AH SDD Project: Game Assist Tool

Contents

[Analysis 1](#_Toc113542233)

[Description of problem 1](#_Toc113542234)

[User Survey 2](#_Toc113542235)

[Scope, Boundaries and Constraints 2](#_Toc113542236)

[UML Use Case Diagram 2](#_Toc113542237)

[Requirements Specification 2](#_Toc113542238)

[End-User Requirements 2](#_Toc113542239)

[Functional Requirements 2](#_Toc113542240)

# Analysis

## Description of problem

* No maintained tool exists which records match data on the game Magic: The Gathering Online. This leaves users to record data manually on an excel document, if at all, which is very time consuming.
* The end users are people who play Magic: The Gathering Online, they are fairly technically adept (as they play an online game), their age range is roughly 18-45. This is relevant to the project because the end users spend a decent to a vast amount of money on the program, sometimes to maximise their win percentage, and are capable of installing, and using, an assist tool.
* The app is built on Electron. On the app, a python file can be run which navigates a specific app (Magic: The Gathering Online), by reading the screen. It navigates to the game history, from which it reads each record of data, each record contains the information of a match (best of 3 games), stored in a server-side video. The app uses Tesseract to read text on the screen. For each game it starts a loop and watches the game and reads all information given, which it categorises and stores in a local database (sqlite)When the loop is exited, it also reads and stores information from a website (using selenium, also name of python library used here), if internet connection is established and compares that data to some of the data from the match to determine which “deck” each player is playing in the match, which it stores in the local database. All information stored in the local database is easily readable in the Electron app.
* The project meets these requirements:
  + Is an SDD project integrated with a local database.
  + The project will use a sort algorithm, use procedural programming, an array of records, create a local database (if it doesn’t already exist) and update, select, delete records (based on what the end-user selects) from that database.

## User Survey

## Scope, Boundaries and Constraints

* Scope
* Boundaries
* Constraints
  + Feasibility Study
    - Economic
      * There are no costs associated with the project as it uses free software. It also generates no revenue. However, it can be further built upon and in such a way that can create revenue.
    - Time
      * The project will take 80 hours to complete.
    - Functional

## UML Use Case Diagram

## 

## Requirements Specification

### End-User Requirements

### Functional Requirements

## Project Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | | Start Date | End date | Resources Required |
| Analysis | | | | |
| 1. | Description of problem | 27/08/22 | 29/08/22 |  |
| 2. | Create user survey | 31/08/22 | 05/09/22 | Survey monkey – create account |
| 3. | Scope, boundaries & constraints |  |  |  |
| 4. | UML Use Case diagram |  |  | Using Lucidchart |
| 5. | Create persona and user stories |  |  |  |
| 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. | Testing with personas and test cases |  |  |  |
| Evaluation | | | | |
| 19. | Evaluation Report |  |  |  |