2018-2019 Fall

CS319 Object Oriented Software Engineering

Term Project

Section: 01

Group: 2E

Group Name: ProCODERs

Lords in Halls

Design Report

Group Members

1. Şamil Iraz
2. Ateş Bilgin
3. Enes Emre Erdem
4. Gülnihal Muslu
5. Can Ozan Kaş

Supervisor: Eray Tüzün

Teaching Assistant: Gülden Olgun

Table of Contents

1. Introduction

1.1) Purpose of the system

In this project, our purpose is develop and improve a game which is named as Lords in Halls. In our game, there will be different colored knights which will determine the score of user, and there will be also walls which will be used to cover the knights depending on some rules defined in the game. Our aim is to implement a game which will be preferred by user.

1.2) Design Goals

For designing proper system, there are various factors we need to take into consideration as software engineers. Especially, non-functional requirements of the systems contain significant role in system designs. We would like to focus deeply on some of them, which are mentioned in requirement elicitation. To focus more on non-functional design goals, we also need to evaluate such non-functional requirements inside a criteria. There are four main criterias we use in order to evaluate our design.

1.2.1) Performance

As mentioned in the analysis report, performance of the game is one of the most significant design traits, because users tend to prefer softwares with higher performance. That is why most software developers concern about the performance. To increase the performance of the game, we will use HTML-javascript to create our UI. To design the logic of the game, we will try to use efficient algorithms to create and design the maps, and check solutions.

1.2.2) User Friendliness-Usability

User friendliness is another significant design feature, because software developers need to consider the user-side to make it more understandable and easy to use. We will design an understandable basic UI to make our application more user friendly and usable. Also playing game will be easy

to control, which will be done with basic mouse clicks and drag-drops. The game will be played on the web browser, which will not require any downloading processes, which is easier for players.

1.2.3) Security

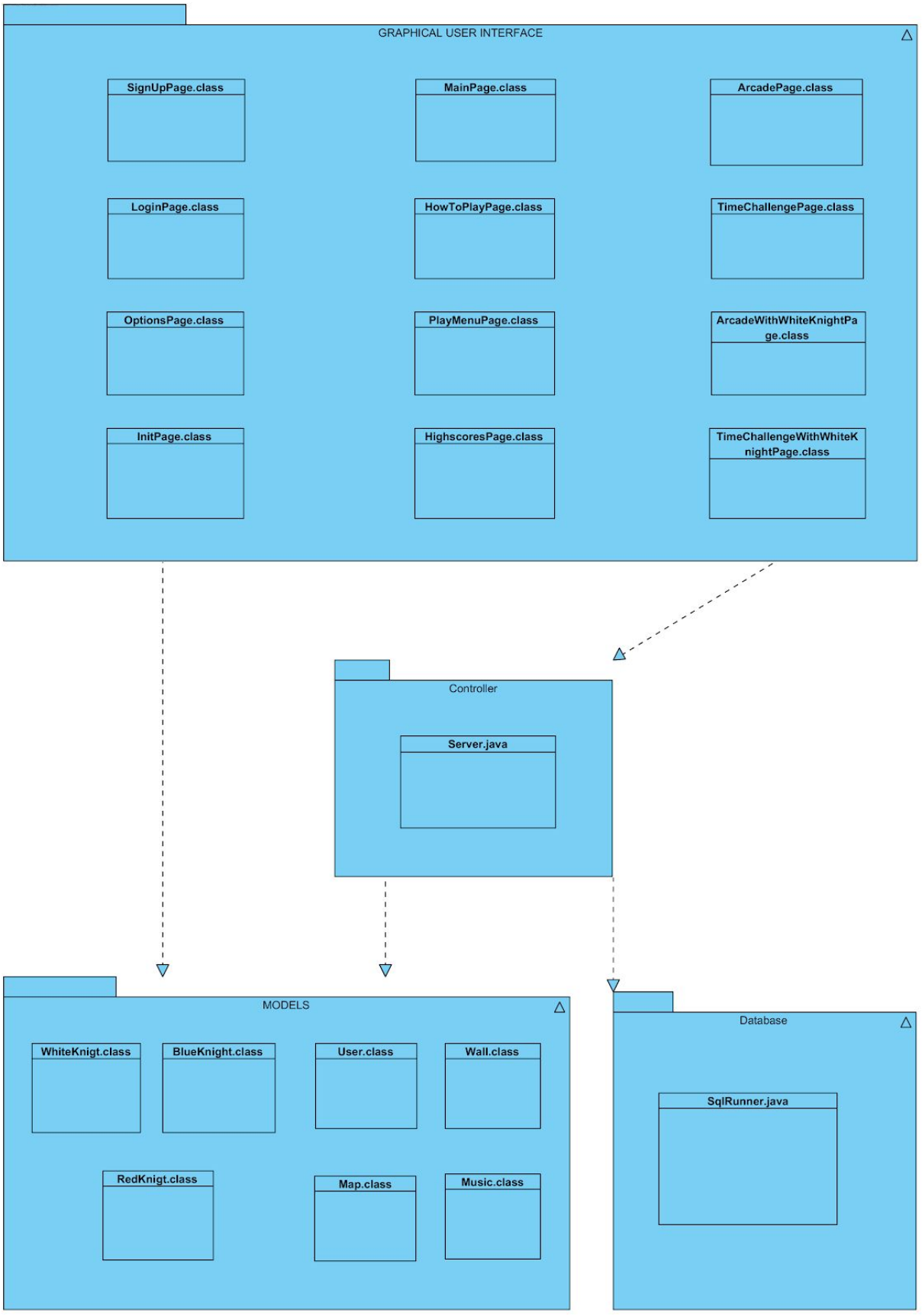
Security of a software is another important design goal, because as a software engineer, we need to keep the data records of the users secure. To make sure users data is private, we will ask them to sign up our game system with IDs and passwords. We will keep their score game records in the reliable database providers databases, such as DigitalOcean.

1.2.4) Maintainability

We need to take the maintainability of our software into consideration because it makes a particular software to be used by more commonly. In our game, we will use random map generator to create more and different type of level, which will make it more maintainable. Also, we are planning to create new game modes if possible. One of the new modes may be allowing user to create his/her own map and playing the other maps that other users create. Moreover, we will take different players opinions to make our design more maintainable.

2. High-level software architecture

2.1) Subsystem decomposition


GRAPHICAL USER INTERFACE
SignUpPage.class
MainPage.class
ArcadePage.class
LoginPage.class
HowToPlayPage.class
TimeChallengePage.class
OptionsPage.class
PlayMenuPage.class
ArcadeWithWhiteKnightPage.class
InitPage.class
HighscoresPage.class
TimeChallengeWithWhiteKnightPage.class
Controller
Server.java
MODELS
WhiteKnigt.class
BlueKnight.class
User.class
Wall.class
RedKnigt.class
Map.class
Music.class
Database
SqlRunner.java

We decided to have four different layers for our web application. Generally we implied MVC rules in our application. Our project was quite applicable for an MVC design pattern. Thus, we decided to design our project according to MVC with small changes.

GUI layer is the default View layer of MVC. In this part, we have classes for visual content. All of the page classes are stored here. With the help of HTML and Javascript, the content of these classes will be displayed on screen.

Controller layer includes the server code for our application. Server contacts with database layer and fills in the contents of the models with the data in gets from database. After the data is read from database and filled into model classes, it will be displayed inside View classes.

Model layer is again a default MVC model layer for our project. We have our classes for modeling: white knight, blue knight, red knight, wall, music, user and map. These models are required to simulate the items that we have in the project. The information about models will be inside its own model class. For example, the information about a white knight will be stored in WhiteKnight.class.

Finally, we have the database layer. In this layer we will only have one source code: SqlRunner.java. This java code includes all of the sql commands that we will need to connect with database such as getting information like scores and updating new informations such as new password, scores. We will setup our database into an online trustable database provides like DigitalOcean and SqlRunner.java will connect to it.

2.2) Hardware/software mapping

We will implement our application in Java language. Although back-end code of the application will be implemented in Java, front-end side of the application will be coded in HTML with Javascript. Key point here is that our application is a web application. Thus, any machine that has a web browser will be able to run our application. Most of the web browser support Java and HTML so it would be appropriate to code this project using Java and HTML.

Considering the performance of our application, since we do not have any advanced graphics such as 3d graphical content or motion effects, the application is expected to run smooth and fast in order to provide usability. On the other hand, we do not have any long stored procedures. So, the database interactions should be quite fast, almost instantaneous. Besides, the performance of the user's CPU should be good as well. The reason is that our application does not force the CPU to make complex calculations or 3d image rendering etc. Only simple calculations and controls are made inside CPU. Thus, it is expected that most CPU's are going to provide high performance.

The memory is not a problem for out application either. We will have the database for memory of application data and user machine's RAM for obtained and displayed data. These data is so small that it would not even be noticed by user at all. The database is the place to hold user information and highscore information tables. The amount of tables is also very few so this would not be a problem for us as well. The database provider will give us more than enough space to keep this data.

The keyboard and mouse will be used to interact with the application. Basically the keyboard is needed when user wants to write anything like

username or password. Mouse will be required to click buttons and navigate through the pages. Besides the interactions, while playing the game some music will be played in background. To hear the music, some kind of audio device is recommended as well but not compulsory.

2.3) Persistent data management

The data management is also an issue to be considered. We plan to setup our database into DigitalOcean which is a trusted online database provider. The database will hold quite small amount of data for us just to enable the application to handle with many users and highscores. We will not have a problem with the memory in that sense. Also, the service DigitalOcean provides is nonstop. Users will have access to application anytime.
Other than that, we will use MySQL to handle the queries. SQL commands will interact with database and do the required actions.

2.4) Access control and security

Lords in Halls is a internet and web-browser based game. Mainly, our application will be runned by the server code which is only be available in one computer instead of uploading to all user's devices. And also, even though there is not multiplayer game modes, there are some features that requires internet. For instance, instead of pre-determined maps, we will use a random map generator that creates maps dynamically. Furthermore, we will provide scoreboard which keeps global high scores and this feature requires internet connection because this leaderboard will be changed dynamically.

Security is another point of our application because mainly, Lords and Halls depends on users and their informations such as progress. Therefore, we will use security system which consists unique username and password.

2.5) Boundary conditions

Initialization

Case: Lords in Halls application will start whenever users go to the application URL via web browser.

Termination

Case 1: Application will be terminated if the user clicks `Exit` button in the first page.

Case 2: Application will be terminated if the user directly closes the browser. In this case, if the browser is reopened and application URL is entered, there is no need to log in again.

Failure

Case 1: In case of a database failure, the program will not crash but user will not be able to change its state. For example passwords changing will not be possible or saving a highscore to database will not be possible, but the game will still be playable.

Case 2: In case of a system failure (power issues, internet connection problems), the application will stop working. Once the problem is fixed, user can restart playing.

3. Subsystem services

3.1 Graphical User Interface

The Graphical User Interface layer will be consisted of two main parts. Firstly, there will be view code which are written in java. The view codes interacts with the graphical parts, provides required data to them. Second part will

consist of different Graphical User Interface files, written in HTML-Javascript. These two parts will interact with each other and with the user. There will be 12 views, each contains one page of the game. Therefore, there will be 12 files for Graphics, and 12 files for page controllers.

3.1.1 SignUpPage

In the sign up page part, system will ask user to enter the username and password he/she wants. The view class will interact with the controller server, to check them.

3.1.2 LoginPage

In the login page, system will ask user to enter his/her username and password to enter the system. View class will interact with the controller server to verify.

3.1.3 OptionsPage

In the options page, there will be two options, One will be change password part, where user will enter new password. Corresponding view class will interact with the controller class. There will be another controller bar, where user can increase or decrease the volume of the game.

3.1.4 InitPage

Initpage class will contain UI of the first page of the game. There will be two buttons, Log in and Sign Up. Both buttons will run the view class, and interact with the controller class, in order to change page.

3.1.5 MainPage

Main page will have 3 buttons, Options, Play, and highscore buttons. Each can interact with the controller class to change the page.

3.1.6 HowToPlayPage

This page will display a informative message about how to play the game. This page will also interact with the controller class via view files.

3.1.7 PlayMenuPage

In the PlayMenu page, there will be four buttons to allow player to choose

which game he/she wants to play. The corresponding view class will interact with the controller to compose level map and changes the game page.

3.1.8 HighScoresPage

Highscores page will display the highest scores ever done in the game, also show the player's highest score. Corresponding view class will interact with the controller, and controller constantly refreshes the displayed data with interacting database.

3.1.9 ArcadePage

Arcade Page is one of the page of game mods. In this page, user will play the arcade mode of game according the rules. This page will have game contents such as knights, walls, map. These contents will be updated dynamically depending on level and because of this, there will always be a connection between ArcadePage and Server to get and send information, contents.

3.1.10 TimeChallangePage

TimeChallange page is just like the ArcadePage. It will have again some contents and will be connected in anytime with Server.java to send and get required informations such as new map for new level.

3.1.11 ArcadeWithWhiteKnightsPage

ArcadeWithWhiteKnights page is just like the ArcadePage. It will have again some contents and will be connected in anytime with Server.java to send and get required informations such as new map for new level.

3.1.12 TimeChallengeWithWhiteKnightsPage

TimeChallangeWithKnights page is just like the ArcadePage. It will have again some contents and will be connected in anytime with Server.java to send and get required informations such as new map for new level.

3.2 Controller

The controller part of our Lords and Halls project consists singular java code which is named as Server.java. This java code will manage our application by using functions of Java and Html. When user tries to start our game, Server.java will respond to this request and answers it accordingly. Furthermore, the transitions between pages will be performed by Server. By using defined types of requests such as @GET, @POST, our server will answer to client and responses depending on requests. Server will also be in interaction with SqlRunner.java which is responsible from database transactions. The reason behind of this connection is that when some actions such as new attempt of login, checking leaderboard or updating scores of user, server will use this connection and gives these actions to SqlRunner. In addition, Server.java will take some informations such as user password from SqlRunner and push them to Client in proper way.

3.3 Models

Models are the java classes which are the main objects of the game. They will contain the main attributes of these objects.

3.3.1 Wall

Wall is the class which keeps informations such as coordinates, image url about walls which is one of the game objects.

3.3.2 Map

Map is the class which contains all the objects on the levels. Map class holds the objects in an array, or arraylist and adjusts their positions. It also checks the solution everytime player moves a wall. It checks whether player tries to place the wall in allowed position, and checks collisions. It creates the random maps for players and interacts with controller class to display them.

3.3.3 User

User class is the user information storage class. In this class, we will have

user ids, usernames, passwords and user scores for each game modes. Objects of this class will be our users.

3.3.4 Song

Song objects will keep information about the music id's and URL's for each music.

3.3.5 WhiteKnight

WhiteKnight is the class which keeps information of white knights. It keeps the the locations and image url of white knights as an information.

3.3.5 BlueKnight

BlueKnight is the class which keeps informations of blue knights. It keeps the the locations and image url of blue knights as an information.

3.3.6 RedKnight

RedKnight is the class which keeps informations of red knights. It keeps the the locations and image url of red knights as an information.

3.4 Database

Our database contains a few tables for user, high scores and musics. In the user table, all of the information about users are kept. For the highscores of each game mode, we have a different table. In these tables, users with their scores are stored according to their ids as foreign keys. In the music table, musics with their urls are stored. We will use SQL runner code to make an interface to handle our operations easily. We will manipulate the data inside the database and get the objects from our classes with the help of this interface.

4. Low-level design

4.1) Object design trade-offs

Understandability

Since our application is easy to play, the usability should be easy too. Another reason is that users should not be bored before playing. For that, we kept the UI and rules simple. The user just launches the game and can play directly. This simple usage also provides increase in enjoyment. Besides, navigation through pages is also simple. Once user wants to go to a page, there is a simple path to that page from the main menu page.

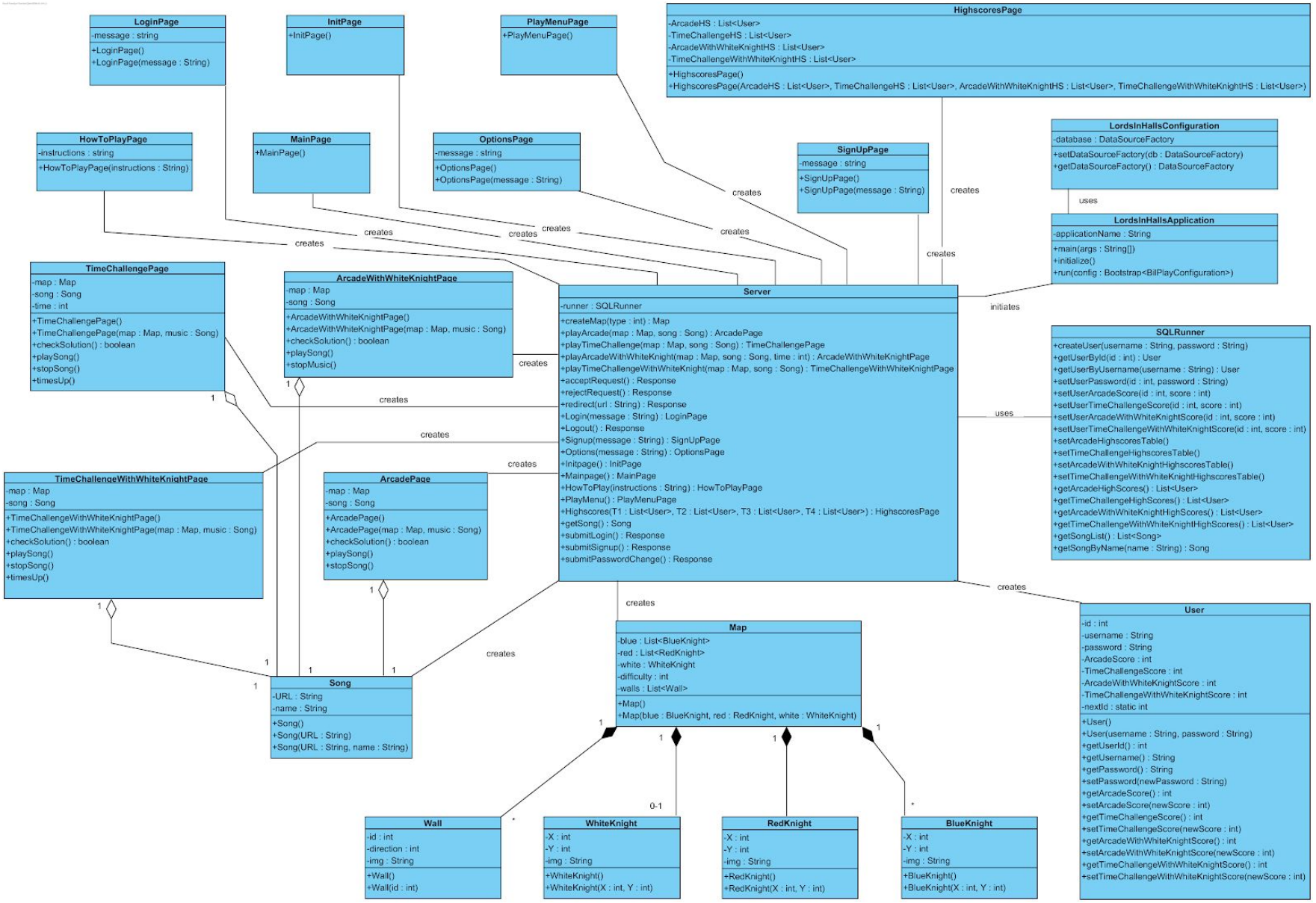
Maintainability

Since our memory usage is very small, it provides the chance to upgrade our game frequently. We, in the beginning, decide low level graphics and that feature increased the maintainability. Whenever we want to upgrade our game, the deletions and downloads will not be costly. Additionally, the small amount of data and low usage of CPU makes the game accessible for every system(very high to very low). Increasing access makes our application maintainable.

Development Time vs User Experience

During the analysis period, we tried to enhance the user experience of our application as much as we can. This actually caused some replication which means some more extra coding for us but that is okay as long as the users are positively affected by it. For instance, provided gameplay modes are quite similar to each other in terms of rules and gameplay. Just for us developers to be sure that users have different playstyles even though difference is small. In that sense, our application is quite maintainable. In appearance, the codes are long, but indeed they are similar to one another thus, easy to change.

4.2) Final object design


LoginPage
-message : string
+LoginPage()
+LoginPage(message : String)
InitPage
+InitPage()
PlayMenuPage
+PlayMenuPage()
HighscoresPage
-ArcadeHS : List<User>
-TimeChallengeHS : List<User>
-ArcadeWithWhiteKnightHS : List<User>
-TimeChallengeWithWhiteKnightHS : List<User>
+HighscoresPage()
+HighscoresPage(ArcadeHS : List<User>, TimeChallengeHS : List<User>, ArcadeWithWhiteKnightHS : List<User>, TimeChallengeWithWhiteKnightHS : List<User>)
HowToPlayPage
-instructions : string
+HowToPlayPage(instructions : String)
MainPage
+MainPage()
OptionsPage
-message : string
+OptionsPage()
+OptionsPage(message : String)
SignUpPage
-message : string
+SignUpPage()
+SignUpPage(message : String)
LordsInHallsConfiguration
-database : DataSourceFactory
+setDataSourceFactory(db : DataSourceFactory)
+getDataSourceFactory() : DataSourceFactory
LordsInHallsApplication
-applicationName : String
+main(args : String[])
+initialize()
+run(config : Bootstrap<BllPlayConfiguration>)
TimeChallengePage
-map : Map
-song : Song
-time : int
+TimeChallengePage()
+TimeChallengePage(map : Map, music : Song)
+checkSolution() : boolean
+playSong()
+stopSong()
+timesUp()
ArcadeWithWhiteKnightPage
-map : Map
-song : Song
+ArcadeWithWhiteKnightPage()
+ArcadeWithWhiteKnightPage(map : Map, music : Song)
+checkSolution() : boolean
+playSong()
+stopMusic()
Server
-runner : SQLRunner
+createMap(type : int) : Map
+playArcade(map : Map, song : Song) : ArcadePage
+playTimeChallenge(map : Map, song : Song) : TimeChallengePage
+playArcadeWithWhiteKnight(map : Map, song : Song, time : int) : ArcadeWithWhiteKnightPage
+playTimeChallengeWithWhiteKnight(map : Map, song : Song) : TimeChallengeWithWhiteKnightPage
+acceptRequest() : Response
+rejectRequest() : Response
+redirect(url : String) : Response
+Login(message : String) : LoginPage
+Logout() : Response
+Signup(message : String) : SignUpPage
+Options(message : String) : OptionsPage
+Initpage() : InitPage
+Mainpage() : MainPage
+HowToPlay(instructions : String) : HowToPlayPage
+PlayMenu() : PlayMenuPage
+Highscores(T1 : List<User>, T2 : List<User>, T3 : List<User>, T4 : List<User>) : HighscoresPage
+getSong() : Song
+submitLogin() : Response
+submitSignup() : Response
+submitPasswordChange() : Response
SQLRunner
+createUser(username : String, password : String)
+getUserById(id : int) : User
+getUserByUsername(username : String) : User
+setUserPassword(id : int, password : String)
+setUserArcadeScore(id : int, score : int)
+setUserTimeChallengeScore(id : int, score : int)
+setUserArcadeWithWhiteKnightScore(id : int, score : int)
+setUserTimeChallengeWithWhiteKnightScore(id : int, score : int)
+setArcadeHighscoresTable()
+setTimeChallengeHighscoresTable()
+setArcadeWithWhiteKnightHighscoresTable()
+setTimeChallengeWithWhiteKnightHighscoresTable()
+getArcadeHighScores() : List<User>
+getTimeChallengeHighScores() : List<User>
+getArcadeWithWhiteKnightHighScores() : List<User>
+getTimeChallengeWithWhiteKnightHighScores() : List<User>
+getSongList() : List<Song>
+getSongByName(name : String) : Song
TimeChallengeWithWhiteKnightPage
-map : Map
-song : Song
+TimeChallengeWithWhiteKnightPage()
+TimeChallengeWithWhiteKnightPage(map : Map, music : Song)
+checkSolution() : boolean
+playSong()
+stopSong()
+timesUp()
ArcadePage
-map : Map
-song : Song
+ArcadePage()
+ArcadePage(map : Map, music : Song)
+checkSolution() : boolean
+playSong()
+stopSong()
Map
-blue : List<BlueKnight>
-red : List<RedKnight>
-white : WhiteKnight
-difficulty : int
-walls : List<Wall>
+Map()
+Map(blue : BlueKnight, red : RedKnight, white : WhiteKnight)
User
-id : int
-username : String
-password : String
-ArcadeScore : int
-TimeChallengeScore : int
-ArcadeWithWhiteKnightScore : int
-TimeChallengeWithWhiteKnightScore : int
-nextId : static int
+User()
+User(username : String, password : String)
+getUserId() : int
+getUsername() : String
+getPassword() : String
+setPassword(newPassword : String)
+getArcadeScore() : int
+setArcadeScore(newScore : int)
+getTimeChallengeScore() : int
+setTimeChallengeScore(newScore : int)
+getArcadeWithWhiteKnightScore() : int
+setArcadeWithWhiteKnightScore(newScore : int)
+getTimeChallengeWithWhiteKnightScore() : int
+setTimeChallengeWithWhiteKnightScore(newScore : int)
Song
-URL : String
-name : String
+Song()
+Song(URL : String)
+Song(URL : String, name : String)
Wall
-id : int
-direction : int
-img : String
+Wall()
+Wall(id : int)
WhiteKnight
-X : int
-Y : int
-img : String
+WhiteKnight()
+WhiteKnight(X : int, Y : int)
RedKnight
-X : int
-Y : int
-img : String
+RedKnight()
+RedKnight(X : int, Y : int)
BlueKnight
-X : int
-Y : int
-img : String
+BlueKnight()
+BlueKnight(X : int, Y : int)
creates
uses
initiates
1
0-1
*

4.3) Packages

4.3.1) java.util [1]

In our LordsInHalls application, we will mainly need list objects to store and send informations about game pieces such as Map, Wall etc. Java.util package will grant us to use these list objects.

4.3.2) java.net.URI [1]

To represent the URLs, we will need this package and its content.

4.3.3) javax.ws.rs & javax.ws.rs.core [1]

To represent the Response class, javax.ws.rs and javax.ws.rs.core will be needed.

4.4) Class interfaces

4.4.1) LoginPage

private string message: This string is for reporting the unwanted events like having wrong id, password and changing password wrongly.

4.4.2) HowToPlayPage

private String instructions: This string instance contains the instructions which describes the game rules and how to play the game.

4.4.3) OptionsPage

private String message: This string instance contains the possible messages to displayed to the user in the Options page. Such messages could be, error messages which implies the fault change password, or can be success message that says the password change request is successful.

4.4.4) HighScoresPage

private ArcadeHS List<User>: Lists contains the users with scores for ArcadeHS mode.

private TimeChallengeHS<User>: Lists contains the users with scores for TimeChallengeHS mode.

private ArcadeWithWhiteKnightHS List<User>: Lists contains the users with scores for ArcadeWithWhiteKnightHS mode.

private TimeChallengeWithWhiteKnightHS List<User>: Lists contains the users with scores for TimeChallengeWithWhiteKnightHS mode.

4.4.5) SignUpPage

String message: to report the wrong sign up trials.

4.4.6) TimeChallangePage

private Map map: The map in the time challenge mode.

private Song song:This is the song that plays at the background.

private int time: This is to show the remaining time since it is time challenge mode.

4.4.7) ArcadeWithWhiteKnightPage

private Map map: This is a variable of ArcadeWithWhiteKnightPage class that keeps map informations to play.

private Song song:The song to play.

4.4.8) TimeChallangeWithWhiteKnightPage

private Map map: contains the map features for the mode.

private Song song: the song that will play while playing this mode.

4.4.9) ArcadePage

private Map map: contains the map features for the mode.

private Song song: the song that will play while playing this mode.

4.4.10) Song

String URL: This string instance contains the URL address of the song to be played.

4.4.11) Map

private List<BlueKnight> blue: This BlueKnight list contains the list of Blue Knight objects, which will be used for creating map.

private List<RedKnight> blue: This RedKnight list contains the list of Red Knight objects, which will be used for creating map.

private List<Walls> walls: This wall list contains the all walls in the map.

private WhiteKnight white: This White Knight instance contains a white knight inside a map.

private int difficulty: The instance of int, which represents the difficulty of the level. The difficulty of the map will be determined when the level is created. Difficulty of the level will be related to the number of possible solutions of the level and the positioning of the walls.

4.4.12) Wall

private int id: contains the id of the unique wall type.

private int direction: will keep the direction of wall.

private String img: This string will keep url of image that will be used in user interface.

4.4.13) WhiteKnight

private int X: contains the x-location of WhiteKnight

private int Y: contains the y-location of WhiteKnight

private String img: contains the image of WhiteKnight

4.4.14) RedKnight

private int X: keeps the X-location of RedKnight.

private int Y: keeps the Y-location of RedKnight.

private String img: keeps the url of image of RedKnight.

4.4.15) BlueKnight

private int X: contains the x-location of BlueKnight

private int Y: contains the y-location of BlueKnight

private String img: contains the image of BlueKnight

4.4.16) User

int id: The user id’s are unique int instances.

String username: Usernames are unique string instances.

int ArcadeScore: Contains the score achieved in the Arcade mode.

int TimeChallengeScore: Contains the score achieved in the Time Challenge mode.

int ArcadeWithWhiteKnightScore: Contains the score achieved in the Arcade With White Knights mode.

int TimeChallengeWithWhiteKnightScore: Contains the score achieved in the Time Challenge with White Knights mode.

static int nextId: Contains the id of the next User. In another words, contain the number of users created.

4.4.17) SQLRunner

This class is only an interface which is consisted of SQL commands.

public void createUser(username : String, password : String): Creates a user in the database with given username and password form parameters.

public User getUserById(id : int): Returns the user with the given ID parameter.

public User getUserByUsername(username : String): Returns the user with the given username.

public void setUserPassword(id : int, password : String): Changes the password of the user denoted by id to the given password.

public void setUserArcadeScore(id : int, score : int): Sets the Arcade mode score of the user given by id.

public void setUserTimeChallengeScore(id : int, score : int): Sets the TimeChallenge mode score of the user given by id.

public void setUserArcadeWithWhiteKnightScore(id : int, score : int): Sets the ArcadeWithWhiteKnight mode score of the user given by id.

public void setUserTimeChallengeWithWhiteKnightScore(id : int, score : int): Sets the TimeChallengeWithWhiteKnight mode score of the user given by id.

public void setArcadeHighscoresTable(), setTimeChallengeHighscoresTable(), setArcadeWithWhiteKnightHighscoresTable(), setTimeChallengeWithWhiteKnightHighscoresTable: Sets up the related highscores table according to the current scores of all users.

public List<User> getArcadeHighScores(), getTimeChallengeHighScores(), getArcadeWithWhiteKnightHighScores(), getTimeChallengeWithWhiteKnightHighScores(): Returns the related highscores table as a list of users.

public List<Song> getSongList(): Returns all the songs in database.

public Song getSongByName(): Returns the song denoted by name.

4.4.18) Server

SQLRunner runner: Non-initialized instance of an SQLRunner interface. Will be used to run SQL Commands.

public Map createMap(type: int): Creates a random map with requested type(with a White Knight or without a White Knight) with Knights in positions. Then returns it as a Map model.

public ArcadePage playArcade(map : Map, song : Song): Instantiates ArcadePage.

public TimeChallengePage playTimeChallenge(map : Map, song : Song) : Instantiates TimeChallengePage.

public ArcadeWithWhiteKnightPage playArcadeWithWhiteKnight(map : Map, song : Song, time : int): Instantiates ArcadeWithWhiteKnightPage.

public TimeChallengeWithWhiteKnight playTimeChallengeWithWhiteKnight(map : Map, song : Song): Instantiates TimeChallengeWithWhiteKnightPage.

private static Response acceptRequest(): returns accepted response.

private static Response rejectRequest(): returns rejected response.

private static Response redirect(): redirects to the given URL.

public LoginPage Login(message : String): Instantiates LoginPage.

public Response Logout(): Clear cookies and logs out from current session.

public SignUpPage Signup(message : String): Instantiates SignupPage.

public OptionsPage Options(message : String): Instantiates OptionsPage.

public InitPageInitpage(): Instantiates InitPage.

public MainPage Mainpage(): Instantiates MainPage.

public HowToPlayPage HowToPlay(instructions : String): Instantiates HowToPlay page with given instructions.

public PlayMenuPage PlayMenu(): Instantiates PlayMenuPage.

public HighscoresPage Highscores(T1 : List<User>, T2 : List<User>, T3 : List<User>, T4 : List<User>): Instantiates HighscoresPage.

public Song getSong(): Gets the songs from database and returns one of them randomly as a Song object.

public Response submitLogin(): Compares the given id-password and the id-password in the database and creates a response for that.

public Response submitSignUp(): Checks the written id and password with the other ids and passwords in the database and also checks situations like password length.

public Response submitPasswordChange(): Compares the two passwords for changing password.

4.4.19) LordsInHallsApplication

This class contains the main method, which will initiate the server and configuration instances.

4.4.20) LordsInHallsConfiguration

This class connects to the DataSourceFactory to access database inside it.

5. Glossary & References

[1] “How can we help?,” *Oracle Help Center*. [Online]. Available: https://docs.oracle.com/en/. [Accessed: 07-Nov-2018].