CS 319 Term Project Design Report

Alara Yaman 21101164
Berfu Deniz Kara 21201662
Can Demirel 21401521
Muhammed Emin Aydın 21002035
Servan Tanaman 21201977

Supervisors: Gülden Olgun, Eray Tüzün

Project Group Name: Oldies but Goldies

Project Topic: Risk Board Game

1.Introduction

1.1 Purpose of The System

Our game RISK will be played over a network. The game allows three to six person to play. The main purpose of the game is to invade all the territories or accomplish all the given missions before other players. In this report we will describe how we will implement our functional and non-functional requirements according to our project plan.

1.2 Design Goals

1.2.1 Easy to Use

We make our game easy to use because we want our game to be more fun for every player without any difference. Plus we use graphical interface to be more understandable.

1.2.2 Performance

We want our game to work as fast fast possible. We measure a time to simulate the performance. Currently we get 924 ms which is the total run time of the existing method and we want to optimise this score even better.

1.2.3 Saving Information to Database

In our game, we want to save all the information on players actions. This database information helps players to continue to game after an exit. Moreover, saving information helps player to see others actions easily too.

2.High-level Software Architecture

2.1 Subsystem Decomposition

In order to understand the game system better, reduce the complexity of the overall system and to see our possible logical mistakes, we decomposed the packages of our game system one by one into classes they belong to and explained the purpose of their operations and specifications. In this way, we also aim to see new possible requirements both functional and non-functional.

2.2 Hardware/Software Mapping

We will use Java programming language and its packages for implementing our game engine and SQLite for our database system. For hardware requirements; a simple keyboard, a mouse, a monitor and a sound system will be enough to enjoy our game.

2.3 Persistent Data Management

We will record our game values(how many player, whose turn, who have where ect.) on a database system, so when the players wants to exit,their game will be saved on their account. When they check-in to their account, our game system will call back the game values belong to that account from database and player will be able move on from their last point on the game.

2.4 Access Control and Security

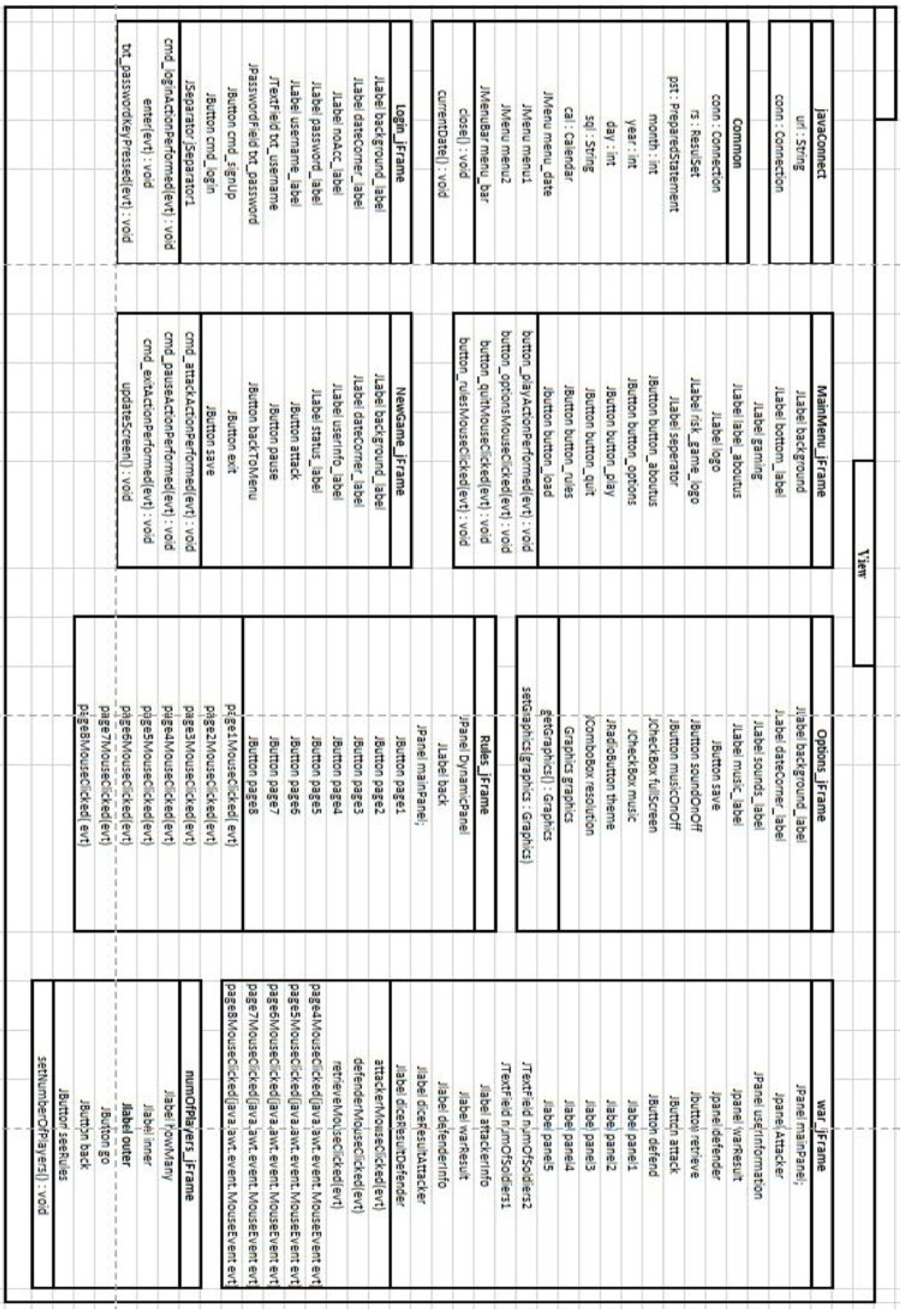
Anyone who has the game application on their desktop would be able to access the game and create an account or login. When creating an account; a username and a password will be asked and recorded into database. For login; username and password will be asked and checked whether they match or not, if they don’t login system will give an error message.In this way we aim to create a secure gameplay by preventing unwanted access to an account.

2.5 Boundary Conditions

Initializing the game will not need any extra program. Our game RISK will come with a .jar file which is executable. Exiting the program is possible by clicking the quit button which exists on the main menu window and on the playGame window. In order to prevent an error which is happened while accessing the resource data, we will provide a function check to approve all the resource data are there or not.

3.Subsystem Services

3.1 User Interface



3.1.1 javaConnect

This class is used Java Library to connect Database which is SQLite. There are no other duty for this class.

3.1.2 login_jFrame

Login frame page is a JFrame Class and this is the beginning of all. User have to login in order to pass this screen and play the game. User either already signed up or can use sign up button to get place in database.

3.1.3 MainMenu_jFrame

Main menu class is JFrame as well and this one is the main progress of the game in order to even play or see rules options exc. User see understandable and easy interface to manage his/her wishes. They can change the settings by clicking options, learn how to play the game both in english and turkish by clicking rules, user can meet with us by using about us button, user can start the game with play button.

3.1.4 newGame_jFrame

Newgame Frame is the brain of the game and war starts here! Users see the same world map and their areas on the map and also their numbers of soldiers. There might be 3, 4, 5 or 6 players in the game and they make moves on this frame if it is their turn.

3.1.5 options_jFrame

User have lots of options like choosing the music, pausing or playing the background sound, themes and lots of futures on this frame. This frame can be accessed via main menu and also from newgame frame. But there will not be all the settings and options if user try to access here from new game which means game is already on.

3.1.6 rules_jFrame

This screen can easily teach the game to user by showing clear examples and rules in english and Turkish.

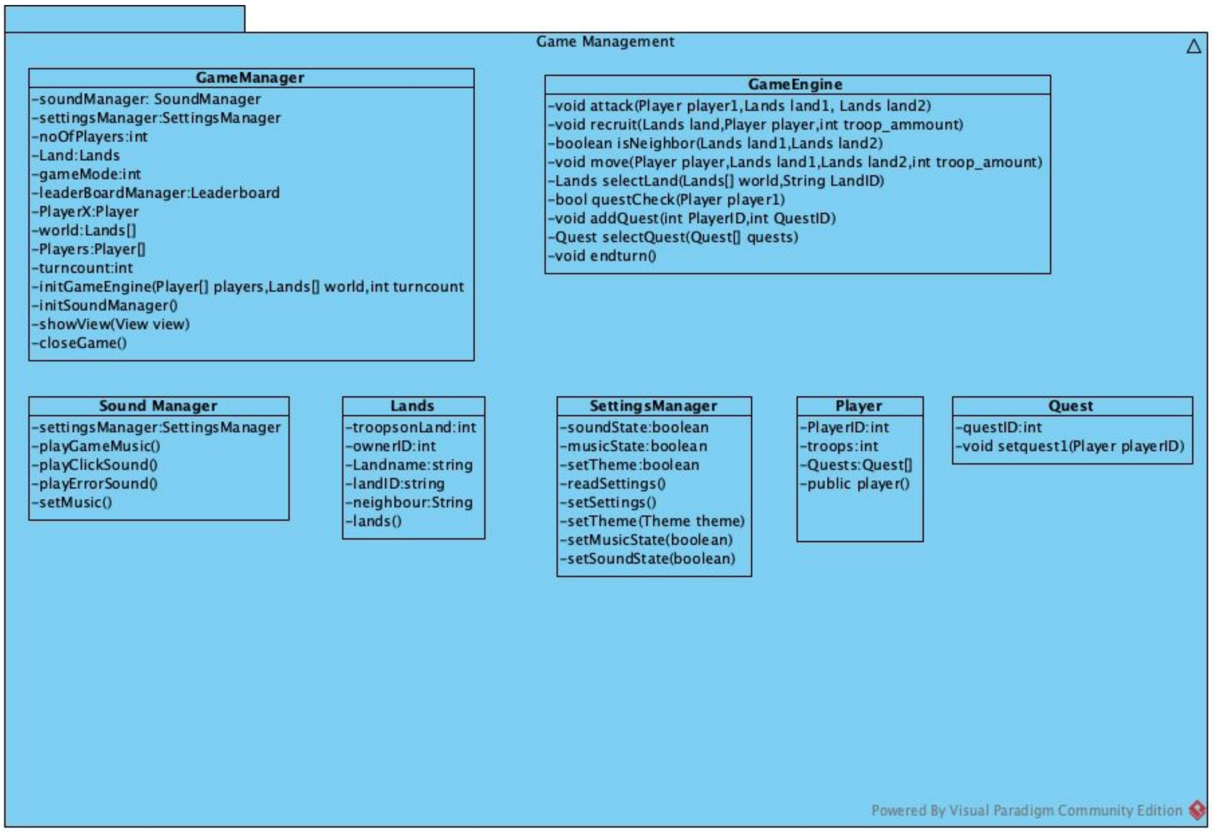
3.1.7 war_jFrame

If the user want to attack to another user's land, if all rules and pre conditions satisfies, as soon as user click to attack button on the map area this window will be shown as secondary window next to new game frame. New game frame will be non accessable until the war frame been finished by the user. War frame is the most exciting and enjoyable window for everyone. This window is the what business is happens! Dices rolls , war begins and there will be only one winner if attacker doesn't choose to retrieve.

3.1.8 numOfPlayers_jFrame

When the user clicks play button on the main menu this window directly appears and it wants user to select how many player will be playing the game and via this windows selection new game settings and options will be selected by the program.

3.2 Game Management


Game Management
GameManager
-soundManager: SoundManager
-settingsManager:SettingsManager
-noOfPlayers:int
-Land:Lands
-gameMode:int
-leaderBoardManager:Leaderboard
-PlayerX:Player
-world:Lands[]
-Players:Player[]
-turncount:int
-initGameEngine(Player[] players,Lands[] world,int turncount
-initSoundManager()
-showView(View view)
-closeGame()
GameEngine
-void attack(Player player1,Lands land1, Lands land2)
-void recruit(Lands land,Player player,int troop_ammount)
-boolean isNeighbor(Lands land1,Lands land2)
-void move(Player player,Lands land1,Lands land2,int troop_amount)
-Lands selectLand(Lands[] world,String LandID)
-bool questCheck(Player player1)
-void addQuest(int PlayerID,int QuestID)
-Quest selectQuest(Quest[] quests)
-void endturn()
Sound Manager
-settingsManager:SettingsManager
-playGameMusic()
-playClickSound()
-playErrorSound()
-setMusic()
Lands
-troopsonLand:int
-ownerID:int
-Landname:string
-landID:string
-neighbour:String
-lands()
SettingsManager
-soundState:boolean
-musicState:boolean
-setTheme:boolean
-readSettings()
-setSettings()
-setTheme(Theme theme)
-setMusicState(boolean)
-setSoundState(boolean)
Player
-PlayerID:int
-troops:int
-Quests:Quest[]
-public player()
Quest
-questID:int
-void setquest1(Player playerID)
Powered By Visual Paradigm Community Edition

3.2.1 GameManager

GameManager is a class that takes player number and game mod selections and creates Player,Land,Quest objects depending on user’s choices. In this class all starting attributes of the game will be created and stored on object arrays such as world[] that will contain Land objects. It will send these arrays and default turncount to GameEngine where all the functions of the game happens. Functions in the GameEngine will modify the values on GameManager and the data will be stored from here. The game view panel will also called in here.

3.2.2 GameEngine

GameEngine is the class that contains several methods like attack and recruit.This class will be called within the GameManager and it will run until user quits the game. Player’s actions in game such as attacking or moving will be selected by a switch case statement. GameEngine also contains many helper methods such as selectLand() or selectQuest() those functions will be used by other methods and they are implemented this way in order to provide an object oriented design.

3.2.3 SoundManager

SoundManager is the class that access the soundtracks in gamefiles. This class will be called in GameManager but its functions such as playClickSound() will be also called in GameEngine class.

3.2.4 SettingsManager

SettingsManager is class that generates game’s default setting values such as theme or soundLevel. It will be called in GameManager and it contains bool returning methods to control the UI features of the game.

3.2.5 Player

This class contains player constructor and it will be called 3 to 6 times in GameManager. Player objects has PlayerID in order to distinguish them. The total troop amount will also be declared in here but it will be modified in GameEngine. A Quests array will hold the quest objects the Player has.

3.2.6 Land

Land class will construct land objects and it will be called by GameManager. LandID,LandName,Neighbour values will be set in GameManager according to the game’s rulebook. These objects have a troopsonland value in order to process user actions in GameEngine. Neighbours and LandID will be defined as string because in GameEngine class isNeighbour() method is using Java's compareTo() function to check these values.

3.2.6 Quest

Quest class will contain a constructor that takes QuestID from GameManager. Each quest in Risk game will be defined as a method in this class and by using selectQuest() method in GameEngine class these Quest objects will be added to the Player's quest array.

4.Low-Level Design

4.1 Packages

4.1.1 SQL Package (java.sql.*)

Provides the API for accessing and processing data stored in a data source (usually a relational database) using the JavaTM programming language. This API includes a framework whereby different drivers can be installed dynamically to access different data sources. Although the JDBCTM API is mainly geared to passing SQL statements to a database, it provides for reading and writing data from any data source with a tabular format.

4.1.2 AWT Package (java.awt.*)

Contains all of the classes for creating user interfaces and for painting graphics and images. A user interface object such as a button or a scrollbar is called, in AWT terminology, a component. The Component class is the root of all AWT components.

Each Component object is limited in its maximum size and its location because the values are stored as an integer. Also, a platform may further restrict maximum size and location coordinates. The exact maximum values are dependent on the platform. There is no way to change these maximum values, either in Java code or in native code. These limitations also impose restrictions on component layout. If the bounds of a Component object exceed a platform limit, there is no way to properly arrange them within a Container object. The object's bounds are defined by any object's coordinate in combination with its size on a respective axis.

4.1.3 Swing Package (java.swing.*)

Java offers two standard library for graphical user interface (GUI), java.awt package and javax.swift package. These two packages are similar but main difference is that when a window, textfield or button created via awt, it created by operating system. Interfaces created with awt will look different on different operation systems like in Windows it will look like Windows window and like a Gnome window in NGU/Linux Gnome interface. However, Interfaces that are created with swing will look like same in every platform because swing objects are drawn by Java.

4.1.4 Toolkit Class (java.awt.Toolkit)

This class is the abstract superclass of all actual implementations of the Abstract Window Toolkit. Subclasses of the Toolkit class are used to bind the various components to particular native toolkit implementations. Many GUI events may be delivered to user asynchronously, if the opposite is not specified explicitly. As well as many GUI operations may be performed asynchronously. This fact means that if the state of a component is set, and then the state immediately queried, the returned value may not yet reflect the requested change.

Most applications should not call any of the methods in this class directly. The methods defined by Toolkit are the "glue" that joins the platform-independent classes in the java.awt package with their counterparts in java.awt.peer. Some methods defined by Toolkit query the native operating system directly.

4.1.5 Event Class (java.awt.event.*)

Provides interfaces and classes for dealing with different types of events fired by AWT components.

4.1.6 GridBagLayout Class (java.awt.GridBagLayout)

The GridBagLayout class is a flexible layout manager that aligns components vertically, horizontally or along their baseline without requiring that the components be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells, called its display area.

Each component managed by a GridBagLayout is associated with an instance of GridBagConstraints. The constraints object specifies where a component's display area should be located on the grid and how the component should be positioned within its display area. In addition to its constraints object, the GridBagLayout also considers each component's minimum and preferred sizes in order to determine a component's size.

The overall orientation of the grid depends on the container's ComponentOrientation property. For horizontal left-to-right orientations, grid coordinate (0, 0) is in the upper left corner of the container with x increasing to the right and y increasing downward. For horizontal right-to-left orientations, grid coordinate (0, 0) is in the upper right corner of the container with x increasing to the left and y increasing downward.

4.1.7 GridBagConstraints Class (java.awt.GridBagConstraints)

The GridBagConstraints class specifies constraints for components that are laid out using the GridBagLayout class.

4.1.8 Calendar Class (java.util.Calendar)

The Calendar class is an abstract class that provides methods for converting between a specific instant in time and a set of calendar fields such as YEAR, MONTH, DAY_OF_MONTH, HOUR, and so on, and for manipulating the calendar fields, such as getting the date of the next week. An instant in time can be represented by a millisecond value that is an offset from the Epoch, January 1, 1970 00:00:00.000 GMT (Gregorian).

The class also provides additional fields and methods for implementing a concrete calendar system outside the package. Those fields and methods are defined as protected.

A Calendar object can produce all the calendar field values needed to implement the date-time formatting for a particular language and calendar style (for example, Japanese-Gregorian, Japanese-Traditional). Calendar defines the range of values returned by certain calendar fields, as well as their meaning. For example, the first month of the calendar system has value MONTH == JANUARY for all calendars. Other values are defined by the concrete subclass, such as ERA.

4.1.9 Locale Class (java.util.Locale)

A Locale object represents a specific geographical, political, or cultural region. An operation that requires a Locale to perform its task is called locale-sensitive and uses the Locale to tailor information for the user. For example, displaying a number is a locale-sensitive operation— the number should be formatted according to the customs and conventions of the user's native country, region, or culture.

4.1.10 GregorianCalendar Class (java.util.GregorianCalender)

GregorianCalendar is a concrete subclass of Calendar and provides the standard calendar system used by most of the world.

GregorianCalendar is a hybrid calendar that supports both the Julian and Gregorian calendar systems with the support of a single discontinuity, which corresponds by default to the Gregorian date when the Gregorian calendar was instituted (October 15, 1582 in some countries, later in others). The cutover date may be changed by the caller by calling setGregorianChange().

5.Glossary & References

[1]’’Oracle’’https://docs.oracle.com/javase/9/docs/api/javax/tools/package-use.html[March, 2019]
[2]’’UltraBoardgames’’http://www.ultraboardgames.com/risk/game-rules.php[March,2019]
[3]’’Boardgamegeek’’https://boardgamegeek.com/boardgame/181/risk[March, 2019]
[4]’’Boardgamegeekthread’’https://boardgamegeek.com/thread/113195/drawing design-boardgame-pc-which-program-use[March, 2019]
[5]’’Instructable’’https://www.instructables.com/id/How-To-Design-Board-Games/[March, 2019]
[6]’’Academia’’https://www.academia.edu/7295897/Board_Game_Design_and_Implementation_for_Specific_Learning_Goals[March,2019]