Visiochess

A Visual Analysis of Chess Openings Over Time

Murray Heymann	Jolandi Lombard	Lisa van Staden
15988694	16994914	18245471
Elan van Biljon	Trandon Narasimulu	Francois Kunz
18384439	19044186	19163630

May 3, 2017

Contents

1 Specifications

The purpose of this project is to develop a web application to provide the user with a detailed analysis of the popularity of chess openings over the years.

1.1 General

The user can view the default database or upload their own data, provided it is in the correct file format, PGN, and contains the needed information.The web application is made fully responsive by using Bootstrap. A user file is restricted to a maximum size of 10Mb (10 000 000 bytes) The visual representation of the analysis is done using D3.js. The user can view all the data or apply the following filters:

1. Data Range by year
2. WhiteElo and BlackElo
3. ECO Codes:
    - Primary ECO codes (A-E)
    - ECO categories(13 categories)
    - ECO subcategories (one of 500)

1.2 Front End

The front end shall make use of D3.js for visualization of processed data. The front end shall give a visual summary of the information after filtering. The front end shall have a list of databases to choose from. The list shall always contain the default database of the back end, along with any databases uploaded by the user. Before a user has uploaded any data, the default data set shall be preselected. Upon loading a pgn file, the newly uploaded data set shall the selected data set.

1.3 Back End

The back end shall have an API for receiving an uploaded PGN file The back end shall parse the PGN file into a MySQL database. The back end shall have an API for receiving filtering commands. The back end shall process filtering commands into MySQL queries. The back end shall perform MySQL queries on databases of chess games. The back end shall return the results of a MySQL query to the user in a way that can be displayed by D3.js,

2 Program Description

2.1 Back End

The Back End is comprised of all code that gets executed on the serverside.

All database queries get done here, mostly because the default database is assumed to be used most often and is found here. This also allows devices with limited processing power to use the service with ease, while also requiring a server with reasonably large memory, storage and processing abilities.

2.1.1 make.sh

This is a Linux bash script that set's up the backend environment. It downloads all necessary dependancies, creates the MySQL database and creates a configuration file for the server.

2.1.2 make_osx.sh

This is a OSX bash script that set's up the backend environment. It downloads all necessary dependancies, creates the MySQL database and creates a configuration file for the server.

2.1.3 .my.cnf

This is a configuration file that is generated. It contains all the information needed to connect to the MySQL server.

2.1.4 user_upload.php

As the specifications dictate that users need to be able to upload their own custom database, user_upload.php receives such uploads and performs some basic checks on the uploaded file, before saving it to the server disk. The file gets hashed using sha256 and this hash is used as the file name when saved, allowing users to easily specify which file they wish to use, while avoiding any possible clashes in the naming domain, should different users upload different files with the same name. Conveniently, if two users upload identical databases, these databases won't be duplicated on the server.

After a file has been uploaded successfully, a database is created with the same name as the local file. This is done in create_db.php with the function create_database.

Once the file has been saved, it get's parsed. For each game, there are three different kinds of data to parse: the seven tags required in PGN, optional tags and the moves made by during the game. It gets parsed by using the parse_pgn_file function in pgn_parser.php.

2.1.5 create_db.php

This script contains a function to create a database for a newly uploaded file. It has to parse the file to determine the string length to be stored in the moves field.

create_database

First the login details for a MySQL user is read from a file named .my.cnf in the root directory of the project. This is then used to connect to the MySQL server on the local server. Once a connection is set up, a database is created with a query, and its success is checked. The connection is then closed.

2.1.6 pgn_parser.php

This file mostly defines functions used by the user_upload.php script. As the name suggests, it parses PGN files.

sscan_tag

The sscan_tag function is used to scan tags when parsing a PGN file. Takes two string arguments: the tag string and a string that describes the start of that line. The second argument is used for getting the offset into the string where the tag value starts. The function the scans the tag until the tag string closes with '"'. This scanned value is then returned to the caller.

parse_pgn_file_to_db

The standard seven tags as required by the PGN are processed with the sscan_tag function and have their values stored in local variables. Thereafter, the optional tags are scanned and each examined as either being an ECO tag or a black or white ELO tag. If any of these are not present, their local variable simply gets the empty string assigned to it.

The moves are often listed over various lines. First, these moves are collected into a single line string. Then, the string is split by spaces, to produce an array. On array indices that are multiples of three, one finds the move numbers. On the very last non-trivial entry, one finds the score of the match. All other entries indicate moves:

$$\begin{aligned} \forall i \in \{x \in \mathbb{Z} \ &: x \text{ a valid index in moves array}\}, \\ i \equiv 2 \pmod 3 &\implies \text{moves[i] denotes a black move} \\ i \equiv 1 \pmod 3 &\implies \text{moves[i] denotes a white move} \end{aligned}$$

The moves are separated into an array of white moves and an array of black moves. These arrays are then stored in an object than can be easily sent to a function that will insert the data into the MySQL database.

get_longest_moves_string

This function is almost identical to parse_pgn_file_to_db, but it serves to determine which chess game in a pgn file has the longest move string. This is helpful when creating the database with the "flat" approach.

2.1.7 query.php

The user sends filters to be applied to the database. These filters are sent to query.php, which then formulates the filter data into MySQL queries. These queries made into the database and the results are sent back to the user.

2.1.8 mysql_interface.php

This file contains the *ServerInterface* class. This class is a PHP interface built ontop of the *mysqli* interface. It adds another layer of abstraction that makes interfacing with MySQL servers much easier.
Among others it provides functions to:

- connect to a MySQL server
- create and delete a database
- create and delete a table
- insert data into a tabel
- fetch data from the database (make queries)
- send SQL commands to the server

2.2 Front End

The front end is presented to the user using HTML5 (index.html) and suplemented with javascript (main.js) to allow for an interactive and dynamic experience. The data returned by a query to the server is displayed using d3.js in a dedicated section. Links to a description of the web application (about.html) and the course website is provided in the footer.

CSS styling is based on Bootstrap and supplemented with a custom styling sheet (css/style.css) generated by Sass (style.scss).

2.2.1 main.js

This file contains the javascript that will run in the clients browser. It will handle user interaction (such as constructing new filters and uploading files). It will make the necessary ajax calls to the server when the user wants to apply filters and upload their own PGN file. This file also validates all user input to ensure it is correct and safe to send to the server.

handle_onsubmit function

This function runs some checks on the submitted PGN file and if it passes all the checks it uses *hex_sha256.js* to hash the file. It then sends the file and a hash of the file to the server using ajax.

handle_filter_submit function

This function sends the filter form data in json form to the server. (It will handle the response properly later but for now it just embeds it into the html)

getFormData function

This function serializes and returns form data in the form of a json. (I believe it was acquired from stackoverflow.com, we should reference that)

2.2.2 visual_response.js

This file contains javascript that will run in the clients browser. It will handle the visualisation of the data, that is received from the server after the filters have been submitted. A lot of the code is based on the Stacked Area Chart Example from `https://bl.ocks.org/mbostock/3885211`.

draw

This function receives a JSON object from handle_response or handle_window_resize in main.js. After this object has been processed, the data it contains is then visualised using the D3 library functions.

process_JSON_to_D3

This function creates and returns a new JSON object from the received data function which will be used to create the graph.

3 Testing

3.1 Javascript Testing

For testing the javascript functions and functionality we used the built in QUnit library to perform JUnit testing.