AH SDD Project: Game Assist Tool

Contents

[Analysis 2](#_Toc115082306)

[Purpose 2](#_Toc115082307)

[Pre-Development User Survey 3](#_Toc115082308)

[Constraints 3](#_Toc115082309)

[UML Use Case Diagram 3](#_Toc115082310)

[Actors 3](#_Toc115082311)

[Use case 3](#_Toc115082312)

[Relationships 3](#_Toc115082313)

[Requirements Specification 4](#_Toc115082314)

[End-User Requirements 4](#_Toc115082315)

[Functional Requirements 4](#_Toc115082316)

[Personas 4](#_Toc115082317)

[Sean Greaves 4](#_Toc115082318)

[Winnie Manning 4](#_Toc115082319)

[3rd person (Least comp) 5](#_Toc115082320)

[User Stories 5](#_Toc115082321)

[User Scenarios 5](#_Toc115082322)

[Project Plan – Initial estimated version 5](#_Toc115082323)

[Project Plan – Final version 8](#_Toc115082324)

[Design 10](#_Toc115082325)

[Pseudocode Design 10](#_Toc115082326)

[UML Class Design 10](#_Toc115082327)

[Project Design 10](#_Toc115082328)

[User-Interface Design 10](#_Toc115082329)

# 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 (% 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, 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 is 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 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 a 2D array and array of records.
* The project will also use a BINARY sort algorithm
* Other Advanced Higher Concepts:
* The app is integrated with:
  + A database (local) with multiple tables, and open (and close) the database connection 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.
* MOREEEE?????

## 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.

## 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.
* 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.
  + 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).

## 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 results that have an unusual deck uncertainty (the probable deck a player was player) and the match logs did not specify if a player won a match.

### 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 let the user edit certain data (data which is empty from match logs and a less probable player deck, which the app assigned), which is updated in the local database.
    - 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.

### Inputs, Processes and Outputs

Main App Page

|  |  |  |
| --- | --- | --- |
| Inputs | Processes | Outputs |
| Sync Button (Click input) | Check if connected to database, then if it is, upload file results to database with additional web scraped information, else hide self, show reload button, and show error message | Optional error message |
| Reload Button (Click input) | Check if connected to database, if it is then show sync button and hide self, else show error message | Optional 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 | CSV file |
| Edit Button (Click and Keyboard inputs) | Updates database record/s | Shows edited results |
|  |  |  |

Create Account Dialog Box

|  |  |  |
| --- | --- | --- |
| Inputs | Processes | Outputs |
| Optional MTGO Account Name (Keyboard Input) | Stores input in local database | End dialog box |
| App Name (Keyboard Input) | Stores input in local database | Next dialog box |
| App Password (Keyboard Input) | Stores input in local database | Next dialog box |

^^Input Validation

Integration

DATA STRUCTURE

Intangible benefits

Sequential, parallel, which tasks can be performed. Gantt chart why.

## 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.
* 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 often x, he needs so that he can

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

## 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 | 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 |  |  |  |
| 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 matchID |
|  | filename | String | Name of file |
|  | p1 | String | Player 1 username (name of end-user if specified) |
|  | p1Deckname | String | Player 1 deck name |
|  | p2 | String |  |
|  | P2Deckname | String |  |
|  | p1Rolls | Array (could be infinite) |  |
|  | p2Rolls | Array (could be infinite) |  |
|  | p1Wins | Integer |  |
|  | p2Wins | Integer |  |
|  | format | String |  |
|  | matchType | String |  |
|  | Date | Date |  |

## 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

Appendix 1 – User Survey Results

Q1

Chart, bar chart

Description automatically generated

Results – Most potential end-users are under 18 to 24, so app design doesn’t have to account for a large number of elderly people.

Q2

Graphical user interface, text, application, email, website

Description automatically generated

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.

Q3

Chart

Description automatically generated

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

Q4Chart

Description automatically generated

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

Q5

Chart, bar chart

Description automatically generated

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

Q6

Chart, bar chart

Description automatically generated

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.

Q7

Chart, bar chart

Description automatically generated

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

Appendix 2

Appendix 3