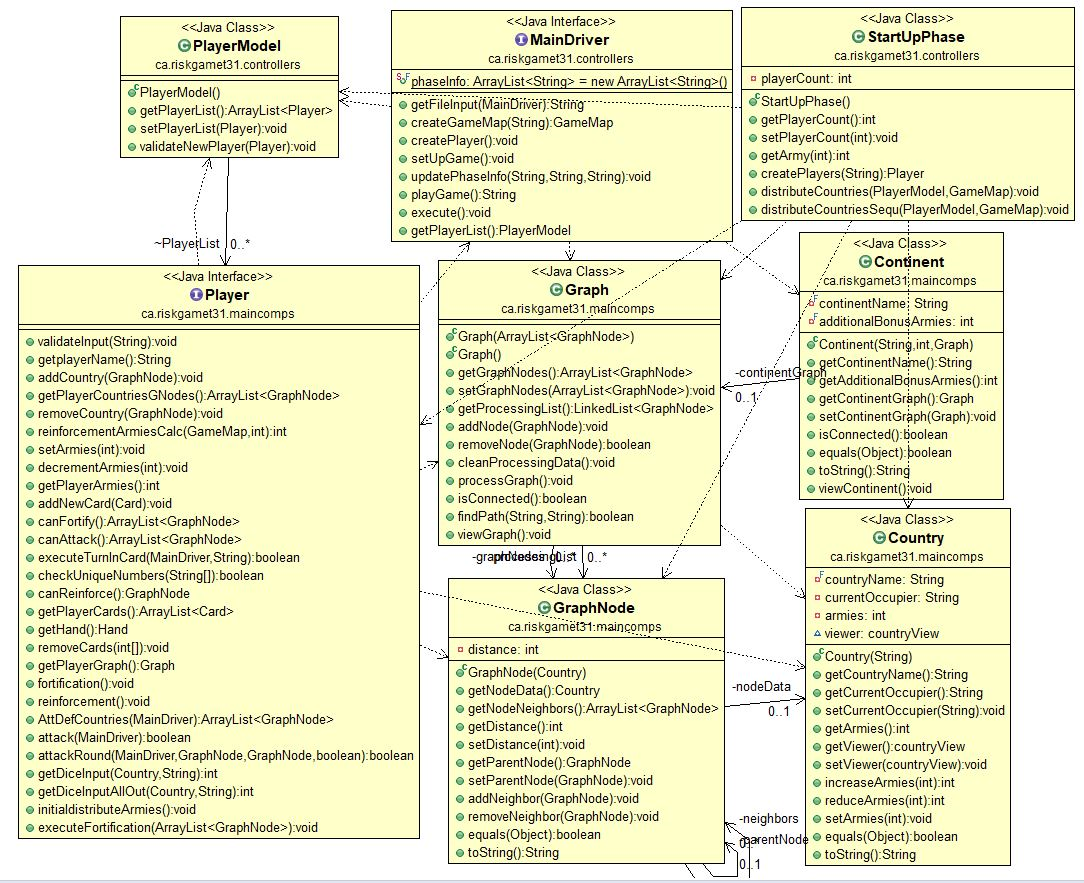
1. Folders and packages structure
   1. Folders structure
      1. src: contains application source code files
         1. /ca
            1. /riskgamet31: contains application source code packages and files
            2. /riskgamet31test: contains Junit test package, suite and classes
      2. ProjDocs: contains project reference files (i.e. coding style reference, assignment 1 documentation and presentation.
      3. Risk\_JDocs: contains the automatically generated java documentation using jdoc tool.
      4. Risk\_MapData: contains several valid and invalid maps source xml files.
   2. Packages structure
      1. ca.riskgamet31.utilitiy: utility classes(constants …etc)
      2. ca.riskgamet31.controllers: controllers classes
      3. ca.riskgamet31.exceptions: customized exception classes
      4. ca.riskgamet31.maincomps: game main model classes
      5. ca.riskgamet31.mapdata: map related classes
      6. ca.riskgamet31.views: views classes

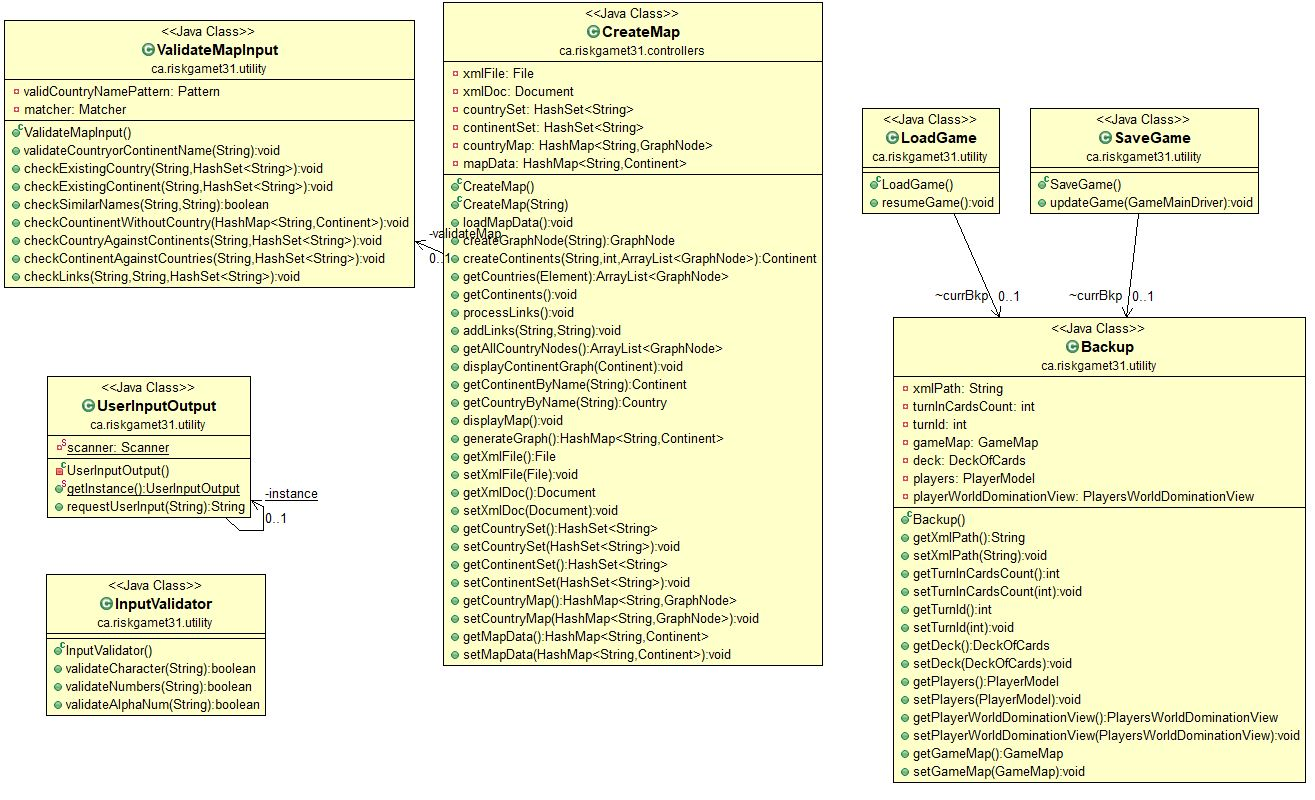
below contains joint test classes and suites for various packages

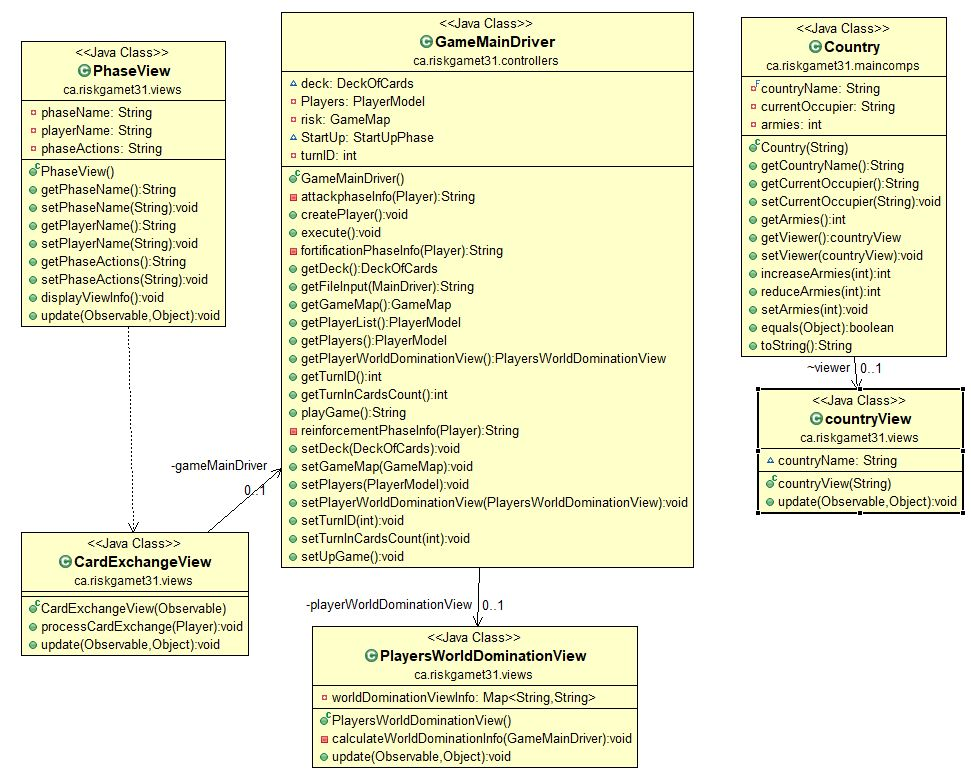
* + 1. ca.riskgamet31test.maincomps
    2. ca.riskgamet31test.controllers
    3. ca.riskgamet31test.mapdata
    4. ca.riskgamet31test.utility

1. Appendix I – UML









1. 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 |