DOCUMENTATION

INTRODUCTION

This project was implemented within the study of the subject *Advanced programming methods* which focuses on the analysis of the Java programming language.

I chose project number A.20 which requires the implementation of the *Towers of Hanoi* game using Java (and other additional tools).

I. MOTIVATION

I chose this project because I already implement this game in C++, using the SFML library, and I wanted to make a more complex, clean version that would highlight my progress as a programmer.

Here is a video of the old project: Towers of Hanoi - with C++ and SFML. This project does not implement the logging system using a database and focuses strictly on the game, it is much more simplistic.

The new project uses animations, execution threads and also implements the logging system. In addition, the automatic mode in which the computer solves the game is also implemented.

II. THE THEMATIC CONTEXT OF THE PROBLEM

As I said, this project was implemented within the study of the subject Advanced Programming Methods . The project was assigned to students, to show their gained knowledge related to the Java programming language. It was mandatory to use a database and connect it to the graphical interface, and my project also required the use of execution threads for windows and animations.

PROBLEM DESCRIPTION

I. THE TASK

My project requirement is as follows:

Write a Java application that visualizes the Towers of Hanoi problem in a suggestive way.

The application should run in two modes: manual mode – in which the user moves the discs on the rods; and in automatic mode – where the computer solves the problem.

- Put the algorithm in automatic mode in a thread so as not to block the interface.
- Provide for manual/automatic mode a counter that measures the time required to the user to resolve the issue.
- Implement the possibility of abandonment.
- Allow specifying the number of disks.
- Insist on the graphic representation, to be as suggestive as possible. Set the working speed in the automatic mode appropriate to human perception. Let every movement made be easily noticeable. Show the disc

moved and the path taken during the movement. Use motion animation with execution threads.

II. TOWERS OF HANOI GAME

But what is the game “Towers of Hanoi” actually about?

The Tower of Hanoi (also called The problem of Benares Temple or Tower of Brahma or Lucas' Tower and sometimes pluralized as Towers, or simply pyramid puzzle) is a mathematical game or puzzle consisting of three rods and a number of disks of various diameters, which can slide onto any rod. The puzzle begins with the disks stacked on one rod in order of decreasing size, the smallest at the top, thus approximating a conical shape. The objective of the puzzle is to move the entire stack to the last rod, obeying the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No disk may be placed on top of a disk that is smaller than it.

With 3 disks, the puzzle can be solved in 7 moves. The minimal number of moves required to solve a Tower of Hanoi puzzle is $\mathbf{2^n - 1}$, where n is the number of disks.

The puzzle can be played with any number of disks, although many toy versions have around 7 to 9 of them. The minimal number of moves required to solve a Tower of Hanoi puzzle is $\mathbf{2^n - 1}$, where n is the number of disks.

With 3 disks, the puzzle can be solved in $\mathbf{2^3 - 1 = 8 - 1 = 7}$ moves.

MY SOLUTION

I. THE MAIN CONCEPTS

The application consists of a graphic interface and a database that are connected.

I chose to develope the game in the Intellij IDE using JavaFX and Maven. The code used for the ghrafic interface respects the MVC design pattern. Every stage consists of a JavaFX class, controller class and a .fxml file where the design is done. In some cases I also used CSS for styling.

The application has a menu window on which the user selects what he wants to do by clicking one of the following buttons:

- Play – play the game (manual game)
- Tutorial – see computer's solution (automatic game)
- Options (change game information)
- Statistics – see statistics about the best performers
- Settings – change account settings
- Quit – close the application

The application has a basic sign up/login system implemented in a class dedicated for this task. All the data is managed using a relational database called: towers-of-

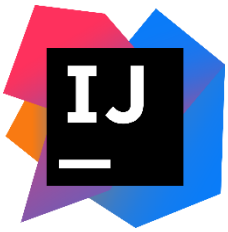
Hanoi. (I also implemented a way to store the data in a NoSQL database (MongoDB))

The application is connected to my database which was constructed in MySQL using the Java JDBC API and the MySQL Connector/J driver.

All coding operations were done under the „supervision" of a version control system: git. My open source repository can be found on my GitHub profile: towers-of-Hanoi.

II. TOOLS USED

Tool 4 Java programming language



Tool 6 Intellij IDE

Tool 5 Maven

Tool 2 MySQL database

Tool 1 CSS

Tool 3 Git VCS

Tool 7 GitHub

THE PROJECT

I. THE APPLICATION

I have a video demo that presents the whole application posted on YouTube (Unlisted mode): This link is also found in the README.md file from my GitHub repository.

II. THE ARCHITECTURE

1. THE GRAPHIC INTERFACE

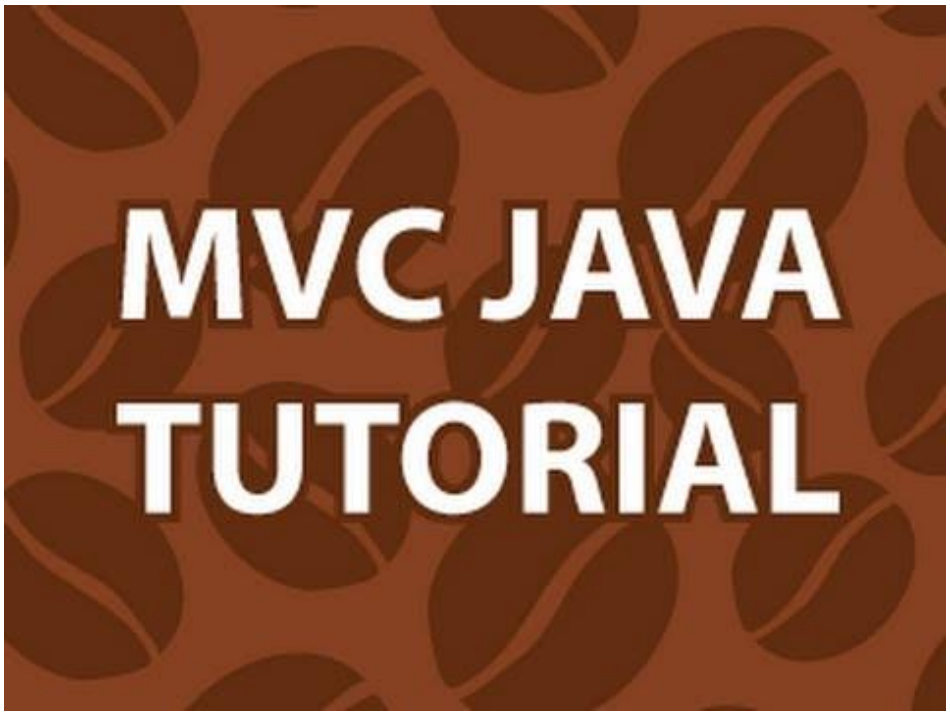
JavaFX applications are designed to respect the MVC architecture.

In JavaFX, the Model-View-Controller (MVC) architectural pattern can be implemented to separate the concerns of data management (Model), user interface (View), and user input handling and event management (Controller). Here's a brief explanation of how MVC can be implemented in JavaFX:

1. Model: The Model represents the data and business logic of the application. It encapsulates the application state and provides methods to manipulate and access the data. In JavaFX, you can create your own classes to serve as models or use existing JavaFX properties and collections to represent your data.
2. View: The View is responsible for presenting the data to the user and handling the visual aspects of the application. In JavaFX, you typically use FXML (an XML-based markup language) along with CSS to define the user interface components and their layout.
3. Controller: The Controller acts as an intermediary between the Model and the View. It handles user input events, updates the Model based on user actions, and updates the View to reflect the changes in the Model. In

JavaFX, the Controller class is usually associated with an FXML file and can be defined using the `@FXML` annotation.

I used this approach:



Video 1 MVC

The local git repository looks like this:

Name	Date modified	Type	Size
.git	6/18/2023 2:46 PM	File folder	
.idea	6/18/2023 2:46 PM	File folder	
.mvn	5/15/2023 4:35 PM	File folder	
doc	6/18/2023 1:59 PM	File folder	
src	6/18/2023 2:37 PM	File folder	
target	5/15/2023 4:38 PM	File folder	
.gitignore	5/15/2023 4:35 PM	Git Ignore Source ...	1 KB
mvnw	5/15/2023 4:35 PM	File	11 KB
mvnw.cmd	5/15/2023 4:35 PM	Windows Comma...	7 KB
pom.xml	6/13/2023 5:27 PM	XML Document	6 KB
README.md	5/15/2023 4:38 PM	Markdown Source...	0 KB

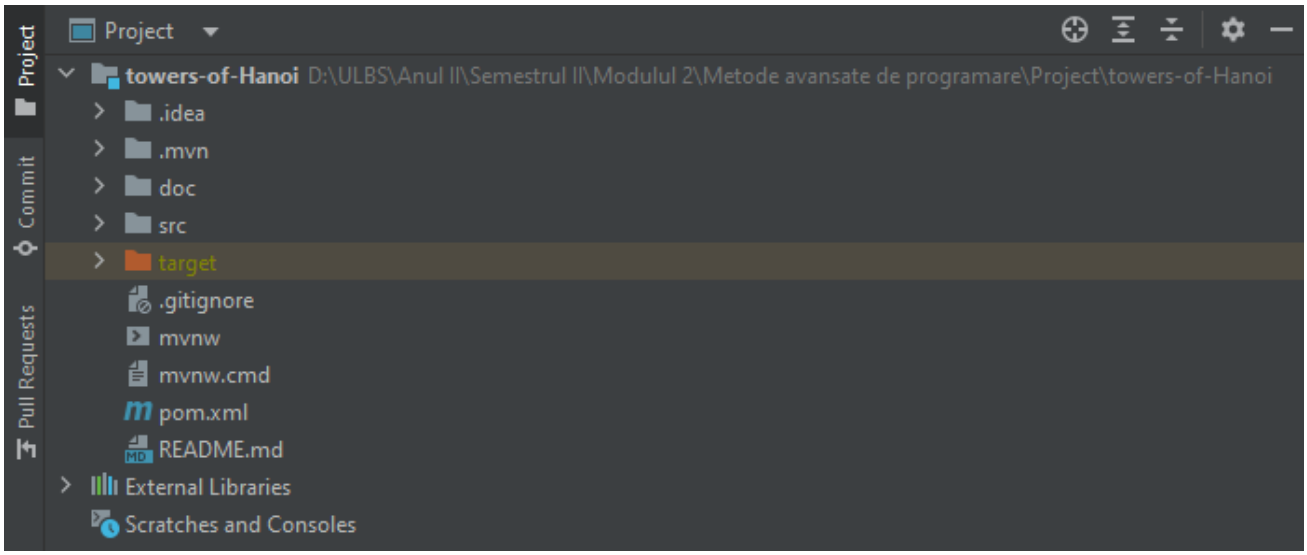
Screenshot 1 git repository

- .git directory contains all information required for version control
- .idea directory contains a set of configuration files (.xml) for the project.
- .mvn is a directory where you can put some Maven configuration files
- doc directory contains the documentation of the project in .docx and .pdf format
- src directory is the most complex and is the root directory for both application code. The application source code goes into src/main/java. Any resource files (e.g. property files) needed by the application goes into the src/main/resources directories. Apart from the .fxml files, I only have one .css file, one image for the background and one .wav file for the background music. I also added at the root level of the directory a subfolder called database where I have the files used to create the databases and some query files.
- target directory contains all the final products that are the result of Maven building the project. The target directory also contain any temporary and intermediate files needed by Maven when building the application.
- .gitignore file specifies intentionally untracked files that Git should ignore.
- mvnw/mvnw & mvn.cmd files are known as maven wrappers. These files let users run maven builds without installing a

maven distribution in your machine. These files allow the users to have a fully encapsulated build system.

- `pom.xml` is the Maven POM file (Project Object Model). It is an XML file that contains information about the project and configuration details used by Maven to build the project.
- `README.md` file is a Markdown file that describes the git repository. GitHub and Gitiles renders it when you browse the repository.

This structure can also be seen in Intellij:



Screenshot 2 Intellij project

2. THE DATABASE

The database respects the 3FN conditions.

What is 3FN? Third normal form (3NF) is a database schema design approach for relational databases which uses normalizing principles to reduce the duplication of data, avoid data anomalies, ensure referential integrity, and simplify data management. It was defined in 1971 by Edgar F. Codd, an English computer scientist who invented the relational model for database management.

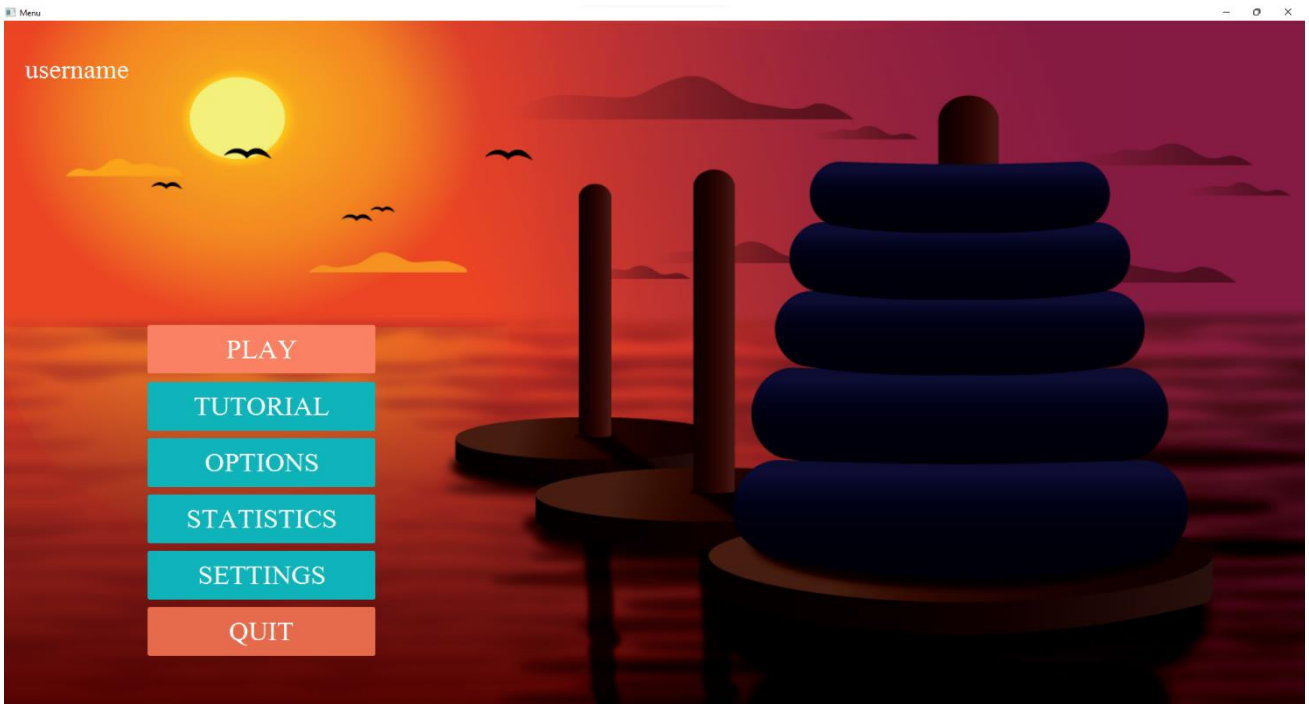
A database relation (e.g. a database table) is said to meet third normal form standards if all the attributes (e.g. database columns) are functionally dependent on solely the primary key. Codd defined this as a relation in second normal form where all non-prime attributes depend only on the candidate keys and do not have a transitive dependency on another key.

My database has 2 tables that meet the requirements:

- They both have a primary key
- The table columns are functionally dependent on solely the primary key.

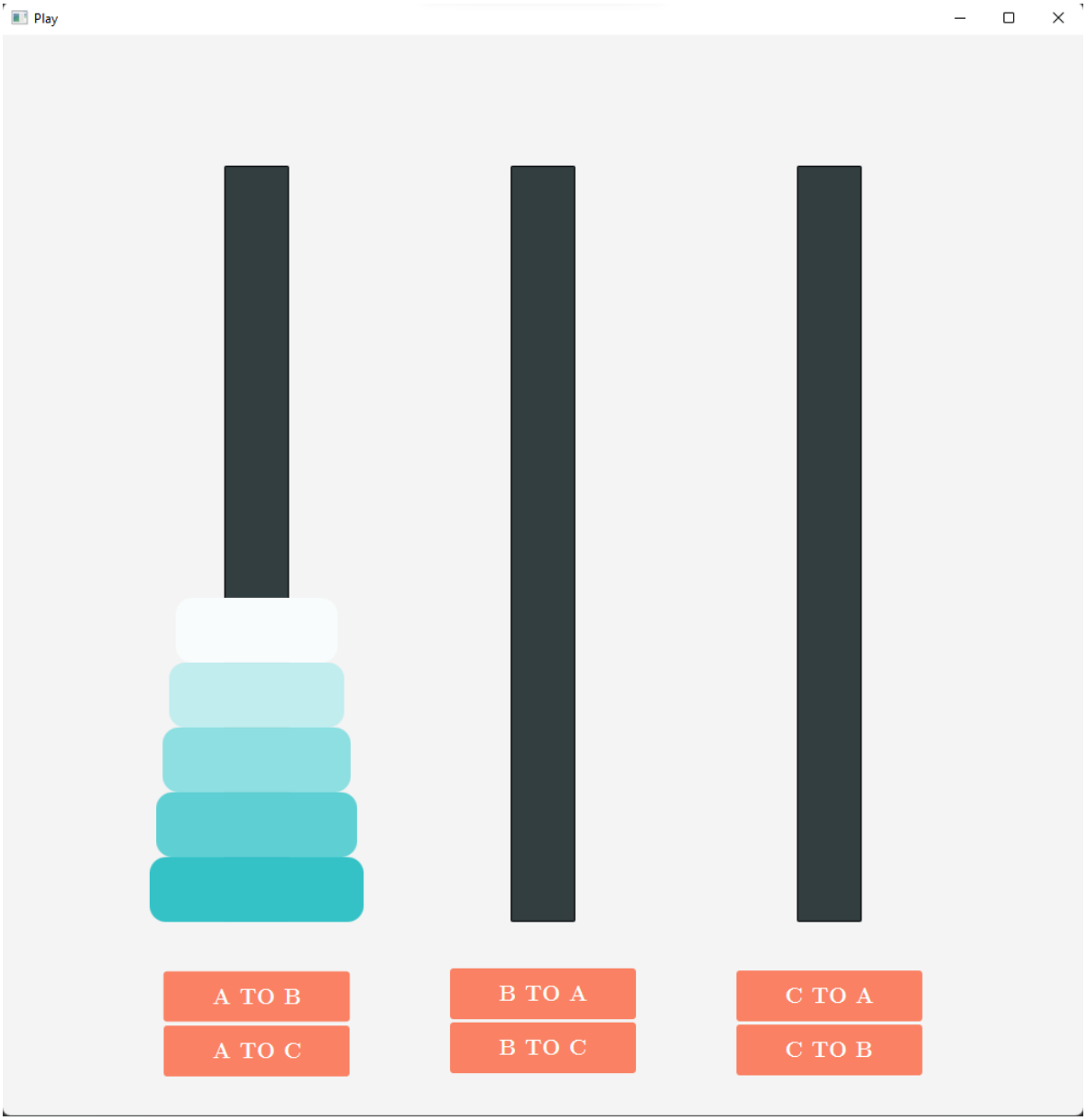
III. STRUCTURE

The graphic interface contains a main menu where the user can decide what to do: he can play but he can also follow a tutorial (computer solution).

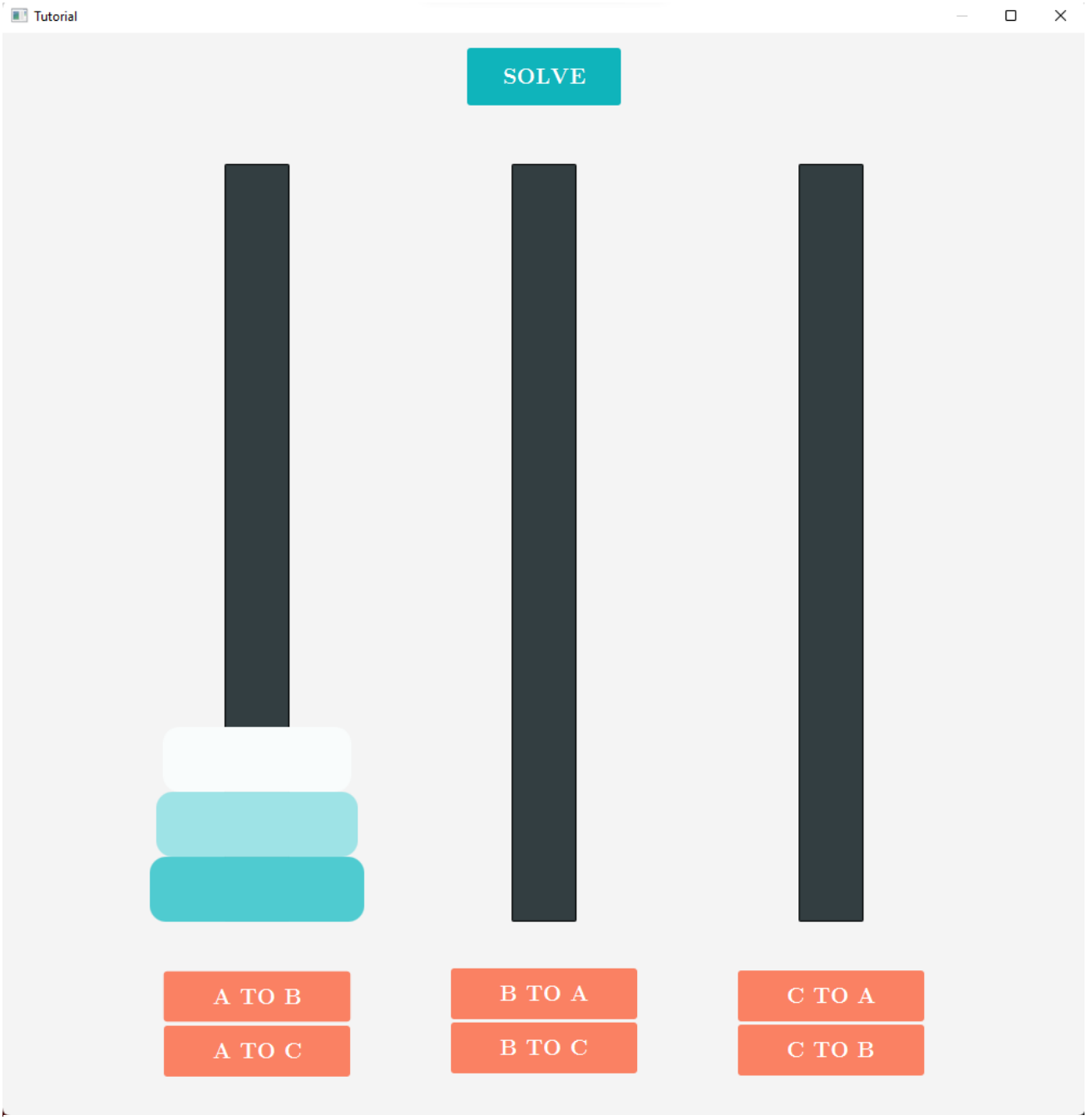


Graphic Interface - Window 1 Menu

The "Play" window and "Tutorial" window are almost identical. The only difference is that the "Tutorial" window contains a button with the text: "Solve". When the user clicks the button, the computer starts to solve the game.

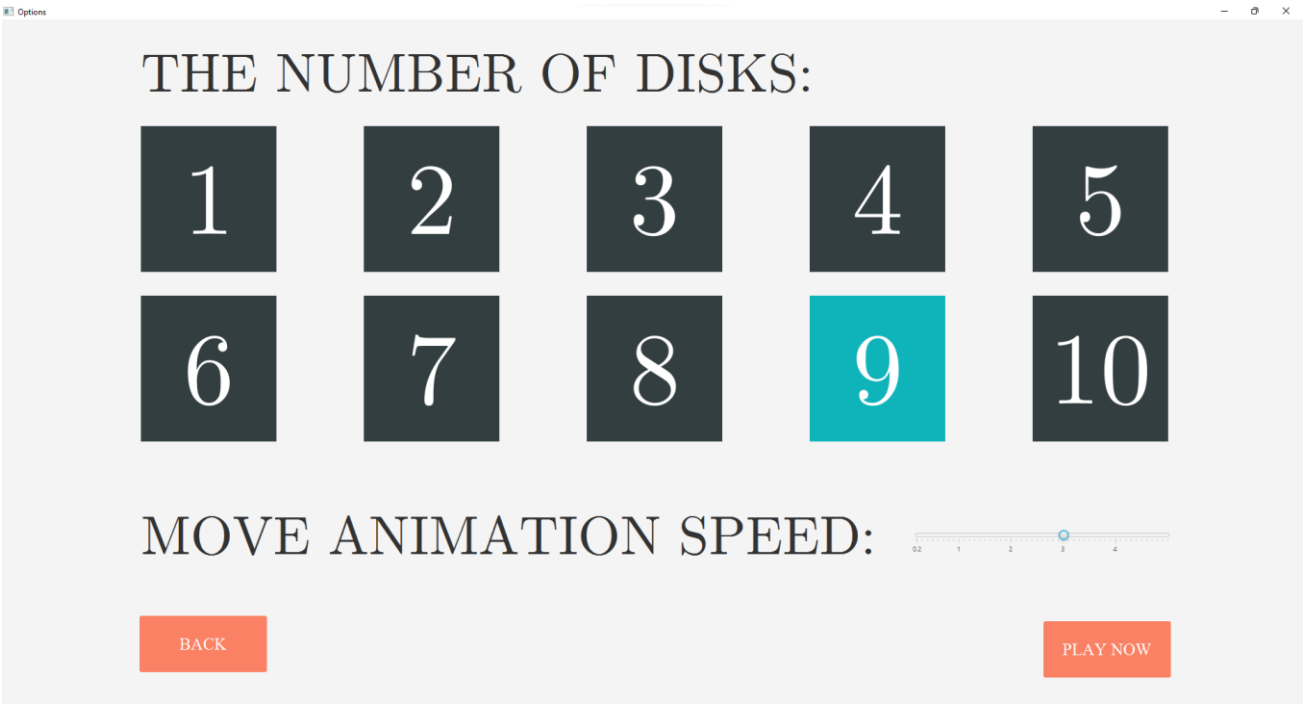


Graphic Interface - Window 2Play



Graphic Interface - Window 3Tutorial

The user can also change the game settings in the "Options window.

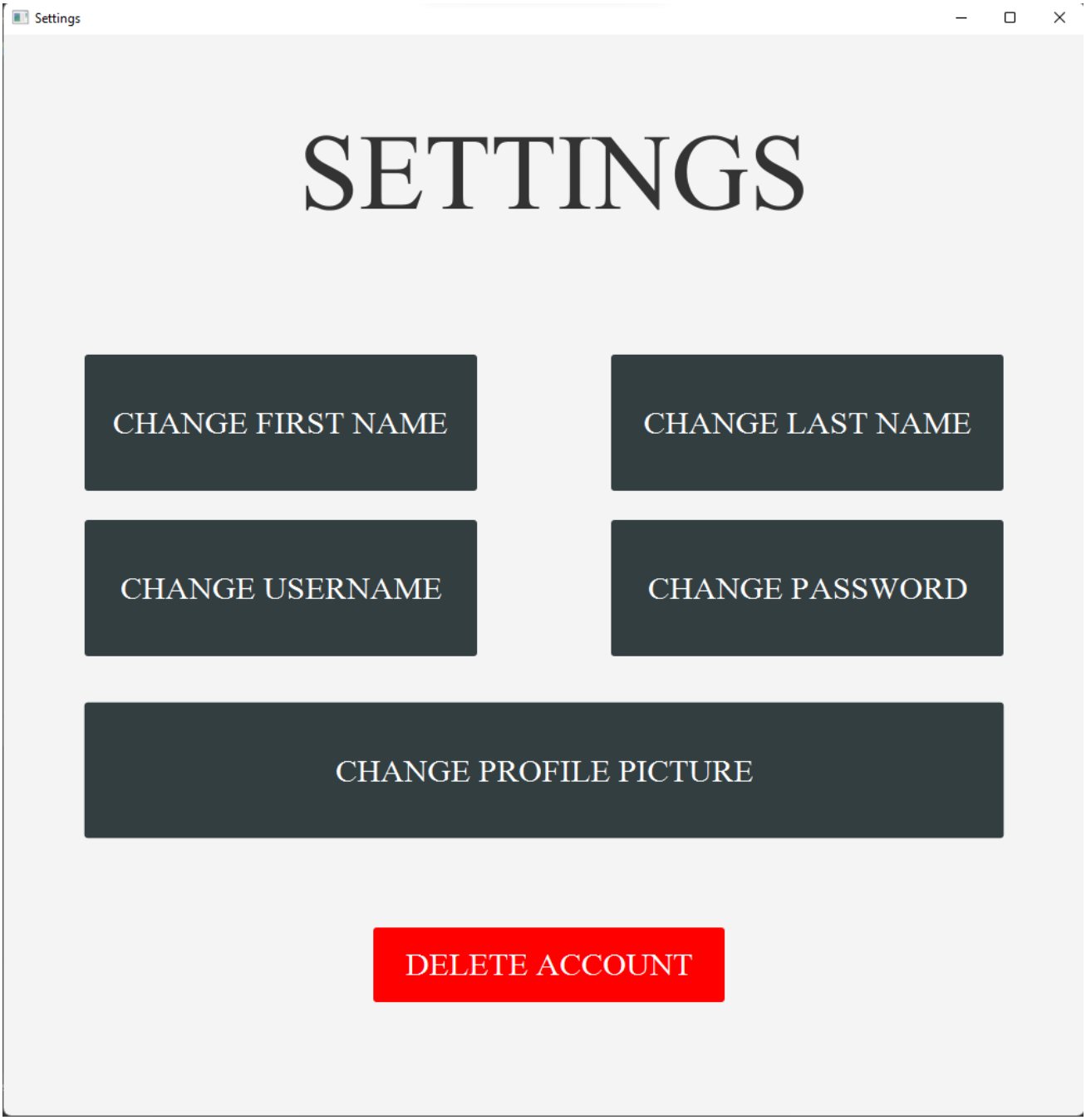


Graphic Interface - Window 4 Options

If the user wants to see who are the best players, he can access the Statistics window where a table with the best players is displayed. (The statistics of the game played are also shown after the game is finished.)

Graphic Interface - Window 5Statistics

Last but not least, we also have the "Settings window where the user can change the information related to the account.



Graphic Interface - Window 6Statistics

IV. DATA MANAGEMENT

All the necessary data is stored in the towers-of-Hanoi database which contains 5 tables: users , statistics , achievements , notifications and complaints . But only the tables "users" and "statistics" are used. The other tables were built to be used when the application will be more complex.

Here is the ERD:

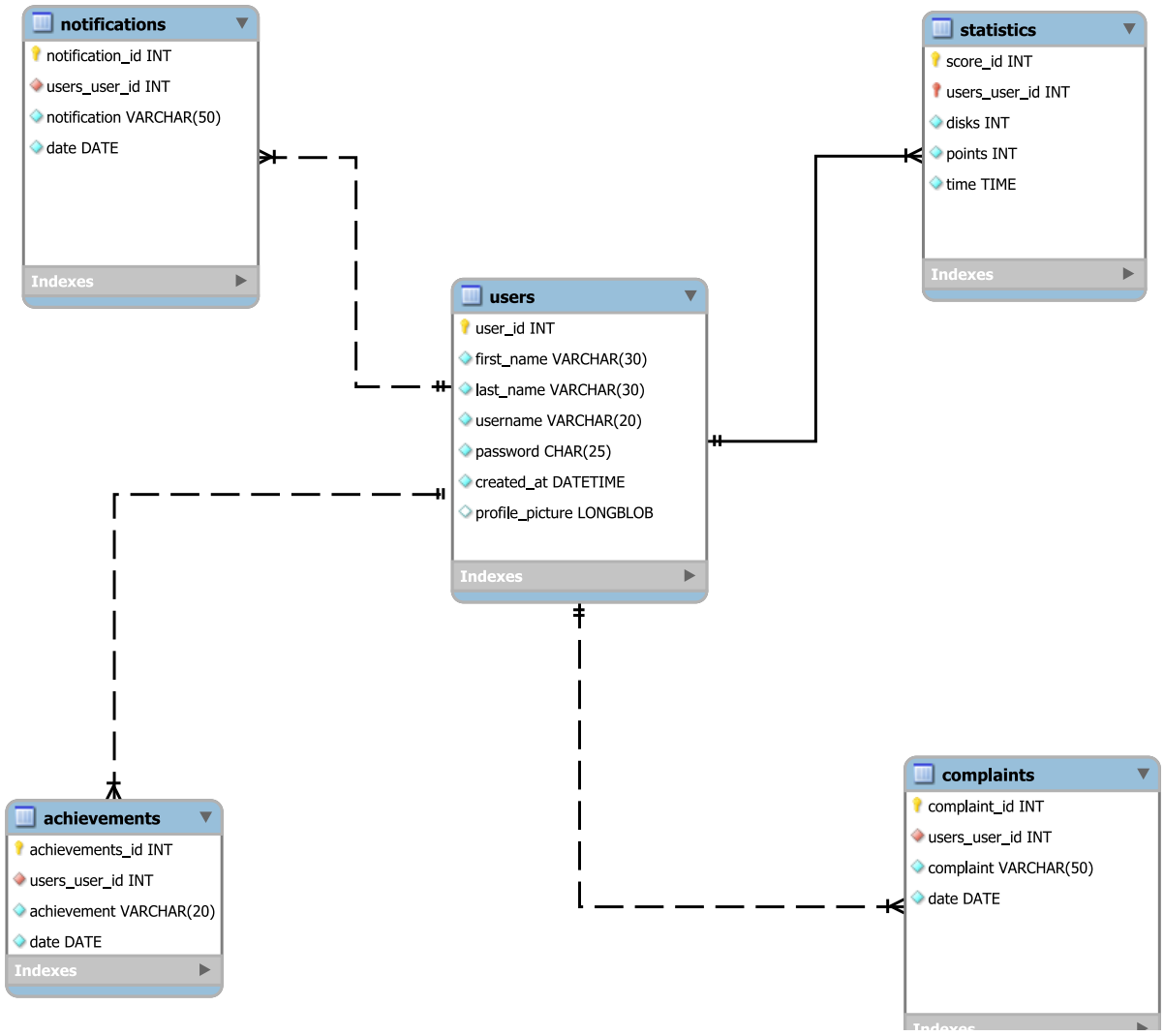


Figure 1 ERD

The generating script for this 2 tables was generated by the entity relationship diagram created in MySQL workbench using forward engineering. All the files used can be found in the "src/database/mysql" directory.

And here is the generating script:

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_
FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema towers-of-Hanoi
-- -----------------------------------------------------
-- This database contains tables used to store user data and other information
related to the classic game "Towers of Hanoi" made by myself using JavaFX. The
database will be connected to Java using the MySQL Connector/J driver.

-- -----------------------------------------------------
-- Schema towers-of-Hanoi
--
-- This database contains tables used to store user data and other information
related to the classic game "Towers of Hanoi" made by myself using JavaFX. The
database will be connected to Java using the MySQL Connector/J driver.
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `towers-of-Hanoi` DEFAULT CHARACTER SET utf8 COLLATE
utf8_bin ;
USE `towers-of-Hanoi` ;

-- -----------------------------------------------------
-- Table `towers-of-Hanoi`.`users`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `towers-of-Hanoi`.`users` (
  `user_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT 'This table stores user
information.',
  `first_name` VARCHAR(30) NOT NULL,
  `last_name` VARCHAR(30) NOT NULL,
  `username` VARCHAR(20) NOT NULL,
  `password` CHAR(25) NOT NULL,
```

```
  `created_at` DATETIME NOT NULL COMMENT 'This table contains the basic columns used for the login process. The table could be updated so that the password is salted and hashed to assure security.',
  `profile_picture` LONGBLOB NULL,
  PRIMARY KEY (`user_id`),
  UNIQUE INDEX `user_id_UNIQUE` (`user_id` ASC) VISIBLE,
  UNIQUE INDEX `username_UNIQUE` (`username` ASC) VISIBLE,
  UNIQUE INDEX `password_UNIQUE` (`password` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `towers-of-Hanoi`.`statistics`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `towers-of-Hanoi`.`statistics` (
  `score_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT 'This table stores some data about user games.',
  `users_user_id` INT UNSIGNED NOT NULL,
  `disks` INT UNSIGNED NOT NULL COMMENT 'This table is used to store the user\'s game statistics.',
  `points` INT UNSIGNED NOT NULL,
  `time` TIME NOT NULL,
  PRIMARY KEY (`score_id`, `users_user_id`),
  UNIQUE INDEX `score_id_UNIQUE` (`score_id` ASC) VISIBLE,
  INDEX `fk_statistics_users_idx` (`users_user_id` ASC) VISIBLE,
  CONSTRAINT `fk_statistics_users`
    FOREIGN KEY (`users_user_id`)
    REFERENCES `towers-of-Hanoi`.`users` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `towers-of-Hanoi`.`complaints`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `towers-of-Hanoi`.`complaints` (
  `complaint_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT 'This table can store information about individual game sessions, including the user who played the game, the start and end times of the session, and the duration of the session.',
  `users_user_id` INT UNSIGNED NOT NULL,
  `complaint` VARCHAR(50) NOT NULL,
  `date` DATE NOT NULL,
  PRIMARY KEY (`complaint_id`),
  UNIQUE INDEX `session_id_UNIQUE` (`complaint_id` ASC) VISIBLE,
  INDEX `fk_games_sessions_users1_idx` (`users_user_id` ASC) VISIBLE,
  CONSTRAINT `fk_games_sessions_users1`
    FOREIGN KEY (`users_user_id`)
    REFERENCES `towers-of-Hanoi`.`users` (`user_id`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `towers-of-Hanoi`.`notifications`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `towers-of-Hanoi`.`notifications` (
  `notification_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT 'This table can be used to maintain a leaderboard of the top scores achieved by users in the game. It would store the user\'s ID, their score, the rank on the leaderboard, and the date when the score was achieved.',
  `users_user_id` INT UNSIGNED NOT NULL,
  `notification` VARCHAR(50) NOT NULL,
  `date` DATE NOT NULL,
  PRIMARY KEY (`notification_id`),
  UNIQUE INDEX `leaderboard_id_UNIQUE` (`notification_id` ASC) VISIBLE,
  INDEX `fk_leaderboard_users1_idx` (`users_user_id` ASC) VISIBLE,
  CONSTRAINT `fk_leaderboard_users1`
    FOREIGN KEY (`users_user_id`)
    REFERENCES `towers-of-Hanoi`.`users` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `towers-of-Hanoi`.`achievements`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `towers-of-Hanoi`.`achievements` (
  `achievements_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT 'This table can track the achievements earned by users while playing the game. It would store the user\'s ID, the name of the achievement, and the date when the achievement was earned.',
  `users_user_id` INT UNSIGNED NOT NULL,
  `achievement` VARCHAR(20) NOT NULL,
  `date` DATE NOT NULL,
  PRIMARY KEY (`achievements_id`),
  UNIQUE INDEX `achievements_id_UNIQUE` (`achievements_id` ASC) VISIBLE,
  INDEX `fk_achievements_users1_idx` (`users_user_id` ASC) VISIBLE,
  CONSTRAINT `fk_achievements_users1`
    FOREIGN KEY (`users_user_id`)
    REFERENCES `towers-of-Hanoi`.`users` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

The first 10 records were added to the tables manually (except for the "statistics" table) using the .sql scripts which can be found in the src/database/mysql/queries folder:

users.sql

```
INSERT INTO users (first_name, last_name, username, password, created_at) VALUES
("Samuel", "Buraga", "samuel", "samuel", NOW());
INSERT INTO users (first_name, last_name, username, password, created_at) VALUES
("David", "Buraga", "david", "david", NOW());
INSERT INTO users (first_name, last_name, username, password, created_at) VALUES
("Adrian", "Oprisor", "adioprisor", "adi", NOW());
INSERT INTO users (first_name, last_name, username, password, created_at) VALUES
("Ion", "Popescu", "ion", "ion", NOW());
INSERT INTO users (first_name, last_name, username, password, created_at) VALUES
("Vasile", "Afloarei", "vasile", "vasile", NOW());
INSERT INTO users (first_name, last_name, username, password, created_at) VALUES
("Andreea", "Amariei", "andreea", "andreea", NOW());
INSERT INTO users (first_name, last_name, username, password, created_at) VALUES
("Eliza", "Popescu", "eliza", "eliza", NOW());
INSERT INTO users (first_name, last_name, username, password, created_at) VALUES
("Tudor", "Miron", "tudor", "tudor", NOW());
INSERT INTO users (first_name, last_name, username, password, created_at) VALUES
("Darian", "Maria", "darian", "darian", NOW());
INSERT INTO users (first_name, last_name, username, password, created_at) VALUES
("Emi", "Munteanu", "emi", "emi", NOW());
```

achievements.sql

```
INSERT INTO achievements (achievements_id, users_user_id, achievement, date)
VALUES
  (1, 1, 'First Win', '2023-06-10'),
  (2, 2, 'Tower Master', '2023-06-12'),
  (3, 3, 'Perfect Score', '2023-06-14'),
  (4, 1, 'Speed Demon', '2023-06-15'),
  (5, 2, 'Champion', '2023-06-15'),
  (6, 3, 'Unstoppable', '2023-06-20'),
  (7, 1, 'Record Breaker', '2023-06-25'),
  (8, 2, 'High Scorer', '2023-06-25'),
```

```
  (9, 3, 'Master Strategist', '2023-06-18'),
  (10, 1, 'Perfectionist', '2023-06-20');
```

notifications.sql

```
INSERT INTO `towers-of-Hanoi-noSQL`.`notifications` (`users_user_id`,
`notification`, `date`) VALUES
(1, 'New achievement unlocked!', '2023-06-01'),
(2, 'You have reached the top of the leaderboard!', '2023-06-02'),
(3, 'Congratulations on your high score!', '2023-06-13'),
(4, 'You have received a new notification.', '2023-06-04'),
(5, 'Important announcement: Game update!', '2023-06-05'),
(6, 'You are now ranked among the top players!', '2023-06-23'),
(7, 'New feature added to the game!', '2023-06-21'),
(8, 'You have been awarded a special bonus!', '2023-06-20'),
(9, 'Reminder: Daily challenges are available!', '2023-06-18'),
(10, 'Check out the latest game events!', '2023-06-10');
```

complaints.sql

```
INSERT INTO complaints (users_user_id, complaint, date) VALUES
(1, 'Game froze during gameplay.', '2023-06-01'),
(2, 'Unable to log in to my account.', '2023-06-12'),
(3, 'Game crashes on startup.', '2023-06-13'),
(4, 'Missing game assets.', '2023-06-24'),
(5, 'In-game purchases not working.', '2023-06-05'),
(6, 'Game lagging during gameplay.', '2023-06-06'),
(7, 'Incorrect scoring calculation.', '2023-06-07'),
(8, 'Account data missing.', '2023-06-08'),
(9, 'Game UI is not responsive.', '2023-06-09'),
(10, 'Bug found in level 5.', '2023-06-10');
```

The same for the MongoDB database, but using .json documents which can be found in the folder src/database/mongodb/documents.

users.json


```
[{
  "_id": {
    "$oid": "649335a9ae622d6458ddaeca"
  },
  "user_id": 1,
  "first_name": "Samuel",
  "last_name": "Buraga",
  "username": "samuel",
  "password": "samuel",
  "created_at": {
    "$date": "2023-06-21T00:00:00.000Z"
  }
},
{
  "_id": {
    "$oid": "649335a9ae622d6458ddaecb"
  },
  "user_id": 2,
  "first_name": "David",
  "last_name": "Buraga",
  "username": "david",
  "password": "david",
  "created_at": {
    "$date": "2023-06-21T00:00:00.000Z"
  }
},
{
  "_id": {
    "$oid": "649335a9ae622d6458ddaecc"
  },
  "user_id": 3,
  "first_name": "Adrian",
  "last_name": "Oprisor",
  "username": "adioprisor",
  "password": "adi",
  "created_at": {
    "$date": "2023-06-21T00:00:00.000Z"
  }
},
{
  "_id": {
    "$oid": "649335a9ae622d6458ddaecd"
  },
  "user_id": 4,
  "first_name": "Ion",
  "last_name": "Popescu",
```

```
  "username": "ion",
  "password": "ion",
  "created_at": {
    "$date": "2023-06-21T00:00:00.000Z"
  }
},
{
  "_id": {
    "$oid": "649335a9ae622d6458ddaece"
  },
  "user_id": 5,
  "first_name": "Vasile",
  "last_name": "Afloarei",
  "username": "vasile",
  "password": "vasile",
  "created_at": {
    "$date": "2023-06-21T00:00:00.000Z"
  }
},
{
  "_id": {
    "$oid": "649335a9ae622d6458ddaecf"
  },
  "user_id": 6,
  "first_name": "Andreea",
  "last_name": "Amariei",
  "username": "andreea",
  "password": "andreea",
  "created_at": {
    "$date": "2023-06-21T00:00:00.000Z"
  }
},
{
  "_id": {
    "$oid": "649335a9ae622d6458ddaed0"
  },
  "user_id": 7,
  "first_name": "Eliza",
  "last_name": "Popescu",
  "username": "eliza",
  "password": "eliza",
  "created_at": {
    "$date": "2023-06-21T00:00:00.000Z"
  }
},
{
  "_id": {
    "$oid": "649335a9ae622d6458ddaed1"
```

```
  },
  "user_id": 8,
  "first_name": "Tudor",
  "last_name": "Miron",
  "username": "tudor",
  "password": "tudor",
  "created_at": {
    "$date": "2023-06-21T00:00:00.000Z"
  }
},
{
  "_id": {
    "$oid": "649335a9ae622d6458ddaed2"
  },
  "user_id": 9,
  "first_name": "Darian",
  "last_name": "Maria",
  "username": "darian",
  "password": "darian",
  "created_at": {
    "$date": "2023-06-21T00:00:00.000Z"
  }
},
{
  "_id": {
    "$oid": "649428a9740cc22b326b4651"
  },
  "user_id": 10,
  "first_name": "Emi",
  "last_name": "Munteanu",
  "username": "emi",
  "password": "emi",
  "created_at": {
    "$date": "2023-06-21T00:00:00.000Z"
  }
}]
```


In Intellij I created a DatabaseConnection.java interface which is implemented by MySQLConnection.java and MongoDBConnection.java.

DatabaseConnection.java:

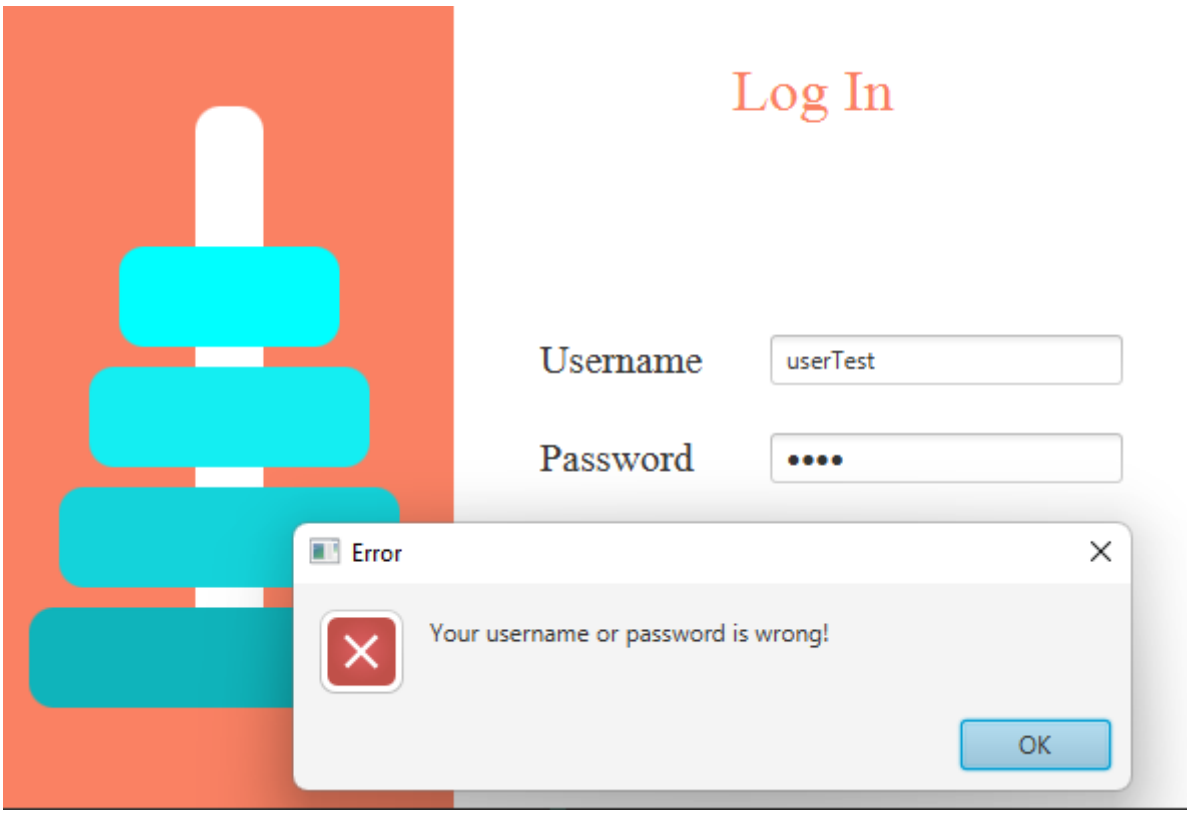
```
package com.example.towersofhanoi;

import javafx.scene.control.TableView;

public interface DatabaseConnection <T> {
    void connect(); // connects with the database
    void disconnect(); // disconnects from the database
    boolean checkIfUserExists(final String username, final String password);
    void deleteAccount(final String username); // deletes account
    void updateUsername(final String currentUsername, final String newUsername);
// updates username
    T getUserByUsername(final String username); // gets user information using
his username
    int getLatestUserId(); // gets the id of the latest added user (the
auto_increment current value)
    void insertNewUser(final String first_name, final String last_name, final
String username, final String password); // inserts new user in to the database
    void extractStatistics(TableView<StatisticsData> statisticsTable); //
extracts data from the statistics table
}
```

Here is an example of the connection between the interface and the MySQL database:

```
public void logInButtonOnAction(ActionEvent e) throws IOException, SQLException {
    if(usernameTextField.getText().isBlank() == false &&
passwordField.getText().isBlank() == false) {
        DatabaseConnection mySQLConnection = new MySQLConnection();
        mySQLConnection.connect();
        if(mySQLConnection.checkIfUserExists(usernameTextField.getText(),
passwordField.getText())) {
            ResultSet resultSet = ((MySQLConnection)
mySQLConnection).getUserByUsername(usernameTextField.getText());
            User.updateData(resultSet);
            Node node = (Node) e.getSource();
            Stage thisStage = (Stage) node.getScene().getWindow();
            thisStage.hide();
            Menu menu = new Menu();
            menu.start(new Stage());
        }
        else {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Error");
            alert.setHeaderText(null);
            alert.setContentText("Your username or password is wrong!");
            alert.showAndWait();
        }
    }
    else {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error");
        alert.setHeaderText(null);
```

```
        alert.setContentText("Please enter your username and password!");
        alert.showAndWait();
    }
}
```

This code checks if the user typed the correct information for the Log In process. The same approach is used for the Sign Up process.



Graphic Interface - Window 7 Log In

CONCLUSIONS

In conclusion, this project represent a successful implementation of the game “Towers of Hanoi” using JavaFX in IntelliJ.

The game provides an interactive and engaging experience for users, allowing them to play the game manually or opt for an automated solution. The integration of a database connection with MySQL or MongoDB enhances the project's functionality by allowing users to save and retrieve game progress, scores, or any other relevant information. Moreover, the implementation of threads ensures smooth gameplay and efficient execution, especially during automated solutions. This project demonstrates the practical application of object-oriented programming principles, graphical user interface development, and database connectivity in Java.

Through its intuitive user interface, seamless database integration, and efficient use of threads, this project showcases the potential of JavaFX, IntelliJ, and database systems in developing engaging and interactive games.

BIBLIOGRAPHY

Banas, D. (n.d.). *MVC Java Tutorial*. Retrieved from YouTube: https://www.youtube.com/embed/dTVVa2gfht8?feature=oembed

Model–view–controller. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Model–view–controller

Third normal form. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Third_normal_form

Towers of Hanoi. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Tower_of_Hanoi

CONTENTS

INTRODUCTION 2
I. MOTIVATION 2
II. THE THEMATIC CONTEXT OF THE PROBLEM 3
PROBLEM DESCRIPTION 4
I. THE TASK 4
II. "TOWERS OF HANOI" GAME 5
MY SOLUTION 7
I. THE MAIN CONCEPTS 7
II. TOOLS USED 8
THE PROJECT 9
I. THE APPLICATION 9
II. THE ARCHITECTURE 10
1. THE GRAPHIC INTERFACE 10
2. THE DATABASE 14
III. STRUCTURE 15
IV. DATA MANAGEMENT 20
CONCLUSIONS 31

ASSETS

Figure 1 ERD 20

Screenshot 1 git repository 11
Screenshot 2 Intellij project 13

Graphic Interface - Window 1 Menu 15
Graphic Interface - Window 2Play 16
Graphic Interface - Window 3Tutorial 17
Graphic Interface - Window 4 Options 18
Graphic Interface - Window 5Statistics 18
Graphic Interface - Window 6Statistics 19
Graphic Interface - Window 7 Log In 30

Video 1 MVC 11