AH SDD Project: Game Assist Tool

Contents

[Analysis 2](#_Toc117847060)

[Purpose 2](#_Toc117847061)

[Pre-Development User Survey 3](#_Toc117847062)

[Constraints 4](#_Toc117847063)

[UML Use Case Diagram 5](#_Toc117847064)

[Actors 5](#_Toc117847065)

[Use case 5](#_Toc117847066)

[Relationships 5](#_Toc117847067)

[Requirements Specification 5](#_Toc117847068)

[End-User Requirements 5](#_Toc117847069)

[Functional Requirements 6](#_Toc117847070)

[Inputs, Processes and Outputs 7](#_Toc117847071)

[Notes on Further Input Validation 9](#_Toc117847072)

[Personas 9](#_Toc117847073)

[Sean Greaves 9](#_Toc117847074)

[Winnie Manning 9](#_Toc117847075)

[James Hill 10](#_Toc117847076)

[User Stories 10](#_Toc117847077)

[User Scenarios 10](#_Toc117847078)

[Project Plan – Initial estimated version 11](#_Toc117847079)

[Project Plan – Final version 14](#_Toc117847080)

[Design 17](#_Toc117847081)

[Pseudocode Design 17](#_Toc117847082)

[UML Class Design 19](#_Toc117847083)

[Project Design 19](#_Toc117847084)

[User-Interface Design 19](#_Toc117847085)

[Appendix 1 – Pre-Development User Survey Results 20](#_Toc117847086)

[Q1 20](#_Toc117847087)

[Results 20](#_Toc117847088)

[Q2 20](#_Toc117847089)

[Numerical Results 20](#_Toc117847090)

[Mean 20](#_Toc117847091)

[Results 20](#_Toc117847092)

[Q3 21](#_Toc117847093)

[Results 21](#_Toc117847094)

[Q4 21](#_Toc117847095)

[Results 21](#_Toc117847096)

[Q5 22](#_Toc117847097)

[Results 22](#_Toc117847098)

[Q6 22](#_Toc117847099)

[Results 22](#_Toc117847100)

[Q7 23](#_Toc117847101)

[Results 23](#_Toc117847102)

[Appendix 2 – Post-Development User Survey Results 23](#_Toc117847103)

# 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, and, trending win rate and useful additional statistics, 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.
* 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, 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, as well as procedural programming with 2D arrays and arrays of records.
* The project will also use a bubble sort algorithm
* Other Advanced Higher Concepts:
  + OOP (Object-Oriented Programming)
  + External database manipulation
* 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)

## 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 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 the design decisions should be followed, and the assumptions about potential end-users were correct.
* 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.

## 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

### Actors

### Use case

### Relationships

## 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 able to filter results.
* 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 filtered and non-filtered match data from the local database.
    - The app must be able to filter most database fields.
    - The app must be able to display the average deck for each player in each match.
    - The app must be able to display additional, curated information when certain database fields are filtered (e.g. win rate when using card, when a card is filtered).
    - 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 or filtered data to a .csv file.
    - The app must be able to display graphs of filtered and non-filtered 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 must let the user update or change their account name, password and optional MTGO name.
    - 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

#### App Load

|  |  |  |
| --- | --- | --- |
| Inputs | Processes | Outputs |
|  | Check whether a user account has been created, if so, then show Create Account Dialog Boxes | Optional create account dialog box |

#### Main App Page

|  |  |  |
| --- | --- | --- |
| Inputs | Processes | Outputs |
| Sync Button (Click input) | Run Check Connection, run File Reader | File Reader: arguments (MTGO name (string), the highest record ID in the local database (integer), whether connected to internet)  User: Possible error message |
| Filter Results Form (Click and Keyboard inputs) | Creates SQL query based on form data | Shows filtered results |
| Upload to CSV Button (Click input) | Creates a CSV file from current displayed results, which is downloaded to the user’s device. | CSV file (.csv) |
| Edit Button (Click and Keyboard inputs) | Updates database record/s | Shows edited results |
| Download Graph Button (Click input) | Creates a .png file of the graph, which is downloaded to the user’s device. | Image file (.png) |

#### Create Account Dialog Boxes

|  |  |  |
| --- | --- | --- |
| Inputs | Processes | Outputs |
| Optional MTGO Account Name (Keyboard Input) | Stores the input in a string variable if it passes a presence check, else, store none in a variable | End dialog box |
| App Name (Keyboard Input) | Stores the input in a string variable if it passes a presence check | Next dialog box |
| App Password (Keyboard Input) | Stores the input in a string variable if it passes a presence check | Next dialog box |
| OK and Cancel (Click input) | Repeats Dialog questions asked if OK is clicked, else, it stores input variables in integrated local database |  |

#### Check Connection

|  |  |  |
| --- | --- | --- |
| Inputs | Processes | Outputs |
|  | Checks connection to the internet | Boolean value of whether connected to internet or not (True, False) |

#### File Reader

|  |  |  |
| --- | --- | --- |
| Inputs | Processes | Outputs |
| Sync Button Arguments | 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: deck lists (array of sets) and date (string) |

#### 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 files, when reading a file, that file is ignored if it doesn’t meet expected formatting (all match data is stored the same way).
* Data is sanitised 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 (as I am a visual learner), so I want it to be displayed visually and 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. The tool solves these problems by displaying manipulatable graphs, has an automatic and consistent format, and helps the user understand their 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. The tool does this by relating win rate of decks to a card being present, and further calculations, to quickly get and clearly display most contribute to success.

## Project Plan – Initial estimated version

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

## Project Plan – Final version

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

# Design

|  |  |  |  |
| --- | --- | --- | --- |
| Table | Column Name | Datatype | Description |
| Matches | matchID | Integer | Unique surrogate key |
|  | filename | String | Name of file |
|  | p1 | String | Player 1 username (name of end-user if specified) |
|  | p1Deckname | String | Player 1 deck name |
|  | p2 | String | Player 2 username |
|  | P2Deckname | String | Player 2 deck name |
|  | p1Rolls | Array (could be infinite) |  |
|  | p2Rolls | Array (could be infinite) |  |
|  | p1Wins | Integer |  |
|  | p2Wins | Integer |  |
|  | format | String |  |
|  | matchType | String |  |
|  | Date | Date |  |

Note: Account Name and Account Password are present for the future possibility of the app being updated, so that users can see other users’ results.

## Pseudocode Design

* Inputs
  + Keyboard Inputs: App Username, App Password, MTGO Username
  + File Inputs: MTGO match logs (.dat files), Renderer.js arguments (sent to python files, to specify what they should carry out, when run)
  + Web driver (ChromeDriver) Inputs: Automated Headless (browser without a graphical interface) Browser (To web scrape data)
  + Mouse Inputs: Navigation of App, Dialog Boxes (Popup alerts and messages)
* Processes

Renderer.js

Event Listeners

App Loaded

* Runs runPythonDB function.

Sync Button Clicked

* Runs runPythonSync function.

Reload Button Clicked

* Runs runPythonDB function.

Functions

runPythonSync

runPythonCreateUser

* Runs dbCMD.py with “createUser” and the user inputs, passed into the function.
* If an error occurs while the python file is run, an error is thrown to the JS console.

runPythonDB

* Runs the dbCMD.py file with “loaded” as an argument.
* If the first output of the python file is “unconnectedDB”, the user details database doesn’t exist yet, so prompts the user for the required data.
  + The dialog box prompts the user to input a username, then password, then MTGO name (which is optional).
  + Runs runPythinCreateUser function with user inputs passed in.
* If the second output of the python file is “unconnectedInt”.
  + The reload button is shown and the sync button is hidden.
* If the second output of the python file is “connectedInt”.
  + The reload button is hidden and the sync button is shown.
* If an error occurs while the python file is run, an error is thrown to the JS console.
  + Outputs

## UML Class Design

## Project Design

## User-Interface Design

Implementation

End Testing

Evaluation

Future: Limited

# 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