SOEN 6441 - Advanced Programming Practices

Project - Risk Game (Build #1)

Fall 2023

Submitted by:-

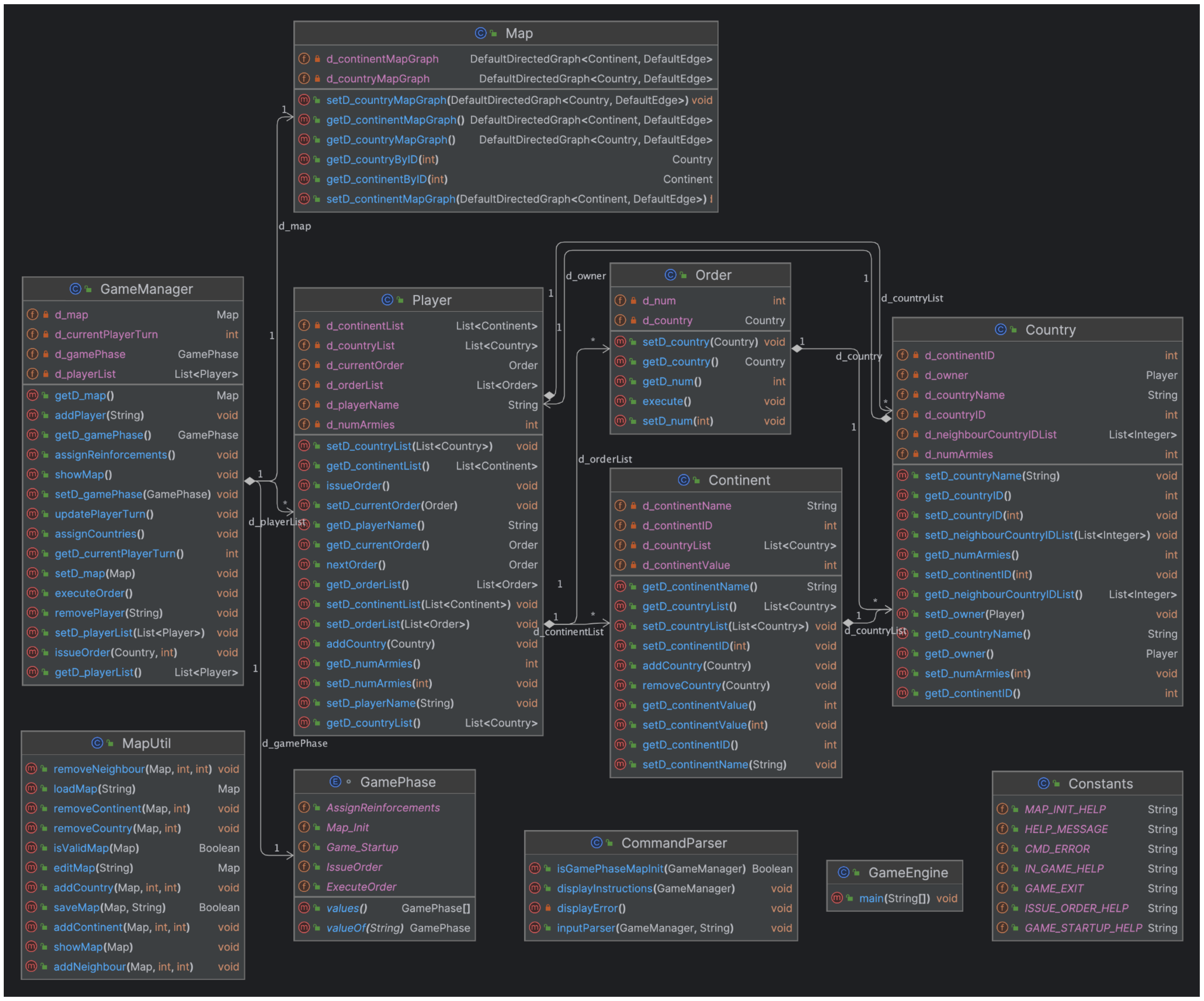
Rishi Ravikumar
Anuja Ajay Somthankar
Yusuke Ishii
Nimisha Mavjibhai Jadav
Abhigyan Singh

Submitted to:-

Dr. Amin Ranj Bar

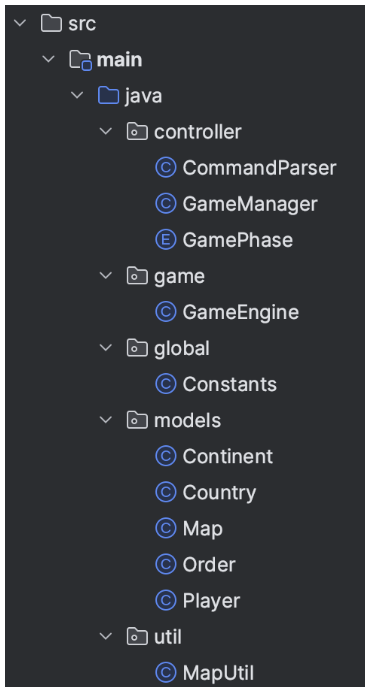
Build #1 - Architectural Design

UML Diagram



Map
d_continentMapGraph DefaultDirectedGraph<Continent, DefaultEdge>
d_countryMapGraph DefaultDirectedGraph<Country, DefaultEdge>
setD_countryMapGraph(DefaultDirectedGraph<Country, DefaultEdge>) void
getD_continentMapGraph() DefaultDirectedGraph<Continent, DefaultEdge>
getD_countryMapGraph() DefaultDirectedGraph<Country, DefaultEdge>
getD_countryByID(int) Country
getD_continentByID(int) Continent
setD_continentMapGraph(DefaultDirectedGraph<Continent, DefaultEdge>)
GameManager
d_map Map
d_currentPlayerTurn int
d_gamePhase GamePhase
d_playerList List<Player>
getD_map() Map
addPlayer(String) void
getD_gamePhase() GamePhase
assignReinforcements() void
showMap() void
setD_gamePhase(GamePhase) void
updatePlayerTurn() void
assignCountries() void
getD_currentPlayerTurn() int
setD_map(Map) void
executeOrder() void
removePlayer(String) void
setD_playerList(List<Player>) void
issueOrder(Country, int) void
getD_playerList() List<Player>
Player
d_continentList List<Continent>
d_countryList List<Country>
d_currentOrder Order
d_orderList List<Order>
d_playerName String
d_numArmies int
setD_countryList(List<Country>) void
getD_continentList() List<Continent>
issueOrder() void
setD_currentOrder(Order) void
getD_playerName() String
getD_currentOrder() Order
nextOrder() Order
getD_orderList() List<Order>
setD_continentList(List<Continent>) void
setD_orderList(List<Order>) void
addCountry(Country) void
getD_numArmies() int
setD_numArmies(int) void
setD_playerName(String) void
getD_countryList() List<Country>
Order
d_num int
d_country Country
setD_country(Country) void
getD_country() Country
getD_num() int
execute() void
setD_num(int) void
Continent
d_continentName String
d_continentID int
d_countryList List<Country>
d_continentValue int
getD_continentName() String
getD_countryList() List<Country>
setD_countryList(List<Country>) void
setD_continentID(int) void
addCountry(Country) void
removeCountry(Country) void
getD_continentValue() int
setD_continentValue(int) void
getD_continentID() int
setD_continentName(String) void
Country
d_continentID int
d_owner Player
d_countryName String
d_countryID int
d_neighbourCountryIDList List<Integer>
d_numArmies int
setD_countryName(String) void
getD_countryID() int
setD_countryID(int) void
setD_neighbourCountryIDList(List<Integer>) void
getD_numArmies() int
setD_continentID(int) void
getD_neighbourCountryIDList() List<Integer>
setD_owner(Player) void
getD_countryName() String
getD_owner() Player
setD_numArmies(int) void
getD_continentID() int
MapUtil
removeNeighbour(Map, int, int) void
loadMap(String) Map
removeContinent(Map, int) void
removeCountry(Map, int) void
isValidMap(Map) Boolean
editMap(String) Map
addCountry(Map, int, int) void
saveMap(Map, String) Boolean
addContinent(Map, int, int) void
showMap(Map) void
addNeighbour(Map, int, int) void
GamePhase
AssignReinforcements
Map_Init
Game_Startup
IssueOrder
ExecuteOrder
values() GamePhase[]
valueOf(String) GamePhase
CommandParser
isGamePhaseMapInit(GameManager) Boolean
displayInstructions(GameManager) void
displayError() void
inputParser(GameManager, String) void
GameEngine
main(String[]) void
Constants
MAP_INIT_HELP String
HELP_MESSAGE String
CMD_ERROR String
IN_GAME_HELP String
GAME_EXIT String
ISSUE_ORDER_HELP String
GAME_STARTUP_HELP String
d_map
d_playerList
d_gamePhase
d_owner
d_orderList
d_continentList
d_country
d_countryList
1
*

Package Structure



```
src
  main
    java
      controller
        CommandParser
        GameManager
        GamePhase
      game
        GameEngine
      global
        Constants
      models
        Continent
        Country
        Map
        Order
        Player
      util
        MapUtil
```

Design Description

We have structured our application into several essential modules to ensure clarity, maintainability, and scalability

- The ***controller*** module serves as the core of our game's logic, handling user inputs to parse and take game decisions that updates the state of the game.

Java Class	Description
CommandParser	This class handles the below commands given by the player. It checks for the given command and action accordingly. ● Add or remove continent ● Add or remove country ● Add or remove neighbor ● Show map ● Save map ● Validate map ● Loadmap ● Adding/removing gameplayer ● Assign countries ● Deploy

GameManager	This class handles the functionality like: ● Assigning countries to the player ● Assigning armies to each player ● Updating the player's turn ● Issuing orders from the player ● Executing the order
GamePhase	This class handles the phase of the game at each stage. The game phases are:- ● Map Initialization ● Game startup ● Assign reinforcements ● Issue Order

- The ***global*** module provides a centralized location for resources, configuration to be accessed by the entire application, promoting code consistency

Java Class	Description
Constants	All the static members are defined here. They are used to display the usage of the commands used for playing the game.

- The ***game*** module serves as the entry point of the game, with a single GameEngine class running an input loop.

Java Class	Description
GameEngine	This class has the main method to drive the logic of the game.

- The ***models*** module defines the artifacts of the game engine to represent the fundamental building blocks of our system. This includes the Player, Order, Country, Continent and Map objects.

Java Class	Description
Player	This class is the representation of the Player entity. It manages the list of countries owned by the player, the orders given by the player. It also handles the functionality of issue order and next order.
Continent	This class is the representation of the continent entity. It contains a list of countries that belong to a particular continent.

Country	This class is the representation of the country entity. A Player object is linked with the Country object to keep track of the countries owned by the player.
Map	This class is the representation of the map entity.
Order	This class handles the deployment of the orders given by a player.

- The ***util*** model holds the static methods required throughout the project. It includes the MapUtil class, which implements various map related functionality like loading, validating and saving the map.

Java Class	Description
MapUtil	This class represents the map utility class. The logic to handle the state of the Map is written here. It has Map functionalities like load map, save map, validate map, show map, add or remove a country, add or remove neighbors.

Game Flow

Map Utility: This part covers the Map related functionalities.

1. Load the map
2. Editing the existing map
3. Save the map
4. Map validation
5. Show map (at any phase of the game)
6. Add or remove continent
7. Add or remove country
8. Add or remove neighbor

Game Play: You can use ‘help’ command to display the commands and its usage for playing the game.

1. Load an existing map to start the game or create a new map.
2. Enter the name of the players.
3. Assign countries to the players.
4. Each player is assigned a number of initial armies.
5. Players issue an order in Round Robin fashion and execute the order.

Tools

- Used IntelliJ for game development.
- JUnit5 is used for unit testing.
- Github repository is used for managing and collaborating the project, as well as for continuous integration. Javadoc implemented for all classes and methods.