CS319 - Object Oriented Software Engineering

Project Design Report

RISK

Group 2-B

Deniz Çalkan - 21703994

Irmak Akyeli- 21803690

Metehan Gürbüz - 21602687

Murat Sevinç - 21702603

Şebnem Uslu- 21802068

Yusuf Alpdemir - 21803035

1.Introduction

1.1 Purpose of the System

Risk is a multiplayer board game, which requires both luck and skill of the players. The purpose of our system is to create an online version of the Risk board game with brand new features while keeping the core features of the original game. The reason for adding new features is to make the game more strategic and more enjoyable for the players. To make the game more strategic, we want our game to focus more on the skills of the players rather than their luck.

1.2 Design Goals

1.2.1 Criteria

User-Friendliness

One of the first aspects that users notice about a software is GUI. So our Risk game will have a GUI that is aesthetically appealing to users and easy to navigate [1]. We will have a simple yet elegant GUI. For example, there won't be unnecessary buttons or labels on the screen and all buttons will have meaningful and simple titles such as ‘START’ button, ‘QUIT’ button, ‘HELP’ button and so on. Also, ‘HELP’ button will navigate the users throughout the game. The users will be able to press this button at any point of the game to get further information about how to play the game.

Another important aspect of a user-friendly software is whether the software can be easily installed and uninstalled [1]. Since we will be using Java to implement our game, users only need to have JRE installed on their computer to be able to play our game and they can easily remove our game by simply deleting the game file with the extension .jar.

Performance

A game with a good performance should have reasonable frame rate and short loading times. Our goal is to keep our game running with 60 FPS and keep our response time less than a second.

Portability

Our game will be portable to any operating system that supports JVM, since we chose to implement our game in Java and Java provides cross-platform desktop applications [2].

Reusability

Some of the classes in our game such as Dice and MainMenu classes can be used in other games without needing to completely change their implementation. These classes have core functionalities and can be easily used in other Java based software.

1.2.2 Trade-Offs

Memory vs Performance

We chose to implement our game in Java so we are not dealing with manual memory management because Java has an automatic garbage collector [3]. However, Java is relatively slow compared to other programming languages because Java code is compiled to an intermediary which is Java bytecode before it is converted to machine code [3].

Cost vs Portability

Since we don't have a budget, our game will be only available for desktop operating systems that support JVM. Moreover, our game won't be available for mobile operating systems such as IOS and Android.

Development Time vs Functionality

Since we have very limited time to implement our game, we aim our game to be able to perform core features of the original Risk game and some of our new features without making the game too complex [4].

1.3 Definitions, Acronyms and Abbreviations

Graphical User Interface (GUI): Computer program that interacts with the user through visual components [5].

Java Runtime Environment (JRE): Software layer that provides resources in order for Java programs to run. Also provides JVM [6].

Frame Per Second (FPS): Frequency of displayed images on the screen [7].

Java Virtual Machine (JVM): Virtual machine which the computer needs in order to run programs that are written in programming languages that are compiled to Java bytecode [8].

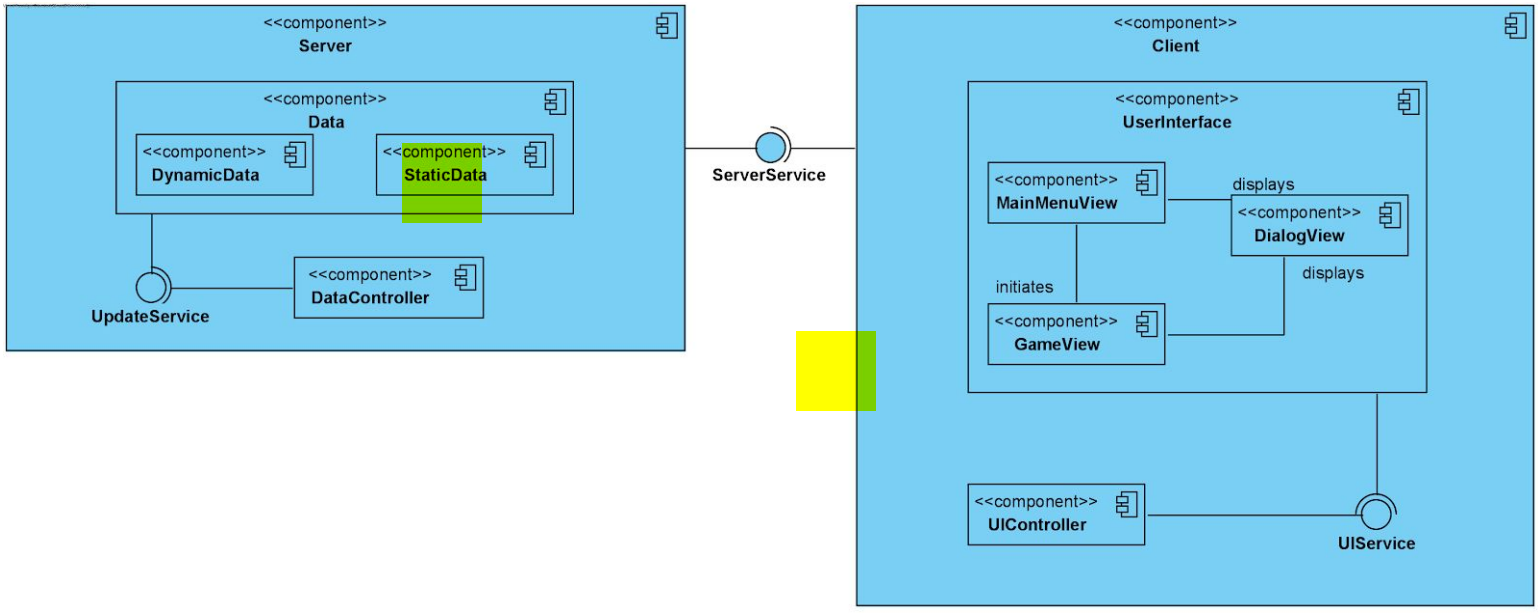
2. Software Architecture

2.1 System Decomposition

Our purpose in this section is decomposing our system into subsystems to reach the design goals of the project. During this process our purpose is to make the design of the project ideal which is with high coherence and low coupling.

Risk is an online game which includes many interactions with users. To provide **high performance** and to serve **many clients**, we have decided to choose **Client/Server architectural style**.

Risk is divided into Client and Server components. The Client component will consist of UsesInterface and UIController components. On the server side, Server component will consist of Data and DataController components.



2.2 Hardware/Software Environment and Dependencies

Our risk game will be implemented in Java Programming languages. The app will be using contemporary java libraries such as javafx. Therefore, a user will need to have JRE.

For hardware, a user will need a monitor to interact with visuals, a pc to install and enter the game, a mouse and a keyboard for input operations and also an internet connection to play and interact with other players.

The game will run in any environment (Linux, Mac, Windows) that has JRE.

2.3 Persistent Data Management

We store some necessary initialization and configuration data for our game engine in the form of JSON files. We use those configuration files to initialize our game engine on startup. However, we do not persist in any intermediate state of games. Therefore once the server is down any states corresponding to the running game are lost. Our game will not save any data for later circumstances. The images will be in .png format to enable transparency and sound files will be stored in .wav format.

2.4 Access Control and Security

Clients will constantly be interacting with our server. They will only be able to access our encapsulated public methods therefore our sensitive data (such as structure of objects, important values) will be protected against external subjects. There will not be any authentication for users. When the app has opened, only a nickname is required. Hence, there will be no private or valuable data of users. So, security will not be our primary concern.

2.5 Boundary Conditions

2.5.1 Initialization

The moment a user opens the game, there will be a dialog asking for a nickname. After the input is provided, the app will wait until it creates a successful communication with the server, there will be a pop-up screen if the waiting time is more than 30 seconds; so, in the beginning stage a user must have an internet connection. Besides, in order to start the game, the client must have the image of the map, needed sound files, and default JSON file for the data of the map, menu, etc. All other communicative data will be provided by the server.

2.5.2 Termination

Users will be able to close the game by using the exit button on the corner. In the main menu there will also be another button to exit.

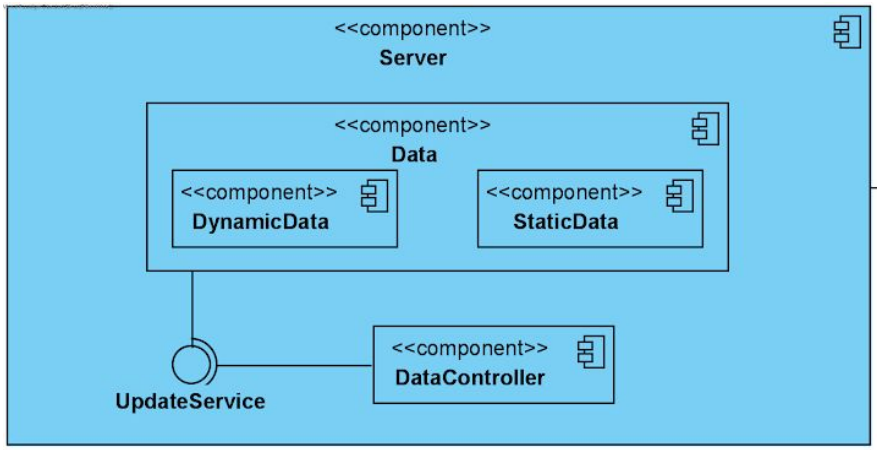
2.5.3 Failure

If an error occurs on the server side (an unexpected response or excessive response time, runtime error), the client side will close the current game and the user will be directed to the main menu.

If an error occurs on the client side, (unable to request, crash, connection loss); server side will not recieve and request and thus the current game will not be playable and the server will close the game, every client will be directed to the main menu.

3. Subsystem Services

3.1 Server Subsystem

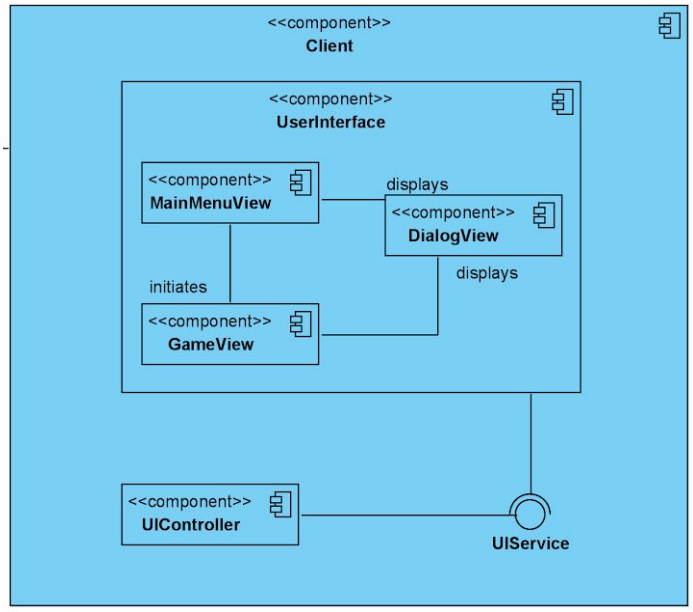


Server component consists of DataController component and Data component. ServerService enables the server to get requests from clients and respond back to clients after necessary processes.

Data component contains any data related to the game. StaticData component basically holds data that does not change in a game, such as map configurations, player nicknames, and any related configuration data for the game. Whereas DynamicData component holds the data that constantly changes such as troop count on a region, room ID number etc.

DataController is a component to communicate with necessary data. By this component, we are able to **boost coherence** and **decrease coupling.**

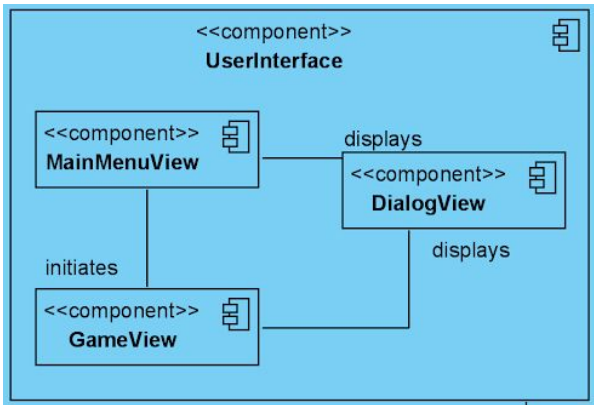
3.2 Client Subsystem



Client component consists of UIController and UserInterface components. UIController is able to communicate with both the user interface and the server. This again helps our system to have high coherence and low coupling. UIController basically adjusts the user interface accordingly to the responses from the server.

3.2.1 UserInterface Subsystem

UserInterface will have different types of view components: MainMenuView, DialogView, and GameView.



- **MainMenuView:** This component will be responsible to display the main menu of the game.

- **DialogView:** In the game, there will be many cases where the user sees a dialog. This view component basically displays different types of dialogs such as nickname dialog (where we get a user's nickname), how to play dialog, and various warnings or errors like connection loss.

- **GameView:** This component is to provide the game screen to the interface. Main map of the game, regions, troop counts on regions, and animations are dealt by this component.

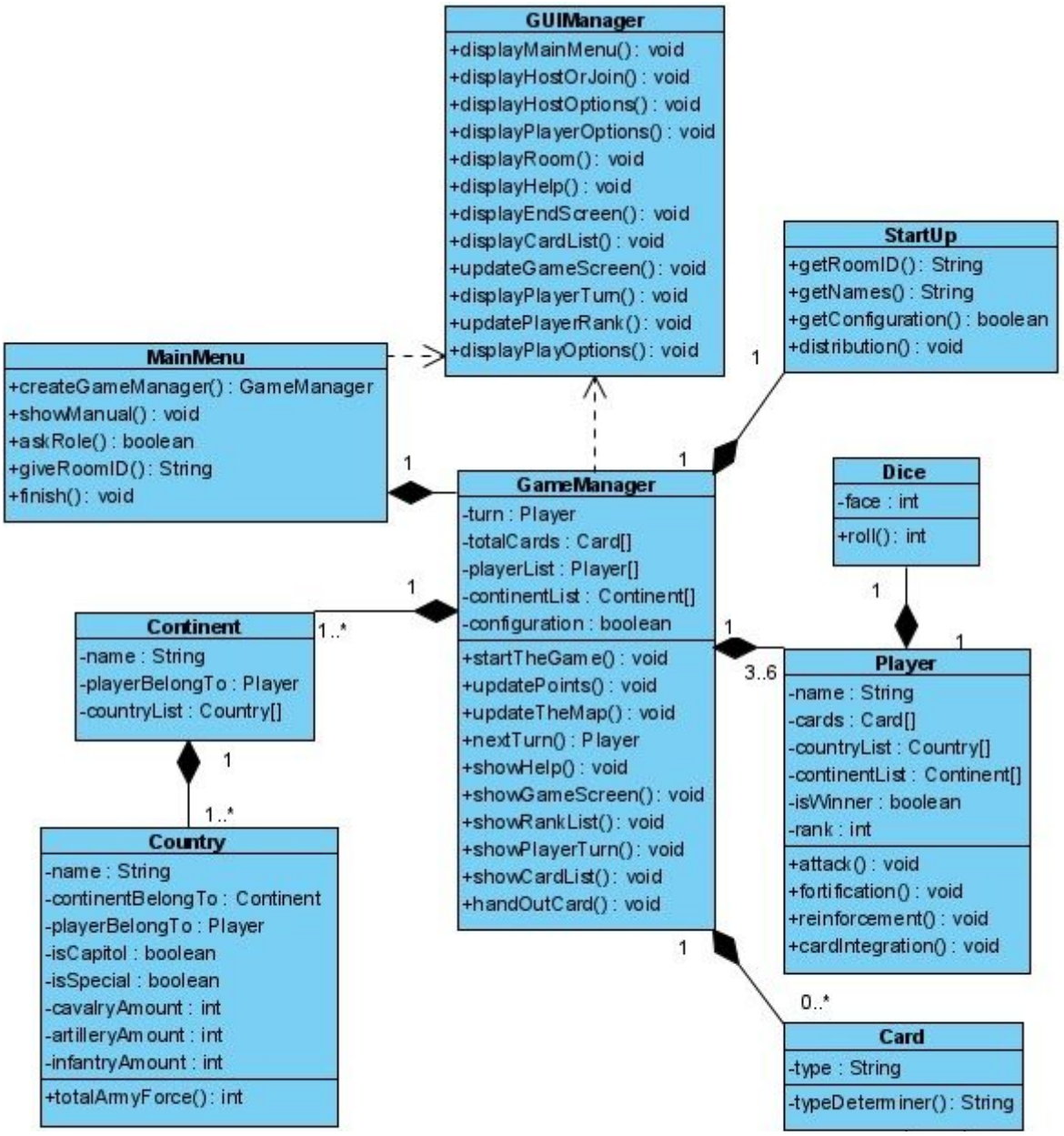
4. Low Level Design

4.1 Object Design Trade-Offs

Performance vs Security

Since risk is mostly an interactive game, our efforts are focused on the performance side. Also since in the game, there will be no sensitive data, security is not in the scope of our purpose.

4.2 Final Object Design

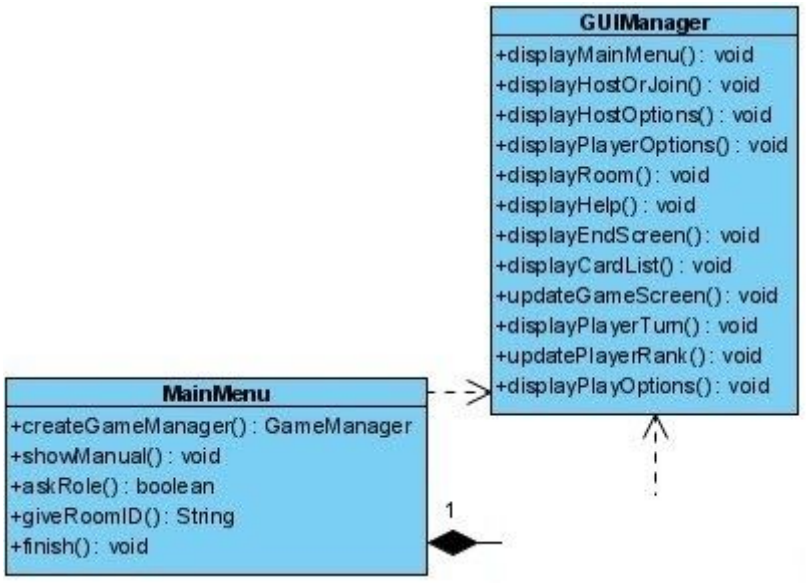


4.3 Layers

Our game has two main layers as discussed previously. The first layer is the Client layer that is visible by the user and the second is Server layer. The two layers always interact with each other and work together to give a good performance game. The client layer consists of user interfaces and GUI's where the server layer consists of everything else.

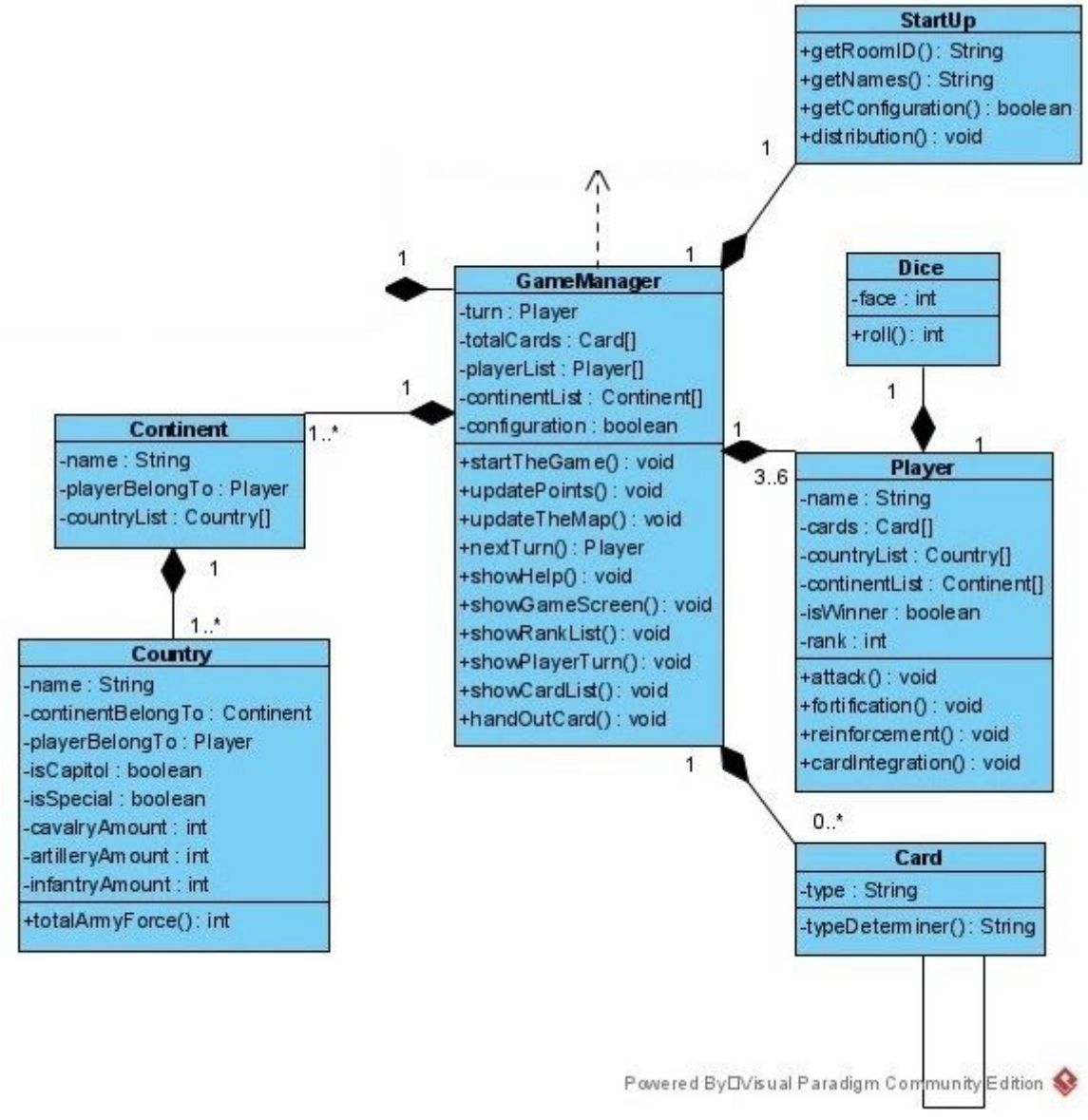
4.3.1 Client Layer

In this layer we only transform the data obtained from the server to GUI and obtain the new information from the client. As our game is mostly operated on a server, this layer's main objective is to create a communication between the user and the server layer. Therefore it is simplified to have only two classes and does not implement a design pattern.



4.3.2 Server Layer

This layer is the most complicated layer and therefore a facade design is implemented. We can see that almost all the classes are accessed through GameManager and with this implementation we prevent further coupling and make it more controlled. However, this also reduces performance as we need to go through multiple classes in order to access some classes.



4.4 Packages

4.4.1 Internal Packages

- **Server Package**

  a) ServerController Package:
  This package consists of classes that are responsible for the management of the server.

  b) Data Package:
  Data package is responsible for static and dynamic data updates during the game.

- **Client Package**

  a) ClientController Package:
  ClientController manages client based processes

  b) View Package:
  View package deals with UI updates, processes. It is responsible for UI management.

4.4.2 External Packages

We use JavaFx in our project to conduct game related UI's and many other operations. As JavaFx is not included on JDE we needed to download it and included it in our game. With javafx.scene we do layouts and use javafx.event to handle objects and interactions. With many other utilities in the javaFx package it is an essential to our project.

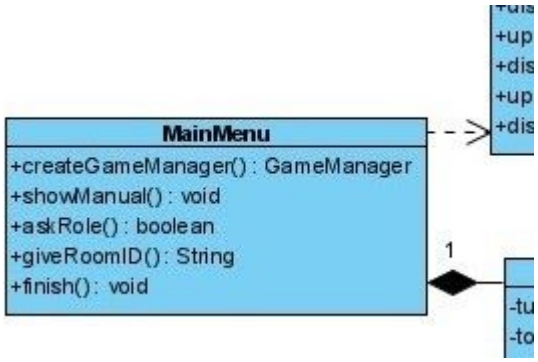
4.5 Class Interfaces

4.5.1 Client Layer Class Interfaces

MainMenu Interfaces

MainMenu class handles the beginning of the game. It takes the input from the user for the options: Manuel, Play, and Exit. In order to offer the user this selection in a proper

interface, it calls GUIManager. If the user selects to play, it calls GameManager.



Operations:

· **public GameManager createGameManager():** It creates a GameManager object to start the game.

· **public void showManual():** It calls displayHelp() from GUIManager to display the Help screen.

· **public boolean askRole():** It calls displayHostOrJoin() from GUIManager and takes the submitted choice of the user and returns it.

· **public String giveRoomID():** It attains an unique Room ID and returns it.

· **public void finish():** It stops the execution.

GUIManager Interfaces

GUIManager class has been established due to the necessity that emerged for the interaction between the players. GUIManager provides the user interface by calling the relevant method. It is called by MainMenu and GameManager.

Operations:

- **public void displayMainMenu():** It displays the user interface of Main Menu.
- **public void displayHostOrJoin():** It displays the user interface of option screen between the host or a participant.
- **public void displayHostOptions():** It displays the user interface of Host screen where the host selects the game mode and enters the player name.
- **public void displayPlayerOptions():** It displays the user interface of Joiner screen where the participant enters the player name and the game code.
- **public void displayRoom():** It displays the user interface of the Room screen where the players wait for others and the host can add more people.
- **public void displayHelp():** It displays the user interface of Help screen.
- **public void displayEndScreen():** It displays the user interface of Finish Screen where the winner is shown.
- **public void displayCardList():** It displays the user interface of the Card List screen where the collected card of the player is seen above the integration request button.
- **public void updateGameScreen():** It updates the user interface of Game Screen.
- **public void displayPlayerTurn():** It displays the user interface of Whose Turn screen.
- **public void updatePlayerRank():** It updates the user interface of the player's ranking and displays it.

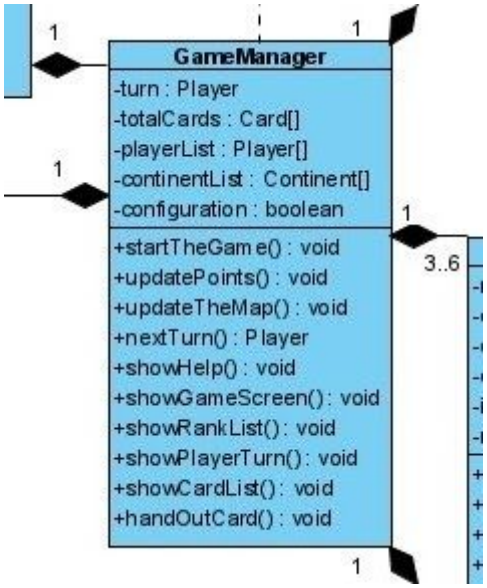
· **public void displayPlayOptions(int option):** It displays the user interface of Play Options bar where the player attacks, fortificates, and/or reinforces according to the parameter.

4.5.2 Server Layer Class Interfaces

GameManager Interfaces

GUIManager class has been established due to the necessity that emerged for the interaction between the players. GUIManager provides the user interface by calling the relevant method.

It is called by MainMenu and GameManager.



Attributes:

· **private Player[] playerList:** Holds the list of Players on the game.

· **private Card[] totalCards:** Holds the card deck.

· **private boolean configuration:** Holds the game mode.

- **private Continent[] continentList:** Holds the list of continents in a way of a map.

- **private Player turn:** Holds the Player that is currently playing.

Operations:

- **public void startTheGame():** It initiates the game by initializing attributes.

- **public void updatePoints():** It updates the points of each Player.

- **public void updateTheMap():** It updates the map according to the latest movements.

- **public Player nextTurn():** It passes to the next Player, calls displayPlayerTurn() from GUIManager, and returns to the Player who will play in the next round.

- **public void showHelp():** It calls displayHelp() from GUIManager.

- **public void showGameScreen():** It calls updateTheMap() and updatePoints() within itself and calls displayGameScreen() from GUIManager.

- **public void showRankList():** It calls updatePlayerRank() from GUIManager to show the ranking bar.

- **public void showPlayerTurn():** It calls displayPlayerTurn() from GUIManager to show Whose Turn screen.

- **public void showCardList():** It calls displayCardList() from GUIManager to show Card List window.

- **public void handOutCard(Player aPlayer):** It attains cards to the earning player.

StartUp Interfaces

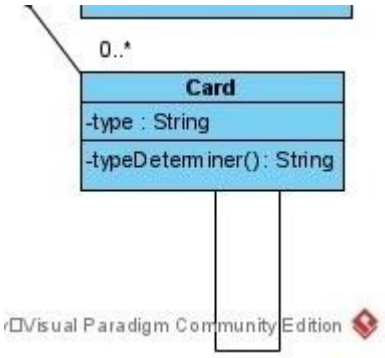
StartUp class was designed for the beginning functions of the game. If the player selects the “Play” option, GameManager will call StartUp to determine player names, configuration status and the initialization of the game.

Operations:

- **public void distribution(Country toCountry):** It initially distributes army forces to the country which is given.
- **public String getNames():** It gets the player name from the user and returns it.
- **public boolean getConfiguration():** It gets the game mode from the host and returns it.
- **public String getRoomID():** It gets the room ID from the participant and returns it.

Card Interfaces

Card class represents a Card that will be drawn within the game. Cards can provide cavalry, infantry... etc. Thus, a card should have a property that indicates the type. The type of the card should be random so that the player cannot count the cards to determine what will come. typeDeterminer() provides a random card to be created. Because this method is only used when the card is created which is in the constructor, the method will stay private.



Attributes:

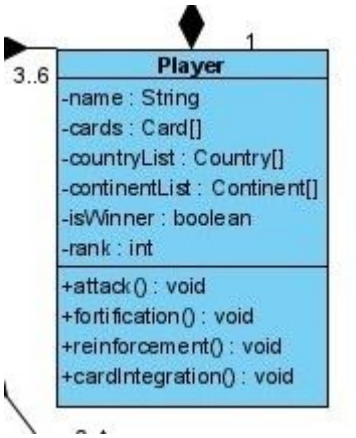
· **private String type:** Holds the type of the Card.

Operations:

· **private String typeDeterminer():** It randomly determines the type of the card within the constructor.

Player Interfaces

Player class will be created when the Game Manager calls it. Thus, the existence of the Player is dependent on the GameManager. If there is no game to play, there will be no player. The player has multiple properties as well as methods. Player will have the attributives of a name, a CardController that will handle the a list of cards, a country list to track down the conquered countries, the continent list so that at the next round, the player can receive a bonus for conquering an entire continent, a boolean to determine that whether the exact player is the winner one or not. Lastly, the rank of the player.



Attributes:

- **private String name:** Holds the information of name
- **private Card[] cards:** Holds a list of Cards that Player possesses
- **private Country[] countryList:** Holds a list of Countries that Player invaded
- **private Continent[] continentList:** Holds a list of Continents that Player invaded
- **private boolean isWinner:** Holds whether the information of the Player is the winner
- **private int rank:** Holds the rank of the Player

Operations:

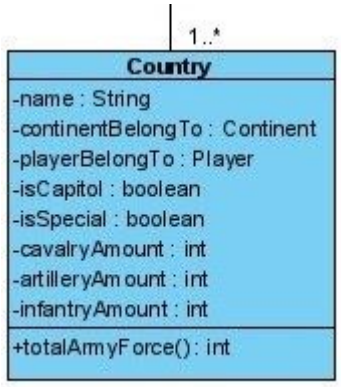
- **public void attack(Country attackerCountry, Country attackedCountry):** It performs the attacking action with the given parameters.
- **public void fortification(Country selectedCountry, Country soldiersAdded):** It performs fortification with the given parameters**.**

- **public void reinforcement(Country fromCountry, Country toCountry):** It performs reinforcement with the given parameters.

- **public void cardIntegration():** It integrates the card and updates the card list accordingly.

Country Interfaces

The creation of the Country class is dependent on the existence of the Continent class. If only if the Continent class is ever created, a list of Country objects will be created. Country class has 8 attributes: the name of the country, the owner of the country, the continent that the country is within, is it a capital or is it a special city like Rome, and lastly, the numbers of three different kinds of soldiers deployed in the country. There is only one method that this class contains which calculates the total army force according to the amount of the different types of soldiers.



Attributes:

- **private String name:** Holds the name of the Continent.

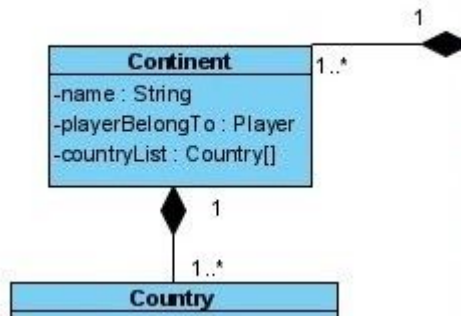
- **private Continent continentBelongTo:** Holds the Continent that Continent in.
- **private Player playerBelongTo:** Holds the Player that invaded the Country.
- **private boolean isCapitol:** Holds the information of whether the Country is a capitol.
- **private boolean isSpecial:** Holds the information of whether the Country is a special.
- **private int cavalryAmount:** Holds the number of cavalries deployed in the Country.
- **private int artilleryAmount:** Holds the number of artilleries deployed in the Country.
- **private int infantryAmount:** Holds the number of infantries deployed in the Country.

Operations:

- **public int totalArmyForce():** It calculates the total army force in the Country.

Continent Interfaces

The creation of the Continent class is dependent on the existence of the GameManager class. If only if GameManager class is ever created, a list of Continent objects will be created. Continent class has 3 attributes: the name of the continent, the owner of the continent, and the list of countries that the continent includes within. This class works as the board of the game which is the map.



Attributes:

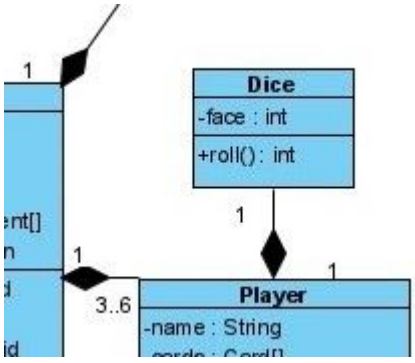
· **private String name:** Holds the name of the Continent.

· **private Player playerBelongTo:** Holds the invader Player of the Continent.

· **private Country[] countryList:** Holds a list of Countries that the Continent encapsulates.

Dice Interfaces

The existence of the Dice class is dependent on the Player class, since if there is no player to roll the dice, no need for the dice. Dice class takes two army and an advantage indicator and returns the army lost from a side after roll function.



Attributes:

· **private int face:** Holds the value of the dice. Which is 100 in our game

Operations:

· **public roll(attackingArmy, attackedArmy, advantage)** Takes the number of each side of the army and the indicator advantage showing if one side has advantage or not and returns the army lost after a single attack.

References

[1] Wallen, J., 2010. *10 Things That Make Software User-Friendly*. [online] TechRepublic. Available at: <https://www.techrepublic.com/blog/10-things/10-things-that-make-software-user-friendly/> [Accessed 26 November 2020].

[2] En.wikipedia.org. 2020. *Cross-Platform Software*. [online] Available at: <https://en.wikipedia.org/wiki/Cross-platform_software> [Accessed 26 November 2020].

[3]McKenzie, C., 2019. *Is Java Slow? Compared To C++, It's Faster Than You Think*. [online] TheServerSide.com. Available at: <https://www.theserverside.com/opinion/Is-Java-slow-Compared-to-C-its-faster-than-you-think#:~:text=In%20fact%2C%20when%20compared%20against,features%20of%20its%20underlying%20architecture.> [Accessed 28 November 2020].

[4] Kanat-Alexander, M., 2008. *The Goals Of Software Design » Code Simplicity*. [online] Code Simplicity. Available at: <https://www.codesimplicity.com/post/the-goals-of-software-design/#:~:text=From%20the%20purpose%20of%20software,of%20software%20design%20should%20be%3A&text=To%20design%20systems%20that%20can,be%E2%80%93as%20helpful%20as%20possible.> [Accessed 26 November 2020].

[5] Levy, S., 2018. *Graphical User Interface | Computing*. [online] Encyclopedia Britannica. Available at: <https://www.britannica.com/technology/graphical-user-interface> [Accessed 26 November 2020].

[6] Ibm.com. 2020. *What Is The JRE (Java Runtime Environment)?*. [online] Available at: <https://www.ibm.com/cloud/learn/jre> [Accessed 29 November 2020].

[7] En.wikipedia.org. 2020. *Frame Rate*. [online] Available at: <https://en.wikipedia.org/wiki/Frame_rate#:~:text=Frame%20rate%20(expressed%20in%20frames,and%20be%20expressed%20in%20hertz.> [Accessed 26 November 2020].

[8]En.wikipedia.org. 2020. *Java Virtual Machine*. [online] Available at: <https://en.wikipedia.org/wiki/Java_virtual_machine> [Accessed 27 November 2020].