Bilkent University

Department of Computer Engineering

CS 319

Object Oriented Software Engineering

Game of "Risk" Design Report

Group 1-E

Mehmet Ege Acıcan-21602186

Ali Ata Kılıçlı-21301530

Ekin Üstündağ-21602770

Mehmet Akif Kılıç-21502487

Seman Arda Yuvarlak-21501402

Contents

1. Introduction

1.1 Purpose of the System

Game of "Risk" is a strategic conquest and conquer game[1]. It' s main purpose is to make the players be in a competitive and entertaining strategic game with each other on the computer environment and therefore it is designed this way[2]. The game of "Risk" is based on the classic board game, "Risk" and its gameplay is almost identical to the classical board game version

however we added 2 different gameplay mods to the game in order to fix the time problem that the original game has. The Game of "Risk" is a game with 3 gameplay mods which are: Classical, Time Mod and Golden Territories. Classical Mod is the classic implementation of the board game while time mod adds a time limit to the classic game, declaring the player who has the most territories at the end of the clock as winner. Golden Territory mod on the other hand declares the player who occupies the adjusted "Golden Territory" as the winner. We increased the entertainment aspect by eliminating the time factor that specifically made the game "boring". Our design and our architecture will be based on subsystems that satisfies the functional requirements in our analysis report.

1.2 Design Goals

We will firstly establish our criterias and trade-offs of our design to simplify the implementation process. Therefore we are going to design the game before implementing. The main criterias and trade offs are derived from our Non functional requirements. The main criterias and trade offs of our system are:

1.2.1 Maintenance Criteria

Ambiguity will be designed to be a single player game. Our main goals in the design process

Adaptability:
Java is a portable OOP language and in CS 101 and CS 102 we have learned Java, due to that reason we chose Java language. Java executed on any platform with JVM. Moreover, it has a class library, garbage collector and simple syntax. Just in time compilers and advanced memory management makes performance better.

Maintainability: The game design is based on object oriented programming and thus game will be consisted of systems and subsystems, which are all going to be connected with each other in hierarchical structure.

Reliability: The game should contain as less bugs and errors as possible so that players are able to play the game without any

disruption. In order to achieve this goal, we are going to look at the interactions between the User and the Interface and decrease the error potential as much as possible. Therefore, the User Interface will handle exceptions without alerting the User. System will be considered about all boundary conditions to avoid the crashes in the system due to the missing some cases. Therefore, for preventing this case we will be testing all situations carefully with each of the development stage at the same time. As a result, system will be bug-free and coherent in the boundary conditions.

Modifiability: Our system will follow the same principles as any other software systems follow: Open to changes and updates. Therefore, Our game is going to allow modification without showing any problems. Also, classes are going to be, even though they are connected, as independent as possible so that whenever a change occurs in one, the others will be affected by it as minimum as possible.

Extendibility: The game design can be modified and can be changed about its functionalities and features in future designs due to the users' feedback or other reasons. For instance, our game can be extended by adding new troop types to the game or adding a new modes.

Reusability: Our system is going to reuse specific components in order to prevent including another subsystem with the same features, creating repetition problem.

1.2.2 User Criteria

User Friendliness: Risk game will be designed to entertain the users as it will provide them a user friendly interface. Therefore, it makes the game easier to play for the players. For instance, there will be a menu that will be understandable for the users and the user-friendly interface of the game will help the players to play the game to avoid misunderstandings about the game. The system will get the mouse inputs from the users, which is the easy use for the players.

Performance: For a better gameplay, the game performance is an important aspect. Therefore, our game will try to be as fast as possible with main purpose to not make the player wait for

more than 0.1 seconds in any of the operations that is in the game.

1.2.3 Trade offs

In order to allow these criterias there are specific trade-offs that should be considered:

Functionality vs User Friendliness: In order to make the game user friendly, the game will be designed in a way to make it as simple as possible. This decreases the complexity of the game and thus decreases the possibility of specific functionalities that can be added.

Efficiency vs Portability: The Game of "Risk" is going to be designed only for the Laptops and Desktops in order to provide better efficiency. Therefore, the portability factor is limited.

Cost vs Simplicity: Since the game is going to be implemented as simple as possible, The cost of time by the developers decreases. Therefore, because of not trying to add any complexity to the game, the time cost is going to be minimized.

Development Time vs User Experience:To implement the user interface of the game Ambiguity, we have chosen to use JAVAFX instead of JAVA. The reason why we chose JAVAFX libraries is because JAVA's standard SWING library uses 8 bits for resolution. On the contrary, JAVAFX uses 32 bits for resolution. Even though JAVAFX is harder to implement and develop the game with it, because SWING is commonly used and known library and more resources are provided. On the other hand, JAVAFX provides a better graphics and user experience.

1.3 Definitions

MVC(Model View Controller): The separation of model, view and controller. The main design pattern of the Game of "Risk". Model, View and Controllers are explicity defined.

GUI(Graphical User Interface): Interface that allows the players to interact with the software. Game of "Risk" uses GUI.

JavaFX: A software platform for delivering desktop applications. JavaFX allows us to make fxml files to manage view.

Boundary conditions: Conditions of the system which may causes runtime errors. They are exceptional cases according to the normal flow of the program. These conditions must be handled carefully for robustness of the system

Java Virtual Machine (JVM): an abstract computing machine for running a Java program.

1.4 Overview

In this section, we presented purpose of the system, which is essentially entertaining the user as much as possible, to manage this purpose we defined our design goals in this part. Our design goals are specified with regards to provide the adaptability, performance, usability, extensibility, maintainability and reliability. In this respect of course, we made some trade-offs to perceive our goals. We give up from functionality to make our system simpler, understandable and for better performance

2. High Level Software Architecture

2.1 Subsystem Decomposition

This section divides the main system into specific subsystems and analyzes them separately. We plan to do this in order to possibly increase the freedom, modifiability and flexibility of our system in the future. Our system uses a closed architecture model which means that the layer can only access the layer that is immediately below itself. Also, our system uses 3 layer architecture in order to maximize maintainability and allow low coupling between layers, thus keeping the effect of change minimum whenever a layer needs a change. Furthermore, our design uses three layer style architecture as presentation, application and data, to closely follow MVC pattern and increase flexibility and maintainability of the code.

Our design is composed of GUI layer, Gameplay Layer and Data Layer.

GUI layer handles all of the GUI tasks such as displaying the game, menus and getting mouse inputs from the user. In other

words, GUI layer is responsible for the screens and screen interactions that the player faces. GUI layer is composed of Game GUI and Menu GUI. While Menu GUI handles the menus that is offered to the user, Game GUI handles the interactions between the User Interface and the Gameplay. In other words, Game GUI is responsible in game screen while Menu Manager is responsible for Main menu, pre-game-settings, credits, display help and adjust settings menu.

Game Logic Layer consists of the gameplay aspect and it is composed of Menu Manager and Game manager. Menu Manager allows the interactions between the player and the menu screens, and thus responsible for menu controllers such as CreditsMenuController, PreGameSettingsController. In other words, Menu Manager subsystem is the collection of menu controllers. Also, Menu Manager is called by the GUI layer, and controls the input from the GUI layer. Game Manager allows the main gameplay and is responsible for the interactions between the in game decisions and player. Game manager is responsible for the controllers such as InGameController and InGameTamrielController. There is also a Game Object Manager which is responsible for managing the game elemets such as Players, Territories, Troops, Troop types. Game Instance Manager is used by Game Manager and communicates with the Data Layer. Game Object Manager is responsible for the model classes of our codes

Data Layer is responsible for handling the data kept in the system. These data include basically the game data that is being saved and loaded. This layer allows the games to be saved and then later loaded and continued from where it was left. Data Layer is the last and bottom layer in our 3 layer architecture, it does not depend on any other layer.

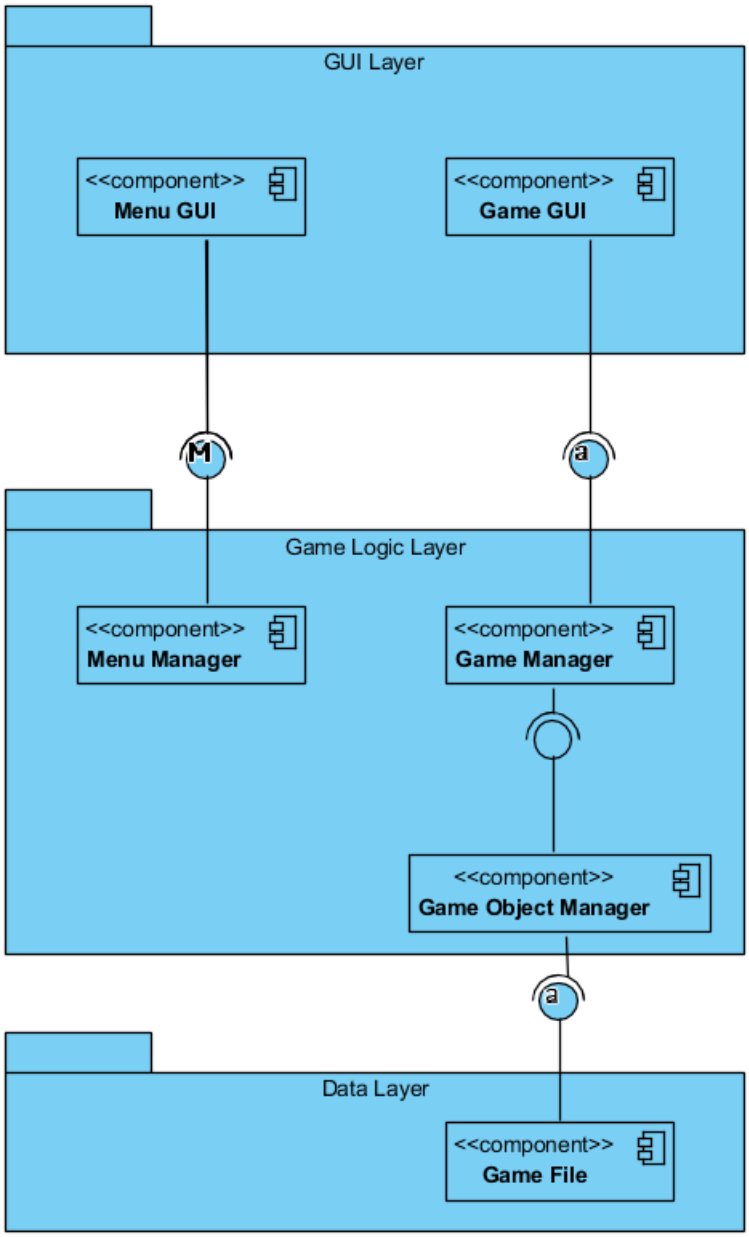


Figure 1: High level software architecture showing GUI Layer, Game Logic Layer and Data Layers

2.2 Hardware/Software Mapping

Game of "Risk" is implemented in Java programming language and it specifically uses Java 8, and we will use JDK 1.8+. Intellij IDE is used in as a platform for programming. Along with this, Game of "Risk" uses Javafx libraries to handle the GUI. Also, the game uses JRM so it will be playable on multiple operating systems such as Linux, Windows and Mac. A mouse and a keyboard is sufficient for interacting the game since the mouse is going to handle all of the GUI inputs for Game and menus while keyboard is going to handle the name giving to the player game and storing those name to the data. Specifically, GUI subsystems requires mouse to get input for Game and Menu GUI. Keyboard is

required for the Menu GUI, in the case of player name giving. File subsystem is working in order to handle the saving and loading the games.

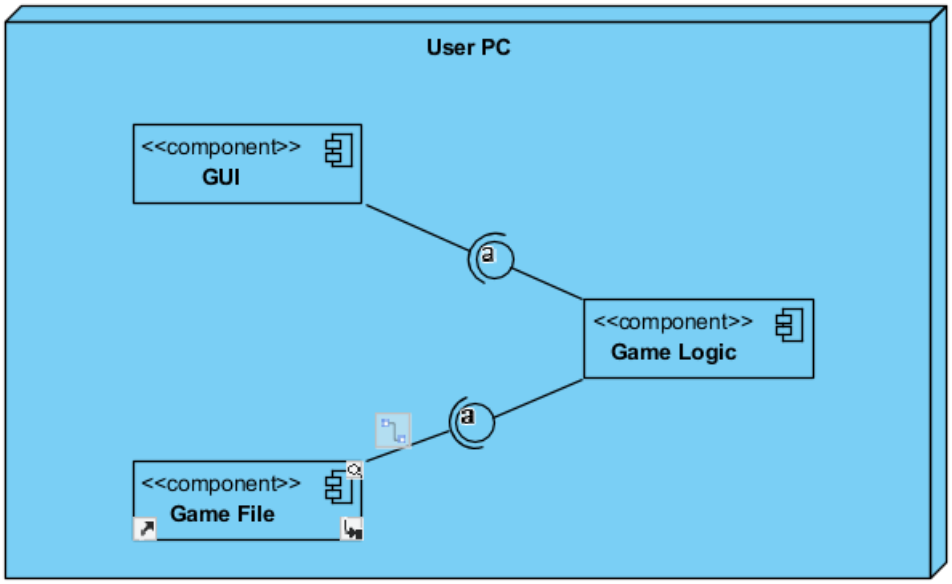


Figure 2: Deployment Diagram showing the Hardware mapping of the software

2.3 Persistent Data Management

In Game of "Risk", we save user data in a text file with JSON format in local storage. We use gson library of java. The save will includes the data of the game such as player names and the owned territories along with the players colors

Our game uses file system for data management since our game saves only player names and current situation of the map and thus creating a low intensity.

2.4 Access Control and Security

Since our game does only hold information about the game itself such as player names and game territory conditions, Our game does not prioritize security. We do not require any personal information that might be valuable to the player and thus did not involved in security measures.

2.5 Boundary Conditions

Error Behaviour and Exception Handling: If there occurs a run time error such as the game not being able to load the data of the previously saved games, We will display a run time error

message. The game will try to handle exceptions without alerting the user.

Starting the Game: As the user starts the game, main menu will be shown without any requirements. After clicking the play button, after the player chooses a new game, adjusts gameplay mod and the pre-game-settings, the system will create a new slot for the players new game, save it and then initialize the game. If the player chooses load game option, if no problem occurs, the system will initialize the loaded game that the player selects from the load games. If one of the loaded games is deleted from the slots, then the slot will be free and a new game is going to be free to start. Otherwise, the slots will show which saved game that they contain along with the last time that they had been played. If one of the loaded files is corrupted, the game will show an error message.

Termination of the Game: The game has two termination options which are terminating from the main menu and termination while playing the game.

The Exit button is presented to the player in the main menu, The player can exit the game in the main menu by using this button.

While playing the game, the player can exit the game by clicking Exit without saving button which is presented under the game map.

The player can save the game by using Save Game button while playing the game. The player can only use this button only to save the game. The player can not terminate the game and exit the game with the same function.

3. Subsystem Services

3.1 GUI Layer

This layer contain two subsystems as Menu GUI and Game GUI.

Game GUI : The GUI subsystem that is consisting of the main game screen GUI part. These game screens include: InGame game view which is a fxml file and InGameTamriel which is also an fxml file. Both these files are containing game screen

along with the troop choice buttons(buttons that allows the player to recruit a specific type of troop) and decision buttons(buttons that allow the player to change between attack, recruit and deplace stages). The views also contains menu buttons(buttons that allow the player to save the game, adjust game settings, return to main menu and exit without saving) and game map. The only difference between these two fxml files is that the maps are different. While one of the maps is Political World map, the other is political Tamriel Map.

Menu GUI: Menu part of the GUI Layer consists of all the menus that the user encounters. These include: The main menu, new-game-settings menu, pre-game-settings menu, play-menu, general settings menu. Main menu is the opening menu that allows users to choose to play the game, settings and directly exit the game. Play menu allows the user to load a game or start a new game. New-game-settings menu allows the user to choose the gameplay mod, pre-game-settings menu allows the user to adjust the pre-game conditions, general settings menu allows the user to adjust the sound and brightness of the game. The respective buttons and sliders that are used in the specific screens are also included.

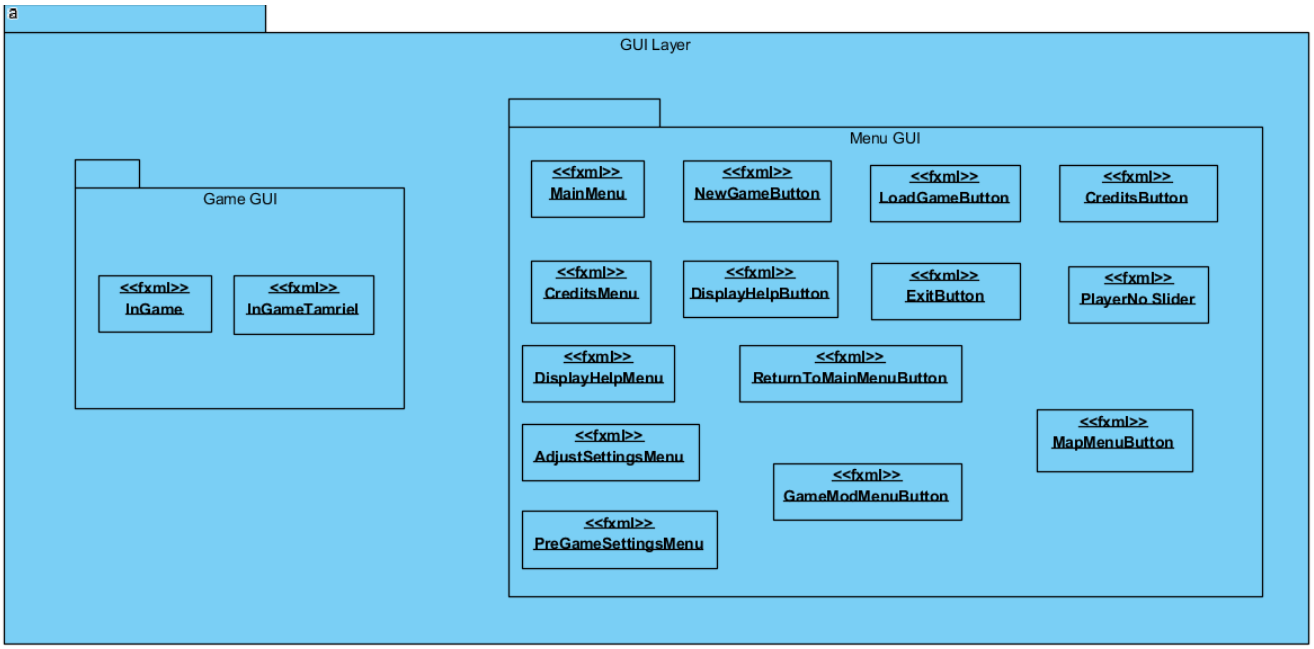


Figure 3: GUI Layer with the respective subsystems

Game GUI

InGame: This fxml file defines the game screen.

InGameTamriel: This fxml file defines the game screen with Tamriel Map.

Menu GUI:

MainMenu: This fxml file defines the Main menu screen.

CreditsMenu: This fxml file defines the Credits menu screen.

DisplayHelpMenu: This fxml file defines the Display Help Menu screen.

AdjustSettingsMenu: This fxml file defines the Adjust Settings Menu screen.

PreGameSettingsMenu: This fxml file defines the Pre game settings menu screen

NewGameButton: This represents the new game button UI image used in main menu.

LoadGameButton: This represents the load game button UI image used in main menu.

CreditsButton: This represents the credits button UI image used in main menu.

ExitButton: This represents the exit button UI image used in main menu and game screens.

ReturnToMainMenuButton: This represents the new game button UI image used in the Credits menu, Game screens andDisplay Help Menu.

GameModMenuButton: This represents the game mod menu button UI image used in adjust settings menu.

MapMenuButton: This represents the map mod menu button UI image used in adjust settings menu.

PlayerNoSlider: This represents slider UI image used in Pre game settings menu.

Game Logic Layer

This layer consists of the subclasses that allow the gameplay of the game. While Menu Manager subsystem contains controllers that are allowing the menu operations, Game Manager subsystem contains classes that allow the main gameplay of the game. Game Object Manager contains the primary objects that the game contains such as Player and Territory. While the Menu Manager contains Menu Controllers and Game Object Manager contains the models of the game when the game is considered in MVC.

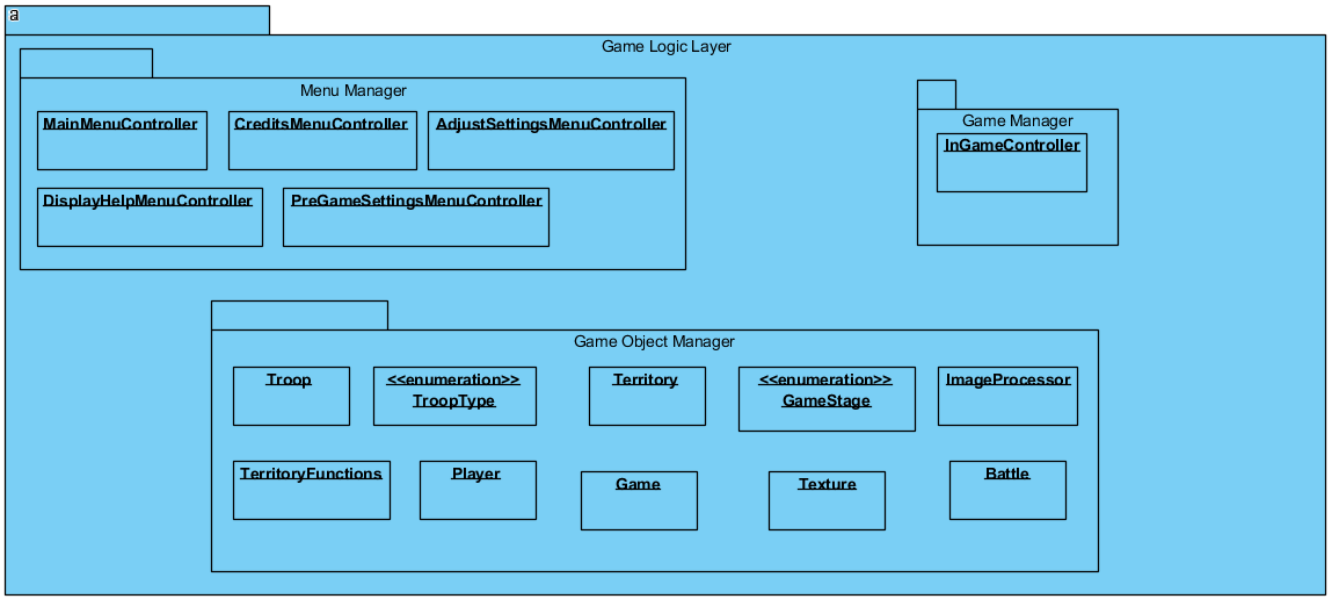


Figure 4: Game Logic Layer with the respective subsystems

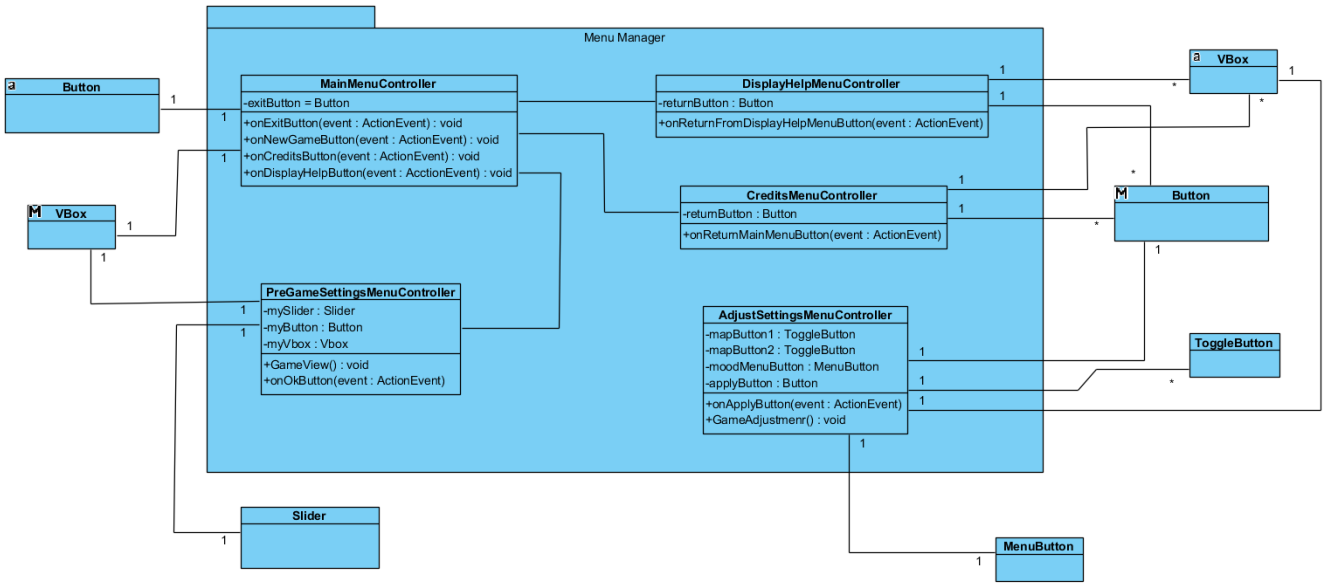
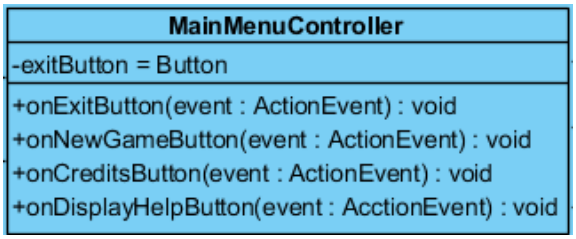


Figure 5: Menu Manager with the respective class diagrams

MainMenuController

This class handles the main menu interactions and Button functions



Attributes

exitButton: This UI element represents the button that allows the player to quit the game.

Methods

onExitButton: This method allows the exitButton to execute the termination function.

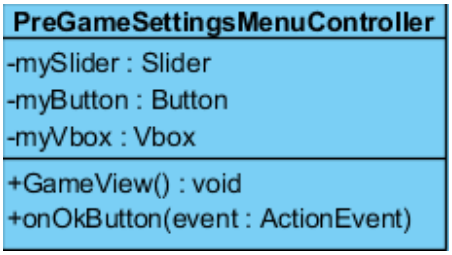
onNewGameButton: This method allows a screen transition between MainMenu and PreGameSettingsMenu.

onCreditsButton: This method allows a screen transition between MainMenu and CreditsMenu

onDisplayHelpButton: This method allows a a screen transition between MainMenu and DisplayHelpMenu.

PreGameSettingsMenuController

This class handles the PreGameSettings Menu interactions and Button functions.



Attributes

mySlider: This UI element represents the slider that decides the number of players in the game

myButton: This UI element represents the button that apply the number of players in the game and allows the showing of the Textfiels where the players enter their player names.

myVbox: This UI element represents the Vbox which allows the displaying of the buttons and sliders.

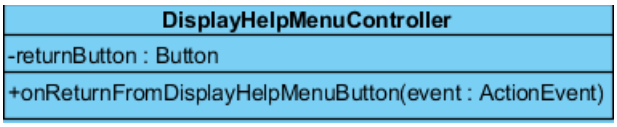
Methods

GameView: This method allows the displaying of the Textfields and getting the players names through them.

onOkButton: This method allows the start of the game by starting the get the names from the textfields and starting the initialization of the game.

DisplayHelpMenuController

This class handles the button interactions of the DisplayHelpMenu



Attributes

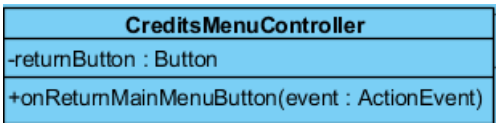
returnButton: This UI element represents the button that allows to return to Main Menu.

Methods

onReturnFromDisplayHelpMenuButton: This method allows the returnButton to make the player return to Main Menu.

CreditsMenuController

This class handles the button interactions of the DisplayHelpMenu



Attributes

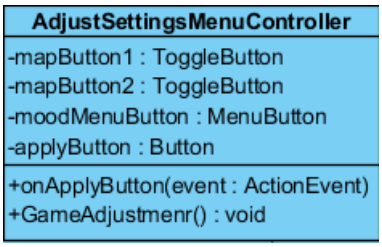
returnButton: This UI element represents the button that allows to return to Main Menu.

Methods

onReturnFromDisplayHelpMenuButton: This method allows the returnButton to make the player return to Main Menu.

AdjustSettingsMenuController

This class handles the button interactions of the AdjustSettingsMenu



Attributes

mapButton1: This UI element represents the button that allows the decision of choosing World Map.

mapButton2: This UI element represents the button that allows the decision of choosing Tamriel Map.

modMenuButton: This UI element represents the menubutton that allows the options of choosing the gameplay mod(Classical, Time, Golden Territory)

applyButton: This UI element represents the button that allows the applying the decisions made in the adjustSettingsMenu.

Methods

onApplyButton: This method allows the applyButton to apply the decisions made in the AdjustSettingsMenu to the game.

Game Manager

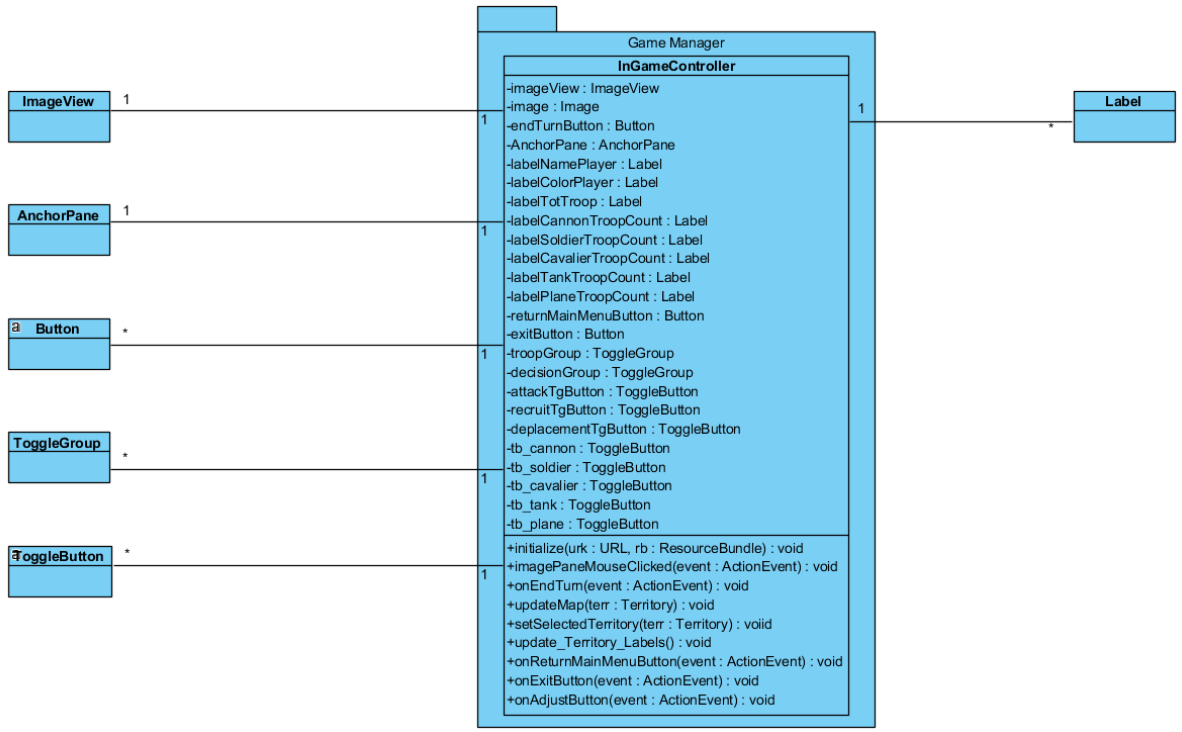
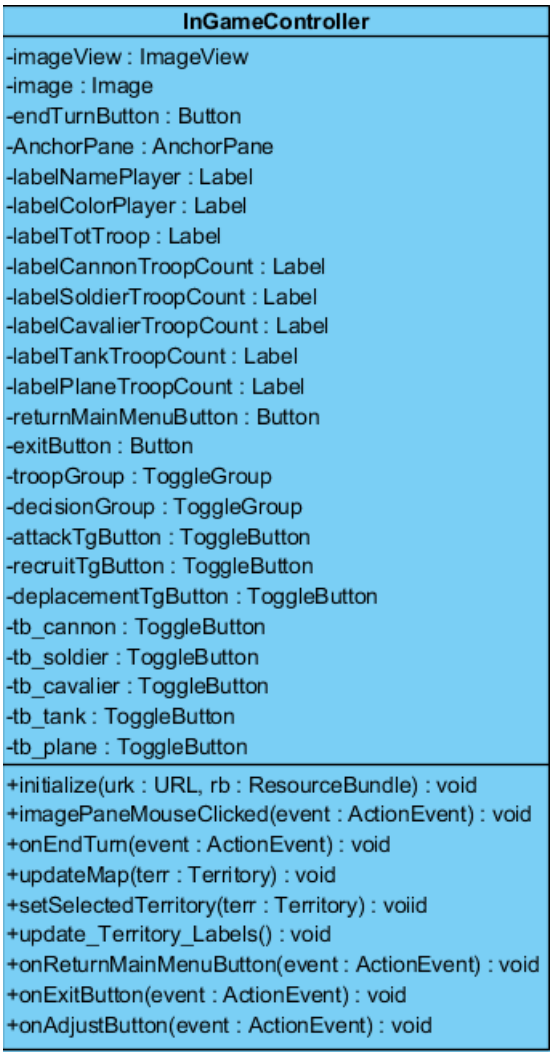


Figure 6: Game Manager subsystem with respective classes



Attributes

imageView: This UI element represents the image screen that allows the image to be shown.

image: This UI element represents the game map image.

endTurnButton: This UI element represents the button that allows to end turn.

returnMainMenuButton: This UI element represents the button that allows to return to main menu.

exitButton: This UI element represents the button that allows to exit the game.

AnchorPane: This UI element represents the Panel that shows the image along with the buttons.

labelNamePlayer: This UI element represents the label that shows player names.

labelColorPlayer: This UI element represents the label that shows the color of the players.

labelTotTroop: This UI element represents the label that shows the total troop count that a player receives in a turn.

labelCannonTroopCount: This UI element represents the label that shows the total cannon troop count that the player has in a territory.

labelCavalierTroopCount: : This UI element represents the label that shows the total cavalier troop count that the player has in a territory.

labelSoldierTroopCount: : This UI element represents the label that shows the total soldier troop count that the player has in a territory

labelTankTroopCount: : This UI element represents the label that shows the total Tank troop count that the player has in a territory.

labelPlaneTroopCount: : This UI element represents the label that shows the total Plane troop count that the player has in a territory.

TroopGroup: This UI element represents the toggleGroup that contains the troop types. It allows only one type of troop to be chosen at one time.

decisionGroup: This UI element represents the toggleGroup that contains the stage types. It allows only one type of Game Stage to be chosen at one time.

attackTgbutton: This UI element represents the toggle button that allows the attack stage to be active

recruitTgbutton: This UI element represents the toggle button that allows the recruit stage to be active

deplacemenTgbutton: This UI element represents the toggle button that allows the deplacement stage to be active.

tb_cannon: This UI elements represents the toggleButton that allows the cannon troop type to be active for recruitment.

tb_soldier: This UI elements represents the toggleButton that allows the soldier troop type to be active for recruitment.

tb_tank: This UI elements represents the toggleButton that allows the tank troop type to be active for recruitment.

tb_cavalier: This UI elements represents the toggleButton that allows the cavalier troop type to be active for recruitment.

tb_plane: This UI elements represents the toggleButton that allows the cannon plane type to be active for recruitment.

Methods

public void initialize(URL url, ResourceBundle rb): This method allows the initialization of the game, displaying the initial map condition and starting the game.

private void imagePaneMouseClicked(MouseEvent event) throws IOException: This method allows the player desired change occurs on the image, along with changing the specific data of the players.

private void onEndTurn(ActionEvent event): This method allows the endTurnButton to end turn.

private void update_Map(Territory terr): This method updates the map according to the changes made in the game.

private void setSelectedTerritory(Territory terr): This method allows the data of the clicked territory to be changed by setting the selected territory to the clicked territory.

public void onReturnMainMenuButton (ActionEvent event): This method allows the returnMainMenuButton to return to main menü..

public void onExitButton (ActionEvent event): This method allows the exitButton to exit the game.

public void onAdjustButton(ActionEvent event): This method allows the transition between InGame to AdjustSettingsMenu

private void update_Territory_Labels(): This method allows the territory labels to change with respect to the changes happening in the game.

Game Object Manager

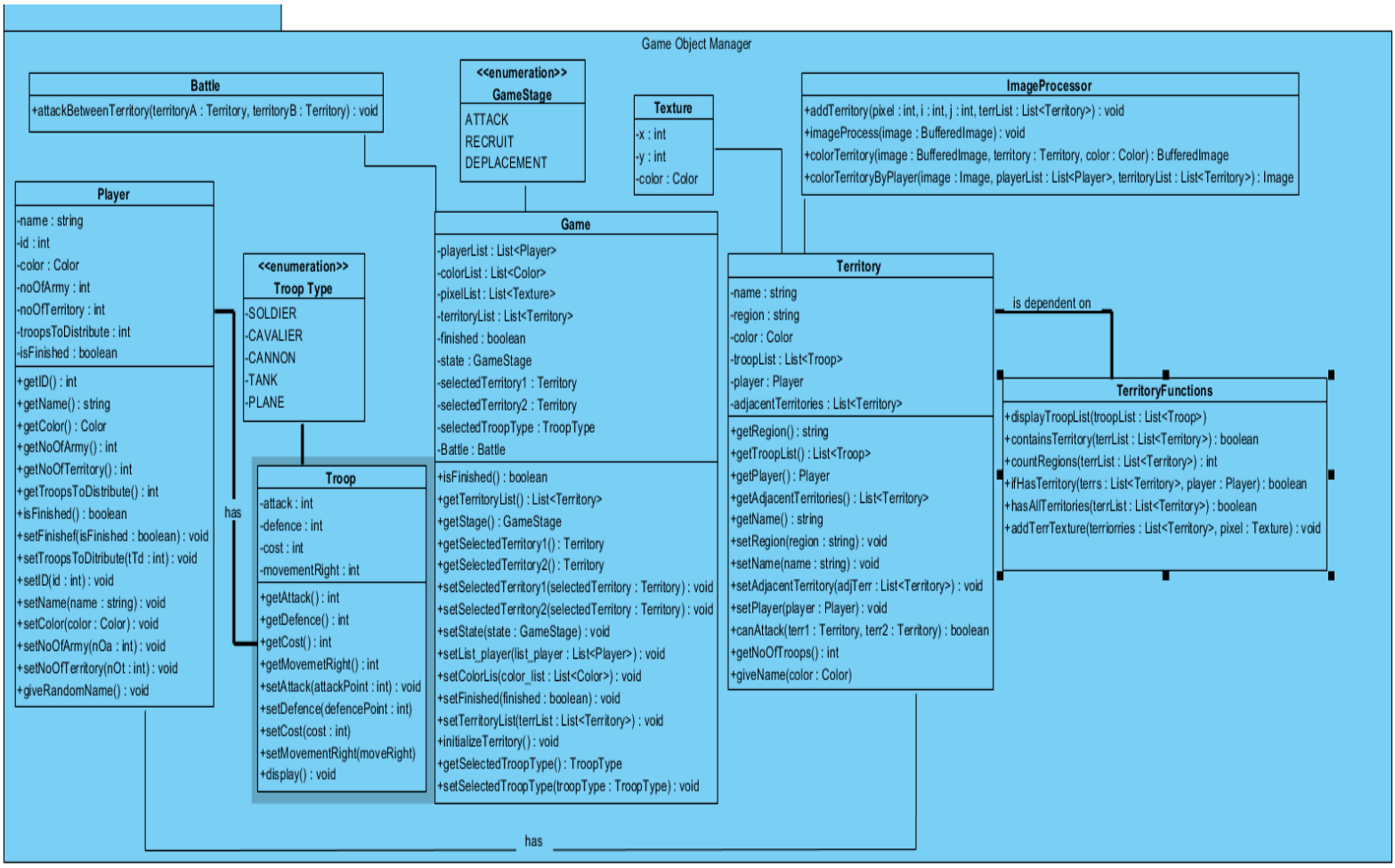


Figure 7: Game Object Manager subsystem along with the respective classes

Battle

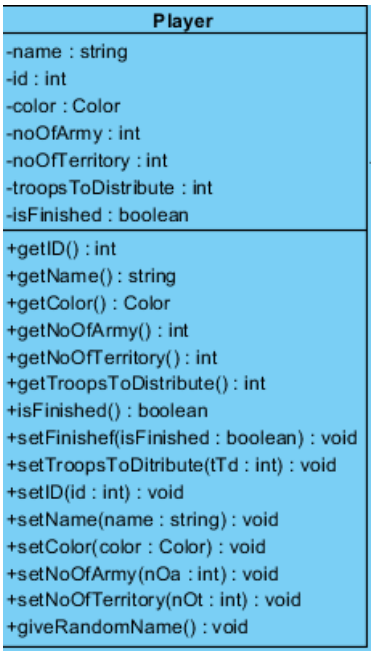
This class defines the battle function.

Battle
+attackBetweenTerritory(territoryA : Territory, territoryB : Territory) : void

public void attackBetweenTerritory (Territory territoryA, Territory territoryB): This method allows the attack stage to be acivated. It is the battle function of the game. Firstly, the two territories are checked whether they are adjacent or not. Then, if adjacent, the attacking territories attacking numbers and territories troops movementRights are being checked. If the battle is allowable based on the conditions, random attacking troops are being chosen in order to attack to the territoryB. If the total strength overpowers the total defence, the territory is occupied. Else, both players lose specific amount of troops.

Player

This class defines a player of the game.



Attributes

private String name:

Name of the player.

private int id:

Id of the player.

private int color:

Color of the player.

private int noOfRegion:

Number of regions that player has.

private boolean isFinished:

shows whether the player's turn has or not.

private Troop troopToDistribute: Troops that will be distributed according to number of regions and continents that player have.

private Boolean hasBiggestRegion: Boolean that shows whether player has biggest region or not.

Constructors:

Player(int id , String name , Color color):This is the constructor of Player class. In this constructor, ID, name and color will be specified by player. If

user does not enter a name for himself/herself , then random name will be generated and assigned to the user internally.

Methods:

public String getName():Getter method that returns the player name.

public void setName(String):Setter method that sets player name to the given string.

public int getID():Getter method that returns the player ID.

public void setID(int):Setter method that sets player ID to the given integer.

public Color getColor():Getter method that returns the color of the player.

public void setColor(Color):Setter method that sets color of the player to the given color.

public int getNoOfArmy():Getter method that returns the number of troops that player has.

public void setNoOfArmy(int):Setter method that sets the number of troops that player has to the given integer.

public int getNoOfRegion():Getter method that returns the number of regions that player has.

public void setNoOfRegion(int):Setter method that sets the number of regions that player has to the given integer.

public boolean isFinished():Getter method that returns whether it is player's turn or not.

public void setFinished(boolean):Setter method that sets the player's turn to the given Boolean.

public Troop getTroopToDistribute():Getter method that returns troops that are distributed to the player at each turn.

public void setTroopToDistribute(Troop):Setter method that sets troops that are distributed to the given troop.

public Boolean getHasBiggestRegion(): Getter method that returns a boolean that shows whether player has the biggest region or not.

public void setHasBiggestRegion(boolean):Setter method that sets hasBiggestRegion to the given boolean.

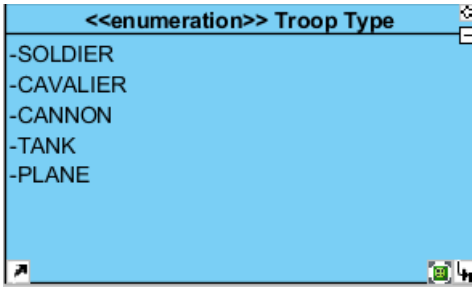
private String giveRandomName():If player does not enter a name for himself/herself, then random name will be generated by using this

method. This method will be used only in the constructor. Random name' s length will be 10.

In this method, random chars will be created and will be appended 10 times. At the end, it will return random name.

Troop Type

This class has only the enumeration that specifies the type of the troop. So, this class does not have any methods inside it. It creates only enumeration that defines the whole possible troop types that are possible and can be used in the game by player.



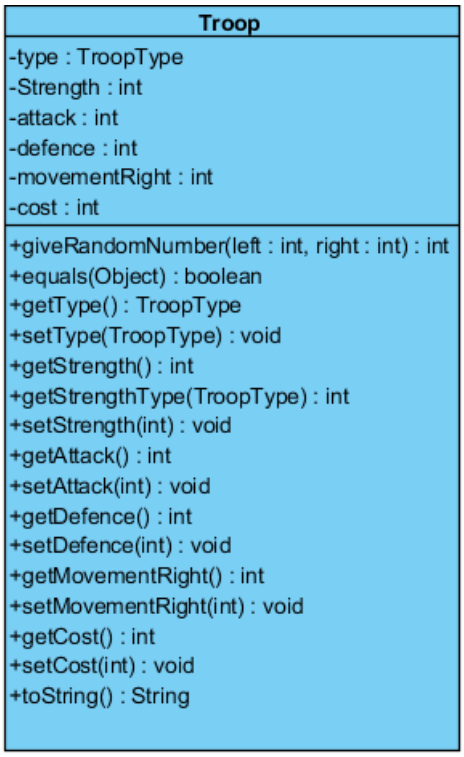
Attributes:

public enum TroopType.

In this enumeration , 5 different types are defined. These types are soldier , cavalier , cannon , tank and plane. Each type will have different cost , movement abilitiy , defence and engage power. However , these attributes will be defined in Troop class.

Troop Class

In this class troop is defined.



Attributes:

private TroopType type:

Type of the troop.

private int strength:

Strength of the troop. This attribute will be generated randomly. Even same types may have different strength. We want to make every single troop be important and make player think about using troop to attack or to defence.

private int attack:

Attack power of the troop.

ptivate int defence:

Defence power of the troop.

private int movementRight:

Movement ability of the troop.

private int cost:

cost of the troop.

Constructors:

public Troop (TroopType type, int untStrength, int unitCost, int unitAttack, int unitDefence , int mvmt)

In this constructor, a single troop is created by using values in the parameter.

public Troop(TroopType type):

In this constructor, only type is specified. Other attributes will be assigned in constructor internally. So, we will assign pre-arranged values to cost, attack, defencw , movementRight and strength. But these values will not harm the game's mechanics.

Type/attributes	strength	atttack	defence	movement	cost
Soldier	Btw 1 and 6	2	1	2	1
Cavalier	Btw 2 and 7	1	3	3	3
Cannon	Btw 4 and 9	3	2	1	7
Tank	Btw 5 and 10	4	3	1	9
plane	Btw 6 and 12	5	3	1	12

Methods:

public int giveRandomNumver(int leftLimit , int rightlimit):

Random number will be generated in this method. Generated random number will be between leftlimit and rightlimit. This method will be used when strength Is assigned in the constructor.

public boolean equals(Object object).

This method compares two objects and returns true if they are same , false otherwise.

public TroopType getType():

Getter method that returns type of the troop.

public void setType(TroopType type):

Setter method that sets type of the troop to the given type.

public int getStrength(TroopType troop):

Getter method that returns strength of the troop.

public void setStrength(int strength) :

Setter method that sets strength to the given integer.

public int getAttack():

Getter method that returns attack power of the troop.

public void setAttack(int attack):

Setter method that sets attack to the given integer.

public int getDefence():

Getter method that returns defence power of the troop.

public void setDefence(int defence):

Setter method that sets defence to the given integer.

public int getMovementRight():

Getter method that returns movement ability of the troop.

public void setMovementRight(int):

Setter method that sets movement ability to the given integer.

public int getCost():

Getter method that returns cost of the troop.

public void setCost(int):

Setter method that sets cost to the given integer.

public String toString():

This method returns a string that includes attributes of an object. In other words , this method is a string representation of troop. For example:

Type: SOLDIER

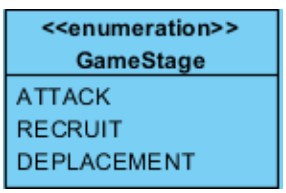
Strength: 3

Attack: 2

Defence: 2

Movement Right: 2

GameStage



This class has only the enumeration that specifies the stage of the game. So, this class does not have any methods inside it. It creates only

enumeration that defines the whole possible game stages that occures during the game.

Attributes

ATTACK:

This stage is used when attack phase. It means after distributing troops, players will try to capture other player’s territories with respect to their goals and after that, one of them attack another eventually.

RECRUIT:

This stage is used when players recruit their troops at the end of the tour. They will recruit new troops with respect to number of territories and regions that occupied.

DEPLACEMENT:

This stage is used when one territory is captured by an enemy player. In this stage, new troops will be deployed to target territory and owner of the territory will be changed. If whole territories are occupied by same player, then the owner of region will be changed too.

Game

Game Class

This is class where the game will be played. This class initializes the game by arranging the troops, setting colors, territories and players. It is basically the starting of the game and every method and attributes in it arranges so that the game can be initialized.

Attributes:

playerList: This is an arraylist consisting of player objects. This list holds the player for the game.

colorList: This is an arraylist consisting of colors. Dark slate grey, purple, olive, dark green, maroon, navy.

maListeDePixel: This is an arraylist of textures. When a place is occupied on the map, color of the area that is occupied should change. Textures holds the coordinates of the places that would change according to the occupation.

territoryList: This is an arraylist, consisting of territories.

finished: This boolean variable checks whether the game is finished or not.

isFirstTour: This boolean variable checks whether it is the first tour or not.

num_tours: This variable holds the number of tours.

state: This GameStage variable shows the current state of the game.

selectedTerritory1: The territory that is selected manually.

selectedTerritory2: The territory that is selected manually.

selectedTroopType: The troop type that has been selected by the user.

instance: This is the instance of the game.

battle: This Battle object is representing the battle of the game.

Constructor:

getInstance() : This constructor creates the game instance if the instance has not been created.

Game(): This constructor creates the attributes of the class within the this constructor method and sets: finished to fasle, selectedTerritory1 and selectedTerritory2 to 0, selectedTroopType to SOLDIER constant of the TroopType class and isFirstTour to true.

Methods

getPlayerList(): This method returns the list consisting of players.

getColorList(): This method returns the list consisting of colors.

isFinished(): This method checks if the player has finished his/her turn.

getTerritoryList(): This method return an arraylist of territories.

getStage(): This method returns the current stage of the game.

getSelectedTerritory1(): This method returns the first selected territory.

getSelectedTerritory2(): This method returns the second selected territory.

getPlayerWithName(String name): This method returns the player according to the given name as parameter.

setSelectedTerritory1(Territory selectedTerritory): This method sets the first selected territory.

setSelectedTerritory2(Territory selectedTerritory): This method sets the second selected territory.

setState(GameStage state): This method sets the current state of the game.

setList_player(List<Player> list_player): This method sets the playerList attribute of the class to the given parameter.

setColorList (List<Color> list_color): This method sets the colorList attribute of the class according to the given parameter.

setFinished(boolean finished): This method sets the finished attribute of the class to the given boolean parameter.

setTerritoryList(List<Territory> list_territory): This method sets the territoryList attribute of the class to the given parameter.

initializeTerritory(): This method initializes the territory according a specific player by setting the troops.

getListColor (int playerNo): This method creates myList and finalList variables. Creates a set of colors and adds them to the myList variable. Gets the color of players to the finalList variable and returns it.

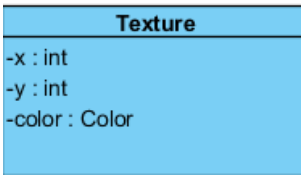
TroopType getSelectedTroopType(): returns the selected troop type.

setSelectedTroopType(TroopType selectedUnitType): Sets the selected trooptype.

tellTerritory(int x, int y): Takes parameters x and y which are the coordinates and this method returns the territory that belongs the given coordinates.

Texture Class

Texture class is used for constructing a map. Moreover, it is a smallest element of the map. Collection of textures will create a map. X and y refer to coordinates and color will determine the color of that coordinate. Since this class is so primitive, it only has constructor and 3 attributes.



Attributes:

public int x:

Refers to x coordinate of the texture.

public int y:

Refers to y coordinate of the texture.

public Color color:

Color of the specified coordinate ((x, y))

Contructors:

Texture(int x , int y , Color color):

In this constructor, coordinates and color set to the given values.

ImageProcessor

Methods:

public static void addTerritory(int pixel, int i, int j, List<Territory> terrList): This method adds a specific coordination in order for coloring it later

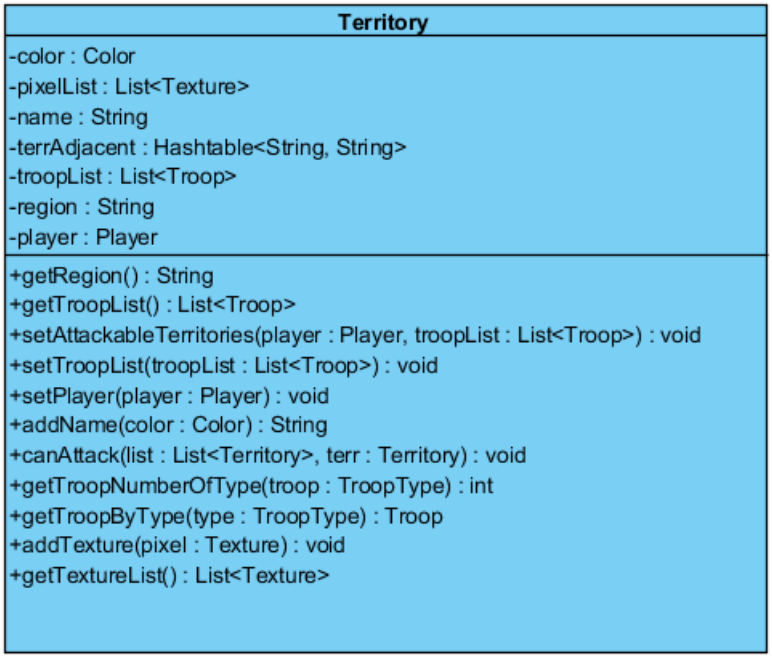
public static List<Territory> imageProcess(BufferedImage image): This method allows to process the image that is, updating the image with the changes that has been added by the coders.

public static BufferedImage colorTerritory(BufferedImage image, Territory territory, Color color): This method allows to color a specific territory

public static Image colorTerritoryByPlayer (Image imageParam, List<Player> playerList, List<Territory> territoryList): This method allows to color a specific territory distinctly based on the player.

Territory

In this class, we define a territory. Name, owner, region and adjacent territories will be arranged here. Properties of the territory are defined in this class. Behavior of territory will be defined in TerritoryFunctions class



Attributes:

public Color color:

Color of the territory.

public List<Texture> pixelList:

A collection of textures. This collection creates a territory. It means that, territory includes all textures in that list. Texture that is not in that list is not in the boundary of the territory.

public Hashtable<String, String> terrAdjacent:

A collection of adjacent territories.

public List<Troop> troopList:

Troops that are on the territory.

public String region:

Region of the territory.

public Player player:

Player that rules the territory.

Construcors:

Territory(Color color):

In this constructor, color of the territory set to the given color. Whole collections are initialized and name is assigned by using addName method. The explanation of that method is below.

Methods:

public String getRegion():

Getter method that returns the region.

public List<Troop> getTroopList():

Getter method that return all troops that are on the territory.

public void setAttackableTerritories(Player player , List<Troop> troopList):

Setter method that sets the attackable territories.

public void setTroopList(List<Troop> list):

Setter method that sets troop list to the given list.

public void setPlayer(Player player):

Getter method that returns the owner of the territory.

public int getTroopNumberOfType(TroopType troop):

Getter method that returns the count of units which type is specified by parameter.

public Troop getTroopByType(TroopType type):

Getter method that returns type of the troop.

public void addTexture(Texture pixel):

This method adds a new pixel to the pixel list. So, we can add expand territory by adding pixel to list.

public List<Texture> getTextureList():

Getter method that returns to texture list.

public String addName(Color color):

In this method, territory is named. Unique color is assigned for each territory. So, we create a switch – case that arranges adjacent territories and give a name to territory. Adjacent territories will be added to the Hash Table here.

public void canAttack(List<Territory> list , Territory terr): This method specifies which territories are attackable from which territory.

TerritoryFunctions

Methods:

public static void displayTroopList(List<Troop> troopList): This method allows to display the troop list.

public static boolean containsTerritory(List<Territory> territories, Color color): This method checks whether a specific territory belongs to a specific player. The checking occurs by checking the color.

public static void addTerritoryPixel(List<Territory> territories, Texture pixel): This method allows a specific color change by adding textures should the need happens

public static int countRegions(List<Territory> territories): This method returns the number of region in a specific territory list.

public static boolean ifHasTerritory (List<Territory> territories, Player players): This method checks whether a specific player has a specific territory

public static boolean hasAllTerritories(List<Territory> territories, Player player): This method checks if a player has all territories.

Data Layer

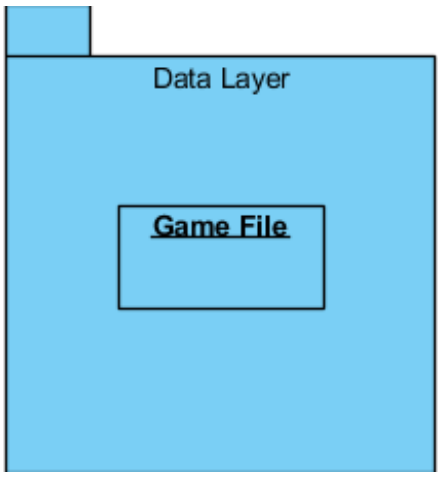


Figure 8: Data Layer Deployment Diagram with its respective subsystem

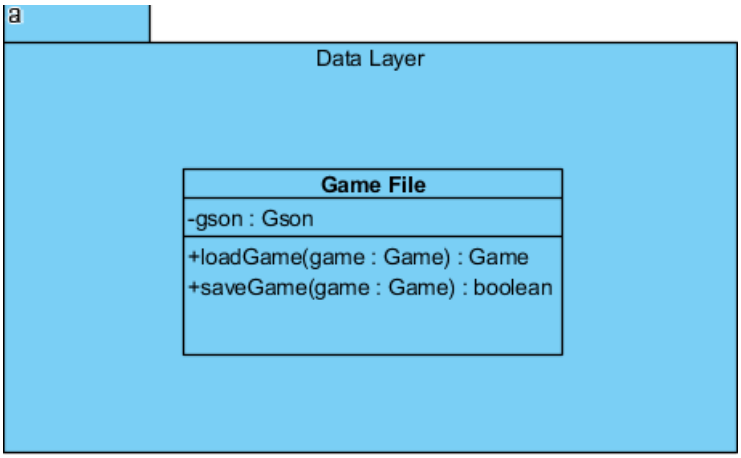


Figure 9: Data Layer wirth its respective class

Constructors:

public GameFile() : This is the constructor that initializes Gson instance.

Attributes:

private Gson gson : This is the gson object that interact with jsons

Methods:

Public Game loadGame(Game game): This method allows to load a previously saved game and start playing from where it was left.

Public Game saveGame(Game game): This method allows to save currently played game.

4. Low Level Design

4.1 Object Design Trade-offs

The difference between Object Design Trade-offs and General Trade-offs (In part 1.2.3) is this part specifically focuses on the implementation design and not the end product.

Loose Coupling vs Tight Coupling: In order to decrease dependence of the classes with each other and make it easier to test for specific code errors, We will use Loose Coupling instead of Tight Coupling. Also, we plan to increase modifiability and maintainability of our code with the Loose Coupling.

Inheritence vs Composition: We design to make our objects in our code as independent from each other as possible. Therefore, we prefer Composition over Inheritence. Thus, our modifiability and maintainability will increase as well

Centralized vs Decentralized: In our game, we mostly use centralized design. Most of the gameplay aspect and menu operations are handled in controller classes. We prefer centralized design in order to increase Maintainability and reusability of the code.

4.2 Final Object Design

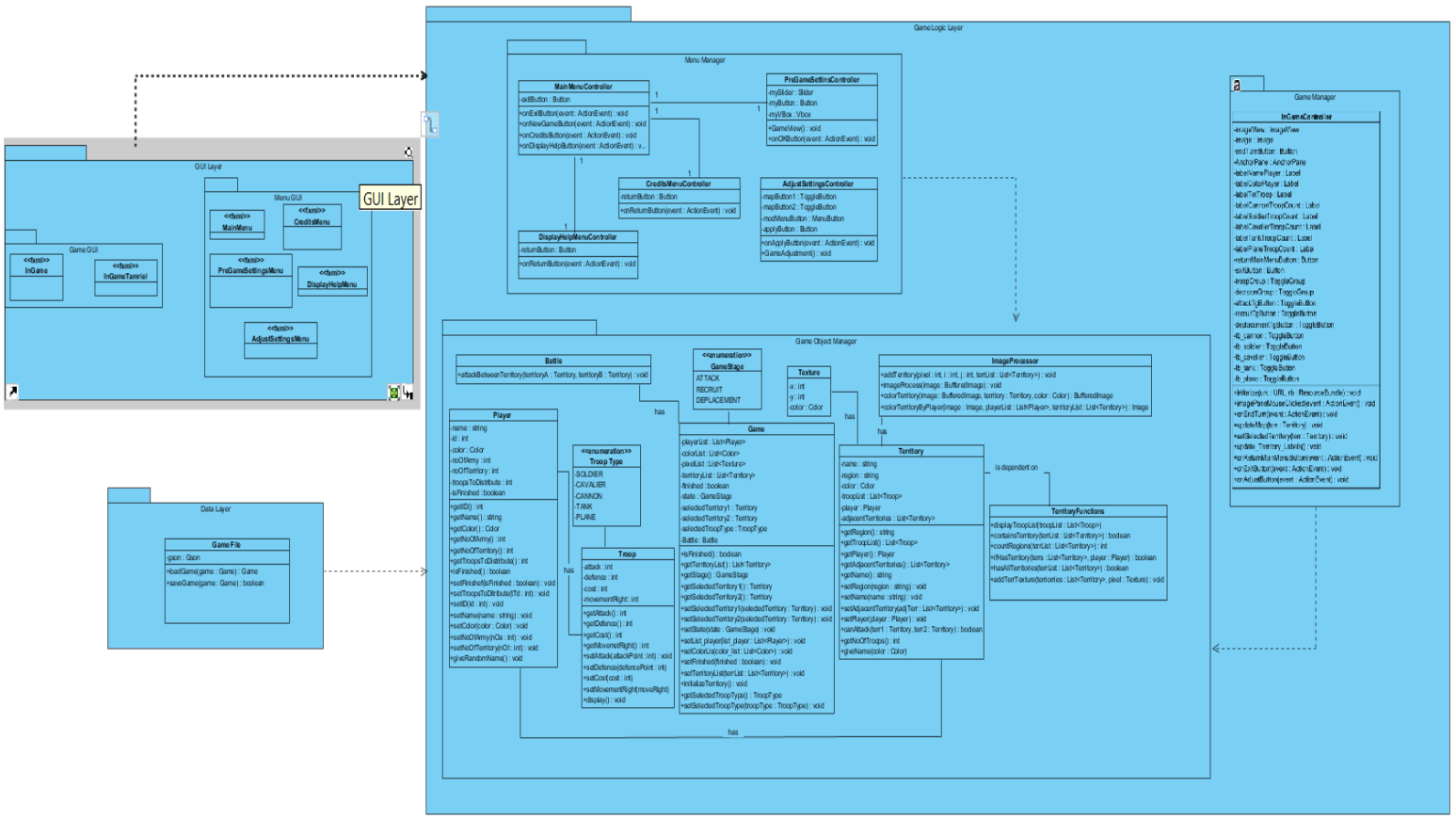


Figure 10: Full Class Diagram along with the layers that they are in

4.3 Packages

Java. util: This package is going to be used for data management. Data management is required in order to hold the datas such as which player holds which territory at the current stage of the game and how many troops does each player has.

JavaFXscene: This package includes sub-packages such as input, layout and image. We will use these subpackages to use handlers for mouse and keyboard events, to fit GUI objects,to support images in GUI.. Input, layout and image will accomplish these events in a respected order.

JavaFXevent: This package handles the events such as key presses and mouse clicks.

4.4 Class Interfaces

ActionListener: This interface catches any action with corresponding action events. This interface will be used to enable buttons in the game.

MouseListener: This interface catches mouse clicks. This interface is going to be used to implement the user input into the game. These inputs include: RecruitStage (for recruiting the newly gained troops), AttackStage(for attacking to different territories), DistributeStage(Distributing the remaining troops), endTurn.

KeyListener: This interface catches keyboard actions. The users entries of name for a save slot will be detected by this listener.

Javax.swing.undo: This interface allows undo/redo functions in the game if necessary.

Initializable: This interface is responsible for initializing an fxml controller. It is used to interact and manipulate the elements in the fxml file.

5. Improvement Summary

In the second iteration of our design report, we have updated and redesigned our system based on the changes we have made on the implementation. Firstly, the changes on the Analysis report has affected our object design and thus allowed changes. Secondly, the updates that we have made on the Design report also changed the object design as well. The changes we have made are:

1.We have changed our models in order to have more clear MVC design

2.We have separated Territory into two classes that are Territory and Territory functions in order to have simplicity. Territory class simply holds the properties and get and set methods of the Territory while the TerritoryFunctions handles more complex methods of the Territory class.

3.We have added new GUI elements, controllers and models that we could not have foreseen during the first iteration. Some of these classes are Texture and ImageProcessor model classes, AdjustSettings GUI element and AdjustSettingsController. The classes emerge as the implementation progress continues.

4. In our design report, we have fixed and rearranged our Subsystem decomposition part.

5. We have fixed and rearranged our subsystem services part completely.

6. We have added new fxml files such as the Menu files and Game Screen files to our game.

7. We have added controllers such as menu controllers and game controllers to our game.

8. We have added Texture and ImageProcessor classes in order to apply color changes on the image.

9. We have added Game File class in order to save and load our games.

10. We have edited the introduction part of our report.

6. References

[1] "Risk Online", Pogo,[Online] Available : https://www.pogo.com/games/risk?sl=2&gamekey=risk

[2] "Risk", Wikipedia,[Online] Available: https://en.wikipedia.org/wiki/Risk_(game)