AH SDD Project: Game Assist Tool

Contents

[Analysis 3](#_Toc119411246)

[Purpose 3](#_Toc119411247)

[Pre-Development User Survey 4](#_Toc119411248)

[Considerations of Pre-Development User Survey 4](#_Toc119411249)

[Post-Development User Survey Plan 4](#_Toc119411250)

[Constraints 5](#_Toc119411251)

[UML Use Case Diagram 6](#_Toc119411252)

[Requirements Specification 7](#_Toc119411253)

[End-User Requirements 7](#_Toc119411254)

[Functional Requirements 8](#_Toc119411255)

[Inputs, Processes and Outputs 9](#_Toc119411256)

[Notes on Further Input Validation 11](#_Toc119411257)

[Personas 11](#_Toc119411258)

[Sean Greaves 12](#_Toc119411259)

[Winnie Manning 12](#_Toc119411260)

[James Hill 12](#_Toc119411261)

[User Stories 12](#_Toc119411262)

[User Scenarios 12](#_Toc119411263)

[Project Plan – Initial estimated version 13](#_Toc119411264)

[Project Plan – Final version 16](#_Toc119411265)

[Design 19](#_Toc119411266)

[Pseudocode Design 19](#_Toc119411267)

[UML Class Design 21](#_Toc119411268)

[Project Design 21](#_Toc119411269)

[User-Interface Design 21](#_Toc119411270)

[Appendix 1 – Pre-Development User Survey Results 22](#_Toc119411271)

[Q1 22](#_Toc119411272)

[Results 22](#_Toc119411273)

[Q2 22](#_Toc119411274)

[Numerical Results 22](#_Toc119411275)

[Mean 22](#_Toc119411276)

[Results 22](#_Toc119411277)

[Q3 23](#_Toc119411278)

[Results 23](#_Toc119411279)

[Q4 23](#_Toc119411280)

[Results 23](#_Toc119411281)

[Q5 24](#_Toc119411282)

[Results 24](#_Toc119411283)

[Q6 24](#_Toc119411284)

[Results 24](#_Toc119411285)

[Q7 25](#_Toc119411286)

[Results 25](#_Toc119411287)

[Appendix 2 – Post-Development User Survey Results 25](#_Toc119411288)

# Analysis

## Purpose

* Currently there is no maintained game assist tool which records match data on the game: Magic: The Gathering Online (MTGO). The game is played by two players, using decks of cards, in a best of three format (called a match). This leaves users to record this match data manually on an excel document, which is very time consuming, or not record the data at all. This is a problem as the game rewards the player for winning matches, so the players are incentivised to increase their win rate (% chance of winning a game). This leads to a highly competitive environment, where players spend lots of money on the game to maximise their win rate. So, if players can get huge volume of match data, which accurately displays their overall winrate and winrate against specific opponent. This means that they can improve their gameplay by developing strategies (or a better strategy), which would hopefully increase their win rate. Which, as I’ve already outlined, they are incentivised to do.
* The end users are people who play Magic: The Gathering Online (MTGO), they are technically adept (as they play an online video game). This means they should be capable of installing and using a game assist tool. Their total age range is between 16-40, with most users being 17-24. The end users also spend lots of money on the game, as they typically want to maximise their win rate. Most end-users live in Europe and North America. They also spend a reasonable amount of time playing video games (10 to 20 hours).
* The app will be built on Electron so that an interface can be easily created using HTML, CSS and JavaScript. The app will display all currently collated data (a spreadsheet of database data of match results), which can be manipulated by the end-user, and manipulatable graphs of the data. The end-user can download the collated data and graphs. On the app, the user can press a button which runs the main processes of the data collection algorithm. This main process navigates to MTGO’s file system, where it gets the users’ match logs (stored in .dat files). The app then reads all information within the logs, which it categorises and stores in a local sqlite3 database. It may require the end-user’s input to clarify data which the match logs leave out, this is conditional on the basis that one of the players don’t concede on the final game. The app also uses the Selenium web driver to get additional information (deck names), to store in the database.
* The Advanced Higher concepts that the app meets are:
* The project will using object-orientated programming (OOP), as well as imperative programming with 2D arrays and arrays of records.
* A SQA Standard Algorithm (bubble sort algorithm).
* Other Advanced Higher Concepts:
  + Connecting to a database via a programming language.
  + SQL query design.
  + Execution of SQL operations.
  + Integrative testing (Software Design and Development and Database Design and Development).
* The app is integrated with:
  + A database (local, sqlite3) with multiple tables, whose connection will be opened (and closed) to execute SQL queries.
* New skills required for the app (which aren’t in the project specification or course specification):
* Running files based on user input, with arguments.
* Implementing regular expression (Regex) in the app.
* Sanitise user inputs which will be used as part of SQL Queries.
* Web Scraping (using the Selenium library).
* Creating and using python modules (files that store a class) to increase modularity.

## Pre-Development User Survey

* This survey was carried out to gauge potential end-user interest in the app, whether design decisions should be followed and whether assumptions about potential end-users were accurate.
* The user survey confirmed that there was an interest in an MTGO game assist tool among potential end-users (this proves the app is viable), that certain design decisions should be followed (open-source, free, etc.), and the assumptions about potential end-users were correct (age demographic, time spent playing games, etc.).
* See Appendix 1 for information on specific results and screenshots of the results.
* Details can also be found in the functional requirements under the heading: Results from Pre-Development User Survey.

### Considerations of Pre-Development User Survey

* The survey didn’t ask the potential end-user how much time they spent playing MTGO. This means that the surveyed group may not contain a wide range of MTGO players.
* The survey was untranslated to other languages, this means that only English speakers could give feedback, so huge numbers of potential end-user’s feedback may not have been collected.

### Post-Development User Survey Plan

* The point of the Post-Development User Survey will be to gauge whether the developed app is successful, whether it meets the requirements of the end-user (requirements which may have been overlooked in the development of the app) and if further work on the app is needed (and if so, what work).
* This means that the Post-Development User Survey must be given to users of the app.

## Constraints

* Technical Constraints
  + The app will run on Windows 7 or later.
  + The app itself does not take up lots of space, however, the databases have no limit on how large it can be (this is down to the user’s discretion).
  + Python’s performance is relatively slow, however, it is very sufficient for this app.
  + All the necessary resources currently exist for the project to be completed.
  + I must learn significantly more Java Script during the development of the app.
* Business Constraints
  + The scheduling and timescales must be met, otherwise the app will be incomplete.
  + There is no budget for the app.
  + There is a single person working on the app.
  + There are no running or single-time costs associated with the project as it uses free software.
  + The app can be further developed which could allow it to follow the freemium model, so the end-user could pay for more features. This would work with a website and a server, which user data will be sent to (collating all user data), so overall evaluations of cards and decks can be done. The users will pay for access levels of this collated data.
  + The app will take 80 hours to complete.
  + The app will generate no revenue and will be open source. This has the intangible benefit of increased trust in the project since anyone can check what the program does with your data.
* Legal Constraints
  + There are no legal issues with development of the app: all web scraping is within current precedent (site doesn’t have a robots.txt, so web scraping isn’t disallowed, and the web driver doesn’t login/create an account) and no personal data is stored (so GDPR isn’t in violation).
  + This project complies with the Copyright, Designs and Patents Act.
* Time Constraints
  + This project must be completed in no more time than 5 months.

## UML Use Case Diagram

Diagram

Description automatically generated with low confidence

## Requirements Specification

### End-User Requirements

#### Results from Pre-Development User Survey

This information has been used to: confirm some assumptions about the end users and see if there is interest in development of the app.

See appendix 1 for more information and the results data.

* The survey found that most potential end-users are under 18 to 24, this means that the app should be designed for that age group so results should be defined clearly, graphs shouldn’t be overused (keep minimal on the page at once), there should always be summaries for large blocks of text (before the large block of text or data), try to keep a clean look (togglable option for large information, for a specific paragraph or generally, could be implemented).
* The survey found that most potential end-users play videogames a fair amount, this means they are likely technology literate, so design doesn’t have to be very simplified like it would have to be for older age groups.
* The survey found that most potential end-users would want the app to display feedback on their game, this means that attention should be paid to making the app extrapolate potential meaning based on data (this also makes the data more accessible).
* This survey found that most potential end-users want a combination of ways to display information stored by the app, this means that there should be toggleable options for types of graphs shown (for data).
* The survey found that cost is a barrier of entry for most potential end-users, this confirms that the app should be free, to enable as many users to use it as possible. This is also useful for when the freemium model is applied, there are lots of end-users who may pay for the additional features.
* The survey found that potential end-users would be interest in installing the finished app, this means that there is enough interest in order to justify creation of the app.
* The survey found that the potential end-user will expect the interface to display:
* Graphs (Pie Charts, Scatter Graphs, etc.)
* Comparisons (e.g. Past win rate Vs Current win rate and Past {specific card} play rate Vs Current {specific card} play rate)

#### Assumed Requirements

The end-user will expect the app to:

* Capture their match logs.
* Display match data in a readable format.
* Be compatible with their device.
* To be able to export match data.
* Display detailed game actions.
* Display individual cards and deck statistics.
* Be integrated with a local database.
* Be able to edit the deck name of a match record that has an unusual deck uncertainty (the probable deck a player was player).
* Be able to download the graphs displayed on the app.
* Be able to edit the exact deck list of a record (helpful if the user knows the deck, since to app will have incomplete information).
* Be able to specify the winner of a match if the match logs did not which player won the final game. This happens because the match logs may contain incomplete information on any number of games. This is a problem if a match goes to three games and the final game is undetermined, since even if no player win is in the logs, whichever player went second in the next game lost the previous game (this workaround isn’t possible if the final game is undetermined).

### Functional Requirements

* + - The app must be able to read .dat file data and insert that data into a local database.
    - The app must be integrated with a local database.
    - The app must be able to display the average deck for each player in each match.
    - On the app, all users will have the same interface.
    - The app will sanitise database inputs to protect against basic against SQL injection.
    - The app must be able to export match to a .json file.
    - The app must be able to display graphs of data.
    - The app must be able to assign deck names and rough deck lists to each deck played, based on limited match data.
    - The app will close the database connection after queries are run to prevent any possible security breach, that could target open database connection vulnerabilities.
    - The app must be able to let the user edit the deck name of the deck, if the probability of the suggested deck name is low (less than 60%) and append that change to the local database.
    - The app must let the user specify which player won the match, if there is incomplete data on who won which game and append that result to the database. The app will help the user determine the result by showing them the final turn log of the match, and if the user cannot determine the win, then it shows the final game log. If the user cannot determine who won, the result stays unchanged.
    - The app must let the user change the deck list of a match record and append the result to the database.
    - The app will create a downloadable .png file for a graph, when it’s download button is pressed (the user must hover the mouse cursor over the image to display the download button).

### Inputs, Processes and Outputs

#### Main App Page

|  |  |  |
| --- | --- | --- |
| Inputs | Processes | Outputs |
| Sync Button Click (Click input) | Runs File Reader, then updates reloads first page | User: Possible error message |
| Profile Button (Click input) | Hides database content, displays profile page and gets average winrate from the database. | Average winrate. |

#### Main App Page

|  |  |  |
| --- | --- | --- |
| Inputs | Processes | Outputs |
|  | Gets results of matches from the database. | Results of matches from the database. |

#### File Reader

|  |  |  |
| --- | --- | --- |
| Inputs | Processes | Outputs |
|  | Iterates over all files where match files are stored.  For each file it outputs deck lists and date to Web Scraper, to get possible deck names (from which it chooses the most probable, and if a deck name isn’t significantly probable (less than 60%), it stores all). It also stores that data in the integrated local app database. |  |

#### Web Scraper

|  |  |  |
| --- | --- | --- |
| Inputs | Processes | Outputs |
| File Data | Web scrapes data from mtgtop8.com to obtain probable deck name data using data inputted | Deck names (dictionary formatted as {“deck name”: percentage of appearance} where ‘percentage of appearance’ is a float) |

### Notes on Further Input Validation

* To validate match files, when reading the file, that file is ignored if it doesn’t meet expected formatting (all match data is stored the same way).
* Data is sanitised by parameterized queries before it’s put into the database to prevent unwanted errors.

## Personas

I have developed 3 Personas, as this allows each user to have different needs, and as must be met by the end-user requirements, each user-type can have their needs met.

### Sean Greaves

* A 23-year-old man who lives in America, on their own in a house. They’ve being playing MTGO for 4 years and are very committed to the game. They have a middling income job at Specsavers and are highly competitive as they are a high-performance swimming athlete in their spare time.

### Winnie Manning

* An 18-year-old woman who lives in the UK, in a flat (with 2 flatmates). They are in their first year of university. They’ve been playing MTGO for 2 years and are fairly committed to the game. They have a part-time job as a barista at a local indie coffee shop. They read the newspaper and always make sure to complete the sudoku, which they enjoy finishing. They also have game night with their flatmates every Saturday night and are quite competitive when it is Monopoly that night.

### James Hill

* A 16-year-old man who lives in Ireland, with their parents and a sibling. They are still in school and have only just picked up MTGO in the last 4 months. They aren’t committed to the game and aren’t very similar with it. Their only income source is pocket, birthday, and Christmas money. They play other competitive video games with their friends during weekends and after school.

## User Stories

* I, Sean Greaves, don’t have much time to analyse my game data (because I have a full-time job and am an athlete), so I want assistance in analysing my game data.
* I, Winnie Manning, struggle to understand my game data, so I want it to be displayed in an understandable format.
* I, James Hill don’t, know which cards most contribute to my success, so I want to know which cards most contribute to my success.

## User Scenarios

* Sean is busy most of the time, whether they’re swimming or working at their job. This means that they don’t have much time to play MTGO and what little time they do play, they can’t put much thought into strategy. Sean is very competitive in everything they do, so winning is tied to their enjoyment they get out of the game. This means that they want to win more and one of the best ways to do that is get better at playing the game. So, to do this, Sean wants a tool which can help him get better at playing the game.
* Winnie Manning has trouble scanning their self-collated match data to analyse their gameplay, they also don’t know exactly what they’re looking for when trying to analyse their self-collated match data and they struggle to maintain a consistent format across their spreadsheet. They want a tool to solve these problems for them by displaying their match data in a consistent format, which would help her to understand her match data.
* James Hill doesn’t have lots of money, so they want to know exactly which cards most contribute to their success in the game, so they don’t spend time and money playing with cards which are ‘bad’ or have a lower win rate than other cards. Therefore, they want a tool to solve this problem for them as they don’t want to learn how to and collect data to analyse all by themselves. They want this the tool to do this by presenting all games, so he can look through the matches, so see which cards most contributing to his success.

## Project Plan – Initial estimated version

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task | | Start Date | End date | Workflow | Resources Required |
| Analysis (27/08/22 – 19/10/22) | | | | | |
| 1. | Description of problem | 27/08/22 | 29/08/22 | Sequential |  |
| 2. | Pre-Development user survey | 30/08/22 | 05/09/22 | Sequential | Using Survey Monkey (to create a survey) and Snipping Tool (to screenshot results) |
| 3. | Constraints | 6/09/22 | 10/09/22 | Sequential |  |
| 4. | UML Use Case diagram | 11/09/22 | 15/09/22 | Sequential | Using Lucidchart |
| 5. | Create persona, user stories and scenarios | 16/09/22 | 19/09/22 | Sequential |  |
| 6. | Requirements specification: end-user req. | 20/09/22 | 25/09/22 | Sequential |  |
| 7. | Requirements specification: functional req. | 26/09/22 | 30/09/22 | Sequential |  |
| 8. | Project plan | 1/10/22 | 8/10/22 | Sequential | Using Gantt Project |
| Design (20/10/22 – 21/10/22) | | | | | |
| 9. | Pseudocode design | 9/10/22 | 14/10/22 | Sequential |  |
| 10. | UML class diagram | 15/10/22 | 18/10/22 | Sequential | Using Miro |
| 11. | Project Design | 19/10/22 | 23/10/22 | Parallel |  |
| 12. | User-Interface Design | 24/10/22 | 27/10/22 | Parallel |  |
| Implementation (01/11/22 – 10/12/22) | | | | | |
| 13. | Implementation (including input validation) | 28/10/22 | 08/12/22 | Parallel |  |
| 14. | Research and development of new skills | 28/10/22 | 08/12/22 | Parallel |  |
| 15. | Log of ongoing testing | 28/10/22 | 08/12/22 | Parallel |  |
| End Testing (12/12/22 – 23/12/22) | | | | | |
| 16. | Final Test Plan | 09/12/22 | 15/12/22 | Sequential | Using pytest |
| 17. | Requirements Testing | 16/12/22 | 17/12/22 | Sequential |  |
| 18. | Testing with personas and test cases | 20/12/22 | 22/12/22 | Sequential |  |
| Evaluation (05/01/23 – 07/01/23) | | | | | |
| 19. | Evaluation Report | 18/12/22 | 19/12/22 | Sequential |  |
| 20. | Post-Development user survey | 03/01/23 | 05/01/23 | Sequential | Using Survey Monkey (to create a survey) and Snipping Tool (to screenshot results) |

## Project Plan – Final version

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task | | Start Date | End date | Workflow | Resources Required |
| Analysis (27/08/22 – 19/10/22) | | | | | |
| 1. | Description of problem | 27/08/22 | 29/08/22 | Sequential |  |
| 2. | Pre-Development user survey | 31/08/22 | 05/09/22 | Sequential | Using Survey Monkey (to create a survey) and Snipping Tool (to screenshot results) |
| 3. | Constraints | 20/09/22 | 21/09/22 | Sequential |  |
| 4. | UML Use Case diagram | 22/09/22 | 27/09/22 | Sequential | Using Lucidchart |
| 5. | Create persona, user stories and scenarios | 01/10/22 | 05/10/22 | Sequential |  |
| 6. | Requirements specification: end-user req. | 07/10/22 | 08/10/22 | Sequential |  |
| 7. | Requirements specification: functional req. | 09/10/22 | 17/10/22 | Sequential |  |
| 8. | Project plan | 18/10/22 | 19/10/22 | Sequential | Using Gantt Project |
| Design (20/10/22 – 21/10/22) | | | | | |
| 9. | Pseudocode design | 20/10/22 | 22/10/22 | Sequential |  |
| 10. | UML class diagram | 24/10/22 | 27/10/22 | Sequential |  |
| 11. | Project Design | 27/10/22 | 30/10/22 | Parallel |  |
| 12. | User-Interface Design | 27/10/22 | 31/10/22 | Parallel |  |
| Implementation (01/11/22 – 10/12/22) | | | | | |
| 13. | Implementation (including input validation) | 01/11/22 | 10/02/22 | Parallel |  |
| 14. | Research and development of new skills | 01/11/22 | 10/02/22 | Parallel |  |
| 15. | Log of ongoing testing | 01/11/22 | 10/02/22 | Parallel |  |
| End Testing (12/12/22 – 23/12/22) | | | | | |
| 16. | Final Test Plan | 10/02/22 | 16/12/22 | Sequential | Using pytest |
| 17. | Requirements Testing | 10/02/22 | 18/12/22 | Sequential |  |
| 18. | Post-Development user survey | 10/02/22 | 20/12/22 | Sequential | Using Survey Monkey (to create a survey) and Snipping Tool (to screenshot results) |
| 19. | Testing with personas and test cases | 10/02/22 | 24/12/22 | Sequential |  |
| Evaluation (05/01/23 – 07/01/23) | | | | | |
| 20. | Evaluation Report | 05/01/23 | 07/01/23 | Sequential |  |

# Design

Design of Advanced Higher Concepts

Design of Integration

User-interface design shows inputs, processes, and outputs, and matches the end-user and functional requirements

Overall design matches the requirements specification

## Pseudocode Design

### Python Files

#### sync.py

IMPORT mtgScraper as MODULE named Scraper

localUser = GET\_LOCAL\_USER()

path = 'C:\Users\' + localUser + '\*AppData\Local\Apps\2.0\Data\'*

for i = 0 TO 2:

    folder = GET\_DIRECTORIES(path)

    path = path + '\' + folder

mainFolders = GET\_DIRECTORIES(path)

scp = Scraper()

for folder in mainFolders:

    folderPath = path + '\' + folder + '\*' + 'Data\AppFiles'*

    for (folderPath, directory, fileNames) in WALK(folderPath):

        if length(directory) > 1:

            directory.REMOVE('.vs')

        folderPath = folderPath + '\' + directory

        exit loop

    scp.ADD\_PATH(folderPath)

scp.RUN()

#### mtgScraper.py

IMPORT mtgtop8 as MODULE named DriverController

IMPORT matchRecord as MODULE named MatchRecord

CLASS Scraper:

    CONSTRUCTOR(self):

        instansiate\_object

        self.paths = []

    METHODS(self):

        method ADD\_PATH(self, path):

            self.paths.append(path)

        method RUN(self):

            self.OPEN\_CONN()

            self.CHECK\_DB()

            top8Conn = self.CHECK\_INTERNET()

            match = MatchRecord

            dc = DriverController

            for path in self.paths:

                fileList = GET\_DIRECTORIES(path)

                self.cursor.EXECUTE("SELECT filename FROM matches;")

                filenames = self.cursor.FETCHALL()

                for filename in fileList:

                    if filename in filenames:

                        exit loop

                    decklists, extra, matchlog, players = match.getDecklists(path+'/'+filename)

                    if decklists is not None:

                        dateTime = filename.GET\_DATE

                        if top8Conn = True:

                            dictNames = dc.RETURN\_DICT\_NAMES(decklists, date)

                        else:

                            dictNames = "NA"

                        self.SQLITE\_DRIVER\_DATA(filename, date, dictNames, extra, decklists, players, matchlog)

            dc.QUIT\_DC()

            self.CLOSE\_CONN()

        method OPEN\_CONN(self):

            self.conn = CONNECT\_TO\_DATABASE

            self.cursor = CURSOR

        method CHECK\_INTERNET(self):

            url = 'https://mtgtop8.com/'

            try:

                URLLIB.REQUEST.URLOPEN(url)

                connection\_to\_internet = True

            except:

                connection\_to\_internet = False

            return connection\_to\_internet

        method CHECK\_DB(self):

            if self.conn = FAILED:

                self.cursor.EXECUTE("""CREATE TABLE matches(

                                    matchID INTEGER PRIMARY KEY,

                                    filename TEXT,

                                    players BLOB NOT NULL,

                                    decknames BLOB,

                                    decklistP1 BLOB NOT NULL,

                                    decklistP2 BLOB NOT NULL,

                                    firstTurns BLOB NOT NULL,

                                    winLoss BLOB NOT NULL,

                                    format TEXT,

                                    type TEXT,

                                    date TEXT NOT NULL);""")

                self.conn.CONNECT()

                self.cursor.EXECUTE("""CREATE TABLE games(

                                    gamesID INTEGER NOT NULL PRIMARY KEY,

                                    gameNum INTEGER NOT NULL,

                                    startingHands BLOB NOT NULL,

                                    gameLog BLOB NOT NULL,

                                    winner BLOB,

                                    matchID INTEGER REFERENCES matches(matchID) ON UPDATE CASCADE);""")

                self.conn.COMMIT()

        method SQLITE\_DRIVER\_DATA(self, filename, dateTime, dictNames, extra, decklists, players, matchlog):

            data = (filename, players, dictNames, decklists[0], decklists[1], extra['play'], extra['winner'], 'NA', 'Constructed', dateTime)

            self.cursor.EXECUTE("INSERT INTO matches(filename, players, decknames, decklistP1, decklistP2, firstTurns, winLoss, format, type, date) VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?);", data)

            self.conn.COMMIT

            self.cursor.EXECUTE("SELECT MAX(matchID) FROM matches;")

            matchID = self.cursor.FETCH\_ONE

            gameNo = 0

            for game in matchlog:

                //parameterized query

                data = (matchID[0], matchlog.INDEX(game)), extra['startingHands'], matchlog[gameNo], extra['winner'][matchlog.INDEX(game)])

                self.cursor.EXECUTE("INSERT INTO games(matchID, gameNum, startinghands, gameLog, winner)  VALUES(?,?,?,?,?);", data)

                gameNo = gameNo + 1

        method CLOSE\_CONN(self):

            self.conn.COMMIT

            self.conn.CLOSE

            self.cursor.CLOSE

#### matchRecord.py

CLASS MatchRecord:

    CONSTRUCTOR(self):

        instansiate\_object

    METHODS(self):

        method GET\_DECKLISTS(self, filename):

            decklists = []

            extra = {'play':[], 'startingHands':[], 'winner':[]}

            self.matchLog = filename.DECODE()

            attempt:

                self.GET\_PLAYERS()

            if error:

                return None

            self.FORMAT\_LINES()

            if length(self.matchLog) < 2:

                return None

            for game in self.matchLog:

                if game doesn't exist:

                    exit loop

                gameDecklists = self.GET\_DECKLISTS(game)

                if gameDecklists doesn't exist:

                    exit loop

                decklists.append(gameDecklists)

                attmept:

                    extra['play'].append(self.GET\_ON\_PLAY(game))

                if error:

                    extra['play'].append('unknown')

                attempt:

                    extra['startingHands'].append(self.GET\_STARTING\_HANDS(game))

                if error:

                    extra['startingHands'].append('unknown')

                extra['winner'].append(self.getWinner(game))

        return decklists, extra, self.matchLog, self.players

    method GET\_PLAYERS(self):

        players = self.matchLog.FIND\_ALL(('@P(\S+) rolled')

        #the first player in the game log is always the user

        self.players = list(players)

    method FORMAT\_LINES(self):

        filteredMatch = self.matchLog.REMOVE\_NON\_ASCII

        filteredMatch = filteredMatch.REMOVE\_RANDOM\_CHARACTERS

        filteredMatch = filteredMatch.SPLIT\_BY\_GAME

    filteredMatch = filteredMatch.SPLIT\_BY\_TURN

        self.matchLog = filteredMatch

    method GET\_DECKLISTS(self, game):

        decklists = {self.players[0]: dict(), self.players[1]: dict()}

        playCardPattenMatches = self.matchLog.GET\_PLAYED\_CARDS

    playCardPattern = '('+self.players[0]+'|'+self.players[1]+') (casts|plays|discards|cycles) (@\[([a-zA-Z\s,-]+)@:[0-9,]+:@\])'

    revealedCardPattern = '('+self.players[0]+'|'+self.players[1]+') (reveals) (@\[([a-zA-Z\s,-]+)@:[0-9,]+:@\])'

        for turn in game:

        playCardMatches = playCardPattern.FIND\_ALL(turn)

            revealedMatches = revealedCardPattern.FIND\_ALL(turn)

            for cardPlayed in playCardMatches:

            //cardPlayer is stored as [player, card]

                if cardPlayed[1] in revealedMatches:

                    revealedMatches.remove(revealedMatches.INDEX([cardPlayed[1]))

                if cardPlayed[1] in decklists[card]:

                    decklists[cardPlayed[0]][cardPlayed[1]] += 1

                else:

                    decklists[cardPlayed[0]][cardPlayed[1]] = 1

            for cardRevealed in revealedMatches:

            //cardRevealed is stored as [player, card]

                if cardRevealed[1] in decklists[cardRevealed[0]]:

                    decklists[cardRevealed[0]][cardRevealed[1]] += 1

                else:

                    decklists[cardRevealed[0]] = 1

        return decklists

     method GET\_ON\_PLAY(self, game):

        onPlay = game.SEARCH('(\S+) chooses to play first')

        return onPlay

    method GET\_STARTING\_HANDS(self, game):

        startingHands = self.matchLog.FIND\_ALL\_STARTING\_HANDS\_IN\_GAME

        return startingHands

    method GET\_WINNER(self, game):

        concededPattern = '(\S+) has conceded'

        winsPattern = '(\S+) wins the game'

        losesPattern = '(\S+) loses the game'

        conceded = concededPattern.SEARCH(concededPattern)

        wins = winsPattern.SEARCH(winsPattern)

        loses = losesPattern.SEARCH(losesPattern)

        if a player wins:

            return wins

        else if a player conceded:

        return conceded

        else if a player loses:

            return loses

        else:

            return 'NA'

#### mtgtop8.py

CLASS DRIVER\_CONTROLLER:

    CONSTRUCTOR(self)

        self.url = "https://mtgtop8.com/search"

        self.driver = webdriver.CONNECT()

    METHODS(self)

        method RETURN\_DICT\_NAMES(self, deckLists, date):

            self.GET\_SITE()

            self.CLEAR\_COOKIE\_BANNER()

            self.INPUT\_FORM\_DATA(format, self.REFACTOR\_DECKLISTS(deckLists), date)

            deckNames = self.GET\_DECK\_NAMES()

            if deckNames are unknown:

                self.QUIT()

                return deckNames

            dictNames = self.GET\_DICT\_NAMES(deckNames)

            return dictNames

        method GET\_SITE(self):

            self.driver.get(self.URL)

        method INPUT\_FORM\_DATA(self, deckList, date):

            self.driver.find(SIDEBOARD).CLICK()

            textarea = self.driver.FIND(CARDS)

            for card in deckList:

                textarea.TYPE(card + RETURN)

            self.driver.FIND(DATE\_ELEMENT).TYPE(date)

            self.driver.FIND(SUBMIT).CLICK()

        method clearCookieBanner(self):

            self.driver.find(COOKIE\_WINDOW).CLICK()

        method getDeckNames(self):

            deckNames = []

            decks = length(self.driver.FIND\_ALL(DECKS\_ON\_PAGE))

            if decks < 1:

                return 'unknown'

            if decks > 5:

                decks = 5

            for deck = 1 To length(decks):

                deckNames.append(self.driver.FIND(DECK\_NAME).TEXT)

            return deckNames

        method QUIT\_DC(self):

            self.driver.QUIT()

        method REFACTOR\_DECKLISTS(self, deckLists):

            cards = associative\_array()

            for game in deckLists:

                for card in game:

                    if cards[card] exists:

                        cards[card] = cards[card]+1

                    else:

                        cards[card] = 1

            return cards

        method GET\_DICT\_NAMES(self, deckNames):

            dictNames = {deckName:0 for deckName in deckNames}

            for deckName in deckNames:

                dictNames[deckName] = dictNames[deckName] + 1

            for deckName in dictNames:

                dictNames[deckName] = dictNames[deckName] / length(deckNames)

        return dictNames

#### dbCMD.py

### Queries

### Data Dictionaries

#### Match table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Field | PK/FK | Datatype | Validation | Formatting (if not null) | Description | Example |
| matchID | PK | Integer | Presence |  | Unique surrogate key | 1 |
| filename |  | Text | Presence |  | Name of file | Gh33-dhhd9-eiuoew-fewfw-dwa23tger.dat |
| players |  | Blob | Presence | (player2 name) | Player2’s username | (CoolGamer, BoringGamer) |
| deckNames |  | Blob |  | {“deckname”: chance, …} | Player1’s and Player2’s deck names | {“AggressiveDeck”:0.75, “ControlDeck”:0.25} |
| deckListP1 |  | Blob |  | ({“cardName”:copies, “cardName2”:copies}, etc.) | Player 1’s decklist each game | ({“Plains”:1}, {“Swamp”:2, “Plains”:3}, {“Bear”:4, “Forest”:1, “Mountain”:3}) |
| deckListP1 |  | Blob |  | ({“cardName”:copies, “cardName2”:copies}, etc.) | Player 2’s decklist each game | ({“Stun Spell”:3,}, {“Trip Spell”:2}, {“Human”:2, “Archery Spell”:3, “Swamp”:3}) |
| extra |  |  |  |  |  |  |
| firstTurns |  | Blob | Presence | (player 1 name, player X name, …) | List of the player’s name which went first | (CoolGamer, CoolGamer, BoringGamer) |
| winLoss |  | Blob |  | (player X wins, player X wins, etc.) | The outcome of the match | (BoringGamer, CoolGamer, CoolGamer) |
| date |  | Text | Presence | "YYYY-MM-DD HH:MM:SS.SSS" | The date of the match | 2022-12-25 00:00:00.000 |

### Arrays of Records and 2D Arrays

## UML Class Design

## User-Interface Design

wireframes

# Implementation

## Python Files

### sync.py

import os

from os import walk

from mtgScraper import Scraper

#gets path of mtgo

localUser = os.getlogin()

path = f'C:\\Users\\{localUser}\\AppData\\Local\\Apps\\2.0\\Data\\'

#gets next two folders and adds them to path

for i in range(0,2):

    folder = os.listdir(path)

    path = path + '\\' + folder[0]

#gets the main mtgo folders, they are prefixed by 'mtgo..tion\_'

mainFolders = os.listdir(path)

scp = Scraper()

for folder in mainFolders:

    folderPath = f'{path}\\{folder}\\Data\\AppFiles'

    #gets name of the user

    scp.getPlayer(f'{folderPath}\\application\_settings')

    #gets the directories of the appfiles folder

    for (folderPath, directory, fileNames) in walk(folderPath):

        #.vs file may be in one of the directories, and is unnecessary

        if len(directory) > 1:

            directory.remove('.vs')

        #gets path of match files

        folderPath = folderPath + '\\' + directory[0]

        #don't look inside any subdirectory

        break

    #adds folder to list of folders to check

    scp.addPath(folderPath)

scp.run()

### mtgScraper.py

from matchRecord import MatchRecord

from mtgtop8 import DriverController

import os.path

from datetime import datetime

import urllib.request

import sqlite3

import json

import re

class Scraper():

    def \_\_init\_\_(self):

        self.paths = []

        self.player = ''

    def getPlayer(self, filename):

        try:

            with open('./playerName.txt', 'rb') as f:

                self.player = f.read().decode(encoding='utf-8', errors='replace')

        except:

            #reads file contents

            with open(filename, 'rb') as f:

                file = f.read().decode(encoding='utf-8', errors='replace').replace(' ', '').splitlines()

            #gets player name from file contents

            self.player = re.findall('"(.\*)"', file[file.index('Setting="LastLoginName"')+1])[0]

    def addPath(self, path):

        #adds path to object

        self.paths.append(path)

    def run(self):

        #opens connection

        self.openConn()

        #checks if database exists

        self.checkDB()

        top8Conn = self.checkInternet()

        #initilises modules

        match = MatchRecord()

        dc = DriverController()

        #gets list of files in database

        self.cursor.execute("SELECT filename FROM matches;")

        filenames = self.cursor.fetchall()

        filenames = [json.loads(i[0])['filename'] for i in filenames]

        for path in self.paths:

            #gets list of files in path

            fileList = [f for f in os.listdir(path) if f.endswith('.dat')]

            #loops through file list

            for filename in fileList:

                #if file already scraped

                if filename in filenames:

                    break

                #gets decklists from MatchRecord

                #other stored data is implemented into database in MatchRecord

                decklists, extra, players = match.getDecklists(f'{path}/{filename}')

                #if the file is valid

                if decklists != {} and decklists is not None:

                    players.remove(self.player)

                    #gets and reformats date

                    dateTime = str(datetime.fromtimestamp(os.path.getmtime(path + '\\' + filename)))

                    x, y, z = dateTime.split(' ')[0].split('-')

                    date = {'date':f'{z}/{y}/{x}'}

                    if top8Conn == True:

                        #gets the possible deck names from DriverController

                        deckName, matchLists = dc.returnDeckName(decklists, date)

                    else:

                        deckName = {'NA': 1.0}

                    #sends info to sqliite db

                    self.sqlliteDriverData({'filename':filename}, dateTime, deckName, extra, {'players':players}, matchLists)

        #closes webdriver

        dc.quitDc()

        #close connection

        self.closeConn()

    def openConn(self):

        self.userConnection = sqlite3.connect("./database/mtgoAssist.db")

        self.cursor = self.userConnection.cursor()

    def checkInternet(self):

        #tries to open mtgtop8.com, if it can't then program will not open later

        url = 'https://mtgtop8.com/'

        try:

            urllib.request.urlopen(url)

            conInt = True

        except:

            conInt = False

        return conInt

    def checkDB(self):

        try:

            self.cursor.execute("SELECT MAX(matchID) FROM matches;")

        except:

            #creates the matches table

            self.cursor.execute("""CREATE TABLE matches(

                                matchID INTEGER PRIMARY KEY,

                                filename BLOB,

                                opponent BLOB NOT NULL,

                                decknames BLOB,

                                decklistP1 BLOB,

                                decklistP2 BLOB,

                                play BLOB NOT NULL,

                                winner BLOB NOT NULL,

                                format BLOB,

                                type BLOB,

                                date BLOB NOT NULL);""")

            self.userConnection.commit()

    def sqlliteDriverData(self, filename, dateTime, deckName, extra, players, matchLists):

        #inserts match into database

        data = (json.dumps(filename), json.dumps(players), json.dumps(deckName), json.dumps({'P1':matchLists['P1']}), json.dumps({'P2':matchLists['P2']}), json.dumps({'play':extra['play']}), json.dumps({'winner':extra['winner']}), json.dumps({'date':dateTime}))

        self.cursor.execute("INSERT INTO matches(filename, opponent, decknames, decklistP1, decklistP2, play, winner, date) VALUES(?,?,?,?,?,?,?,?,?,?);", data)

        self.userConnection.commit()

    def closeConn(self):

        self.userConnection.commit()

        self.cursor.close()

        self.userConnection.close()

### matchRecord.py

import re

class MatchRecord:

    def \_\_init\_\_(self):

        pass

    def getDecklists(self, filename):

        decklists = {}

        extra = {'play':[], 'startingHands':[], 'winner':[]}

        #gets matchLog

        with open(filename, 'rb') as f:

            #unknown characters are replaced by \ufffd (those question marks)

            self.matchLog = f.read().decode(encoding='utf-8', errors='replace')

        #tries to create self.players

        try:

            self.getPlayers()

            if self.players == []:

                return None, None, None

        except (IndexError, ValueError):

            #if there is a problem reading the player names or the match is not 1v1 (fie invalid)

            return None, None, None

        #formats self.matchLog

        self.formatLines()

        #loops through each game

        for gameNo in range(1,len(self.matchLog)):

            #gets decklists from game

            gameDecklists = self.getDeckLists(self.matchLog[gameNo])

            if gameDecklists is None:

                break

            decklists[gameNo] = gameDecklists

            #gets player on play

            try:

                extra['play'].append(self.getOnPlay(self.matchLog[gameNo]))

            except:

                extra['play'].append('NA')

            #get number of cards in each player's hand

            try:

                extra['startingHands'].append(self.getStartingHands(self.matchLog[gameNo]))

            except:

                extra['startingHands'].append('NA')

            #gets winner of game

            extra['winner'].append(self.getWinner(self.matchLog[gameNo]))

        #if only two matches were played, then the person who was on the draw game 2, won the match

        if 'NA' in extra['winner'] and len(extra['play']) == 2:

            #gets player on draw in game 2

            player = [i for i in self.players if extra['play'][1] != i]

            #loops through each 'NA' result, and replaces it with the winner

            for indice in [i for i, elem in enumerate(extra['winner']) if elem == 'NA']:

                extra['winner'][indice] = player[0]

        #if 1 or fewer winners were recorded, and only two games were played, the player in the draw in game 2 won the match

        if len(extra['winner']) < 2 and len(extra['play']) == 2:

            #gets player on draw in game 2

            player = [i for i in self.players if extra['play'][1] != i]

            #while there are less than 2 recorded winners

            while len(extra['winner']) < 2:

                extra['winner'].append(player[0])

        return decklists, extra, self.players

    def getPlayers(self):

        #finds all players

        #the first player in the game log is always the user, or the second player in this list

        self.players = list(re.compile('@P(\S+) rolled').findall(self.matchLog))

        #if more than two players are detected, then the dice rolls where equal at least once in the game opening

        if len(self.players) > 2:

            while len(self.players) > 2:

                self.players.pop()

    def formatLines(self):

        #formats self.matchLog in [['random text', 'random text', 'player rolled', 'player rolled'], [game 1], [game 2...]] and removes noise

        #if there is a full stop in a player's name, then it must obtain to prefix, to prevent it being removed

        altPlayers = {}

        for player in self.players:

            if '.' in player:

                altPlayers[player] = []

                prefix = player.split('.')

                for count, pre in enumerate(prefix):

                    if count % 2 == 0:

                        altPlayers[player].append(pre)

        #removes all non-ascii characters and some non-random characters

        filteredMatch = re.split(r'[**\x00**-**\x08\x0b\x0c\x0e**-**\x1f\x7f**-**\xff**\ufffd**\{\}\|\\**=#**\^**><$]', self.matchLog)

        filteredMatch = [f'{line}.' for line in filteredMatch if len(line) > 4]

        #removes all full stops

        if len(altPlayers.keys())> 0:

            for player in altPlayers.keys():

                for pre in altPlayers[player]:

                    filteredMatch = [i for i in ''.join(filteredMatch).replace(' ',' {').split('{')]

                    filteredMatch = [i.replace('.', '}') if pre not in i and '.' in i else i for i in filteredMatch]

                    filteredMatch = ''.join(filteredMatch).split('}')

                    filteredMatch = [re.sub('^.\*@P', '@P', line) for line in filteredMatch if len(line) > 4]

        else:

            filteredMatch = ''.join(filteredMatch).split('.')

            filteredMatch = [re.sub('^.\*@P', '@P', line) for line in filteredMatch if len(line) > 4]

        #splits by game

        txt = ' '.join(filteredMatch)

        regex = f'@P{re.escape(self.players[1])} joined the game @P{re.escape(self.players[0])} joined the game|@P{re.escape(self.players[0])} joined the game @P{re.escape(self.players[1])} joined the game'

        filteredMatch = re.compile(regex).split(txt)

        #splits by turn

        regex = f'Turn [1-9]: {re.escape(self.players[0])}|Turn [1-5][0-9]: {re.escape(self.players[0])}|Turn [1-5][0-9]: {re.escape(self.players[1])}|Turn [1-9]: {re.escape(self.players[1])}'

        filteredMatch = [re.compile(regex).split(turn) for turn in filteredMatch]

        #removes last random characters

        filteredMatch = [[re.split('(@P)', turn) for turn in game] for game in filteredMatch]

        filteredMatch = {gameNo:{turnNo:[re.sub(r'@P|[^**\x00**-**\x7F**]', '', line) for line in turn if len(line) > 4] for turnNo, turn in enumerate(game)} for gameNo, game in enumerate(filteredMatch)}

        #deletes random characters at start of match

        del filteredMatch[0][0]

        self.matchLog = filteredMatch

    def getDeckLists(self, game):

        decklists = {self.players[0]: dict(), self.players[1]: dict()}

        #stores cards each player has played, revealed, discarded, cycled

        #game actions are formatted as @P(player\_name) (casts|plays|discards|cycles|reveals)

        #card names are formatted as @[Card Name@:numbers,numbers:@]

        playCardPattern = re.compile(f'({re.escape(self.players[0])}|{re.escape(self.players[1])}) (casts|plays|discards|cycles) (@\[([a-zA-Z\s,-]+)@:[0-9,]+:@\])')

        revealedCardPattern = re.compile(f'({re.escape(self.players[0])}|{re.escape(self.players[1])}) (reveals) (@\[([a-zA-Z\s,-]+)@:[0-9,]+:@\])')

        for turn in range(0, len(game.keys())-1):

            #finds matched patterns

            playCardMatches = playCardPattern.findall(' '.join(game[turn]))

            revealedMatches = revealedCardPattern.findall(' '.join(game[turn]))

            for actions in playCardMatches:

                #if a card has been revealed, and has interacted with the game, remove it from revealedMatches

                for match in revealedMatches:

                    if actions[3] in match:

                        revealedMatches.remove(match)

                #adds card to decklists

                if actions[3] in decklists[actions[0]]:

                    decklists[actions[0]][actions[3]] += 1

                else:

                    decklists[actions[0]].update({actions[3]:1})

            for revealed in revealedMatches:

                #adds card to decklists

                if revealed[3] in decklists[revealed[0]]:

                    decklists[revealed[0]][revealed[3]] += 1

                else:

                    decklists[revealed[0]].update({revealed[3]:1})

        return decklists

    def getOnPlay(self, game):

        # Who is on the play in this game?

        # Returns 'player' or 'opponent'

        text = ' '.join(game[0])

        onPlay = re.compile(f'({re.escape(self.players[0])}|{re.escape(self.players[1])})\ chooses\ to\ play\ first\ ').search(text).group(1)

        return onPlay

    def getStartingHands(self, game):

        #format [(player1, cards), (player2, cards)]

        regex = re.compile(f'({re.escape(self.players[0])}|{re.escape(self.players[1])}) begins the game with (no|a|two|three|four|five|six|seven) (?:card|cards) in hand | ({re.escape(self.players[0])}|{re.escape(self.players[1])}) puts (?:a|two|three|four|five|six|seven) (?:card|cards) on the bottom of their library and begins the game with (no|a|two|three|four|five|six|seven) cards in hand')

        startingHands = re.findall(regex, ' '.join(game[0]))

        startingHands = [[b for b in i if len(b) > 1] for i in startingHands]

        result = {i[0]:i[1] for i in startingHands}

        return startingHands

    def getWinner(self, game):

        #determines the winner or loser

        concededPattern = re.compile(f'({re.escape(self.players[0])}|{re.escape(self.players[1])}) has conceded')

        winsPattern = re.compile(f'({re.escape(self.players[0])}|{re.escape(self.players[1])}) wins the game')

        losesPattern = re.compile(f'({re.escape(self.players[0])}|{re.escape(self.players[1])}) loses the game')

        conceded = concededPattern.search(' '.join(game[len(game)-1]))

        wins = winsPattern.search(' '.join(game[len(game)-1]))

        loses = losesPattern.search(' '.join(game[len(game)-1]))

        if wins:

            return self.players[self.players.index(wins.group(1))][0]

        elif conceded:

            return [i for i in self.players if i not in self.players[self.players.index(conceded.group(1))]][0]

        elif loses:

            return self.players[self.players.index(loses.group(1))][0]

        return 'NA'

    def getWinner(self, game):

        #determines the winner or loser

        concededPattern = re.compile('(\S+) has conceded')

        winsPattern = re.compile('(\S+) wins the game')

        losesPattern = re.compile('(\S+) loses the game')

        conceded = concededPattern.search(' '.join([' '.join(i) for i in game]))

        wins = winsPattern.search(' '.join([' '.join(i) for i in game]))

        loses = losesPattern.search(' '.join([' '.join(i) for i in game]))

        if wins:

            return self.players[self.players.index(wins.group(1))][0]

        elif conceded:

            return [i for i in self.players if i not in self.players[self.players.index(conceded.group(1))]][0]

        elif loses:

            return self.players[self.players.index(loses.group(1))][0]

        else:

            return 'NA'

mtgTop8.py

from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.common.by import By

from selenium.webdriver.support import expected\_conditions as EC

from webdriver\_manager.chrome import ChromeDriverManager

from selenium.webdriver.support.select import Select

from selenium.webdriver.common.keys import Keys

class DriverController():

    def \_\_init\_\_(self):

        self.url = "https://mtgtop8.com/search"

        #adds options to the webdriver, in this case, to let webpage load, and bypass rate limiting

        driverOptions = webdriver.ChromeOptions()

        driverOptions.add\_argument('--ignore-certificate-errors')

        driverOptions.add\_argument('--ignore-ssl-errors')

        driverOptions.add\_argument('--start-maximized')

        driverOptions.headless = True

        #check validity of comment

        #downloads and runs a webdriver, that stops and is uninstalled after the program exits, using the options declared above

        chrome\_path = ChromeDriverManager().install()

        chrome\_service = Service(chrome\_path)

        self.driver = webdriver.Chrome(options=driverOptions, service=chrome\_service)

    def returnDeckName(self, deckLists, date):

        #calls the 'getSite()' method to open the url through the driver

        self.getSite()

        #calls the 'cookieBanner()' method to try and clear the cookie banner

        self.clearCookieBanner()

        deckLists = self.refactorDecklist(deckLists)

        #calls the 'inputFormData()' method to get all decks to be scraped

        self.inputFormData(deckLists['P1'][0], date)

        #gets the deck urls and names from the 'getDeckUrls()' method

        deckNames = self.getDeckNames()

        #if there are no decks found, then return deckNames (will have value of 'unknown')

        if deckNames == 'unknown':

            deckNames = {'NA', 1.0}

        #calls getDeckName() to create a dictionary of deckNames and %

        dictNames = self.getDictNames(deckNames)

        return dictNames, deckLists

    def getSite(self):

        self.driver.get(self.url)

    def inputFormData(self, deckList, date):

        #includes cards found in games played so far, as the what cards were sideboarded or not is impossible to find out

        self.driver.find\_element(By.XPATH, '//input[@name="SB\_check"]').click()

        #loops through the list of cards, writing each card into the <textarea>

        textarea = self.driver.find\_element(By.XPATH, '//textarea[@name="cards"]')

        #writes the cards to the textbox

        for card in deckList:

            textarea.send\_keys(card + Keys.RETURN)

        #sets date, so deck possibilites from after that game aren't considered (as they don't exist at the time of playing)

        self.driver.find\_element(By.XPATH, '//input[@name="date\_end"]').send\_keys(date['date'])

        #clicks the submit button of the form

        self.driver.find\_element(By.XPATH, '//td[@colspan="2"]/input[@type="submit"]').click()

    def clearCookieBanner(self):

        #if cookie banner present, click it, as it obstructs webdriver's view

        try:

            self.driver.find\_element(By.XPATH, '//\*[@id="cookie\_window"]/div[2]/button').click()

        except:

            pass

    def getDeckNames(self):

        deckNames = []

        #gets number of decks on page

        decks = len(self.driver.find\_elements(By.XPATH, '//td[@class="S12"]'))/3

        #if there are no decks on page, then return unknown as deckname, and quit DriverController()

        if decks < 1:

            return 'unknown'

        #sets number of decks to 5, if not under 5, so as to reduce time taken

        if decks > 5:

            decks = 5

        #loops through all decks on the page and gets their and name

        for deck in range(2, int(decks)+2):

            deckNames.append(self.driver.find\_element(By.XPATH, f'//body/div/div/table/tbody/tr/td[2]/form/table/tbody/tr[{deck}]/td[2]/a').text)

        return deckNames

    def quitDc(self):

        #stops the driver

        self.driver.quit()

    def refactorDecklist(self, deckLists):

        #reformats decklists

        cards = {}

        for game in deckLists:

            for i, player in enumerate(deckLists[game]):

                if i == 0:

                    tempPlayer = 'P2'

                else:

                    tempPlayer = 'P1'

                cards[tempPlayer] = [{}]

                for card in deckLists[game][player]:

                    cards[tempPlayer][0].update({f'{card}':deckLists[game][player][card]})

        return cards

    def getDictNames(self, deckNames):

        #creates a dictionary of deckNames

        dictNames = {'deckNames':[{deckName:0 for deckName in deckNames}]}

        #gets number of decks with same name

        for deckName in deckNames:

            dictNames['deckNames'][0][deckName] = dictNames['deckNames'][0][deckName] + 1

        #gets percentage chance of said deck

        for deckName in dictNames['deckNames'][0]:

            dictNames['deckNames'][0][deckName] = dictNames['deckNames'][0][deckName] / len(deckNames)

        return dictNames

### dbCMD.py

import sqlite3

import sys

import json

#gets data from database to be displayed on table

def openData():

    userConnection = sqlite3.connect("./database/mtgoAssist.db")

    cursor = userConnection.cursor()

    try:

        matches = cursor.execute("SELECT \* FROM matches ORDER BY matchID DESC;").fetchall()

        results = {}

        for matchTuple in matches:

            results[matchTuple[0]] = [{}]

            for data in matchTuple:

                if isinstance(data, int) == False:

                    data = json.loads(data)

                    results[matchTuple[0]][0][list(data.keys())[0]] = data[list(data.keys())[0]]

        print(json.dumps(results))

    except:

        pass

    finally:

        cursor.close()

        userConnection.close()

#gets user averages

def getUserData():

    try:

        with open('python\playerName.txt', 'rb') as f:

            player = f.read().decode(encoding='utf-8', errors='replace')

    except:

        print('NA')

        return

    userConnection = sqlite3.connect("./database/mtgoAssist.db")

    cursor = userConnection.cursor()

    matches = cursor.execute("SELECT winner FROM matches;").fetchall()

    results = {}

    for num, match in enumerate(matches):

        results[num] = json.loads(match[0])

    winList = []

    count = 0

    for match in results:

        for res in results[match]['winner']:

            if res != 'NA':

                if res == player:

                    count += 1

                winList.append(res)

    print(count / len(winList))

    cursor.close()

    userConnection.close()

def getOppWinrate(opponent):

    try:

        with open('python\playerName.txt', 'rb') as f:

            player = f.read().decode(encoding='utf-8', errors='replace')

    except:

        print('NA')

        return

    userConnection = sqlite3.connect("./database/mtgoAssist.db")

    cursor = userConnection.cursor()

    matches = cursor.execute("SELECT opponent, winner FROM matches ORDER BY opponent;").fetchall()

    #[[opponent, [name, name...]],[]]

    results = []

    for num, match in enumerate(matches):

        results.append([])

        for nom, item in enumerate(match):

            if nom == 0:

                results[num].append(json.loads(item)['players'][0])

            else:

                results[num].append(json.loads(item)['winner'])

            #results[num] = match[0]

    record = []

    while True:

        low = 0

        high = len(results) - 1

        mid = 0

        while low <= high:

            mid = (high + low) // 2

            if results[mid][0] < opponent:

                low = mid + 1

            elif results[mid][0] > opponent:

                high = mid - 1

            else:

                for win in results[mid][1]:

                    record.append(win)

                results.pop(mid)

        #element not present

        break

    count = 0

    for play in record:

        if play == player:

            count += 1

    if count != 0:

        print(count / len(record))

    else:

        print(count)

#getOppWinrate('Pazmaster')

if \_\_name\_\_ == "main":

    pass

else:

    if sys.argv[1] == 'loaded':

        openData()

    elif sys.argv[1] == 'profile':

        getUserData()

    elif sys.argv[1] =='opponent':

        getOppWinrate(sys.argv[2])

## Javascript Files

### renderer.js

function runPythonSync(){

  var pyshell =  require('python-shell');

  var pjson = require('./package.json');

  let options = {

    mode: 'text',

    pythonOptions: ['-u'],

    args: [pjson.version]

  }

  pyshell.PythonShell.run('./python/sync.py', options, function  (err, results)  {

    if (err)  throw err;

    console.log('sync.py finished.');

  });

}

function loadFirstPage(){

  var pyshell =  require('python-shell');

  var pjson = require('./package.json');

  let options = {

    mode: 'json',

    pythonOptions: ['-u'],

    args: ["loaded", pjson.version]

  }

  pyshell.PythonShell.run('./python/dbCMD.py', options, function  (err, results)  {

    console.log(results);

    let keys = Object.keys(results[0])

    if (results){

      let leng = Object.keys(results[0]).length;

      for (let i = 0; i < leng; i++){

        let append = `<tr>

                <th class="matchID" id="row${i}">${Object.keys(results[0])[i]}</th>

                <th class="filename"><p class="hidden">${results[0][keys[leng-i-1]][0]['filename']}</p><button class="replaceImageFilename">...</button></th>

                <th class="players"><p class="replaceTextOpponent">${results[0][keys[leng-i-1]][0]['players'][0]}</p><button class="replaceImageOpponent">...</button></th>

                <th class="decknames"><p class="replaceText">${JSON.stringify(results[0][keys[leng-i-1]][0]['deckNames'][0]).replace(/[[**\]**]|"|{|}/g, '').replace(/:/g, ': ').replace(/,/g, '<br>')}</p><button class="replaceImage">...</button></th>

                <th class="p1Deck"><p class="replaceTextDeck">${structureDecklist(results, keys, leng, i, 'P1')}</p><button class="replaceImageDeck">...</button></th>

                <th class="p2Deck"><p class="replaceTextDeck">${structureDecklist(results, keys, leng, i, 'P2')}</p><button class="replaceImageDeck">...</button></th>

                <th class="turnOrder"><p class="replaceTextExtra">${JSON.stringify(results[0][keys[leng-i-1]][0]['play']).replace(',','<br>').replace(/[[**\]**]|"/g, '')}</p><button class="replaceImageExtra">...</button></th>

                <th class="winList"><p class="replaceTextExtra">${JSON.stringify(results[0][keys[leng-i-1]][0]['winner']).replace(',','<br>').replace(/[[**\]**]|"/g, '')}</p><button class="replaceImageExtra">...</button></th>

                <th class="date">${JSON.stringify(results[0][keys[leng-i-1]][0]['date']).replace('"','').split(' ')[0]}</th>

              </tr>`

        document.getElementById('databaseContent').insertAdjacentHTML('beforeend', append);

      }

    }

    if(err) throw err;

  });

}

function structureDecklist(results, keys, leng, i, player){

  let deck = JSON.stringify(results[0][keys[leng-i-1]][0][player][0]).replace(/[[**\]**]|"|{|}/g, '').split(/:|,/g)

  let res = ''

  for (let b = 0; b < deck.length; b++){

    if (b % 2 == 0){

      res += deck[b+1] + ' ' + deck[b] + '<br>'

    }

  }

  return res

}

function loadProfile(){

  document.getElementById('mid').style.display = 'none';

  document.getElementById('bottom').style.display = 'none';

  document.getElementById('profile').style.display = 'block';

  var pyshell =  require('python-shell');

  var pjson = require('./package.json');

  let options = {

    mode: 'text',

    pythonOptions: ['-u'],

    args: ["profile", pjson.version]

  }

  pyshell.PythonShell.run('./python/dbCMD.py', options, function  (err, results)  {

    console.log(results);

    document.getElementById('winrate').innerHTML = results;

    if(err) throw err;

  });

}

function getOpponentWinrate(){

  let name = document.getElementById('oppName').value;

  var pyshell =  require('python-shell');

  var pjson = require('./package.json');

  let options = {

    mode: 'text',

    pythonOptions: ['-u'],

    args: ["opponent", name, pjson.version]

  }

  pyshell.PythonShell.run('./python/dbCMD.py', options, function  (err, results)  {

    console.log(results);

    document.getElementById('opponentWinrate').innerHTML = results;

    if(err) throw err;

  });

}

function closeProfile(){

  document.getElementById('mid').style.display = 'block';

  document.getElementById('bottom').style.display = 'block';

  document.getElementById('profile').style.display = 'none';

}

document.querySelector('#pyBtnProfile').addEventListener('click', () => {

  loadProfile();

})

document.querySelector('#close').addEventListener('click', () => {

  closeProfile();

})

document.querySelector('#pyBtnSync').addEventListener('click', () => {

  runPythonSync();

})

document.addEventListener('DOMContentLoaded', () =>{

  loadFirstPage();

})

document.querySelector('#submitUser').addEventListener('click', () => {

  getOpponentWinrate();

})

### preload.js

window.addEventListener('DOMContentLoaded', () => {

    const replaceText = (selector, text) => {

      const element = document.getElementById(selector)

      if (element) element.innerText = text

    }

    for (const type of ['chrome', 'node', 'electron']) {

      replaceText(`${type}-version`, process.versions[type])

    }

})

### index.js

const { app, BrowserWindow } = require('electron')

const path = require('path')

const createWindow = () => {

    const win = new BrowserWindow({

      width: 800,

      height: 600,

      webPreferences: {

        nodeIntegration: true,

        contextIsolation: false,

        preload: path.join(\_\_dirname, 'preload.js'),

      }

    })

    win.loadFile('index.html')

  }

app.whenReady().then(() => {

    createWindow()

})

app.on('window-all-closed', () => {

    if (process.platform !== 'darwin') app.quit()

})

## Electron Backend

## UI Files

### index.html

<html>

  <head>

    <title>MTGO Assist</title>

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet" href="./css/index/bottom.css">

    <link rel="stylesheet" href="./css/index/top.css">

    <link rel="stylesheet" href="./css/index/mid.css">

    <link rel="stylesheet" href="./css/index/main.css">

  </head>

  <body>

    <section id="top">

        <section id="bar">

            <section id="topbar">

                <ul class="listBar">

                    <li class="listBarItem">Hello</li>

                </ul>

            </section>

            <section id="navbar">

                <ul class="listBar">

                    <li class="listBarItemNav" id="profileListItem"><button id="pyBtnProfile"><img src="./profile.png" id="profileImage"></button></li>

                    <li class="listBarItemNav"><button id="pyBtnSync">Sync</button></li>

                </ul>

            </section>

        </section>

    </section>

    <section id="mid">

        <section id="databaseSection">

                <table>

                    <tbody id="databaseContent">

                        <tr id="colNames">

                            <th class="tableHeader"><label id="matchID"><label id="matchText">Match </label>ID</label></th>

                            <th class="tableHeader"><label>File<label id="fileText">name</label></label></th>

                            <th class="tableHeader"><label>Opponent</label></th>

                            <th class="tableHeader"><label>Deck<label class="extraHeaderText">name</label>s</label></th>

                            <th class="tableHeader"><label class="altHeaderText">P1</label><label class="extraHeaderText">Player 1 Decklist</label></th>

                            <th class="tableHeader"><label class="altHeaderText">P2</label><label class="extraHeaderText">Player 2 Decklist</label></th>

                            <th class="tableHeader"><label class="extraHeaderTextExtra">Turn </label><label>Order</label></th>

                            <th class="tableHeader"><label id="altHeaderTextExtra">Wins</label><label class="extraHeaderTextExtra">Win List</label></th>

                            <th class="tableHeader"><label>Date</label></th>

                        </tr>

                    </tbody>

                </table>

        </section>

    </section>

    <section id="bottom">

        <div id="nextPage">

                <button id="pyBtnNextPage">Next Page</button>

        </section>

    </section>

    <section id="profile" style="display:none;">

        <div id="results">

            <p>Winrate: <label id="winrate"></label>%</p><br>

            <p>Opponent Username:

                <input type="text" id="oppName">

                <button id="submitUser">Submit</button>

            </p>

            <p>Winrate against opponent: <label id="opponentWinrate"></label>%</p><br><br>

        </div>

        <button id="close">Close</button>

    </section>

    <script>

      require('./renderer.js')

      require('./preload.js')

    </script>

  </body>

</html>

### bottom.css

#nextPage {

    display: inline-block;

    position: absolute;

    width: 99%;

    text-align: center;}

#pyBtnNextPage {

    color:black;}

### mid.css

table,td,th{

    border:1px solid black;}

table{

    border-collapse: collapse;}

#databaseSection {

    color:black;}

.tableHeader {

    padding-left:1.3vw;

    padding-right:1.3vw;

    white-space: nowrap;

    max-width:200px;}

.matchID, .filename, .players, .decknames, .p1Deck, .p2Deck, .turnOrder, .winList, .format, .type, .date{

    max-width:200px;

    word-wrap:break-word;}

.filename{

    max-width:100%;

    white-space:nowrap;}

.replaceImage, .altHeaderText, #altHeaderTextExtra, .replaceImageOpponent, .replaceImageDeck, .replaceImageExtra, .hidden{

    display: none;}

.decknames{

    white-space: nowrap;}

#matchText, #fileText, .extraHeaderText, .replaceText, .replaceTextDeck{

    display:inline-block;}

.winRecord{

    background-color: #59cf78;}

@media only screen and (max-width:1600px){

    #matchtext, #fileText{

        display:none;}}

@media only screen and (max-width:1480px){

    .extraHeaderTextExtra, .replaceTextExtra{

        display:none;}

    #altHeaderTextExtra, .replaceImageExtra{

        display:inline-block;}}

@media only screen and (max-width:560px){

    #matchID, .matchID, tr th:nth-child(1){

        display:none;}}

@media only screen and (max-width:1280px){

    .replaceTextOpponent{

        display:none;}

    .replaceImageOpponent{

        display:inline-block;}}

@media only screen and (max-width:1240px){

    .replaceTextDeck{

        display:none;}

    .replaceImageDeck{

        display:inline-block;}}

@media only screen and (max-width:1090px){

    .tableHeader, .matchID, .players, .decknames, .p1Deck, .p2Deck, .turnOrder, .winList, .format, .type, .date{

        font-size:14px;}}

@media only screen and (max-width:960px){

    .tableHeader, .matchID, .players, .decknames, .p1Deck, .p2Deck, .turnOrder, .winList, .format, .type, .date{

        font-size:12px;}}

@media only screen and (max-width:830px){

    #type, .type, tr th:nth-child(10){

        display:none;}}

@media only screen and (max-width:750px){

    .tableHeader, .matchID, .players, .decknames, .p1Deck, .p2Deck, .turnOrder, .winList, .format, .type, .date{

        font-size:12px;}

    .tableHeader {

        padding-left:1.3vw;

        padding-right:1.3vw;

        white-space: nowrap;

        max-width:200px;}

    .matchID, .players, .decknames, .p1Deck, .p2Deck, .turnOrder, .winList, .format, .type, .date {

        word-wrap:break-word;}

    .extraHeaderText, .replaceText{

        display:none;}

    .altHeaderText, .replaceImage{

        display: inline-block;}}

### top.css

/\* #top, body{width:1000%} \*/

#bar{

    height: 60px;

    width:auto;

    background-color: rgb(134, 134, 134);}

#topbar, #navbar {

    height: 60px;}

#topbar {

    width:30%;

    float:left;}

.listBar {

    list-style-type: none;

    padding: 0px;

    margin: auto;

    height:100%;}

.listBarItem, .listBarItemNav {

    text-align: center;

    vertical-align: middle;

    padding:20px 30px;}

.listBarItem {

    float: left;

    font-size:1.4vw}

.listBarItem a {

    display: block;

    color: rgb(0, 0, 0);

    text-align: center;

    padding: 14px 16px;

    text-decoration: none;}

.listBarItemNav {

    float: right;}

#navbar {

    width:70%;

    float:right;

    background-color: rgb(134, 134, 134);}

#pyBtnSync {

    color:black;}

#profileImage {

    width: 40px;

    height: auto;}

#profileListItem {

    padding-top:7px;

    padding-left: 30px;}

#colNames {

    width:100%;}

#lowHeader {

    color:black;}

@media only screen and (max-width:1250px){

    .listBarItem {

        float: left;

        font-size:16px;}

}

### main.css

h1 {

    color:black;}

# Testing

## Final Test Plan

## Requirements Testing

## Testing with Persona and Use Cases

# Evaluation

## Post-Development User Survey

# Appendix 1 – Pre-Development User Survey Results

## Chart, bar chart Description automatically generatedQ1

Results: Most potential end-users are under 18 to 24, so app design doesn’t have to account for many elderly people.

## Graphical user interface, text, application, email, website Description automatically generatedQ2

Numerical Results: 5, 50+, 34, 1, 23, 1, 4, 6, 14, 2, 0, 0, 10, 10, 12, 10, 10, 31, 17, 18, 24, 16

Mean: 14 (2 s.f.)

Results: Most potential end-users play videogames a fair amount.

## Chart Description automatically generatedQ3

Results: Most potential end-users would want the app to display feedback on their game.

## Q4

Chart

Description automatically generated

Results: Most potential end-users would want the app to try and help them improve.

## Chart, bar chart Description automatically generatedQ5

Results: Most potential end-users want a combination of ways to display information stored by the app.

## Chart, bar chart Description automatically generatedQ6

Results: Cost is a barrier of entry for most potential end-users, this confirms that the app should be free, to enable as many users to use it as possible.

## Chart, bar chart Description automatically generatedQ7

Results: Most potential end-users would be interest in installing the finished app.

# Appendix 2 – Post-Development User Survey Results

Appendix 3