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

In this report we aim to show you how the subsystems are interacting with each other for our design of RISK.

1.1 Purpose of The System

Our implementation of board game RISK will be a turn-based game which can be executed on Windows and OSx operating system. 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. This game based on playing pieces, world map, dices and mission cards.

Other than standard version we add new features such as new continent in Atlantic Ocean the name of "oldies but goldies republic" which has 3 territories, epidemics on units to make the game more interesting, timer on turns, season on continents, new missions, theme and sound.

1.2 Design Goals

1.2.1 Easy to Use

In order to improve the usability of risk game, we focused to make the game more intuitive and satisfactory. We make our game easy to use because we want our game to be more fun for every player without any difference. We design our buttons named shortly, explicitly and readable. Our colour schemes are content related. Plus we use a graphical interface to be more understandable. Instead of light, pastel colours that represent happiness and joy, dark matt colours are dominant. We added themes are added to make the game more customizable (such as Light, Dark, World War 2 theme). Thus, players can configure the game in their own way.

1.2.2 Performance

We want our game to work as fast as 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.

According to our tests, resources that risk game needs are approximate:

-256 MB RAM

-100-120 MB storage space

-Windows (XP and higher) OS

-Sound card

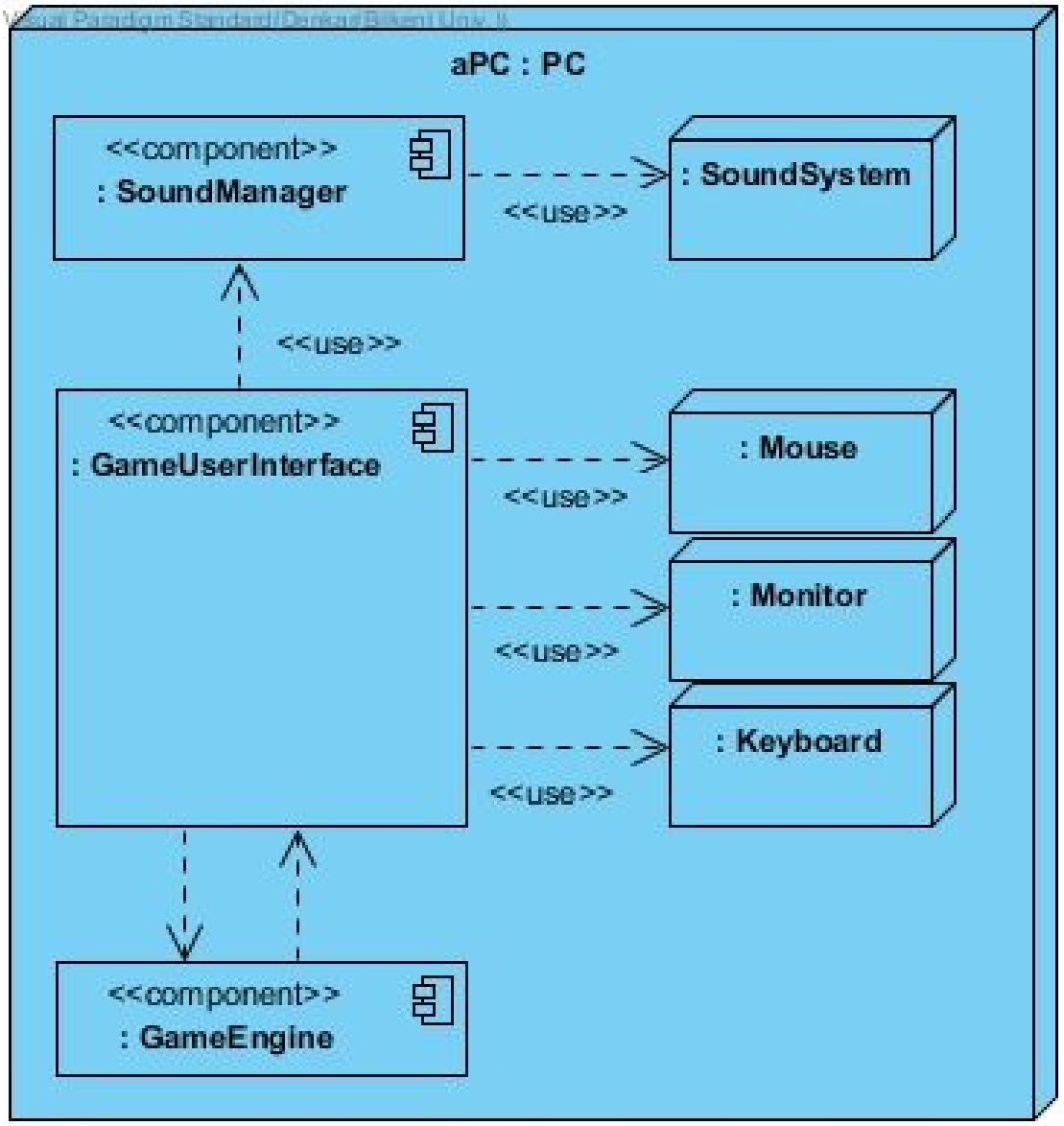
-Broadband internet connection

1.2.3 Saving Information to Database

In our game, we want to save all the information on players actions this helps to the game be secured. This database information helps players to continue to game after they want to quit the game or there is freeze or crash. The player can restart the game from last saved point. Also, we designed an auto-save system that saves game progress every 10 minutes. However, if the player don't save game and a crash occurs, he/she can continue from auto-save.

2.High-level Software Architecture

2.1 Hardware/Software Mapping

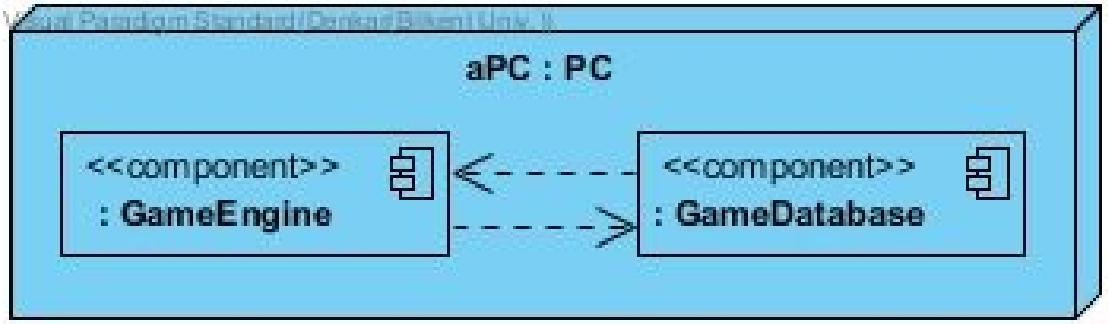


We are using Java programming language and its packages to implement our game engine and user interface. We are using SQLite for our database system implementation.

For hardware requirements; a simple keyboard, a mouse, a monitor and a sound system will be enough to play our game. Keyboard will be used to enter user name and password. Mouse and monitor are the main hardware components of our game as we use them to demonstrate the game images and to make the player select their moves during the game and in the main menu. Sound system is not a must but it is used by the user interface during the gameplay.

Hardware-software interactions are in one-way only; UI is using the keyboard, monitor, mouse and soundsystem, in software part however, UI and game engine work together simultaneously.

2.2 Persistent Data Management



We will hold some of our data in the game database because our design has a save/load system. The game database holds user name, user password, date, Player objects and Land objects for the loadGame option. User name and user password are hold because our design has a login system and players need to have an account to play the game. We made our design choice in this way because we want our players to have their own saved game. We don't have a server yet but we are thinking about to add one in order to give players the option to be able to reach their accounts and saved games from other computers via internet connection.

When users check-in to their account, our game system will call the related data which belong to that account from the database in the same computer and player will be able move on from their last point on the game.

2.3 Access Control and Security

Anyone who has the game setup file on their desktop would be able to setup 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 will be taken and checked whether it exists or not in the database, if it exists than the password is checked to see if it matches the user name, 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.

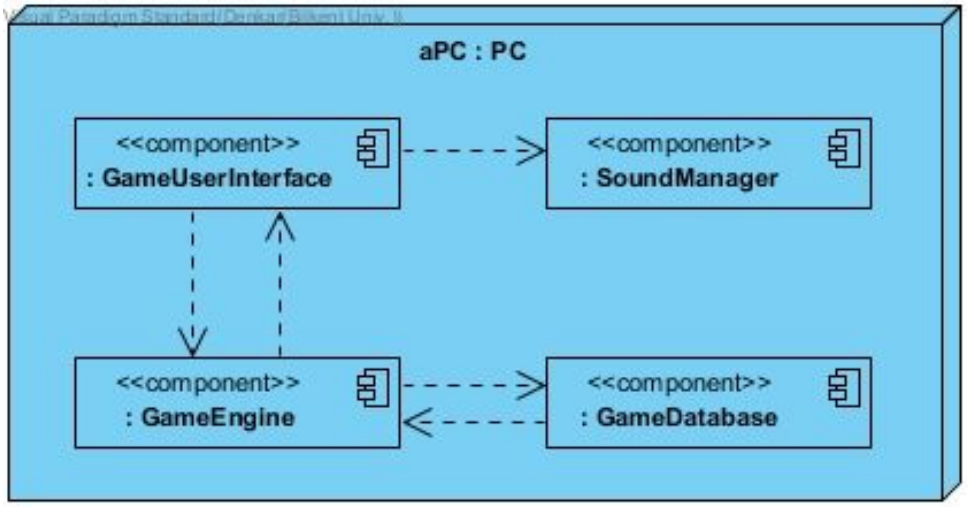
We added this login system because we want every player to have their own gameplay because RISK is a

2.4 Boundary Conditions

To upload our game, there will be a setup file which has all the soundtracks and images for the UI and the code files in it. Initializing the game will not need any extra program. Our game will come with a .jar file which is executable on any computer which has java. After the setup, double click on the game icon will run our game. 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 can happen while accessing the resource data, we will provide a function check to approve all the resource data are there or not.

3.Subsystem Services

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.



3.1 Game Engine System

Game System contains the classes and methods that performs the user action in game. It communicates with every other subsystem and act as a manager system. It takes event actions from Game UI System and updates it according to the modifications on datas. It can send data to Game Database System for saving purpose and it also can withdraw data from Game Database System to load an existing game. For specific events like battle Game System can call Sound Manager System to perform sound effects.

3.2 Game UI System

Game UI System contains the frames, panels and buttons that enables user interactions. It communicates only with Game System and it tells the Game which event has chosen. The frame classes in Game UI System shows the game data to players and they can be updated by Game System.

3.3 Game Database System

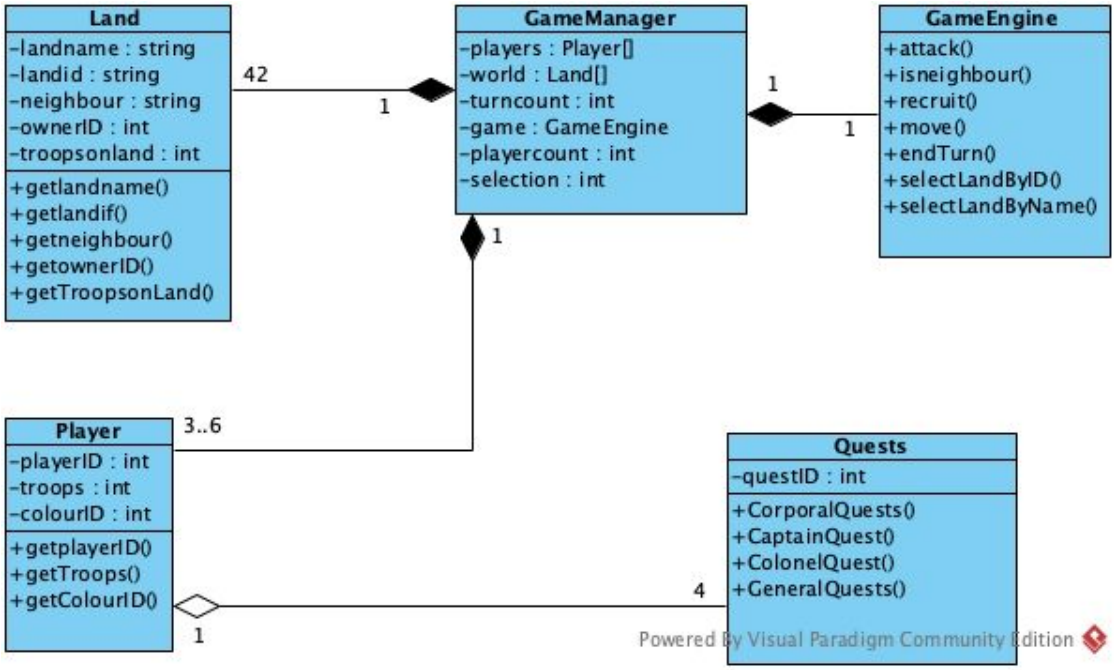
Game Database System works as a data storage in this project. It holds the players and land objects data when a game is saved. When User selects to load an existing game ıt will send the data to Game subsystem to initialize that existing game.

3.4 Sound Manager System

Sound Manager System holds the soundtracks and musics that the Game subsystem will call according to user events.

4. Low Level Design

4.1 Game Engine Class Diagram



4.1.1 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.

4.1.2 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.

4.1.3 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.

4.1.4 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.

4.1.5 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.

4.1.6 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.

4.1.7 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.

4.2 User Interface Class Diagram

4.2.1 javaConnect

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

4.2.2 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.

4.2.3 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.

4.2.4 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.

4.2.5 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.

4.2.6 rules_jFrame

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

4.2.7 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.

4.2.8 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.

4.3 Final Object Design

4.3.1 Game Object

Game Object in Risk System creates and holds the player, land and quest objects as object arrays. It declares a turn count and by using these datas it initialize the game. Users will interact with this object by selecting buttons on Game Frames. According to their selection methods in Game Engine class will modify the data on Game Object. There will be only 1 Game object in a game.

4.3.2 Player Object

Player Objects represents each user that plays RISK game. Each object has a unique playerID and colourID. These object will also hold an integer called ”troopamount”. This integer will represent the amount of troops that player can recruit. Player objects will hold and integer array named “Quests”. This array will hold questIDs and system can check the quests are completed or not by checking this array. There will be 3 to 6 player objects according to game set-up.

4.3.3 Land Object

Land Objects are representing the territories in the game board. They have a unique landID. Land Objects will also hold their owner’s playerID and an integer named “troopsonland” that shows the military force on that Land Object. Each Land Object has a string called “neighbours”. This string will include the adjacent territories landIDs by using this string Game Engine will determine the selected territories are neighbours or not. There will be 42 Land Objects in RISK game but we might increase that to 45 by adding new territories.

4.3.4 Quest Object

Quest Objects in RISK system will only have a unique questID as attribute but there will be many boolean returning methods that represents the quests in RISK game. System will check the returning values of these methods to determine the quest is completed or not. Each Player object will hold 4 quest objects therefore the Quest count will vary between 12 to 24 in a game.

4.4 Packages

4.4.1 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.

4.4.2 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.

4.4.3 Swing Package (java.swing.*)

Java offers two standard libraries 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.

4.4.4 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.

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

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

4.4.6 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.

4.4.7 GridBagConstraints Class(java.awt.GridBagConstraints)

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

4.4.8 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.

4.4.9 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.

4.4.10 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().

4.5 Class Interface

4.5.1 Players

Player class provides the methods that keep information of every player in the RISK game.

- +PlayerID(int): give every player a dedicated ID (1,2,…,6)
- +troopamount(int) : determine player’s troop count.
- + colourID(int): give color to every player to indicate lands ownership.
- +Questes[]: to store player’s quest progression in an array. The array involves quests’ IDs.
- -getID:return the player’s ID.

- -player(PlayerID,colourID,troopamount): create Player object.

4.5.2 Lands

This class provides the methods that keep information of lands.

- +LandID(string): gives every land a dedicated string name (like "indonesia","venezuela").
- +troopsonLand(int): determines how many troops are on the current land.
- +ownerID(int): determines the land belongs to which player.
- +neighbour(string):
- -getLandID(): returns land's ID.
- -lands(LandID,neighbour): creates Lands object.

4.5.3 Quests

This class provides the methods that keep the player's progression on quests.

- +GeneralQuest(): returns true or false according to the player's progression. If the quest in Quests[] array, return true.
- +CaptainQuest(): returns true or false according to the player's progression. If the quest in Quests[] array, return true.
- +ColonelQuest(): returns true or false according to the player's progression. If the quest in Quests[] array, return true.
- +GeneralQuest(): returns true or false according to the player's progression. If the quest in Quests[] array, return true.
- -questID(int): the constructor that takes questID from GameManager.

4.5.4 GameManager

GameManager mainly stores game data such as players and land IDs and send them to GameEngine.

- -players[]: to store playerIDs in array.
- -world[]: to store landIDs in array.
- -turncount: an integer variable to store turn count
- -game(players[],world[],turncount):
- -playercount: an integer variable to store player count.
- -selection: an integer variable to store mode selection.

4.5.5 GameEngine

- ❏ +attack(Player,Lands,Lands): compare attacker and defender sides dice results. Print out the end of the comparison. Update player's troop amount according to result.
- ❏ +recruit(Lands,Player,int): add troops to lands that are owned only by the current player. Update player's troop amount according to result.
- ❏ +move(Player,Lands,Lands,int): move troops by decrease troops from origin land and add them to destination land. Update player's and land's troop amount.
- ❏ +isNeighbour(Lands, Lands): check lands are neighbors by Lands class' neighbor variable. Return true or false.
- ❏ +endTurn():
- ❏ +selectLandByID(int): return a Lands object via its integer ID from world[] array.
- ❏ +selectLandByName(Lands[],string): return a Lands object via its string ID from world[] array.

5. Improvement Summary

In our design of RISK, we decided to add some new features and their implementations will take place in the UI and game engine.

One of these features is the new continent in the Atlantic Ocean the name of “Oldies but Goldies Republic” which has 3 territories. This new continent will make the game more interesting than the standard version.

The timer on turns such as 30 seconds for a fast turn and 1 minute on a usual turn will make the game more excited and fluent during gameplay.

We will add epidemics to reduce units to make the game more compatible. Moreover, we want to make the game more interesting.

Also we will add new missions to our cars and we add seasons to change the unit count.

Plus, if it is winter, unit count will be reduced from the attacker.

6.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]