Team Proposal Report – Matrices Quiz Tool

Team 13: Aishwarya Bandaru, Hannah Marie James, Michael Fan, Nandini Chavda

Introduction

Our educational tool will be used to test students on their knowledge of the topic Matrices. Through randomly generated mixed-practice quizzes, the tool will test the students on the following sub-topics:

- Matrix addition
- Matrix subtraction
- Finding the determinants of a 2x2 or 3x3 matrix
- Matrix multiplication
- Finding the inverse of a 2x2 or 3x3 matrix
- Eigenvectors and eigenvalues of a 2x2 or 3x3 matrix.

Our tool also incorporates a revision notes feature, in the form of a formular sheet, for these sub-topics which can allow students to go over the quiz content before and/or after attempting the quiz. For the quiz, the student can choose a difficulty between easy, medium, and hard. Each difficulty level contains pre-selected sub-topics for a set matrix size. For example, level easy includes addition, subtraction, multiplication, determinant, and inverse questions for 2x2 matrices.

The student can also select how many questions they want to attempt out of 5, 10, or 20. At the end of the quiz, the questions they got correct and incorrect will be displayed along with the correct solution. The user will be given an overall score depending on the number of correct answers compared to the total number of questions. Students can re-attempt the quiz to receive a different set of random questions to test if they have improved. They will be able to view their scores from past attempts to track their progress by using the leader board feature of our tool.

This tool is aimed at A-levels students, those in college and undergraduate students. The tool will give them the opportunity to improve their understanding of the topic of matrices. It can also help students to identify their weakest sub-topics to focus their revision on. The revision pages and the quiz can be used as a supplement for their revision.

Functional Requirements

To conceptualise the tool better, a set of requirements were determined alongside their priorities to see which elements and features were within the scope of the project. Table 1 outlines these using the MoSCoW method.

Table 1: The tool's functional requirements their priorities for implementation.

ID	Functional Requirements	Priority
1	Display the main menu to the user, where the user can select either to do a quiz or revise.	S
2	If "revise" is selected, display revision notes on the matrices topic and sub-topics the quiz tests the user on.	C
3	The user will be able to select number of questions they want to answer.	S
4	The user will be able to select what difficulty of questions they want to be quizzed on.	S
5	The program should display new questions to the user by generating new, random matrices.	M
6	The program should carry out calculations to calculate the correct solutions to the questions.	M
7	The program will store the questions in a text file called "questions.txt".	S
8	The program should take in input, the user's answers, as integers.	M
9	The program will compare the user's answer with the true solution and append to a list, if whether the user got the question correct or the true solution if the user's answer was incorrect.	S
10	The program will calculate the user's score for the attempt as a percentage.	M
11	At the end of an attempt, display to the user all the questions they were tested on, if whether they got the answer correct or incorrect and the correct solution if they answered incorrectly.	C
12	The program should store the username, score, time taken to do the quiz and date in a textfile called "leaderboard.txt", in one line.	C
13	At the end of the quiz, the "leaderboard.txt" file should be read and displayed line by line to show the recent 5 previous attempts.	C
14	The "leaderboard.txt" stores the top 10 scores across multiple players so users can compete with other people and compare eachothers' progress.	W
15	The quiz can be printed out to be used as worksheets.	W

Framework/Language

Our tool will be programmed in Python using object-orientated programming. The following classes will be used: Calculator, Questions, UI, Leader board.

We decided to use Python due to its extensive software ecosystem and ease of debugging. Python provides the modules NumPy and PyQt. NumPy makes generating and calculating with matrices easier and efficient. It also increases our scope for implementing other features. PyQt is used to create an easy-to-follow graphical user interface. The downside of Python, is its slow performance, will not be a problem for our project.

Implementation Plan

The flowchart in Figure 1 summarises the implementation process. The chosen data structures and pre-defined methods from Figure 1 (for generating random questions, checking the user's solution, and displaying incorrect answers) are outlined in the upcoming sections.

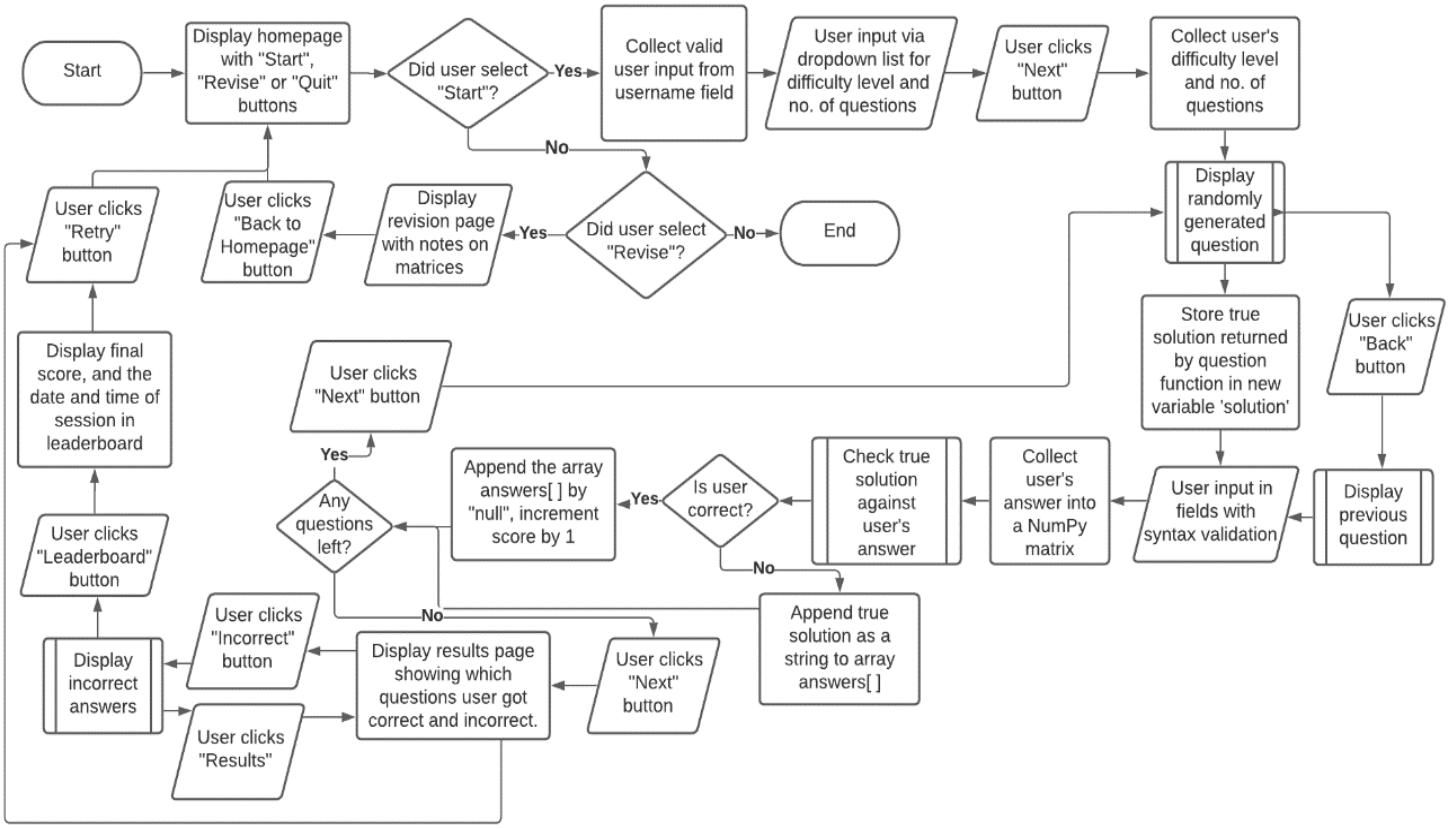

Figure 1: High-level implementation flowchart

Data Structures & Data Storage

- Strings: All output and input will be in string type.
- NumPy Matrices: a specialized 2D array which retains its 2D nature through operations such as matrix multiplication.
- Dictionary easy{ }, where the keys 0-6 inclusive are used to represent the 7 topics. The values are function calls for generating an easy question for that topic.
- Dictionaries medium{ }, and hard{ } are structured in the same way as easy{ }.
- String array (Python list) answers[ ] of length n where n = 5, 10 or 20, answers[i] gives "null" or the correct solution. For example, Figure 2 shows the array when n = 5:

	0	1	2	3	4
answers[ ]	"null"	"[[1 0] [5 4]]"	"null"	"18"	"null"

Figure 2: Array answers[ ] where there are n = 5 elements

- "leaderboard.txt": A text file, storing the username, date when the quiz was taken and score as a percentage e.g.: John 11/02/2022 80.
- "Questions.txt" : A text file where we write the questions to. The first line being the first question and the last line being the last question of the quiz.

Display randomly generated question:

1. Choose a random key using key_val = random.randint(0, 6) where key_val represents the key.

2. Use key_val to find the function call for generating a question for the topic from the dictionary easy{ }, medium{ } or hard{ }.
3. The function should generate matrices with random integers.
4. Turn the matrices generated into strings.
5. Fill in the gaps in the question template with the new matrices using string concatenation.
6. Output the question by displaying it on the screen as text.
7. Store the output text in question.txt to retrieve after the quiz when displaying results.
8. Carry out matrix operations to calculate the true solution of the question.
9. Return the true solution

Display incorrect answers:

1. Int x = 0
2. print("Question", x + 1)
3. Read and output the x^{th} question in question.txt.
4. Look up the x item in the array answers[ ].
5. If the item is not equal to "null"
6. then print("Incorrect")
   print("The correct answer is: ", answers[x])
7. Continue until we reach the end of the array.

Collect and check user's answers:

```
62  // function that takes in the integer values entered by the user to create a matrix and returns
63  // the new matrix.
64  def collect_user_input():
65      Wait for user to enter integer values into input boxes and confirm their answer.
66      Store the values in an array, user_input[r][c] where r,c = rows, columns of the matrix.
67      user_input = format_arg(user_input)
68      user_matrix = np.matrix(user_input)
69      return user_matrix
72  // function that compares the calculated answer to the user's answer and stores if whether
73  // they got the answer correct or incorrect.
74  def check_answer(user_matrix, solution, question):
75      if (solution == user_matrix)
76          then answers[question] = True
77      else answers[question] = False
78      end if
79
```

Figure 3: Pseudocode for collecting the user's answer and checking it.

Matrix Calculations

The matrix operations for checking the user's solutions will be carried out using the NumPy module – some of the specific NumPy functions we will use are illustrated in Figure 4. For our implementation, all the randomly generated NumPy matrices will have the same number of rows and columns, with values in the range 15 to 15. This prevents a ValueError when the matrix is passed to NumPy's determinant and inverse functions which require a square matrix. Additionally, NumPy's inverse function returns a matrix with floating point values for matrices with values of our range. As a result, to ensure the user can input an answer in exact form, we have decided to create an inverse method that allows the user to submit their answer as:

$$\frac{1}{det}[\ \ ]$$

where *det* is the determinant and [ ] represents the adjugate matrix.

This ensures that input values are integers. Similarly, NumPy also returns floats for eigenvalues and eigenvectors. For both, our method will round the answers to 2dp for higher precision.

```
1  import numpy as np
2
3  // all answers will be calculated to 2 decimal places.
4
5  // --- numpy functions explained ---
6  // matrix(data) - returns a matrix from a string of data
7  // linalg.det(a) - returns the determinant of the matrix a
8  // linalg.inv(a) - returns the inverse of the matrix a
9  // linalg.eigvals(a) - returns the eigenvalues of the matrix a
12 // function to format the argument as a string that the matrix function takes in.
13 def format_arg(M):
14     M = str(M)
15     In the string M, replace "]" with " " and replace "[" with ";" and replace "\n" with " ".
16     Remove the first two characters in the string M.
17     return M
21 def generate_matrix(rows, columns):
22     Generate an array, M[rows][columns], consisting of random integer values in range -15 to 15
23     M = format_arg(M)
24     matrix = np.matrix(M)
25     return matrix
```

Figure 4: NumPy functions we will utilise and the pseudocode for generating random matrices.

Input Syntax Validation

When the user selects a username, the program checks if the username entered is valid. The username must have a length in the range 1 and 24 inclusive and have no special characters or spaces to be considered a valid username.

Users can input their answers to the questions in the input fields via their device keyboard. To prevent errors when the program processes the input, the input strings are checked for invalid characters: this includes all characters besides 0-9 and the full stop for decimal answers which should be rounded to 2 decimal places. When the user submits an invalid answer i.e., anything apart from an integer or float, the input box is cleared as another valid input is expected and an error message is displayed stating the input was invalid. An error message is also produced when a field has been left empty, preventing the user from proceeding. After the program collects the user input, whitespaces are removed, and the user's answer is transformed into a NumPy matrix for comparison with the true solution.

Displaying the Leader board

1. When the user clicks the “Leader board” button after reviewing their answers. Information on their username, score as a percentage, date the quiz is saved into a text file called “leaderboard.txt”.
2. When displaying the leader board, the program will read the data saved on the leaderboard.txt line by line to show the top 5 attempts.

3. The first 5 lines are read, then for any next line, if the score is greater than the scores in the first 5 lines, this attempt is slotted into the list of 5 and the lowest ranked attempt is removed.
4. We considered having the leader board rank the scores of multiple users by using the PyMongo module to connect to MongoDB. However, due to time constrains, this may be outside the scope of this project.

Examples of How Our Tool Can be Used

The home menu is displayed to the user, providing the options 'Revise', 'Start' or 'Quit' as shown in Figure 5.

Example 1:

1. The user selects the 'Revise' button.
2. As a result, the user is redirected to the revision notes page.
3. The user can then scroll through the notes on the different sub-topics of matrices that the quiz also tests their knowledge on.
4. Once the user has finished reviewing the content, they can select the 'Back to Homepage' button which will take them back to the home menu.

Example 2:

1. The user selects the 'Start' button.
2. The user is redirected to a page where they can select their settings for the quiz and enter a username. Changing the difficulty displays details of each difficulty as shown in Figure 6.
3. The user types in a username, selects medium and 5 questions on the drop-down menus.
4. First question is displayed, and the user answers the question by typing in integer values into the boxes provided as shown in Figure 7. The user selects 'Next' which displays the next question. This is repeated for a further 4 questions.
5. Once the user has completed the quiz, they select the 'Submit' button on the final question page.
6. The results page is displayed as shown in Figure 8. This shows all the questions 1-5, and whether they got the question correct or incorrect. The user answered question 2 incorrectly and therefore clicks the question to view their incorrect answer and the correct answer as shown in Figure 9.
7. After viewing the corrected answer, the user goes back to the results page by selecting the 'Results' button.
8. From the results page, the user selects 'Leaderboard' to view the leader board showing their top 5 attempts as shown in Figure 10.
9. The user selects 'Retry' which redirects them back to the home menu.

Example 3:

1. The user selects 'Quit' button.
2. The program terminates.

User Interface

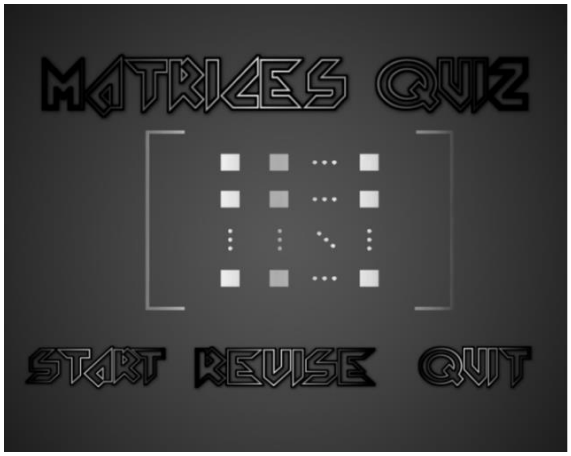

Figure 5: Prototype of home menu page.

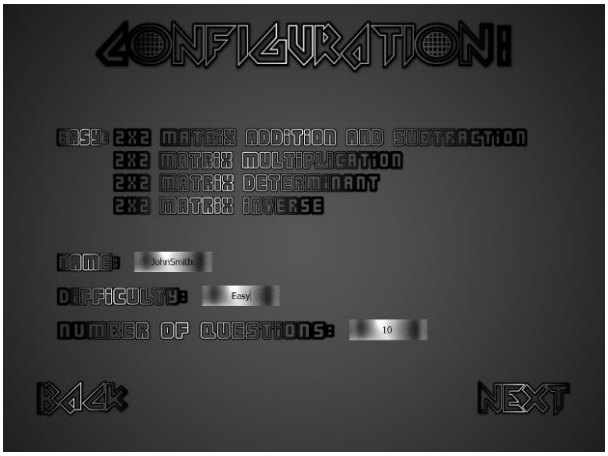

Figure 6: Prototype of configuration page.

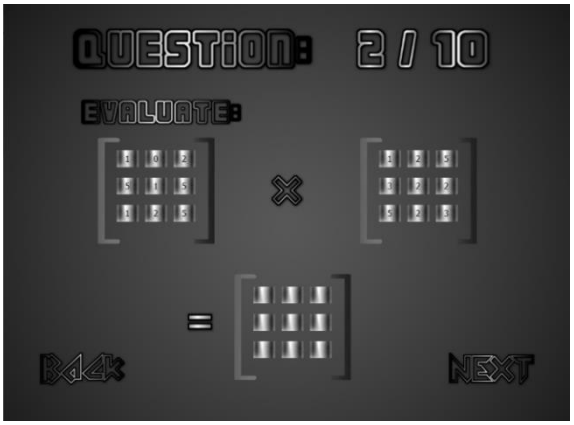

Figure 7: Prototype of a multiplication question page.

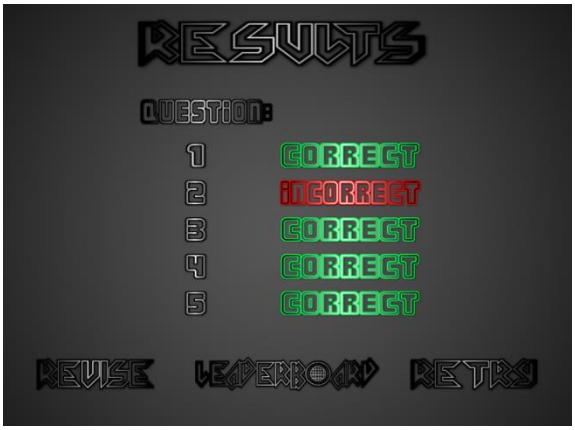

Figure 8: Prototype of results page.

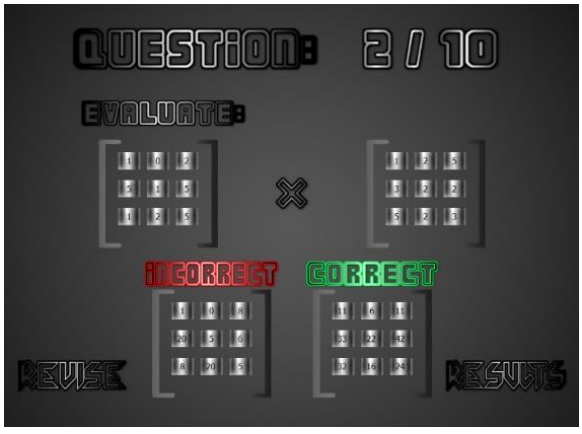

Figure 9: Prototype of a question page with corrections.

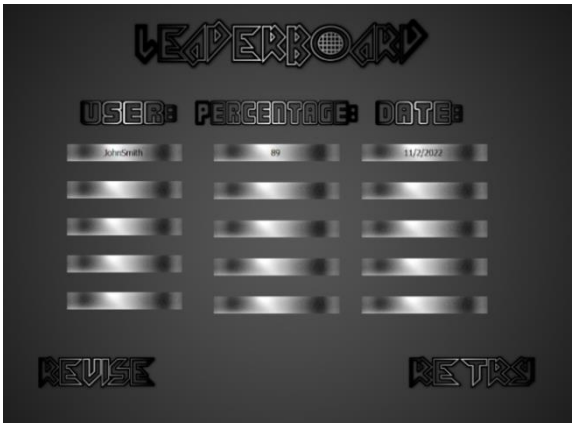

Figure 10: Prototype of a leader board page.

Project Timeline

The Gantt Chart in Figure 8 demonstrates how we plan to allocate our time. For the first 3 ½ weeks, our team will focus on programming and building our quiz tool. The team will be split into pairs to work on tasks simultaneously. Aishwarya and Nandini will work on the back-end components concerning matrix calculations and generating questions. Michael and Hannah will work on the UI components. We will also work on integrating the front-end and back-end components during this period and test that the overall program works. For the 2nd half of the 4th week and the 5th week, we will plan and record our demo presentation.

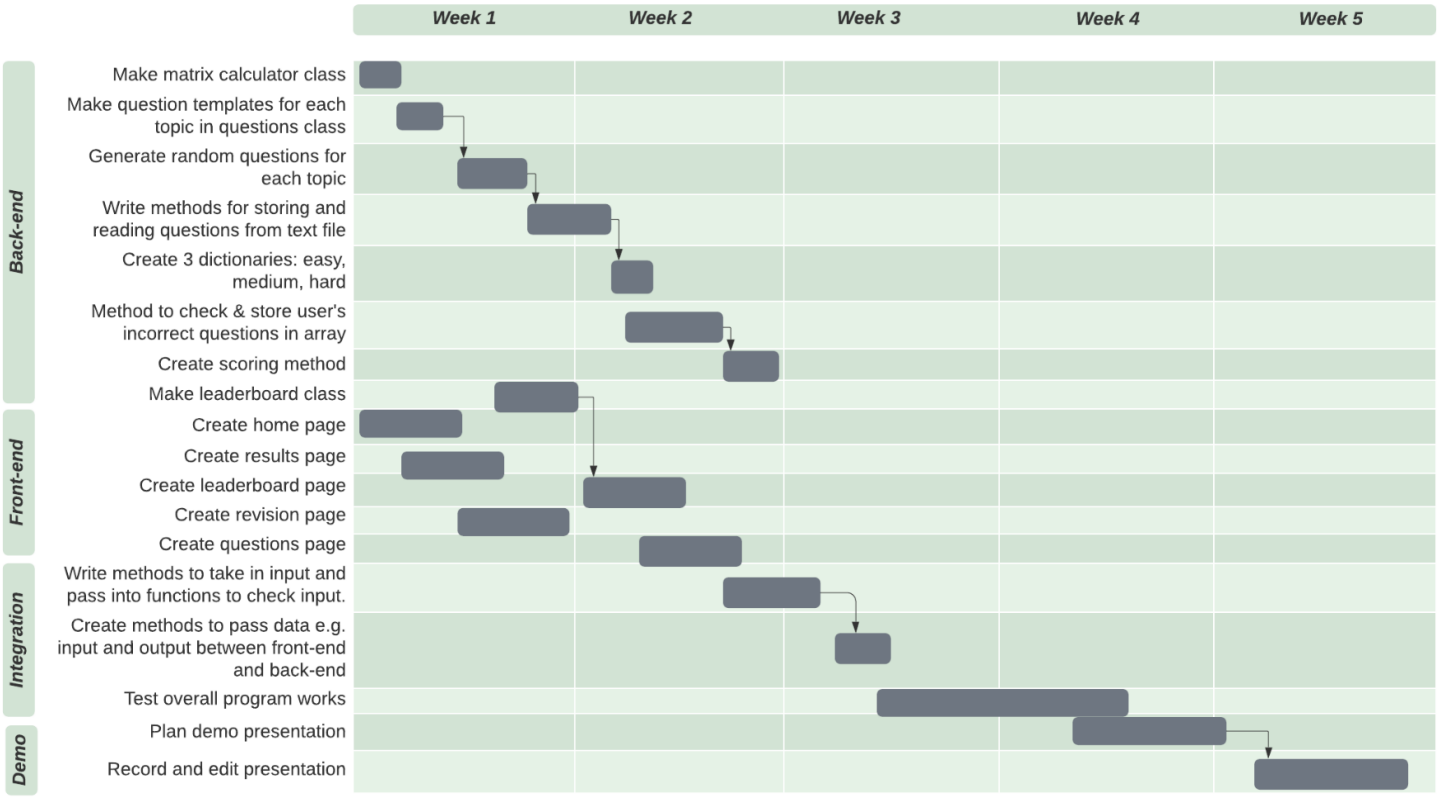

Figure 8: Gantt Chart showing the timeline for the implementation of the matrix tool.