SOEN6441 Project Document: Risk-like Strategy Game

1. Introduction

This document outlines the design and implementation of a Java-based strategy game inspired by "Risk." The project consists of two main modules:

- **Map System Module**: Manages continents, countries, and their adjacency, handles map creation, editing, and file I/O.
- **Game Play Module**: Manages the game logic, including the game start-up phase and the main game loop phase.

The application follows a Model-View-Controller (MVC) architecture to separate concerns and ensure modularity, scalability and maintainability.

- **Model**: Represents the data and business logic.
- **View**: Handles the user interface (console-based)
- **Controller**: Manages user input and updates the model and view accordingly.

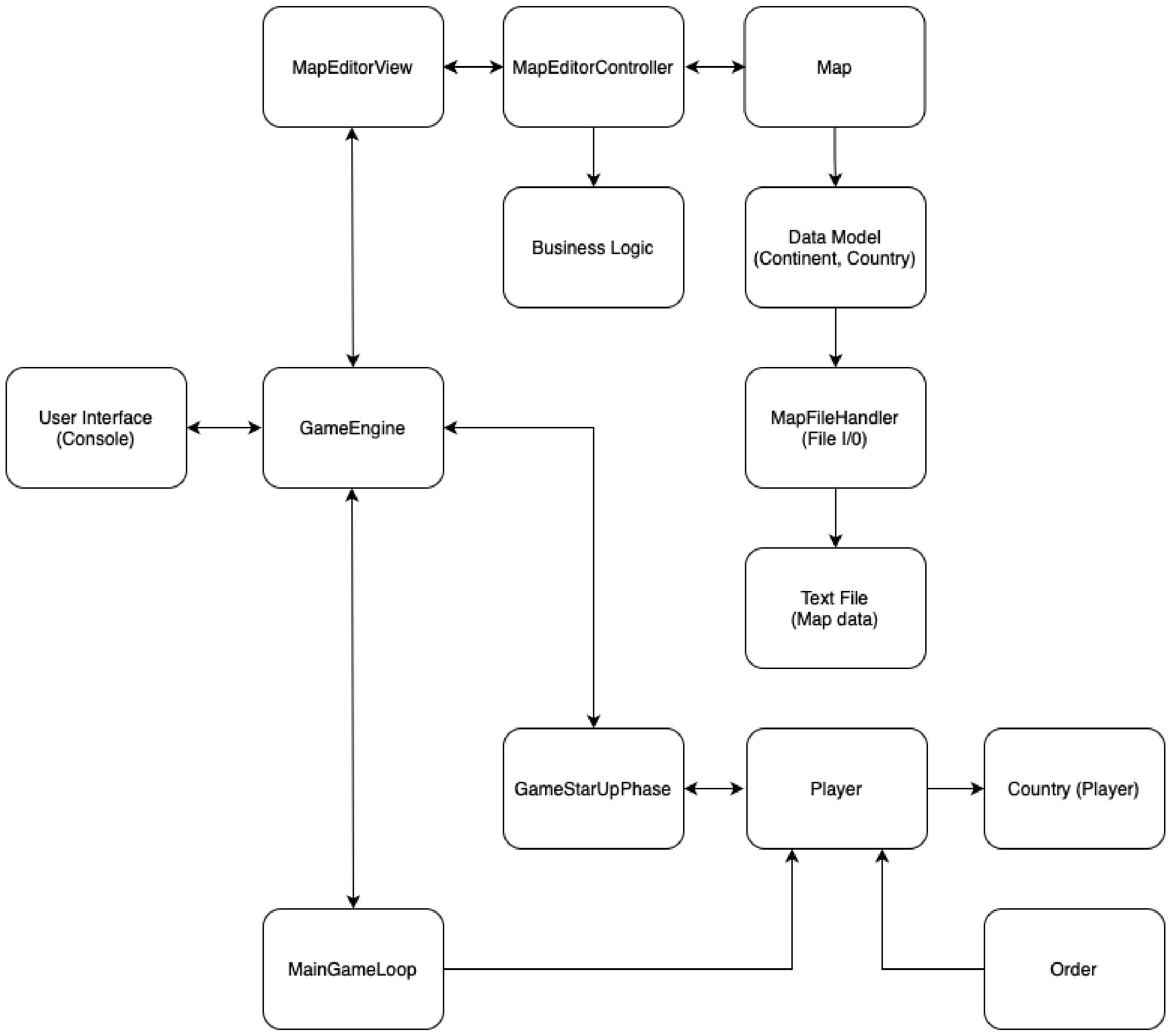
2. Project Structure

```
src/
├──mapeditor/
│   ├── controller/
│       ├── MapEditorController.java
│   ├── model/
│       ├── Map.java
│       ├── Continent.java
│       ├── Country.java
│   ├── view/
│       ├── MapEditor.java
├──gameplay/
│   ├── engine/
│       ├── GameEngine.java
│   ├── phase/
│       ├── StartUpPhase.java
│       ├── MainLoopPhase.java
│   ├── model/
│       ├── Order.java
│       ├── Player.java
│       ├── DeployOrder.java
└── utils/
   ├── MapFileHandler.java
   ├── Helper.java
```

```
    ├── InvaildMapException.java
test/
├──mapeditor/
│   ├── controller/
│         ├── MapEditorControllerTest.java
│   ├── model/
│         ├── MapTest.java
│         ├── ContinentTest.java
│         ├── CountryTest.java
│   ├── view/
│         ├── MapEditorTest.java
├──gameplay/
│   ├── engine/
│         ├── GameEngineTest.java
│   ├── phase/
│         ├── StartUpPhaseTest.java
│         ├── MainLoopPhaseTest.java
│   ├── model/
│         ├── OrderTest.java
│         ├── PlayerTest.java
│         ├── DeployOrderTest.java
└── utils/
    ├── MapFileHandlerTest.java
    ├── HelperTest.java
    ├── InvaildMapExceptionTest.java
lib/
├──junit-platform-console-standalone-1.12.0.jar
docs/
├──javadoc/
├──Risk project document - TYRCL.pdf
map/
├──mymap.txt
├──test_map.txt
```

---

3. Architecture Diagram



---

4. Detailed Design

4.1 Model Layer

Map Class

Manages continents, countries, and adjacency.
Attributes:

- `Map<Integer, Continent> continents`: Key = continentID, Value = Continent object.
- `Map<Integer, Country> countries`: Key = countryID, Value = Country object.

- `Map<Integer, List<Integer>> adjacencyList`: Adjacency relationships between countries.

Methods:

1. **Continent Management**
   - `addContinent(int continentID, String continentName, int armyValue)`
   - `removeContinent(int continentID)`
2. **Country Management**
   - `addCountry(int countryID, String countryName, int continentID)`
   - `removeCountry(int countryID)`
3. **Adjacency Management**
   - `addNeighbor(int countryID, int neighborID)`
   - `removeNeighbor(int countryID, int neighborID)`
4. **Map Operations**
   - `showMap()`: Prints continents, countries, and neighbors.
   - `saveMap(String filename)`: Saves to a text file in the specified format.
   - `editMap(String filename)`: Loads or creates a map file.
5. **Validation**
   - `validateMap() throws InvalidMapException`
     - Checks:
       - All continents and countries are connected.
       - Countries belong to valid continents.
       - No duplicate IDs.

4.2 File I/O Class

MapFileHandler Class

Handle map file operations such as saving and loading.

Methods:

- `saveMap (Map map, String filename)`
- `loadmap (Map map, String filename)`

4.3 GameEngine Class

Controls game flow and phases.

Phases:

1. **Startup Phase**
   - `loadMap(String filename)`: Loads the map using `MapFileHandler`.

    - `addGamePlayer(String name)`, `removeGamePlayer(String name)`: Manages players.
    - `assignCountries()`: Randomly assigns countries to players.
2. **Main Game Loop (**For each player)
    - `assignReinforcements()`: Calculates reinforcements based on owned countries/continents.
    - `issueOrdersPhase()`: Players create orders (e.g., deploy armies).
    - `executeOrdersPhase()`: Processes orders in round-robin fashion.

Methods:

- `showMap()`: Displays country ownership, armies, and connectivity.

4.4 Player Class

Represents a player with owned countries and orders.
Attributes:

- `List<Country> ownedCountries`
- `Queue<Order> orders`
- `int reinforcementPool`

Methods:

- `issueOrder(Order order)`: Adds an order to the queue.
- `Order nextOrder()`: Returns and removes the next order.
- `Commit()`: Commit all orders in the order queue.

4.5 Order Class Hierarchy

- **Order** ()
    - `execute()`:
- **DeployOrder** ()
    - Attributes: `targetCountryID`, `numArmies`
    - `execute()`: Adds armies to the target country.

---

5. Flow of Control with GameEngine

5.1 Initialization

The GameEngine initializes the map, players and phases.

5.2 Game Start-Up Phase:

The GameEngine calls the StartUpPhase to:

- Load the map
- Add players.
- Assign countries to players.

5.3 Main Game Loop Phase:

The GameEngine enters a loop and calls the MainLoopPhase to:

- Assign reinforcements.
- Issue orders.
- Execute orders.

---

6. File Format Specification

Example `map.txt`:

```
[continents]
1 Europe 5     # ID 1
2 Asia 7       # ID 2

[countries]
1 France 1    # Belongs to continent 1 (Europe)
2 China 2     # Belongs to continent 2 (Asia)

[borders]
1 2          # Country 1 borders country 2
2 1          # Country 2 borders country 1
```

---

7. Key Features

- **Map Validation**: Ensures playability via connectivity checks.
- **Round-Robin Order Execution**: First come first serve order processing.
- **Reinforcement Calculation**.

---

8. Testing Plan

1. **Unit Tests**
    - Map validation.
    - File I/O for saving/loading maps.
    - Game engine.
2. **Integration Tests**
    - Full game loop with reinforcement assignment and order execution.