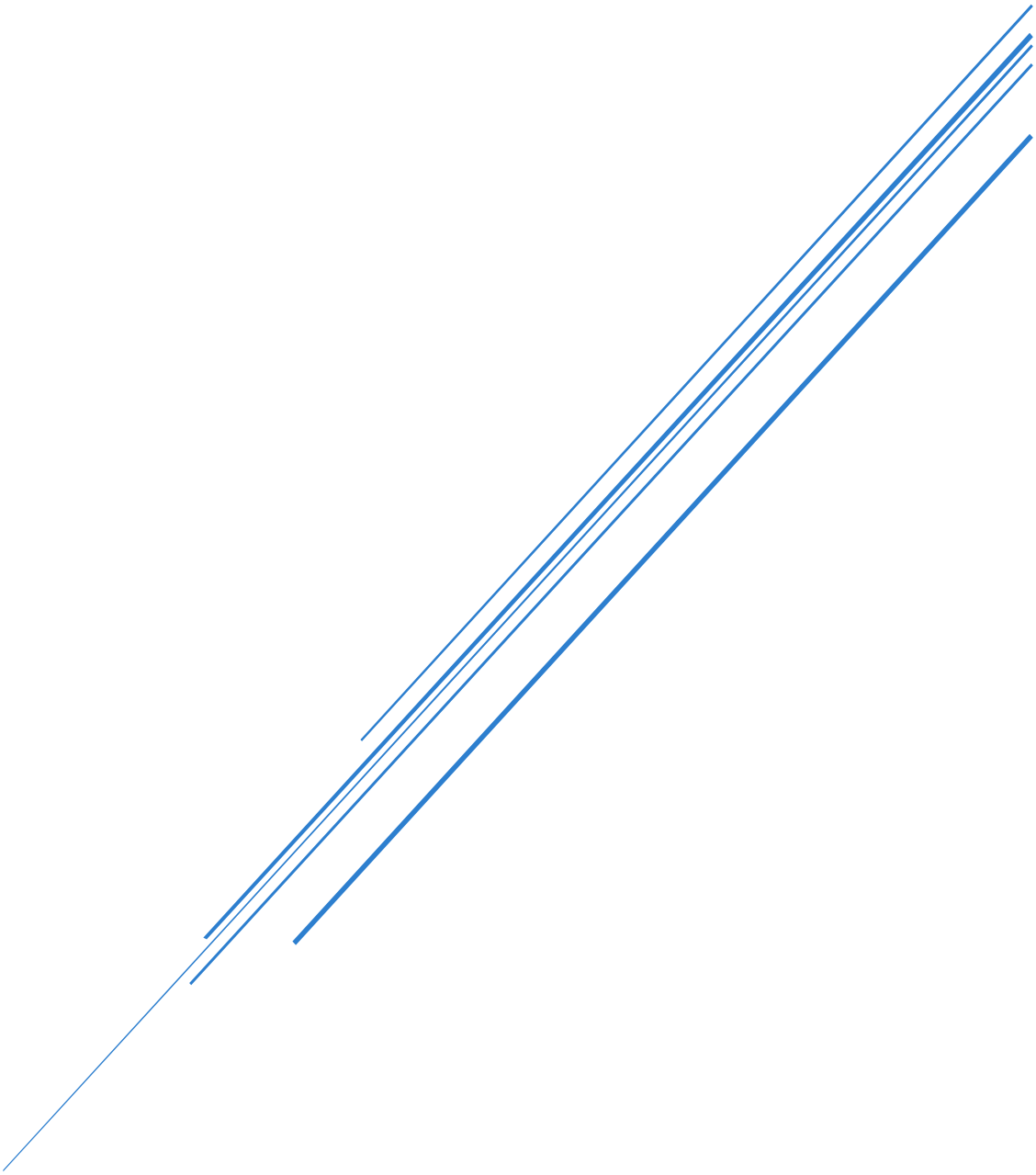
MATHS REVISION PLATFORM

ABHI KALAKOTI



LEICESTER GRAMMAR SCHOOL
COMPUTER SCIENCE NEA

Contents

Analysis

Client Background

Client: Nikhil Tej
School: Leicester Grammar
Year: 9
Potential Users: Any Year 9 student studying Maths

Current Study Method Description

Nikhil is a student currently studying Maths (alongside other subjects) at Leicester Grammar in Year 9. At home, he studies self-dependently, generally using textbooks, material provided from the school itself, and occasionally researching online resources. I aim to provide him with an all-in-one software that allows him to revise concepts from notes, answer topic questions, and track his progress with ease.

Client Interview

What is your current method of study for Maths?

Nikhil: I just do past paper questions or specific topic questions. I answer the questions, mark them, and go back to check where I went wrong. Afterwards, I recap these topics where wrong answers more frequently occur until I am good with everything.

What are the advantages of your current method?

Nikhil: It mainly helps to reconsolidate my information. It also ensures that I have covered everything that I need to go over for a test or general revision.

What are the disadvantages of your current method?

Nikhil: It can be hard for me to get into the mood as I usually do passive revision such as writing notes before I do active revision such as revising notes and answering questions. It is also somewhat time consuming.

How could the method be improved upon?

Nikhil: Incorporating other forms of passive revision such as going through notes. And preferably make it faster to carry out by reducing the number of tasks I need to do.

Are there any new features that you think should be added to the proposed system?

Nikhil: Perhaps going through notes, and anything else that could make it better for remembering key equations and concepts and making it easier to perform and faster to carry out than the current system of studying Maths. It could also be like a game or have some gamified experience that makes it more fun to use.

Client Game Interview

What types of games do you like?

Nikhil: I like open world exploration games, level-based platformers, fast-paced games, combat and shooter games too. I also really like the games where you grind (repeat different tasks in-game multiple times) to gain rewards.

Do you like study-related games, are they effective?

Nikhil: Yeah, I enjoy them. They aren't the most efficient form of revision and they aren't normally the greatest games, but they are fun to play at times.

Which study-related games have you played?

Nikhil: I have played a lot of the MyMaths games and Quizlet flashcard games. I haven't seen many of those games that are more focused towards the game experience than education though.

What specific features would you like in the game?

Nikhil: I would like a fast-paced game for sure, possibly a game that uses a timer to see how fast you can beat each level or the entire game. It would be cool if you could play as a username and there would be a leaderboard to see which user has the fastest times. I think a platformer would suit this style of game best.

What movement mechanics would you like for the platformer-style game?

Nikhil: Well, obviously the basics like walking, jumping, and basic physics and gravity. As the game is fast, sprinting and wall-jumping would be cool, to make it easier to perform parkour-style movement across different obstacles.

How would you like the game to look, colour and theme-wise?

Nikhil: I don't care too much for that, when I play games the core gameplay is what interests me, so I would prefer if you would focus on that. However, character animations, and a variety of world textures and sprites would be pretty cool.

Current System Observation

Date – 04/04/2024
Observing client studying at home

Client Study Actions:

- Gets past papers / topic questions from online school resources from OneDrive or physical papers provided by teachers in school
- Answers a section of the paper, generally 3 – 4 questions of medium length under timed conditions (not normally strict)
- Takes a short 5–10 minute break
- Repeats last 2 steps until past paper is finished

- If mark scheme is available, he marks the questions, normally taking around 1 minute for short answer questions and 3 minutes for long answer questions
- Total score calculated and compared to previous results (normally from memory) to reflect overall performance
- Incorrect / low-scoring answers are checked to see where mistakes were made and where most marks were lost
- Frequently wrong-answer topics are recapped via notes / textbook

Overall, this process can take several hours to complete. Therefore, the client doesn't always implement the full method, only performing certain tasks each study session.

Client Study Recordings:

- Timings – how long it took to do the test compared to the time limit of the test
- Overall Mark – the mark calculated
- Wrong Answer Qs – which questions were answered mostly incorrectly
- Difficult Topics (ones with highest Wrong Answer Qs) – which topics had the most incorrect questions
- Topic Notes (with more focus to Difficult Topics) – make notes on those difficult topics or get notes from other teacher / online resources

These are generally noted down in the school-provided exercise book.

Current System Overview

Answering Questions:

The student gets either past papers or topic questions from sheets provided by his teacher as part of homework/revision, or from the school's online resources. This is a problem due to the limitation of resources that the teacher can provide due to time restrictions and the lack of resources available with the school's online files, mainly that they can't provide full coverage of the specific learning needs of each pupil.

The student generally does a certain number of questions before taking a break, but this can vary depending on how many parts the question has and how difficult the question itself is. This poses a problem as it is difficult for the student to consistently make a reliable estimate for how many questions they should answer in a given amount of time. Therefore, it is often led to guessing which is less optimal and can lead to a lot of time wasted, or adversely spending too much time on a particular question.

Marking Questions:

The mark scheme is not always available if the teacher doesn't provide it as a hard copy. This is a problem as the marking part is the key method for fixing gaps in his knowledge, as he often works out the question without knowing the end result, which can be time consuming.

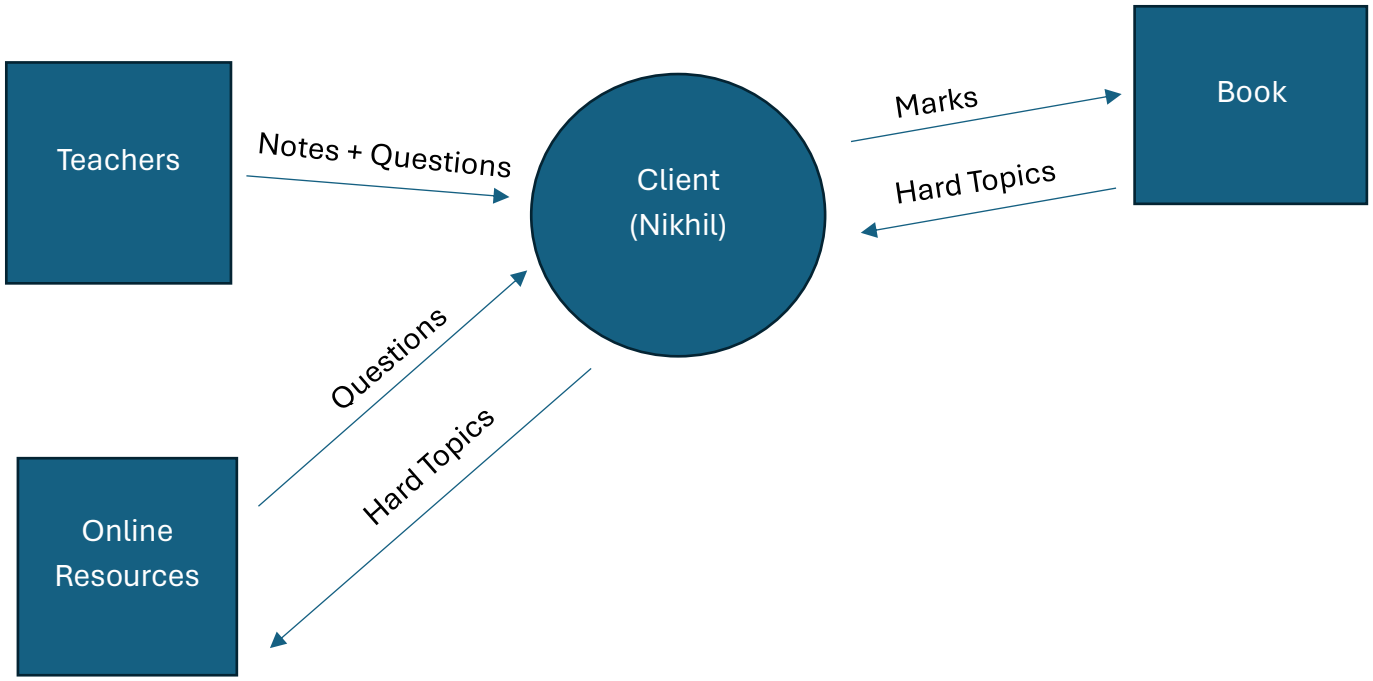
The student has to individually mark each question using the mark scheme. The student also needs to figure out the topics that each incorrect question applies to, e.g. the student needs to note these topics down and deduce which ones appear the most. The student rewrites the notes on these topics to reconsolidate his learning from the information. This is a big problem as the entire process is extremely time consuming if carried out in full, that's why he only performs certain quick tasks on occasion.

Timing Processes:

Alongside all of this, he needs to keep timings for how long he does the questions in, for as not to spend too long. He also needs to make sure his breaks don't go on for too long as not to waste time meant for studying. Maintaining this timing on top of answering questions, keeping track of mistakes, and other requirements all together, is very difficult for my client to carry out.

To conclude, he needs a solution that can: provide a large amount of questions from all the topics he's learning, automatically mark these questions so he can get instant feedback, allow him fast access to notes so he can reconsolidate his learning, and manage the timing of his studies.

Data Flow Diagram for Current System



The diagram shows the flow of data for the current system, the teacher provides notes from class learning, the student can get questions from the teacher or online resources. The student answers the questions, carrying out the marking and recapping processes. Using the marks, he can find out the hard topics, which he gets notes and questions for from his textbook, he can then also get questions from online resources, specifically for the hard topics.

IPSO Table for Current System – Questions

IPSO	Information
Input	Topic / Past Paper Questions (from teachers/online)
Process	Answer topic / past paper questions under timed conditions
Storage	Question data sheet Answers to question sheet Mark scheme sheet (These can be either physical or digitally stored, the student's use of resources varies)
Output	Completed answers

This is the first process used by the student to answer the questions.

IPSO Table for Current System – Marking

IPSO	Information
Input	Mark Scheme Questions/Answers
Process	Calculate overall mark Calculate how many wrong answers Get topics for each wrong answer and count
Storage	Overall mark Topics most frequently wrong
Output	List of topics in order of descending difficulty

This is the main process used for marking the questions already completed and getting specific topic work to do based on that. Each step of the method constitutes a part of the IPSOs, each of which can be implemented via computer programming.

IPSO Table for Current System – Recapping

IPSO	Information
Input	Mark Scheme Questions/Answers
Process	Calculate overall mark Calculate how many wrong answers Get topics for each wrong answer and count
Storage	Overall mark Topics most frequently wrong
Output	List of topics in order of descending difficulty

This is the secondary optional process used by the student to go over difficult parts of papers, and any specific topics they struggle with.

Data Dictionary for Questions

The questions shown here represent the scheme of work from Maths for Yr 9 students.

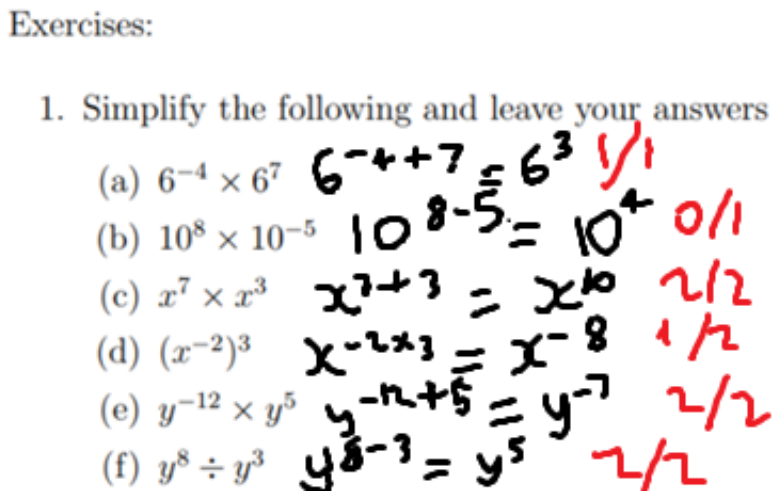
Exercises:

1. Simplify the following and leave your answers in index form:
   (a) $6^{-4} \times 6^{7}$
   (b) $10^{8} \times 10^{-5}$
   (c) $x^{7} \times x^{3}$
   (d) $(x^{-2})^{3}$
   (e) $y^{-12} \times y^{5}$
   (f) $y^{8} \div y^{3}$

No.	Item	Data Type	Sample Data
1	Title	String	Exercises:
2	Q1	String	1. Simplify the following and leave…
3	Q1a	String	$6^{-4} * 6^{7}$
4	Q1b	String	$10^{8} * 10^{-5}$
5	Q1c	String	$X^{7} * X^{3}$
6	Q1d	String	$(X^{-2})^{3}$
7	Q1e	String	$Y^{-12} * Y^{5}$
8	Q1f	String	Y^{8} / Y^{3}

Here is an example of a current question that the client would use and how the current method could be implemented for further study improvement via the use of a data dictionary. This will help to me identify the different tables and data types used for the new system and how the tables will link together. The main tables that take the basis of the current method are Questions, Marking, and Recapping.

Data Dictionary for Marking

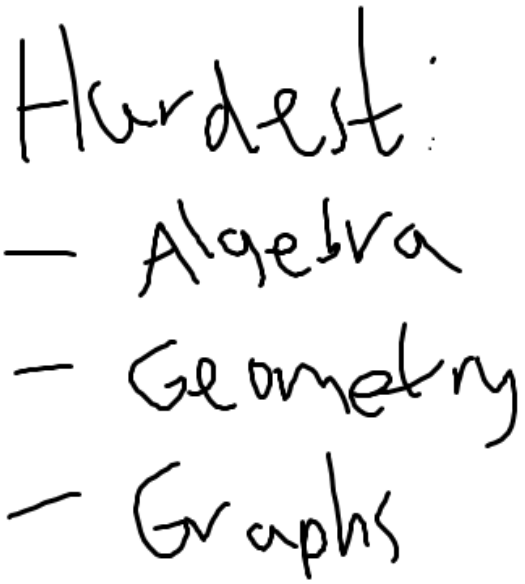

Exercises:

1. Simplify the following and leave your answers in index form:
   (a) $6^{-4} \times 6^{7}$ $6^{-4+7} = 6^{3}$ 1/1
   (b) $10^{8} \times 10^{-5}$ $10^{8-5} = 10^{4}$ 0/1
   (c) $x^{7} \times x^{3}$ $x^{7+3} = x^{10}$ 2/2
   (d) $(x^{-2})^{3}$ $x^{-2\times 3} = x^{-8}$ 1/2
   (e) $y^{-12} \times y^{5}$ $y^{-12+5} = y^{-7}$ 2/2
   (f) $y^{8} \div y^{3}$ $y^{8-3} = y^{5}$ 2/2

No.	Item	Data Type	Out of	Mark	Topic
2	M1	Integer	Sum(3,8)	Sum(3,8)	Numbers
3	M1a	Integer	1	1	Numbers
4	M1b	Integer	1	0	Numbers
5	M1c	Integer	2	2	Numbers
6	M1d	Integer	2	1	Numbers
7	M1e	Integer	2	2	Numbers
8	M1f	Integer	2	2	Numbers

The table stores the correct answer for each question, the total mark for the question, the marks achieved on the question, and the topic that the question relates to, a list of topics can be provided if the question links to multiple topics. This links using a relational database, with the primary key being the No. value.

Data Dictionary for Recapping



No.	Topic	Data Type	Correct	Wrong	Proportion
1	Numbers	String	8	2	0.80
2	Graphs	String	21	9	0.70
3	Shapes	String	14	7	0.67
4	Handling Data	String	14	8	0.64
5	Algebra	String	22	16	0.58

Example table for Maths Topics

Proportion = Correct / (Correct + Wrong)

The client orders the proportion from highest to lowest. The client picks the topics they should focus on more and guesses how many questions they will do based on the proportion which they normally also guess/estimate. However, if the client wished to practice 50 questions, and accurately calculate it in a way so they cover lower proportion topics more, they could do something like this in the proposed system.

Questions(Topic) = ((1 – Proportion) / Sum(Proportion)) * 50

Further Example Questions

Percentages Revision Example Question Sheet

Percentages Revision

Section A: Percentage Change

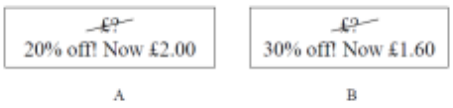
1. The population of a village increased from 234 to 275 during one year. Find the percentage increase.
2. When a beaker of sand is dried in a hot oven its mass reduces from 1.2kg to 870g. Find the percentage reduction in its mass.
3. A battery was tested and found to power a camera for 12 hours before it needed recharging. An improved version of the battery powered the camera for an extra 30 minutes. Find the percentage increase in the life of the batteries.
4. The average cost of a local telephone call dropped by 8p to 27p. Find the percentage reduction in the average cost of a local call.

Section B: Increasing and Decreasing

1. In a sale, all the prices are reduced by 30%. Calculate the sale price of the following items:
   a. a bike that cost £250
   b. a pair of gloves that cost £3.20
2. In 2004, 180 parents applied to a school for a place for their child. The following year saw an increase of 35% in the number of applications. Find the number of applications in 2005.
3. Following the opening of a new supermarket nearby, the number of customers using a small store decreased by 21%. If 2,400 customers used to use the store each week, find the number of customers after the store opened.
4. A car costs £9,999.90 before VAT (value added tax). Work out the cost including VAT if it is charged at 20%.
5. Sally's investment of £450 has gone up by 30%, while Susie's investment of £650 has gone down by 10%. Who now has the larger amount of money, Sally or Susie?
6. A train company increases its rail fares by 4% one year and by 6.5% the following year. Find the percentage increase in cost over the two years.

Section C: Reverse Percentage Problems

1. A jacket is reduced by 12% to £66 in a sale. Find the original price.
2. A baby's weight increases by 8% over a month from birth to 4.05kg, what wasthe weight at birth?
3. Which product has the greatest original price? Show your working.



A B

4. The air pressure increases by 1.2% to 1,214.4 mbar. What was the original air pressure?
5. A dress in a sale is reduced by 7% to £60.45. What is the original price?
6. A stereo system is sold for £1,998 and an 11% profit is made. Find the original cost of the stereo.
7. A shop sells a television to a man and makes a 15% profit. The man sells it to another man for £414 at a loss of 10%. Find the original price of the television.

Section D: Compound Interest

1. Jane invests £1,200 in a bank account which earns interest at the rate of 6% per annum. Find the value of a) the investment after 5 years b) the interest earned.

2. Craig puts £240 into a savings account. Each year the savings earn interest at 6% of the amount in the account at the start of the year. What will his savings be worth after 3 years? Give your answer to the nearest penny.

3. Each year a car loses value by 11% of its value at the start of the year. If it was worth £8000 when it was new, what will it be worth after 2 years?

4. A population of bacteria is estimated to increase by 12% every 24 hours. The population was 2000 at midnight on Friday. What was the population (to the nearest whole number) by midnight the following Wednesday?

5. Ambrose invested £3500 in a six-year bond that added 5% to the amount each year for the first three years and 7.5% each year for the next three years. What is the amount in the bond, to the nearest penny after six years?

****

Section E: Bonus Questions

1. Two numbers b and c are 20% and 28% less than the third number d. By what percentage is the number c less than the number b?

2. The ratio of two numbers is $\frac{5}{6}:\frac{2}{3}$. By what percentage is second number more or less than the first number?

3. 250% of x is increased by 250% to become 350. What is the value of x?

Answers

Section A
1) 17.5%
2) 27.5%
3) 4.2%
4) 22.9%

Section B
1) a) £175 b) £2.24
2) 243
3) 1,896
4) £11,999.88
5) Same (£585)
6) 10.76%

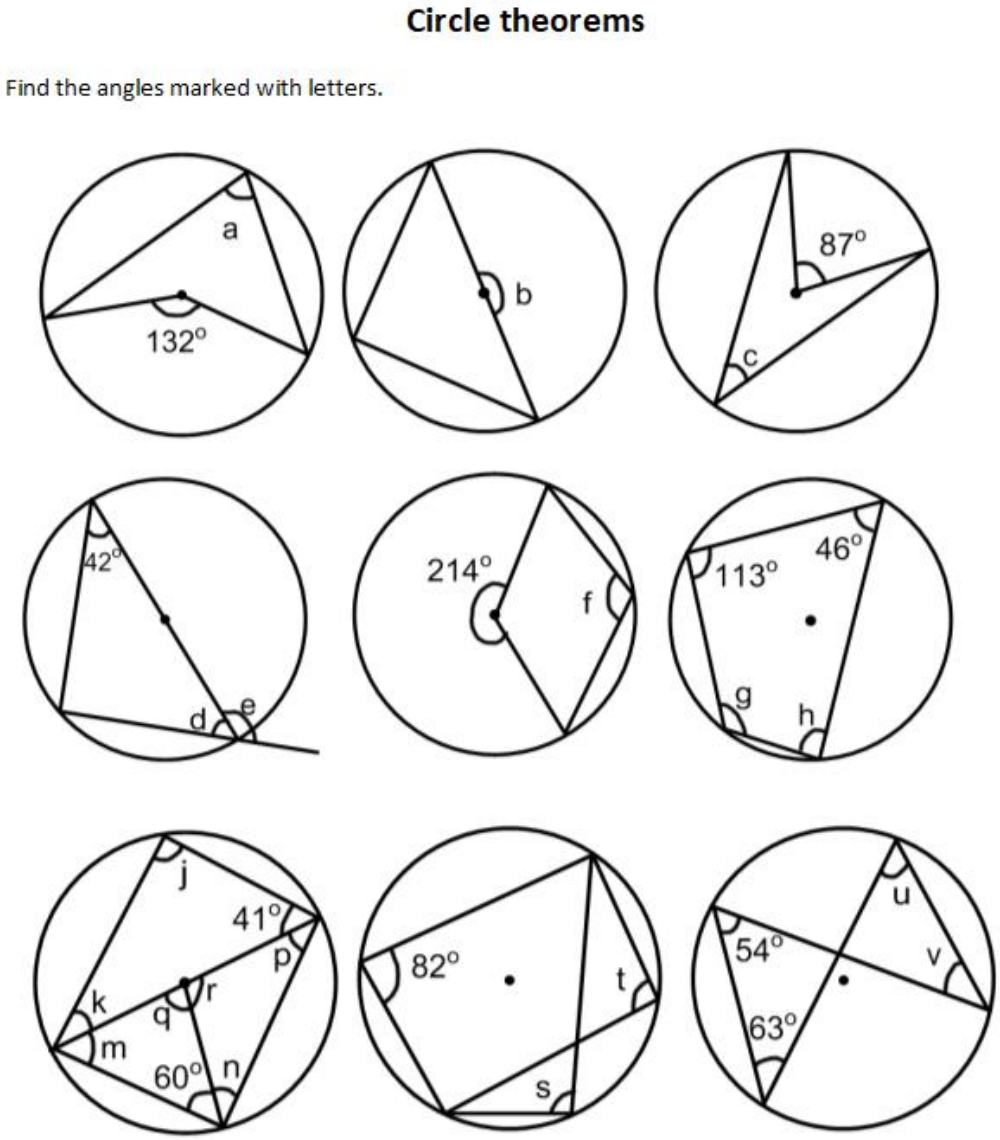
Section C
1) £75
2) 3.75kg
3) A
4) 1,200mbar
5) £65
6) £1800
7) £400

Section D
1) a) £1,605.87
   b) £405.87
2) £285.84
3) £6,336.80
4) £3,524.68
5) £5,033.40

Section E
1) 10%
2) 20% less
3) 40

Using the current suggested form of data dictionary formatting, this sheet could be implemented into the system via a database.

Circle Theorems Example Sheet



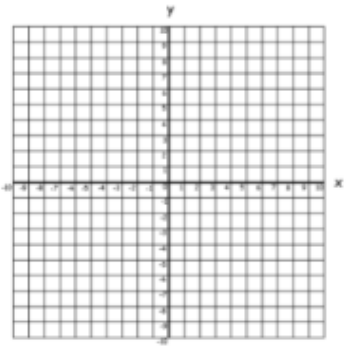
The current system isn't adapted to hold images, this could be added in a separate column for the Questions Table. The images could be displayed according to the question number and question being shown.

Graphs Example Sheet

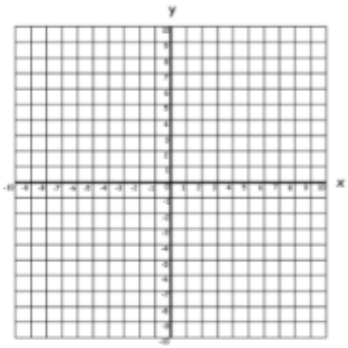
Solving Simultaneous Equations Graphically Revision

Find graphical solutions to each of these pairs of simultaneous equations.

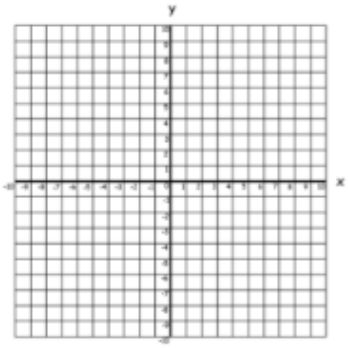
y = 3x - 2 and y = x - 2
x = _____ , y = _____



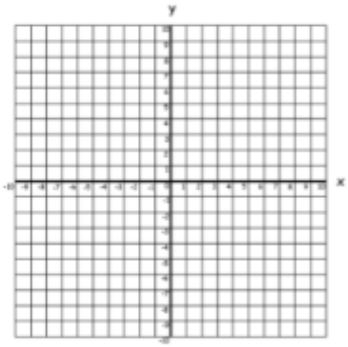
y = 3 - 2x and y = x
x = _____ , y = _____



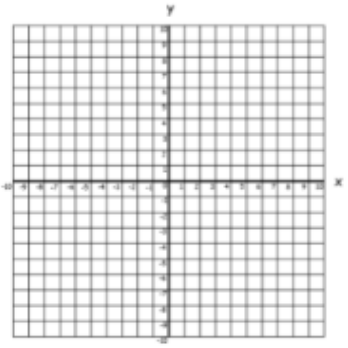
-x + y = 5 and y = 2x - 1
x = _____ , y = _____



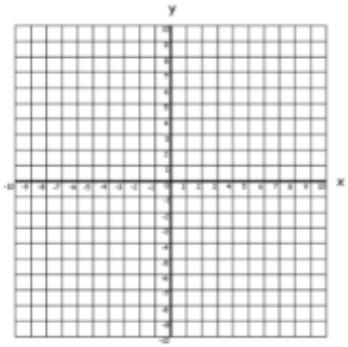
2x + y = 6 and x + y = 0
x = _____ , y = _____



y = 3x - 2 and x + y = 2
x = _____ , y = _____

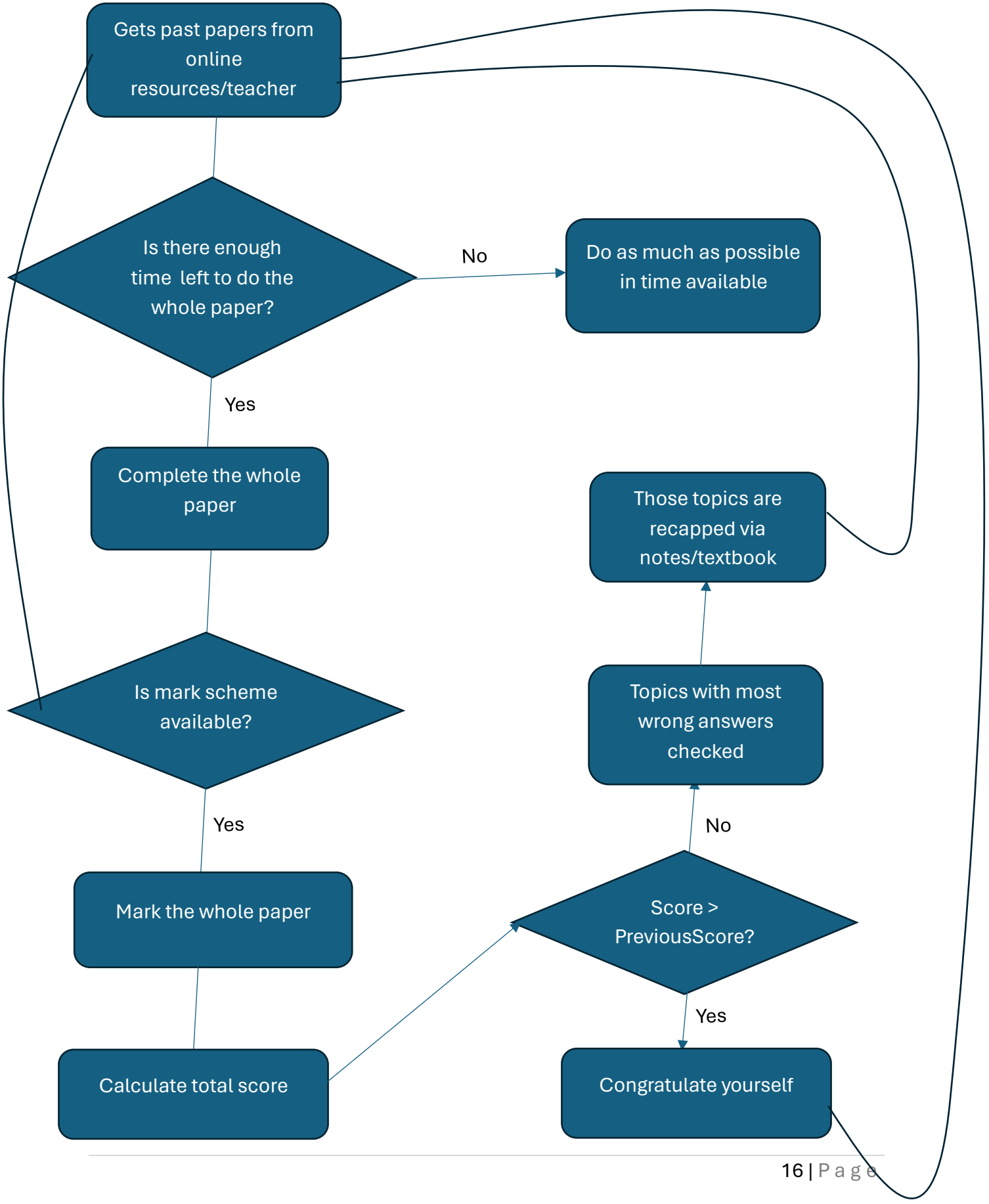


2x - 3y = 6 and x + 3y = 3
x = _____ , y = _____



The system isn't adapted to manage questions where you draw something such as graph sketching questions. This could be implemented via the turtle library in python, but I don't plan to add it currently. To bypass this without adding new features, the user could use paper to answer the questions and they can then ask the proposed system to show the answers and check themselves based on their answer.

Flow Chart of Current System's Manual Operation



Limitations of the Proposed System

This system is being made for the primary use of the client, Nikhil. However, its purpose is applicable to any student within his year group and age range. Additionally, teachers could use the program for classroom implementation. Almost all teachers and most Year 9 students should have basic computer fluency, generally knowing how to launch and use a program. Therefore, it isn't largely important if the system is complex. However, it should still be somewhat easy to understand and interact with, in order to reduce time wasted using the system. The questions should be formatted in a style similar to that of a sheet of questions or an exam paper, as it helps ease of use.

- Software limitations – The program should be able to run on the client's computer with relative ease and it should be easy to interact with. The client already has Python installed but this may pose an issue when it comes to the modules used. However, this can be solved by using a compiled version.

- Question formatting – The questions should be formatted in a style that the user is familiar with such as exam papers. They also need to be sorted by difficulty which must be done by me individually. Questions with unusual formatting that are difficult to implement in a standard database may cause issues.

- Programming knowledge – I should be able to make it in Python using my knowledge and the aid of other resources. This is definitely feasible for the intended purposes of the system but may require some additional learning on my part.

- Time limitations – The system needs to be completed within a certain time frame, so it shouldn't be overly complex or have any extra features that would take too long to implement.

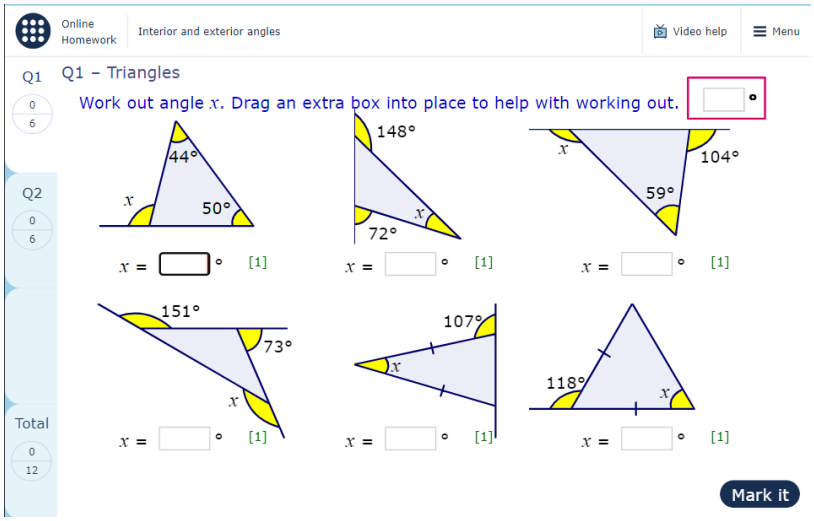
- Fulfilling requirements – I should be able to include the features of the current system in the proposed system, while also building upon those features to improve them through computerization. Additionally, I should implement new features that will help my client with his revision for Maths.

Comparison against Existing Online System – MyMaths

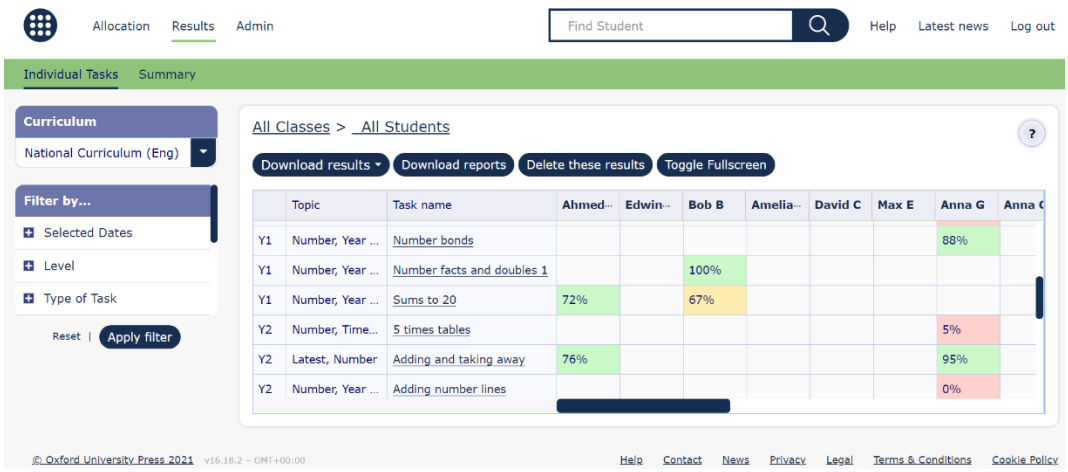
Existing System Description (Provided by MyMaths) – MyMaths is an interactive online teaching and homework service developed for teaching, practising and assessing children's maths fluency.

MyMaths provides a wide range of maths resources for use including: full coverage of school curriculums, ready-made lessons paired with randomised homework questions, revision activities, and maths games. Alongside this, the student's progress can be tracked. The teachers can easily track progress and create homework assignments. They also support teachers through training resources, helping them to use the platform more easily.

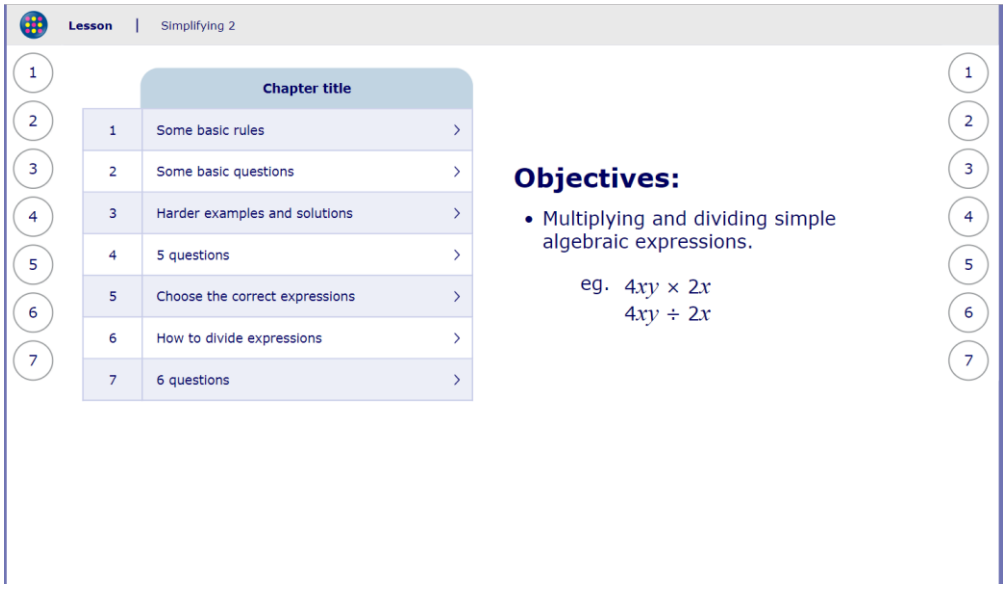
Examples:



Here is an example of a homework assignment for the subtopic of Interior and Exterior angles. The system outputs the questions by having 2 different menus, each holding a different question. It displays the images, providing an input field, and the number of marks for each one. The marking button can be used to activate the marking process. The task can then be submitted as complete on the final page. This system is slightly inefficient as it would make it harder for the user to concentrate on one question at a time if presented with multiple. Additionally, if a user doesn't know what they're doing and they answer all of them at once, they will presumedly get all questions wrong. Doing one at a time could allow them to know where they went wrong, and correct this mistake for future questions, rather than spending more time incorrectly answering each question. For these reasons, my system will present 1 question at a time.



Here is an example of the progress page that the teacher uses to track the students' scores on different homework tasks that have been set. It uses a relational database, storing the scores obtained on each task in the appropriate pupil column and topic row. My system already plans on implementing a table in this form to use for recapping. Additionally, I could allow the client to view their progress in an easily understandable table similar to MyMaths.

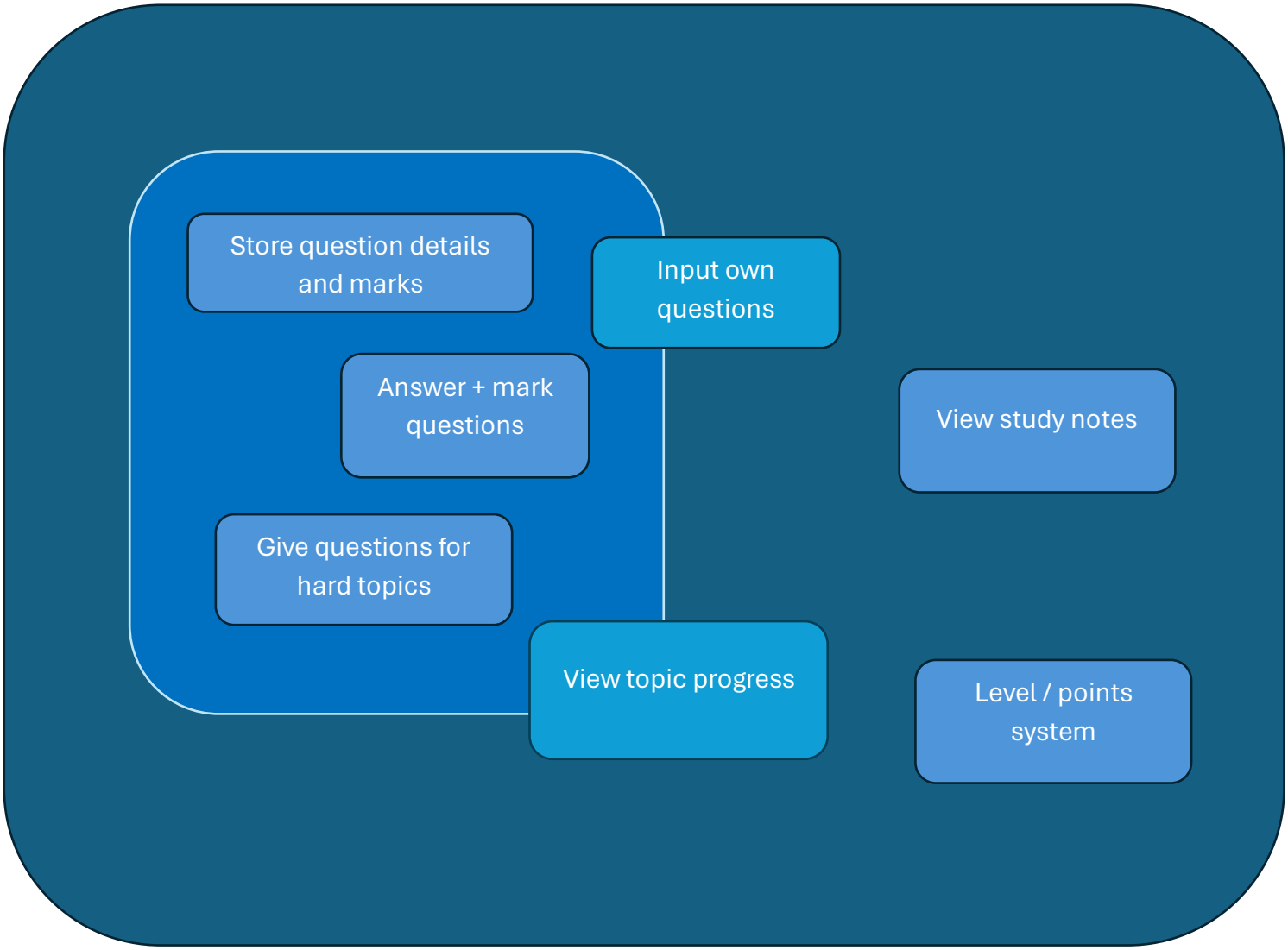


Each homework is paired with an online lesson that the student can go through if they haven't learnt the topic yet, they want to recap something, or they want to revise the material. It provides a list of chapters, each with details on how the concepts work and step by step guides to answer those types of questions. This is a very good way to present info as it doesn't clutter the user with a large wall of text information in the form of notes, making it easier to digest in parts. This may be slightly harder to implement while also making it user friendly with my understanding of Python and Tkinter (the python library I plan to use for building the user interface).

Objectives for Proposed System

Objectives	Performance criteria
1. Store all question details	Store question, question no, answer, topic, total mark, steps required for the marks, should be able to support all types of questions including image-based questions and graph questions. This can be achieved using a table in a relational database.
2. Answer questions	The system should be able to display the question (showing one part of the question at a time) with a working input field for the answer. This can be achieved using a standard input element.
3. Mark questions as they go	The system should provide a mark button that checks the answer automatically, if possible, else provides a list of the marking steps for the question. If the answer is wrong, show the correct answer after marking the question. This can be achieved by comparing values from the submitted answers to the stored actual answers.
4. Store marks for answers	The system should store the mark obtained from the question in a database, while also assigning it to a specific place using the topic name associated with the question. This can be achieved by using a table in a relational database.
5. Give questions for harder topics	Calculate proportion from the topic marks, this can be achieved by using the formula Questions(Topic) = ((1 – Proportion) / Sum(Proportion)) * Q to work out how many questions to give for each topic.
6. Input their own questions	Provide an interface for entering each of the question details necessary – question, question no, answer, topic, total mark, steps required for each mark. This can be achieved using a standard input element which stores the questions into a database.
7. View their progress	The system should allow the user to view their progress for each topic, how well they do on each of them comparatively, in the form of a menu tab that shows a table. This can be achieved by outputting a table from the database that stores progress.
8. View notes to study from	Show notes, each in digestible lesson style parts, by showing images or text with a Next button to move onto the next part, for the user to learn new concepts or revise old ones. This can be achieved by storing notes via files in a database, and outputting those files.
9. Level system with points	For every correct answer, gain points to level up and unlock items, provides positive reinforcement making it more fun for the user. This can be achieved by setting a level variable and adding score to level up for each correct answer, additionally using an exponential growth rate for the amount of EXP required to level up.

Scope Diagram for Features



In the Level 1 scope, the inner most scope, I have stated the objectives that I am certain I can complete within the given time frame of the NEA. I think I can perform all of these with my knowledge. These features are parts of the current system that need to be included so that my client can perform everything he normally does, but in a computerised setting. The Level 2 scope features are 'View topic progress' and 'Input own questions'. These are very useful features which are somewhat implemented in the current system but it may take time to create and isn't required. The other 2 features in the Level 3 scope are features that would help the user as they can track progress much more easily with physical papers. They also help with more studying new concepts or revising them, and keep the user engaged through a game-style system. Therefore, they are somewhat useful but not nearly as important and difficult to add.

Chosen Solution for Proposed System

The final solution that I have decided to implement is a system that implements all Level 1 scope objectives: Store question details, store marks, inputs for answering questions, automatically mark questions, if possible, otherwise display marking steps, store harder topics, provide more questions for harder topics. I will also implement the Level 2 Scope features of viewing topic progress and input own questions as I feel that is essential for improvement so the user knows what they can work on themselves and so they can use their own questions within the program. Within the Level 3 Scope, I probably won't implement the ability to view study notes as it is difficult to implement and I don't think it would be of much use for studying a subject like Maths, or of high importance for the time required to make it. However, I will try to include the game-like system with points and levels as I think it would engage the user and make the learning experience more fun which could also influence them to spend more time studying. I will also include the ability for the user to input their own questions as it would be very useful and otherwise, they would have to rely on the built-in questions provided by the database itself which could be limited in amount.

I will implement these features using Python 3.12, Tkinter, and SQLite. Python is for the base program itself, Tkinter is a python module for building the User Interface, and SQLite is a database engine using the SQL language which I would implement via the sqlite3 module in Python. I have sufficient skills and knowledge to code in Python already. I know the SQL language to a good extent but I also need to learn how to apply the language via the SQLite database engine and the sqlite3 module. However, for Tkinter I am mostly unfamiliar so I would have to learn it myself to create the system. This will also be easier for the user to run in most environments as the client already has the Python environment installed and all Leicester Grammar School systems for students have Python pre-installed.

Design

IPSO Chart for Proposed System Overview

Input: • Topic / Past paper questions - Question number - Question text - Question marks - Question images - Question topic • Mark schemes - Question no - Steps for marks - Question marks • Topics	Process: • Get questions from database • Sort questions by topic • Analyse mark schemes - Calculate total mark - Find out difficult topics for user - Provide paper feedback • Find questions from difficult topics
Storage: • Questions table • Topics table • Mark schemes table	Output: • Menu UI where user can: - Answer questions - Mark questions - View progress - Recap topics

Detailed Objectives for Proposed System

Objectives	Performance criteria
1. Store all question details	1.1 – Store question text 1.2 – Store question image 1.3 – Store question number 1.4 – Store question answer 1.5 – Store question topic 1.6 – Store question marks 1.7 – Store question type 1.8 – Store question working stepsd 1.9 – Store question images for image-based question 1.10 – Store question answers as images for image-based-Qs 1.11 – Delete questions from the database By storing all of these components, the system should be able to support all types of questions, including manual-mark questions and image-based questions. I will achieve this using a table called tblQuestion in a relational database. -- +-
2. Answer questions	2.1 – Display question details in box format 2.2 – Display question image 2.3 – Display input field for entering answer 2.4 – Display mark/show answer button 2.5 – Display answer result box The system should be able to display the question with a working input field for the answer. This can be achieved using a standard input element. It should also display a button to mark the question, and an answer result box to display the correct answer and the result.
3. Mark questions as they go	3.1 – Mark answer when button pressed 3.2 – Display correct answer 3.3 – Display choicebox for choosing correct/wrong (if manual) 3.4 – Update result box to correct/wrong 3.5 – Display new box with question steps 3.6 – Update button functionality to next-question button The system should provide a mark button that checks the answer automatically, if possible, else provides a list of the marking steps for the question. If the answer is wrong, show the correct answer after marking the question. This can be achieved by comparing values from

	the submitted answers to the stored actual answers. Then, updating the mark-answer button to a next-question button.
4. Update question details	4.1 – Update question details 4.2 – Clear steps box 4.3 – Clear answer result box 4.4 – Remove choicebox (prevQ is manual and newQ is automatic) 4.5 – Add choicebox (prevQ is automatic and newQ is manual) 4.6 – Update button functionality to mark-answer button This updates the question details when the next-question button is pressed, clearing any previous results. It adds or removes the choicebox depending on what the previous question and next question types are. Then, the button is reverted to the mark-answer button.
5. Store marks for answers	5.1 – Check if answer is correct/wrong 5.2 – Check question topic 5.3 – Increment correctQs / answerQs in tblTopic for specific topic The system should store the mark obtained from the question in a database, while also assigning it to a specific place using the topic name associated with the question.
6. Give questions for harder topics	6.1 – Fetch a list of all the question topics from tblTopic 6.2 – Using formula, calculate proportion for each topic for the number of questions 6.3 – Make a dictionary using the topic names as the key, and the proportions as a value 6.4 – Use question function to select random question from topic selected based on weighting from dictionary This calculates proportion from the topic marks, using the formula “Questions(Topic) = ((1 – Proportion) / Sum(Proportion)) * Q” to work out how many questions to give for each topic. This allows questions to be selected based on the user’s proficiency within that topic.
7. Input their own questions	7.1 – Display text labels for each of the inputs for question details 7.2 – Display input boxes for each question detail 7.3 – Display add-question button 7.4 - Run SQL command to add question to tblQuestion in the database when button pressed Provide an interface for entering each of the question details necessary – question, question no, answer, topic, total mark, steps required for each mark. This can be achieved using a standard input element which stores the questions into a database.

8. View topic progress	8.1 – Fetch list of topics with SQL 8.2 – Fetch stats for each topic 8.3 – Display menu with selector for topics 8.4 – Display topic statistics 8.5 – Update statistics when topic selected is changed 8.6 – Allow user to add own topics The system should allow the user to view their progress for each topic, how well they do on each of them comparatively, in the form of a menu tab that shows a table. This can be achieved by outputting a table from the database that stores progress.
9. View notes to study from	9.1 – Display list of notes files from notes folder 9.2 – Display text content of note selected 9.3 – Allow user to edit note with textbox 9.4 – Save note when button pressed Show notes, each in digestible lesson style parts, by showing images or text with a Next button to move onto the next part, for the user to learn new concepts or revise old ones. This can be achieved by storing notes via files in a database and outputting those files.
10. Level system with points and rewards	For every correct answer, gain points to level up and unlock items, provides positive reinforcement making it more fun for the user. This can be achieved by setting a level variable and adding score to level up for each correct answer, additionally using an exponential growth rate for the amount of EXP required to level up. As they level up, they can get rewards.

Flowcharts for Proposed System

Here are some more detailed versions of the previous flowchart, with the new features of the current system that I most likely plan to implement, covering each individual part of the proposed system's IPSO steps.

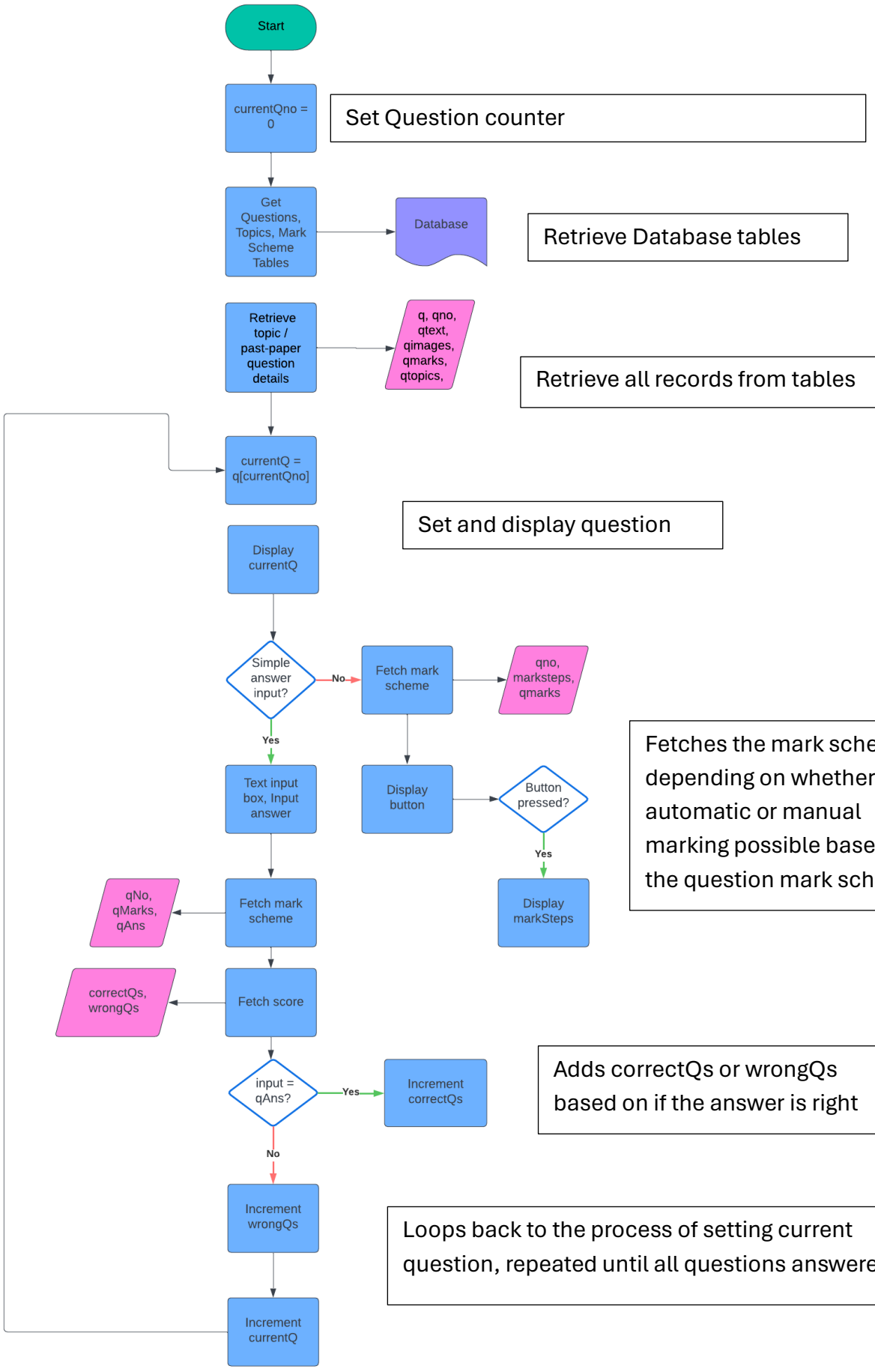
The first version of the flowchart shows the necessary elements for implementing basic functionality, with the base features of Displaying questions, taking in inputs, and returning whether the answer is correct or not, while keeping track of correct and incorrect questions. This is achieved by fetching questions, and answers from the database tables, specifically Questions and Mark Scheme, fulfilling objectives 2, 3, and 4, and 5.

The second version of the flowchart shows how topics would be implemented and keeping track of topic progress, and how the formula mentioned earlier is used to display questions based on the proportion of Incorrect Answers to Correct Answers for questions answered. This is the Level 1 objectives from the Scope Diagram completed (PG 20), fulfilling objective 6.

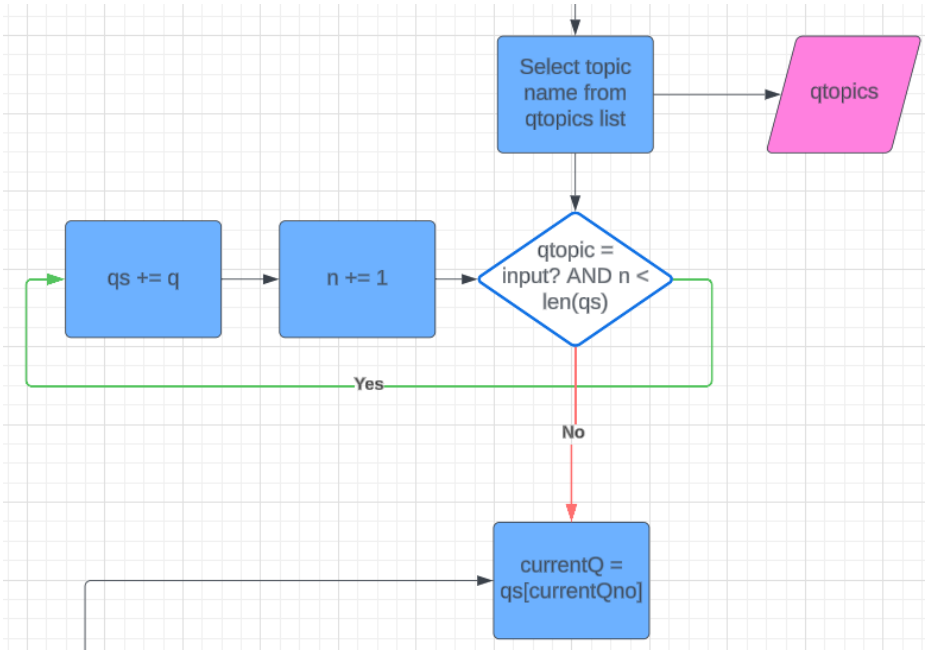
The third and fourth flowchart show separate branches of the menu, where the client can add questions, by appending to the table in the database, and viewing topic progress, while making it more user friendly to view. These are the Level 2 objectives from the Scope Diagram, fulfilling objectives 1, 7, and 8.

Objective 9, the note viewer, isn't a complex process, so I didn't make a flowchart as it's not necessary to understand. As mentioned before, I don't plan to add Objective 10, the Level system.

Implementing basic functionality

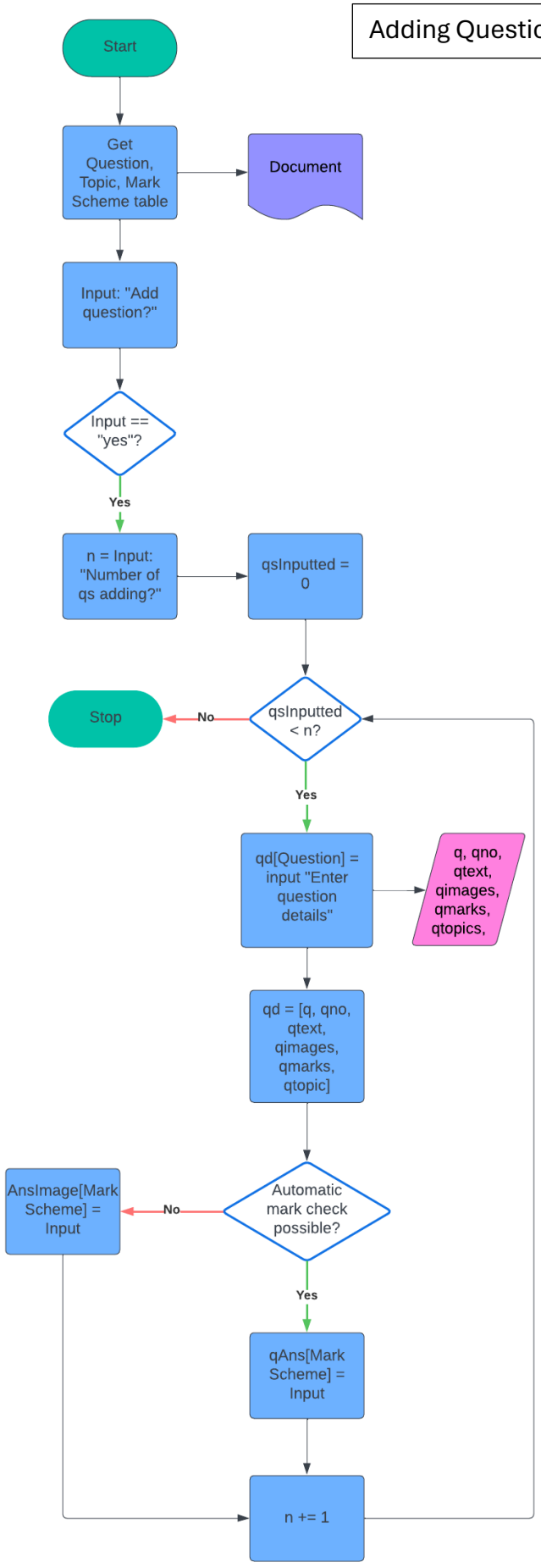


Topic implementation



This is an implementation of a while loop where it allows the user to add questions from a specific topic to a set amount of questions that in a list. I could also add another input to allow them to choose the number of questions that they want to answer, which are automatically added, or how much time they want to spend on work.

Adding Questions



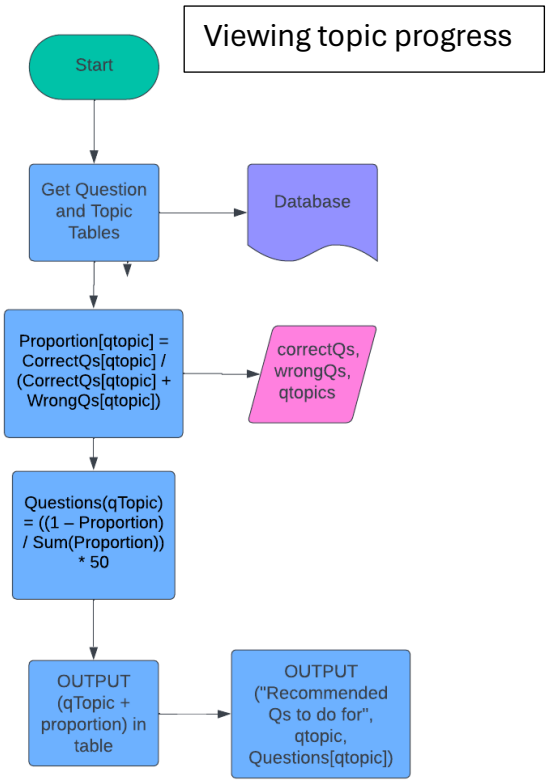
The system retrieves all tables from the database.

It asks the user if it wants to add questions, and if so, how many questions should be added.

This is a flowchart implementation of how a while loop with a counter would be used to repeat the question input until it has reached value n that was input.

The system gets the question details in the form of an array from the user, appending each of these to a record in the questions table.

Based on whether the mark scheme is a basic answer, or contains more complex steps, the marking method and details are set in the Mark Scheme table.



Gets the number of correct and wrong questions, calculating proportion for each topic, and outputting it via a table.

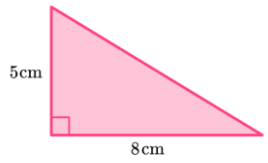
Alongside this, it also provides a recommended amount of questions to do based on the total amount of questions that the user wants to do, in this example I used 50.

In the actual process, I would use conditional statements with text prompts to display messages rather than values, would be more user-friendly.

By combining this functionality together, along with a menu UI to select each of these options, adding questions, answering questions, and viewing progress. These flowcharts represent the basic processes of the system and how they are carried out.

Data Dictionary for Proposed System with Data Validation

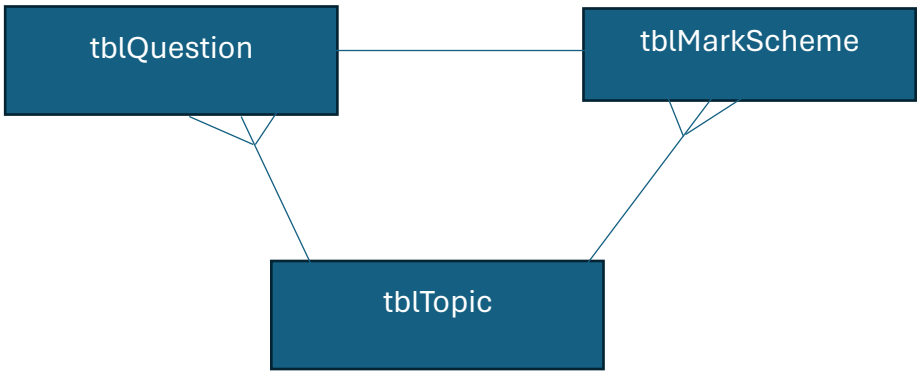
This contains all possible fields that can be entered across the 3 tables of the relational database, providing the key details, such as whether it's used as a primary key or foreign key, field name, data type, length, and most importantly how I would validate that an input for the database is acceptable to be added.

Field	Key	Data Type	Length	Validation	Example Data
Question text	N/A	String	10-500 chars	Must be in the length range	"What is the area of a triangle with height 5cm and base 8cm in cm^2?"
Question images	N/A	String (file path/link for image)	10-200 chars	Must be a valid file path or working image web link	
Question topic	N/A	String	5-40 chars	Must be on list of curriculum topics	"Geometry"
Question number	Primary, Foreign	Integer/Auto index	1-99999	Not applicable as it uses automatic indexing	7
Question marks	N/A	Integer	1-50	Must be in the value range	2
Question difficulty rating	N/A	Integer	1-5	Must be in the value range	1
Marking steps	N/A	String (text steps or file path/ link for image)	10-500 chars	Must be in the length range for text or a valid file path or working image web link	5 x 8 * 0.5 = 40 cm^2 Must show working for 1st mark

Mark scheme number	Primary	Integer	1-99999	Must be in range and match question number value	7 (same as question number)
Mark scheme answer	N/A	String	10-500 chars	Must be in length range	“40”
Correct answers	N/A	Integer	1-99999	Must be in range	10
Wrong answers	N/A	Integer	1-99999	Must be in range	8
Topic number	Primary	Integer	1-999	Must be in range	3
Topic name	N/A	String	5-40 chars	Must be a valid topic on list of curriculum topics	“Geometry”

Entity Relationship Diagram for Proposed System

The diagram shows the relationship between the tables in the proposed system.



For each question record in tblQuestion, there is an exact record linking to the question which holds marking details for that in tblMarkScheme. This makes it a one-to-one relationship. For each topic record in tblTopic, there are multiple questions that could have that topic, and therefore multiple answers that could have that topic, making the relationship between tblTopic a one-to-many for both of the tables.

Database Normalisation

First Normal Form – 1NF

Requirements:

- 1 cell only holds 1 value (Atomic values)
- Primary key for identification
- Each column only has 1 value for each row
- No duplicated rows or columns

Here is my db in first normal form:
tblQuestion

qNo	qText	qImage	qRating	qMarks	qType	qTopic	mSteps	mAns

tblTopic

topicNo	topicName	correctQs	wrongQs

All the entries are atomic, every single field is also dependent on qNo as a primary key, as the details for each field of each record, will be unique to that record. This is also true for tblTopic with the primary key topicNo and the other fields. For a table to be in second normal form (2NF), it must have no partial dependencies, this is already achieved by this table, therefore this is also 2nd normal form.

Third Normal Form – 3NF

The only requirements for 3rd normal form are that the table is already in 2nd normal form and has no transitive partial dependency. This means that any additional dependencies should be removed, those where an attribute that isn't part of the candidate's key is dependent on one that is part of the key.

Here is my db in third normal form:

tblQuestion

qNo	qText	qImage	qRating	qMarks	qType	qTopic

tblTopic

topicNo	topicName	correctQs	wrongQs

tblMarkScheme

qNo	mSteps	mAns

I'm going to use the first form for the actual database, as it will be mostly free on UPDATE, INSERT, and DELETE anomalies/errors, which is the operations that the user will mostly be performing through the interface.

Potential SQL Commands for Proposed System

Create Topic table – Objective 3

```
CREATE TABLE tblTopic(
     topicNo int primary key,
     topicName varchar(40),
     correctQs int,
     wrongQs int,
);
```

topicNo is the integer primary key for the table, it auto increments for every topic added. topicName is a string for the name of the topic. correctQs is the number of questions answered correctly by the user for that topic. wrongQs is the number of questions answered incorrectly by the user for that topic.

Create Questions table – Objective 1

```
CREATE TABLE tblQuestions(
     qNo int primary key,
     qText varchar(500),
     qImage varchar(200),
     qRating int,
     qTopic varchar(40),
     qType varchar(9),
     qMarks int,
);
```

qNo is the integer primary key for the table, it auto increments for every question added. qText is a string for the text content of the question. qImage is a string for the filepath of the image, if the question uses an image. qRating is an integer used to rate the difficulty of the question. qTopic is a string for the name of the question's topic. qType Is a string for the type of marking that needs to be used for the question, I have automatic or manual marking, and qMarks is an integer for the number of marks.

Create Mark Scheme table – Objective 5

```
CREATE TABLE tblMarkScheme(
     mNo int primary key,
     mSteps varchar(500),
     mAns int,
);
```

mNo is the integer primary key, it auto increments for every question added. mSteps is a string used for the text content for the marking steps for the question. mAns is an integer used to store the answer for the question if the answer can be a single integer.

View topic progress for all topics in order of worst to best

```
SELECT topicName, correctQs, wrongQs
FROM tblTopic
ORDER BY (correctQs / (correctQs + wrongQs)) ASC;
```

Display a question from a specific topic with a specific rating

```
SELECT qNo, qText, qImages, qRating
FROM tblQuestion
WHERE qNo = currentQ AND qTopic = "Topic here" AND qRating =
integer;
```

Add a question to the database

```
INSERT INTO tblQuestion(qNo, qText, qImage, qRating, qTopic,
qType, qMarks)
VALUES(?, ?, ?, ?, ?, ?);
```

```
INSERT INTO tblMarkScheme(mNo, mSteps, mAns)
VALUES(?, ?, ?);
```

? is used as a placeholder for data inputs with SQlite3 which you can use to add multiple records at once, useful when adding several question records at once.

Update a specific question's topic

```
UPDATE tblQuestion
SET qTopic = "Algebra"
WHERE qNo = integer;
```

Add a topic to the database

```
INSERT INTO tblTopic(topicNo, topicName, correctQs, wrongQs)
VALUES(?, ?, ?, ?);
```

SQLite Python Statements for Proposed System

Connect to database

```
import sqlite3
con = sqlite3.connect("math.db")
cur = con.cursor()
```

The variable con represents the connection to the database.
The variable cur represents the cursor created.
The connect statement should have a filepath input that references the db.

Execute a statement

```
cur.execte("SELECT * FROM tblQuestion")
```

Insert multiple rows of data at once:

```
INSERT INTO tblQuestion(qNo, qText, qImage, qRating, qTopic,
qType, qMarks)

questions = [
                    (1, "x + 3 = 7, Solve for x", null, 1,
"Algebra", "Auto", 1)
                    (2, "Plot graph f(x-2)", "..\Q2image.png",
"Graphs", "Manual", 7)
                    (3, "Area of this triangle?",
"..\Q3image.png", "Geometry", "Auto", 7)
]

cur.executemany("INSERT INTO tblQuestion VALUES(?, ?, ?)",
questions)
```

Commit (save) the database

```
con.commit()
```

Close the database

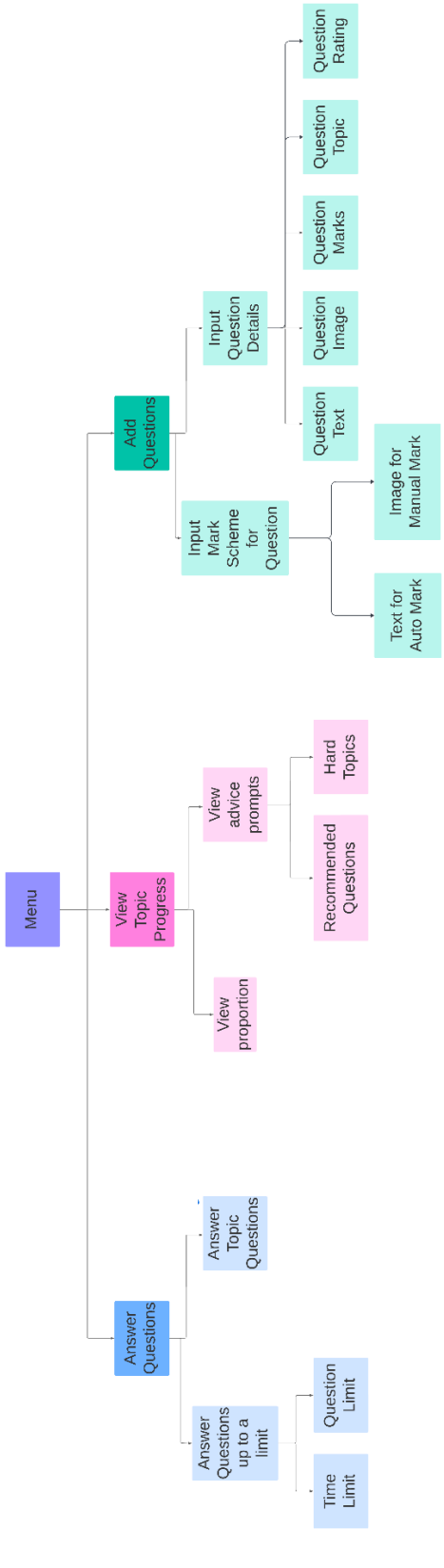
```
con.close()
```

Verify database has been written to

```
con2 = sqlite3.connect("math.db")
cur2 = con2.cursor()
fetch = cur2.execte("SELECT qNo, qText FROM tblQuestion WHERE
qNo=1")
qNo, qText  = fetch.fetchone()
print(f'{qNo}. {qText})
```

Should return "1. x + 3 = 7, Solve for x"

Proposed System Structure Diagram



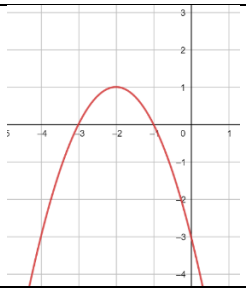
Add Questions – The user should have the ability to add questions, by inputting the question details, and the answer to the question, and steps to answering the question. The system should provide the ability to add questions that can be automatically marked, and manually marked, based on the type of answer.

View Topic Progress – Topic progress should be provided for each topic, showing how well the user is doing, statistics such as proportion should be shown and advice prompts should be given to help guide the user towards improvement in these areas.

Answer Questions – The system should allow the user to answer questions up to a limit number of questions, answer questions up to a certain time limit, and answer questions from specific topics geared towards the ones they struggle with the most.

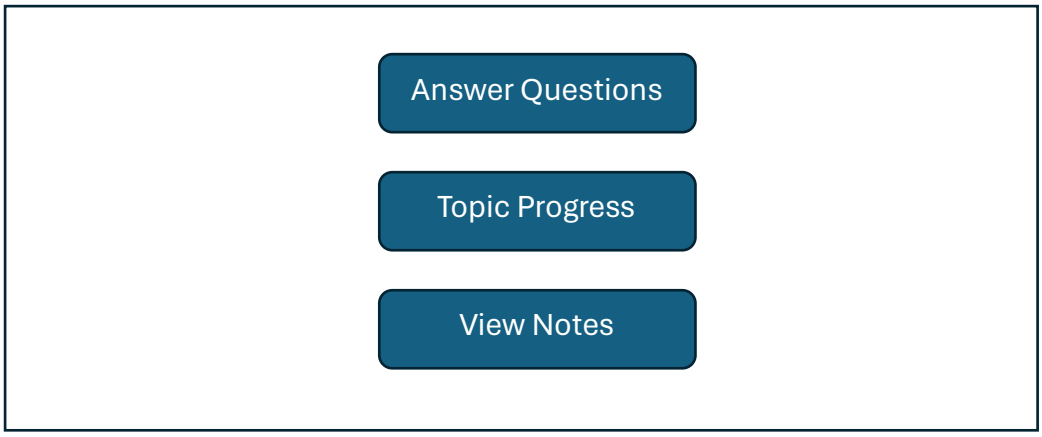
Final Question Types

In the Analysis, I went through some example questions and looked at the possibilities of what can be easily implemented, what could also be implemented but required some workarounds to allow them, and what most likely couldn't be used. I am finalising these decisions on what type of questions will be accessible via the proposed system.

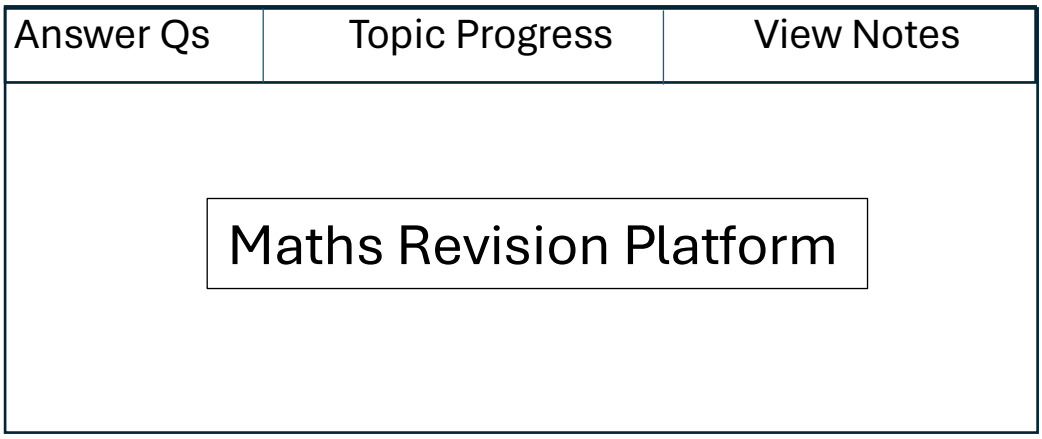
Format	Planned	Why?	Example Question
Numerical Answer	Yes	It is a common question in most Maths exams, easy to implement for automatic marking in the proposed system.	$x^2 - 2x + 1 = 0$ Solve for x x = -1 Correct Input: "-1"
Word / Multi-Numeric Answer	Yes	It's another common question in most Maths exams, fit for implementation with manual marking.	P(A) = 0.8 P(B) = 0.6 P(A shared B) = 0.48 Are A and B independent events? – "Yes"
Graph / Image	Yes	Very common topic and question in Maths exams, can be implemented with manual marking.	
Multiple choice	No	It is possible to implement them and they can appear in certain circumstances, but in Maths exams undertaken by the client, these types of questions will not be present.	7x – 4y = 0 3x – 2y = 5 Solve for x A) 7 B) -10 C) 10 D) 5 Correct Input: "B"

A disadvantage of the proposed system is that for a wide range of question types it relies on manual marking. This is also true for other systems. However, especially if they wanted to be accurate with their marking according to the mark scheme, the only real way to get around this is some form of machine learning which still wouldn't be entirely accurate and would probably be extremely difficult to implement, and harder for the user to even use, as they would have to somehow submit a picture of their work too.

User Interface for Proposed System

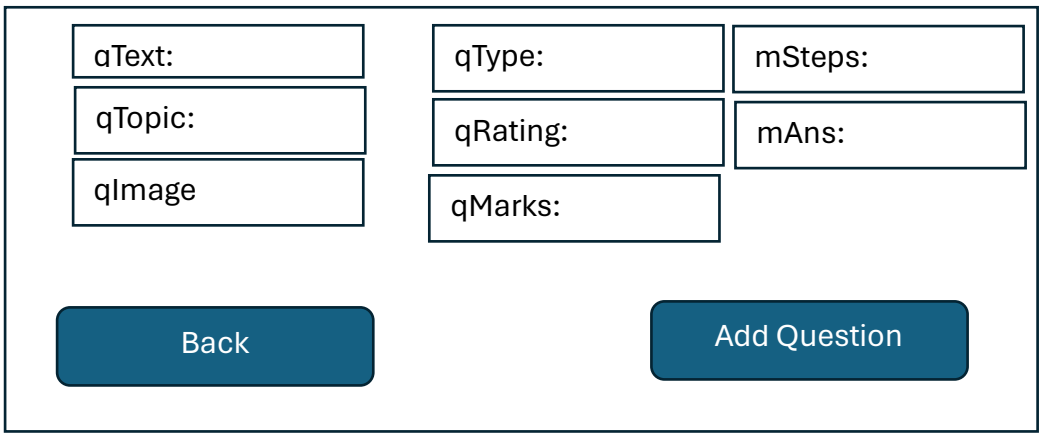


Starting Menu UI – what appears when you first start up the application
1st button redirects to "Answer Questions" Menu
2nd button redirects to "Topic Progress" Menu
3rd button redirects to "View Notes" Menu
This is a basic main menu implementation with 3 buttons for each submenu. This is easy to understand and use, but somewhat slower than other implementations.

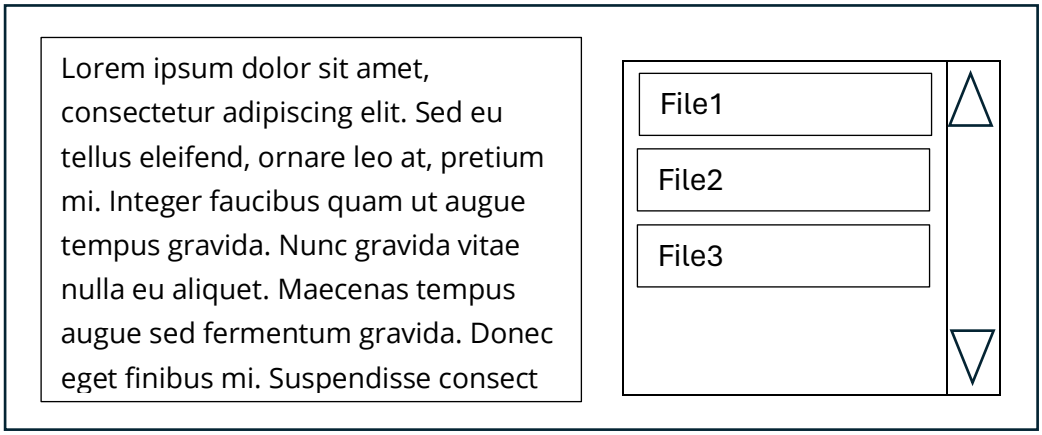


Alternatively, I could use tabs, which redirect to the desired submenu, making every menu accessible from every submenu, including the main menu. It can also show which menu is currently being selected too, making it easier to understand what is happening for the client.

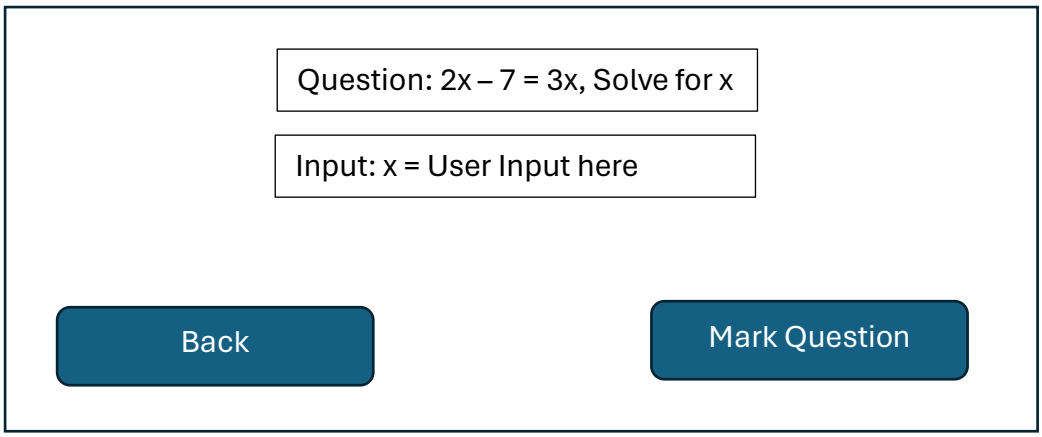
Both examples are practical and not that time-consuming, while also being fairly easy to implement and easy for the client to use. I will use the 1st implementation as I think the user would be more familiar with using buttons for navigation and not using tabs allows the user to keep multiple windows for separate tasks all open at once.



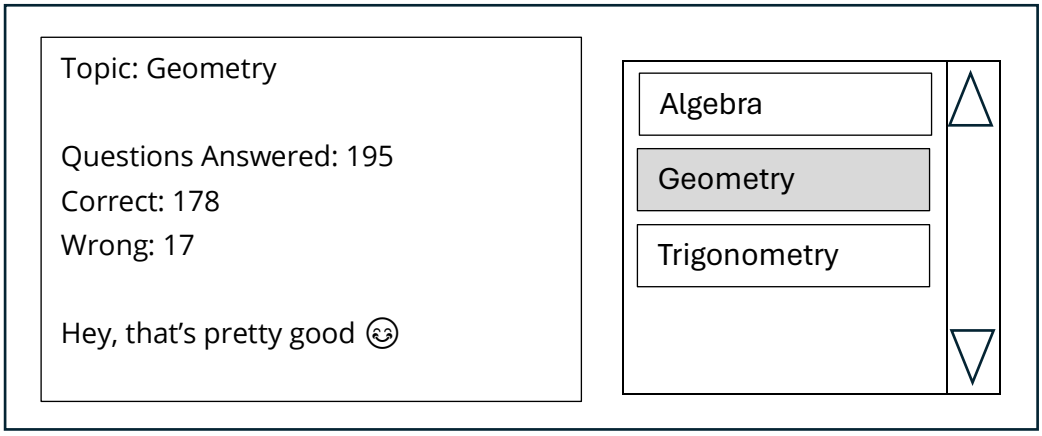
Add Questions Menu: There are 6 input fields for entering all the question details, the "Add Question" button makes a new record in tblQuestion with each of those details placed in the associated fields, with the right data type and validation restrictions put in place automatically. There's also a back button to return to the main menu. This is a fairly basic user interface that is easy for the client to understand and use but is somewhat inefficient when it comes to entering multiple questions at once, as the user will have to manually enter each data item into each input field, and click the add question button, each time they enter a question. The system also has to clear the contents of the input fields each time the Add Question button is pressed.



This is the user interface for the **Notes viewer**, which is separate from the main processes and operation. It simply acts as a document viewer for files stored on the user's system, so they can view their notes in the same software as where they can answer their questions, which improves ease of use for the client, who won't have to have multiple applications open at once, potentially impacting speeds. (The actual impact on speeds is trivial because this software is very small, and a document viewer isn't going to be that CPU intensive). This UI is easy for the client to understand and use, with a scroll wheel for the user to choose the files from a directory of their choice, and a box that displays the content of the files on the right.



This is the user interface for the **Answer Questions submenu**. There's a dynamic text box, that changes its text each time a question has been answered and the Mark Question button has been presed, and an input field for the answer. The Mark Question button is used to take the final input, compare it to tblMarkScheme in the database, and output correct, or incorrect, and then append values in the tblTopic according to whether the question was answered correctly or not, and show the updated values for that topic in the interface too.



This is the user interface for the **Topic Progress submenu**. There's a box on the right, which outputs the topic selected, the total amount of questions answered from that topic, the number of questions answered correctly, and the number of questions answered wrong, and a statement based on how good they are doing at that topic. On the right, it shows a list of topics, with a scroll where, where the user can select a different topic, to change the content displayed in the box on the left.

However, as a complete alternate to the tab system, I could use the start menu as a base navigator to open up pop-up windows, this would allow the user to do multiple things at once, instead of switching between menus. This is much more efficient and allows for ease of use between different tasks so I have decided I will do this instead.

Security and Integrity of Data

Even if there's unauthorized access, they won't be able to access any secure data as the system doesn't store that, only public access questions that are stored, and how many questions the user has answered. The system will only be installed on the local computer itself, so other people getting access to it is unlikely, and either way it wouldn't be an issue.

To keep data integrity, each field of data that could possibly be inputted to the database via the system has validation processes in place (as shown earlier in the data dictionary) to ensure that everything works as intended, and no errors occur when using the system.

Potential Options for Distribution and Storage

The system is only going to be used on a single computer, and only requires the installation of Python 3, which doesn't take up much storage at all, and can be run on most modern operating systems, including the one that the user has, Windows 10. The program could be given in several forms of storage.

The modern standard for distributing software, is simply just sending an executable file to the user, either via a file-hosting platform or a messaging service that supports file transfers beyond the storage limit of the executable.

The user can then download the file, which shouldn't be difficult, as they have internet access. The file can be stored on a USB drive, for easy transportation of the software, but the read-write times would be slower. The client probably doesn't need to transport the software to multiple places as they use a laptop which is fairly portable.

The client's laptop has a solid-state drive which is optimal for the program (and the database which is also stored on the system locally), it has the fastest read-write speeds out of any software, it has enough storage space for the program, and it has additional free space, so no matter how much the database increases in size, it should be able to support it no matter what. Even with millions of questions + mark schemes, and thousands of topics stored, the storage size still wouldn't exceed 500 MB according to my approximations. This may change based on the number of questions though.

Data Manipulation Algorithms in Pseudocode

For when I'm adding questions to the tblQuestion and the associated tblMarkScheme inputs, I will be adding 1 question at a time as shown in the interface I used earlier. Therefore, I will only need to use regular INSERT statements, doing them one at a time, and not implement extra sqlite3 functionality.

```
qInputs = [qNo, qText, qImage, qRating, qTopic, qType, qMarks]
mInputs = [mNo, mSteps, mAns]
EXECUTE('INSERT INTO tblQuestion VALUES' + qInputs)
EXECUTE('INSERT INTO tblMarkScheme VALUES' + mInputs)
```

I haven't added a user interface for adding topics, as I planned to just add all the topics beforehand, for a Year 9/10 Maths curriculum that would be necessary, as the main use of the software is for the client. However, the software could be used by practically anyone studying any subject whatsoever, due to the support for manual marking. Therefore, I will add a way to add topics, which is possible using this algorithm.

```
topicName = INPUT "Topic name: "
correctQs = 0
wrongQs = 0
topicInputs = [topicName, correctQs, wrongQs]
EXECUTE('INSERT INTO tblTopic VALUES' + topicInputs)
```

Technically, I don't need to add a topicNo value as I can just make it auto increment with sqlite3. Since the starting value for the number of correctQs and wrongQs for a topic will always be 0, as they haven't answered any questions yet, I can just automatically add the same value, so I'd only need to add a record, with the topicName field changed.

File Management

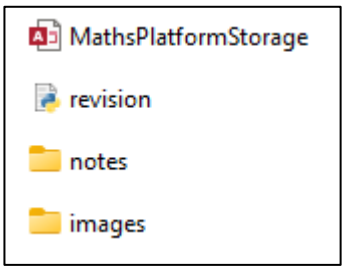
I only plan on using 1 python file for my entire program. I know for a lot of bigger projects, people will split the program into multiple files, one to start it, and each file providing a different part such as functions that can be carried out, or user interface. My code itself, isn't that advanced, with the main processes, being switching between questions, adding questions, and keeping track of progress. Therefore, I can probably do the user interface and put all the OOP functionality within the same file.

I'm only using 1 database, which has 3 tables, so that only requires a singular database file, which I'll place in the same directory as the python script.

The only other files I would need to store, are images and notes that need to be displayed for the questions. I will have folders in the same directory and the images will be named according the question number of the question they are used for.

Overall, it looks like this:

- Maths Revision Platform
  - MathsPlatformStorage.db
  - revision.py
  - /notes
  - /images



Potential Testing Plan

There are 3 main types of data that I need to check for in the test:

- Normal data: valid data that is within the accepted range
- Boundary data: valid data that falls at the boundaries of the accepted range
- Erroneous data: invalid data that falls outside the accepted range

Test Details	Normal	Boundary	Erroneous	Prediction
Adding a question				
Answer a question				
Delete a question				
Adding a topic				
Update topic progress				
Navigate topic progress				
View topic progress				
Navigate notes directory				
View notes				

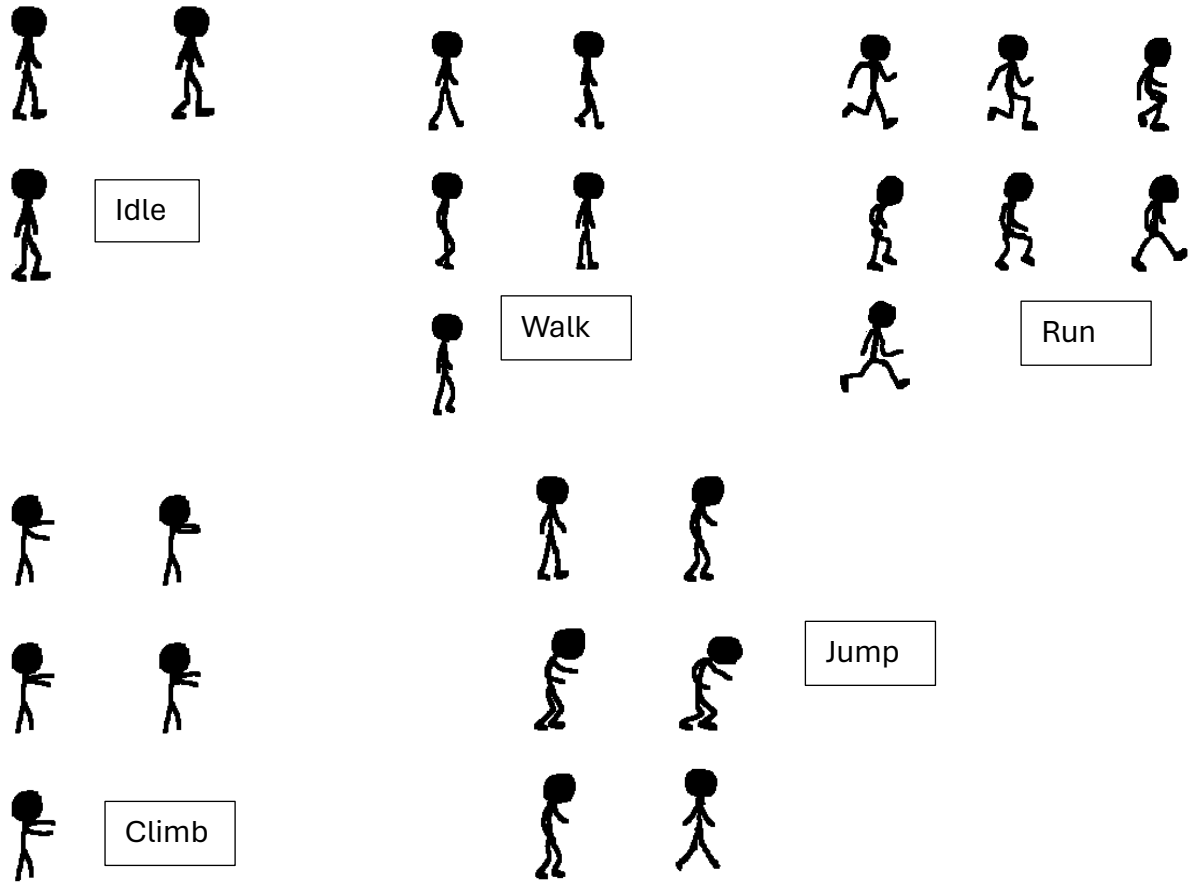
I will add this data when I have finished writing the software, as it is possible that I will make changes in terms of validation. This is because predicting the exact test data that I'm going to use before writing the program itself wouldn't be as efficient or easy to do.

However, I have put all the main processes that the user should be able to perform in some way through the menu UI or the command line itself. Through this, I want to ensure that I haven't made any errors in my restrictions for data inputs.

Game Implementation

The game, called Jimmy, is being made using Godot, a popular game engine. Godot utilizes objects called scenes, to load the current window being displayed. Scenes can also be used to create game items which can be fetched and added to other scenes. I'm using these scenes to create levels, which use nodes as the level objects. The scripts I've made in Godot use its own language GDScript which is very similar to Python.

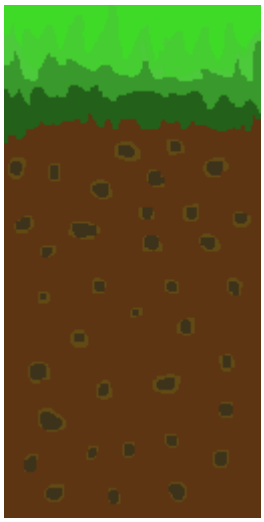
Firstly, I had to design a character (a simple stickman), so I drew some animations for him, an idle animation (when not moving), a walking animation, running animation (when SHIFT key is held), jumping animation (when the character is in mid-air), and a climbing animation (when staying on a wall).



Since the animations are for different actions and have varying frame counts, I'm actually playing these animations at different framerates within Godot. This is so that I can control the speed that I want them to loop at. The idle animation plays at 3 FPS, the climb animation at 8 FPS, the jump animation at 12 FPS, the run animation at 12 FPS, and the walk animation at 6 FPS.

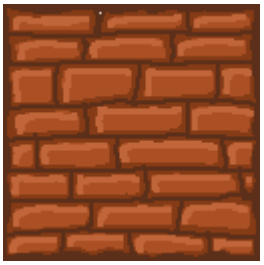
The animations have to loop to form a smooth pattern so I have made them loop.

The player requires a physics object so I created some simple world textures. These textures will be used for building the levels throughout the game.

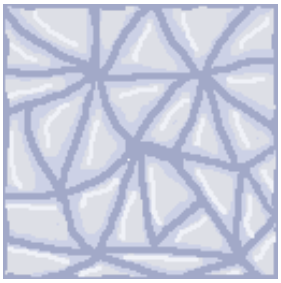


Dirt/grass combination

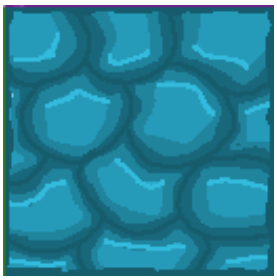
Stone



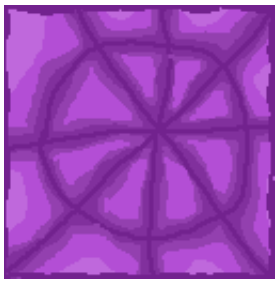
Red brick



White Diamond



Blue Sapphire



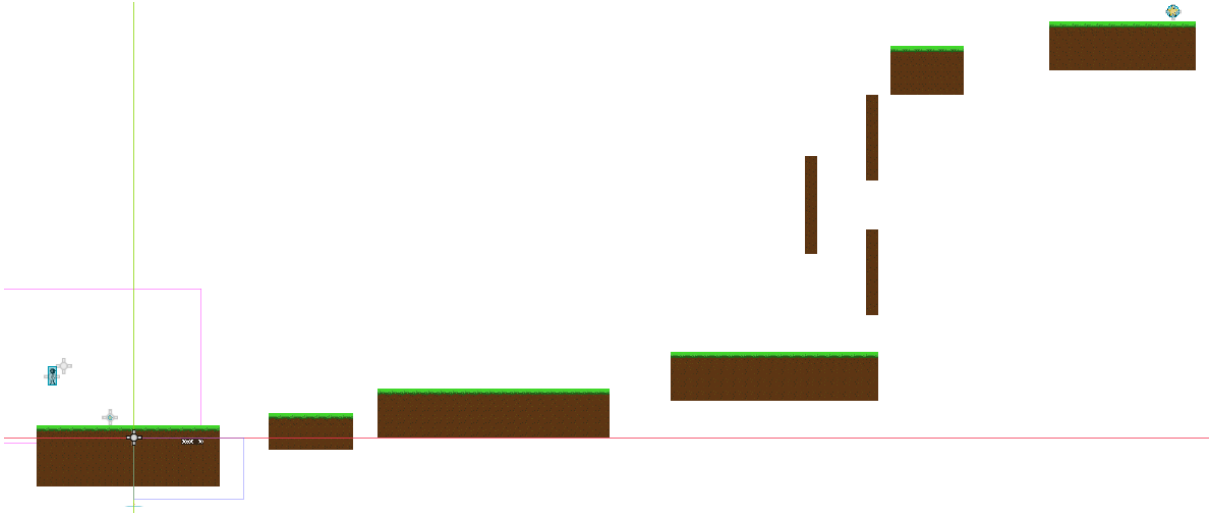
Purple Amethyst

Green Emerald

I also need something that stops the player, "killing" them and resetting their position to the start of the level, so I added this spike. When the level is finished, I need an object that sends the player to the next level, so I added this star.

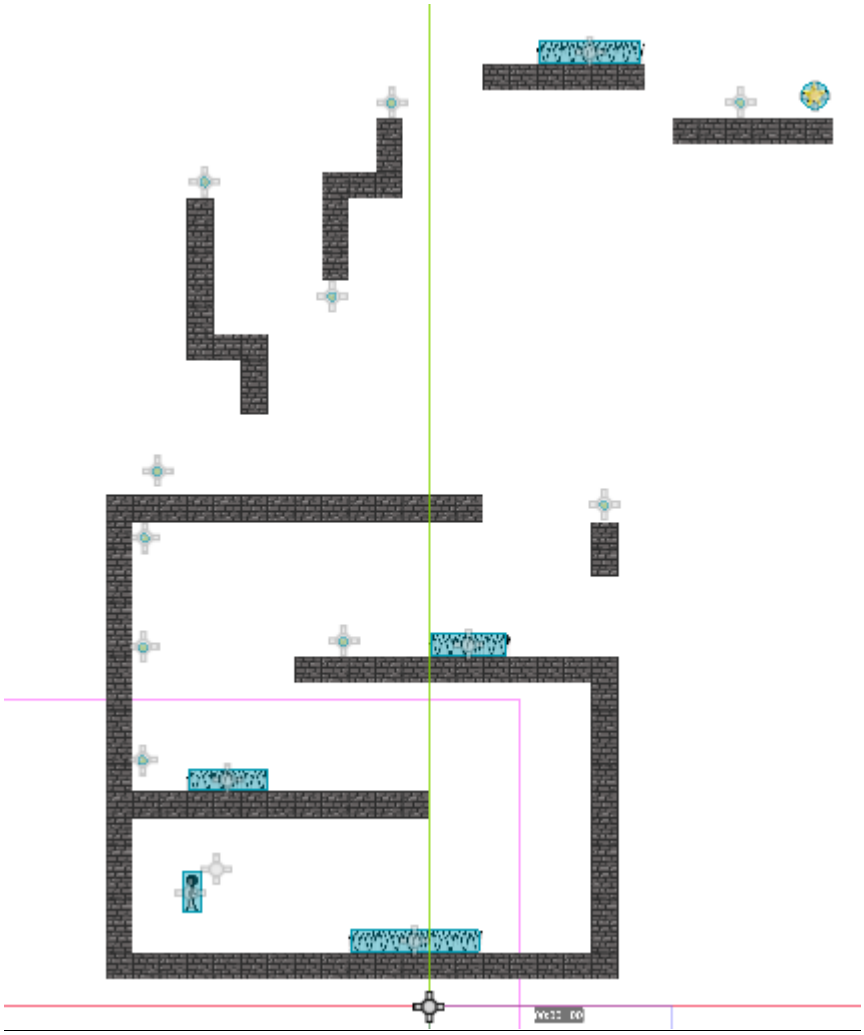
I have shown the levels I built using these textures below.

Level 1

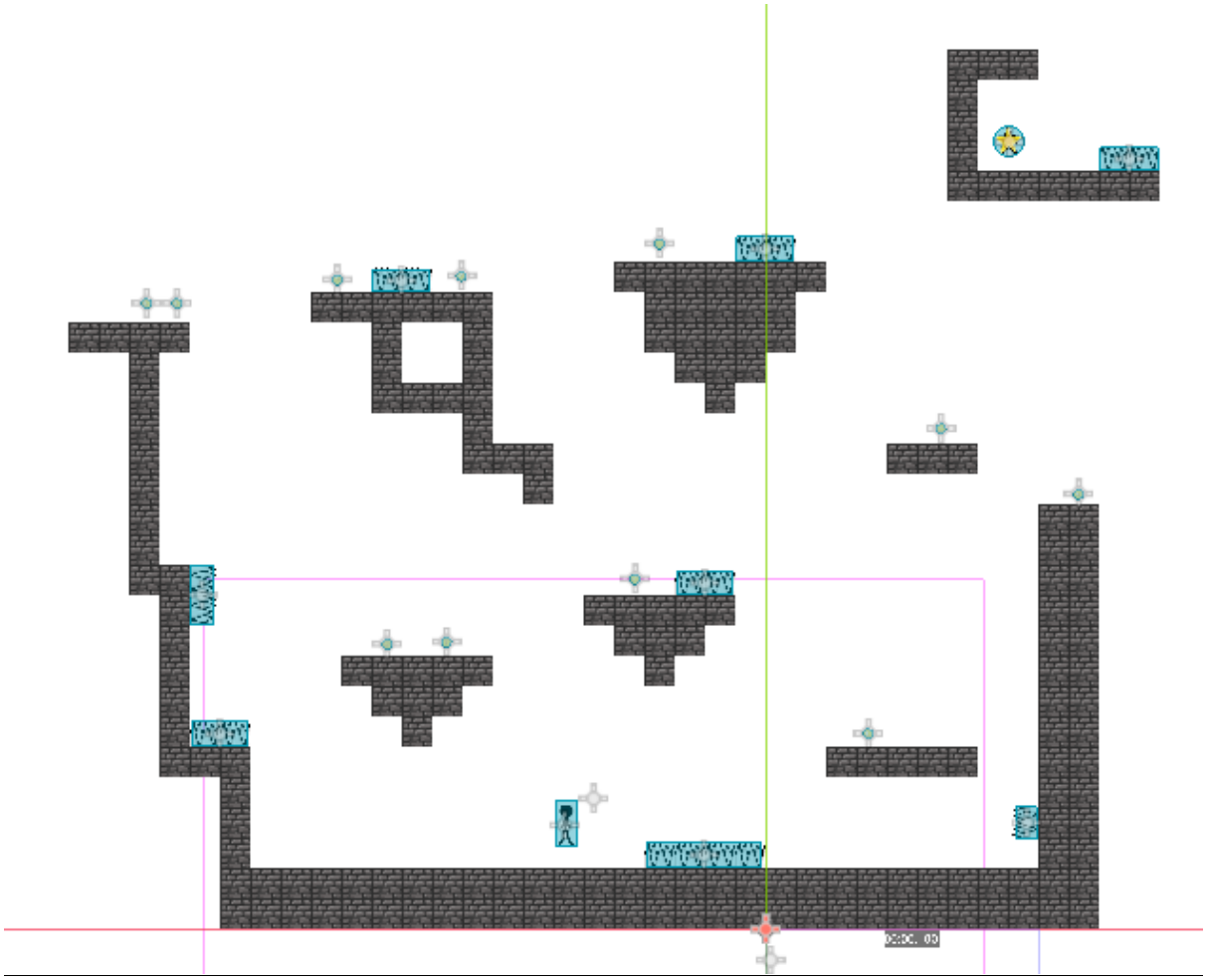


I want to implement these walls, which with a simple movement system wouldn't be possible to get past, so it appears I need to implement some form of wall jumping / sliding.

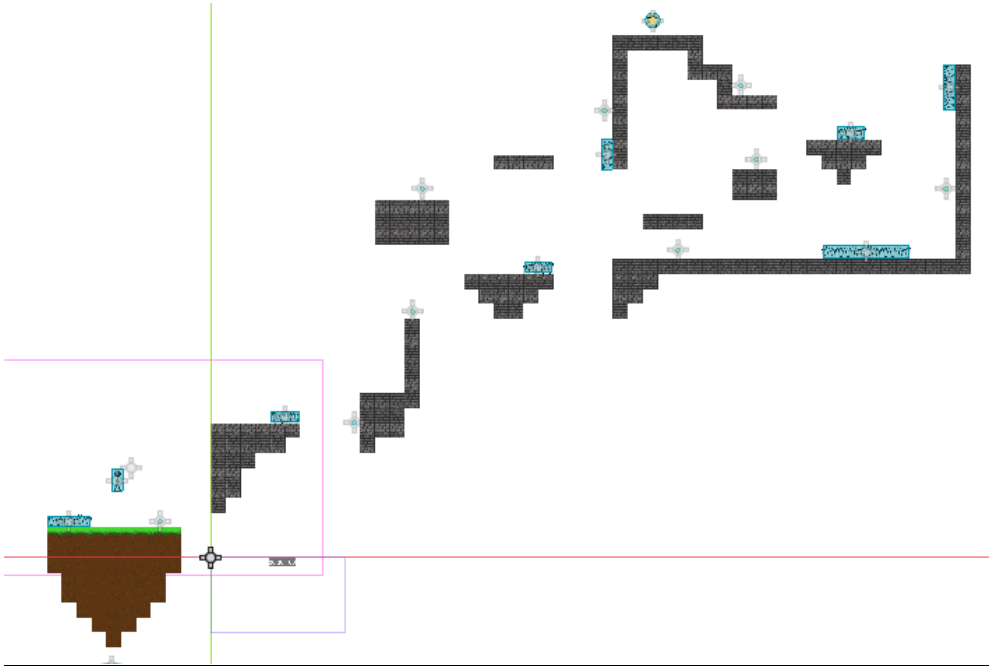
Level 2



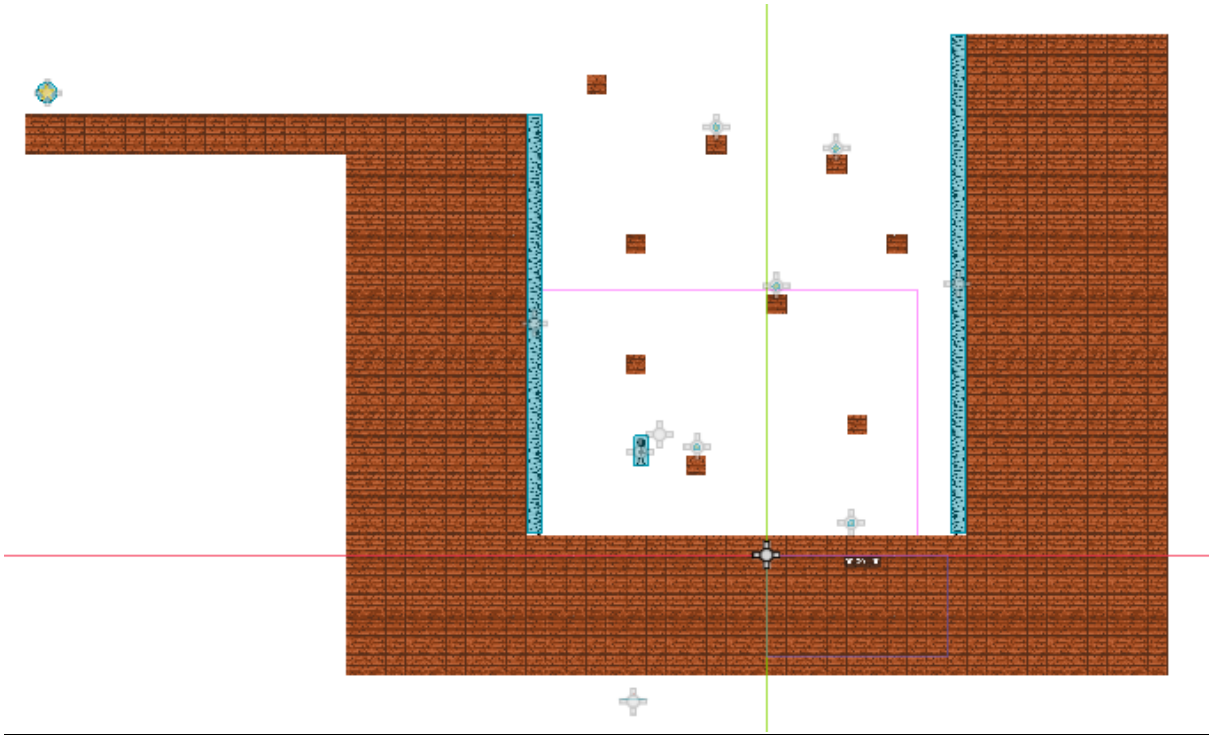
Level 3



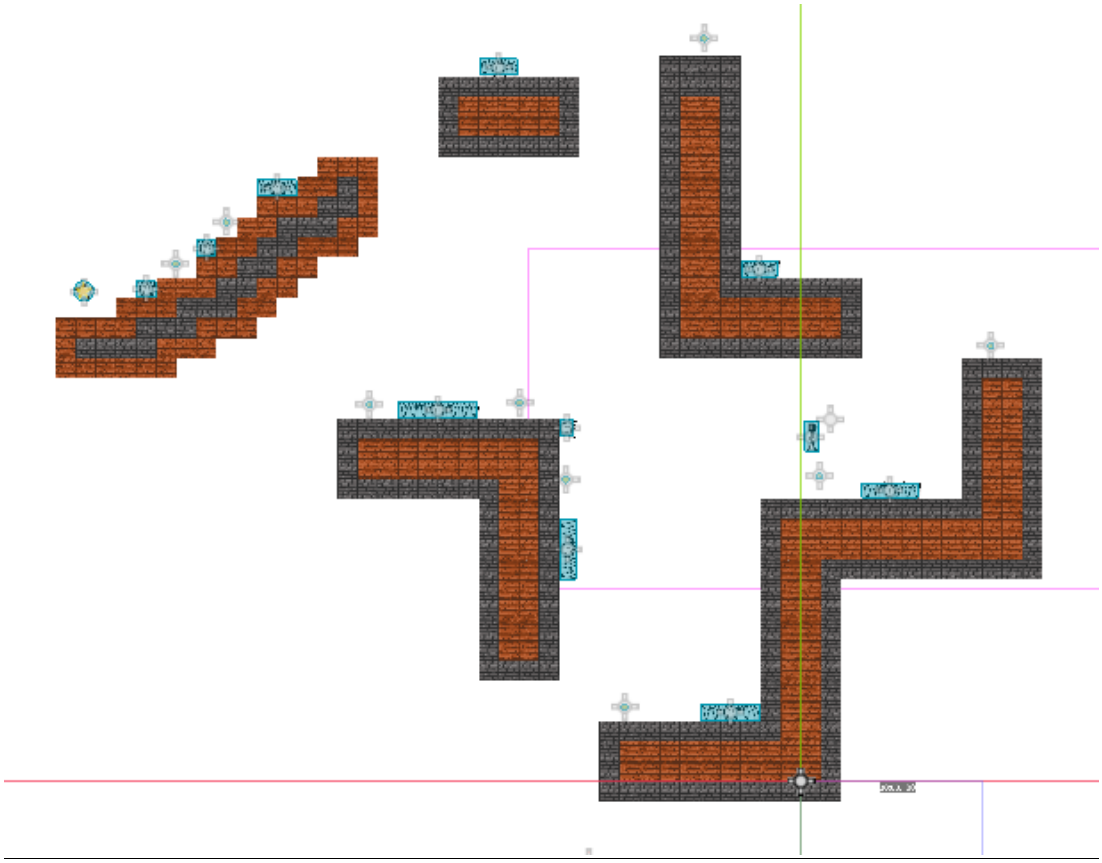
Level 4



Level 5



Level 6



I have created these levels for World 1 of the game using a Godot 2d node called a Tilemap. This allows you to place square tiles as textures. By creating a physics layer for these textures, it allows them to be used as a physical object with which the player can stand and move on.

However, the spikes are not physics objects, because the player should be able to interact with them, by entering them, and triggering a scene reload.

Additionally, it is possible to fall off the map in all of these levels from certain positions. To prevent the player from falling infinitely, I have placed a world boundary node at the bottom of each level, which should trigger a scene reload when entered.

Player Movement

The movement should allow the player to: stand still, move in both directions, walk at a slow speed, run at a fast speed, jump, and fall while jumping. This can be implemented with a basic movement script.

Alongside that, all of my levels require a form of wall-jumping and sliding to be added, allowing the user to progress past certain points.

Additionally, the movement should allow for more leeway, so the user doesn't have to perfectly time jumps when landing. This can be achieved through implementing a jump buffer, giving the user a time gap between the point at which they land on the platform, so that the jump will still register even when pressing the jump key.

The movement should also give leeway when jumping after coming off a platform, so the user doesn't have to jump at the pixel perfect moment. This can be achieved through implementing coyote timing, which allows the user to jump, even when not in contact with a platform, within a certain amount of time of coming off the platform.

Shown below is the movement script I created to implement all of this.

```
extends CharacterBody2D


var SPEED = 1200 * time_manager.multiplier
var JUMP_VELOCITY = -1500.0 * time_manager.multiplier
@onready var sprite = $AnimatedSprite2D
@export var coyote_time = 0.1
var jump_available = true
var jump_buffer = false
var jump_buffer_time = 0.1
@onready var sfx = $"../sfx"
var accel = 20
@onready var global_timer = %GlobalTimer


func _physics_process(delta):
	if Input.is_action_pressed("menu"):
		time_manager.time = 0

	get_tree().change_scene_to_file("res://scenes/menu2.tscn")
	#if Input.is_action_pressed("reset"):
		#global_timer.get_time_formatted()
		#get_tree().call_deferred("reload_current_scene")
	# Add the gravity.
	if not is_on_floor() and not is_on_wall():
		if jump_available:

	get_tree().create_timer(coyote_time).timeout.connect(coyote_ti
meout)
		sprite.play("jump")
		velocity += get_gravity() * delta * 3
		if velocity.y > 0:
			velocity += get_gravity() * delta * 1.5
```

```
            elif velocity.y < 0 and not
Input.is_action_pressed("jump"):
                  velocity += get_gravity() * delta * 1
      else:
            jump_available = true
            if jump_buffer == true:
                  jump()
                  jump_buffer = false
      if is_on_wall_only():
            velocity.y += 220 * delta * 3
      if is_on_wall():
            if Input.is_action_pressed("move_right"):
                  sprite.flip_h = false
                  sprite.play("climb")
            elif Input.is_action_pressed("move_left"):
                  sprite.flip_h = true
                  sprite.play("climb")



      if Input.is_action_pressed("dash") and not is_on_wall():
            SPEED = 1600.0 * time_manager.multiplier
            if Input.is_action_pressed("move_left"):
                  sprite.play("run")
                  sprite.flip_h = true
            elif Input.is_action_pressed("move_right"):
                  sprite.play("run")
                  sprite.flip_h = false
      if Input.is_action_just_released("dash"):
            SPEED = 1200.0 * time_manager.multiplier


      if Input.is_action_pressed("move_left") and is_on_floor() and
not Input.is_action_pressed("dash"):
```

```
            sprite.play("walk")
            sprite.flip_h = true
      if Input.is_action_pressed("move_right") and is_on_floor() and
not Input.is_action_pressed("dash"):
            sprite.play("walk")
            sprite.flip_h = false
      if not Input.is_action_pressed("move_right") and not
Input.is_action_pressed("move_left") and is_on_floor():
            sprite.play("idle")
      # Handle jump.
      if Input.is_action_just_pressed("jump"):
            if jump_available:
                  if not sfx.playing:
                        sfx.stop()
                        sfx.jump_sound()
                  jump()
            else:
                  jump_buffer = true

      get_tree().create_timer(jump_buffer_time).timeout.connect(on_j
ump_buffer_timeout)

      # Get the input direction and handle the
movement/deceleration.
      # As good practice, you should replace UI actions with custom
gameplay actions.
      var direction = Input.get_axis("move_left", "move_right")
      if direction:
            velocity.x = move_toward(velocity.x, direction * SPEED,
150.0)
      else:
            #if sprite.flip_h == false and velocity.x > 0:
                  #velocity.x -= accel
            #if sprite.flip_h == true and velocity.x < 0:
```

```
                #velocity.x += accel
            velocity.x = move_toward(velocity.x, 0, SPEED)
      move_and_slide()



func jump()->void:
      if is_on_wall():
            velocity.y = JUMP_VELOCITY * 1.3
            velocity.x += 900 * get_wall_normal().x
      else:
            velocity.y = JUMP_VELOCITY
      jump_available = false

func coyote_timeout():
      jump_available = false

func on_jump_buffer_timeout()->void:
      jump_buffer = false
```

Level Progression System

I used a script to send the player to the next level. The next_level script is linked to the Area region of the star object. When the player's body has been detected within the area, the name of the current scene is fetched (e.g. 'level_4) and using string slicing it gets the level number (4) and adds 1 to it to get the next level number. Then, if the new level isn't level 7 (because level 7 doesn't exist in World 1), the next level scene is loaded. However, if level 6 has just been completed, the time obtained by the player is saved to the leaderboard client-server backend via the SilentWolf module.

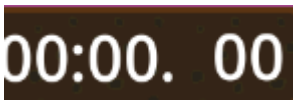
```
extends Area2D

@onready var coin_manager = $"../coin_manager"

@onready var coin_count = $"../CanvasLayer/coin_count"

@onready var sfx = $"../sfx"

@onready var global_timer = %GlobalTimer


func _on_body_entered(_body):

	var current_scene = get_tree().current_scene.name

	var current_level_number = int(current_scene.get_slice("_",
1))

	var new_level_number = current_level_number + 1

	global_timer.get_time_formatted()

	if new_level_number == 7:

		Global.save_score(button_manager.player_name,
time_manager.time, "main")

	if new_level_number != 7:

		var new_level_path = "res://scenes/level_" +
str(new_level_number) + ".tscn"


	get_tree().call_deferred("change_scene_to_file",new_level_path
)

	else:

		time_manager.most_recent_time = time_manager.time

		time_manager.time = 0

		time_manager.multiplier = 1
```

```
	get_tree().call_deferred("change_scene_to_file","res://scenes/menu2.tscn")
	coin_manager.coins = 0
	coin_count.text = str(coin_manager.coins)
```

Timer System

Each level has a timer at the top of the screen, which starts when the game is started, and resets to 0, after each level is cleared. I also wanted to store the total time across all levels, for use in a leaderboard system. However, Godot does not have a simple way to store values, across different scenes. To counter this, I created a script which takes the time value after a level's completion and adds that value to the total_time variable of a global script, which prevents the value from being reset, when going to a new level, or reloading the current level.

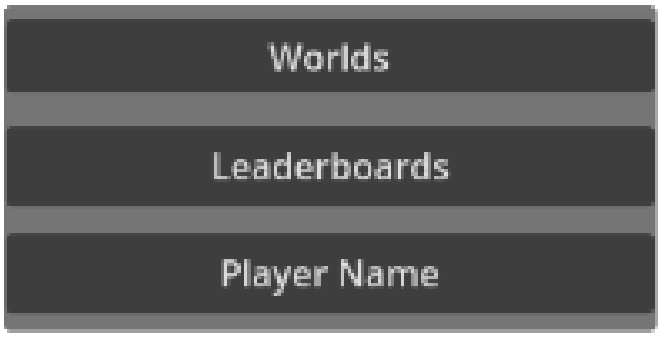
```
@onready var tminutes = %Minutes
@onready var tseconds = %Seconds
@onready var tmilliseconds = %Milliseconds
@onready var star = $"../../star"


# Called when the node enters the scene tree for the first time.
func _ready():
	pass # Replace with function body


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
	time+= delta
	milliseconds = fmod(time,1) * 100
	seconds = fmod(time, 60)
	minutes = fmod(time, 3600) / 60
	tminutes.text = "%02d:" % minutes
	tseconds.text = "%02d." % seconds
	tmilliseconds.text = "%02d" % milliseconds
```



```
func stop() -> void:
	set_process(false)


func get_time_formatted() -> String:
	time_manager.time += time
	return "%02d:%02d.%02d" % [minutes, seconds, milliseconds]
```

User Menu

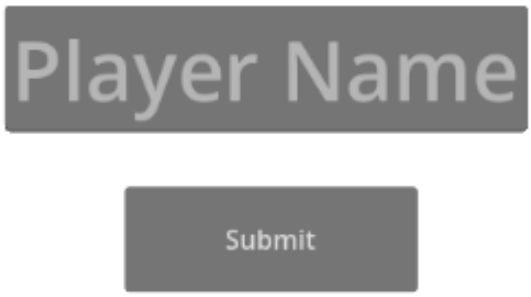
I created a user menu to allow the user to navigate between the leaderboard, the game itself, and the player name menu. It's fairly simple and just uses a few buttons. Below them, I also added an option that shows the most recent time that the user just achieved, as finishing the game takes you back to the menu, and there's an option to view your best time relative to the player name that you have selected.



Main Menu for navigation and viewing best time and most recent time

Script for showing best time:

```
extends Label

@onready var label_3 = $"."

func load_best_time():

	var sw_result = await
SilentWolf.Scores.get_top_score_by_player(button_manager.player_name
,0,"main").sw_top_player_score_complete

	var time = 1 / (sw_result.top_score.score - 1)

	var time_formatted = "%.3f" % time

	label_3.text = button_manager.player_name + "'s Best Time: " +
str(time_formatted)

func _on_refresh_pressed():

	load_best_time()
```



Setting player name menu, character range limit of 3-16

The player name set by the user is the one used for saving times to the leaderboard and fetching the player's best time.

Worlds Menu for starting the game

Leaderboard System with Client Server Backend

Using a module called SilentWolf, which is made for use with Godot, I wanted to utilise a leaderboard system to store the fastest times that the game was completed in by users. SilentWolf doesn't have a simple way to set up leaderboards for games that utilise timers, where the times need to be displayed from lowest at the top, to highest at the bottom. Additionally, the time value for a user needs to be updated once they get a lower time than their current fastest time. To implement this, it requires some workarounds so I modified the built in SilentWolf leaderboard script.

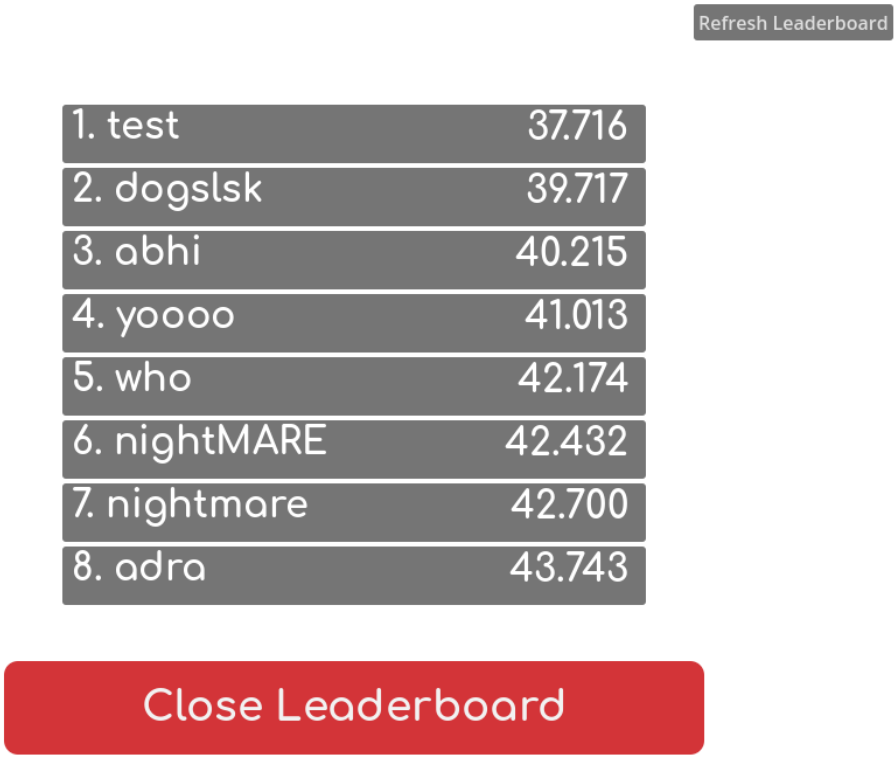
First of all, I set up the leaderboard to store the highest value obtained. Then I used the formula, 1 + (1/time), as this would mean a lower time gives a larger value to convert time to a score. I had to add 1 because SilentWolf leaderboards don't register decimal values between 0 and 1.

```
func save_score(name, time, lb):
	SilentWolf.Scores.save_score(str(name), 1+(1 / time), lb)
```

Then, when instantiating the leaderboard elements, I displayed the elements in reverse value order and then converted the values back to time.

```
func reverse_order(scores: Array) -> Array:
	if len(scores) > 1 and scores[0].score > scores[-1].score:
		scores.reverse()
	return scores
func add_item(player_name: String, score_value: String) -> void:
	var item = ScoreItem.instantiate()
	var score_val = 1/(float(score_value) - 1)
```

The scene I used below is the one provided by SilentWolf, as a default leaderboard style.



Maths Revision Platform Link

The game does also need to be maths-based, as this is a game for a Maths revision platform. To allow for this, I've drawn and added these coin objects. The user can collect these coins, and it increments a coin counter, which is displayed. I have shown below the script used to increase and update the coin counter.

```
extends Area2D
@onready var coin_count = $"../../CanvasLayer/coin_count"
@onready var coin_manager = $"../../coin_manager"

func _on_body_entered(_body):
	coin_manager.coins += 1
	coin_count.text = str(coin_manager.coins)
	queue_free()
```

I think I will set up a bubble powerup / obstacle. This has a random mathematical operation on it and a random value within a range of 1-20. When the player enters the bubble, that operation is performed with that value, on the current coin count. The goal is to avoid bad obstacles (subtract and divide) and interact with good powerups (add and multiply), in order to get the maximum amount of coins. I intend to add 4 these throughout each level. This is an example script I may use to perform this.

```
extends Area2D

var bubble = $"../../bubble"

func generate_values():
      operation = random.choice(['+', '*', '-', '/'])
      value = random.randint(1,20)
      bubble.text = operation + str(value)

func _on_body_entered(_body):
      coin_manager.coins =
eval(str(coin_manager.coins)+operation+str(value))
      coin_count.text = str(coin_manager.coins)
      queue_free()
```

Once I finish building the entire base system, with some small changes to the level progression script, making a new leaderboard, and making a set of new levels, I can create new worlds extensively. This makes my game easily updatable, allowing for improvements in the future, according to the changing needs of the client. I also want to make more of a direct link to the platform, so I may implement features such as a limited amount of time to play the game, depending on how much score you get from correct questions while revising.

How the Client-Server system works for fetching and saving times

save_score() – The player name is checked to ensure that it's a valid name, the leaderboard name is checked to make sure it's a string, and the metadata is checked to make sure it's in dictionary format. A HTTP request is prepared, and once it is successfully connected, the payload is fetched. The payload is a series of details for the unique score being saved, specifically the score_id, player_name, game_id, score, and leaderboard_name. The api-key and version headers (game-id, plugin-version, godot-version) are fetched, and the data being sent is converted to JSON string format. Finally, a HTTP request is made via the POST method, with the parameters being the request_url (being the server api link – api/save_score), the headers (version infos, api-key, and a hash value for error checking), and the query itself (dictionary containing player name, score, and the payload data).

This process is carried out in full every time a player gets a time that is lower than their best time. Their best time is initially fetched via this process below. The function is called get top score but that's because I'm saving the times as scores using an inverse function (1 + 1/time) as mentioned before, so a "top" score would be their lowest time.

get_top_score_by_player() – Similarly to before, a http request is prepared. When it has successfully connected, a GET request is made via the api through this request_url (api/get_top_score_by_player). Additionally, the request_url has parameters depending on the parameters of the function. This decides which leaderboard the time is being fetched from, whether it's a max or minimum time, and what player name is being searched for specifically. Data is being fetched so the only data submitted via a get request are the headers. If successful, the value returned is the best time for the player.

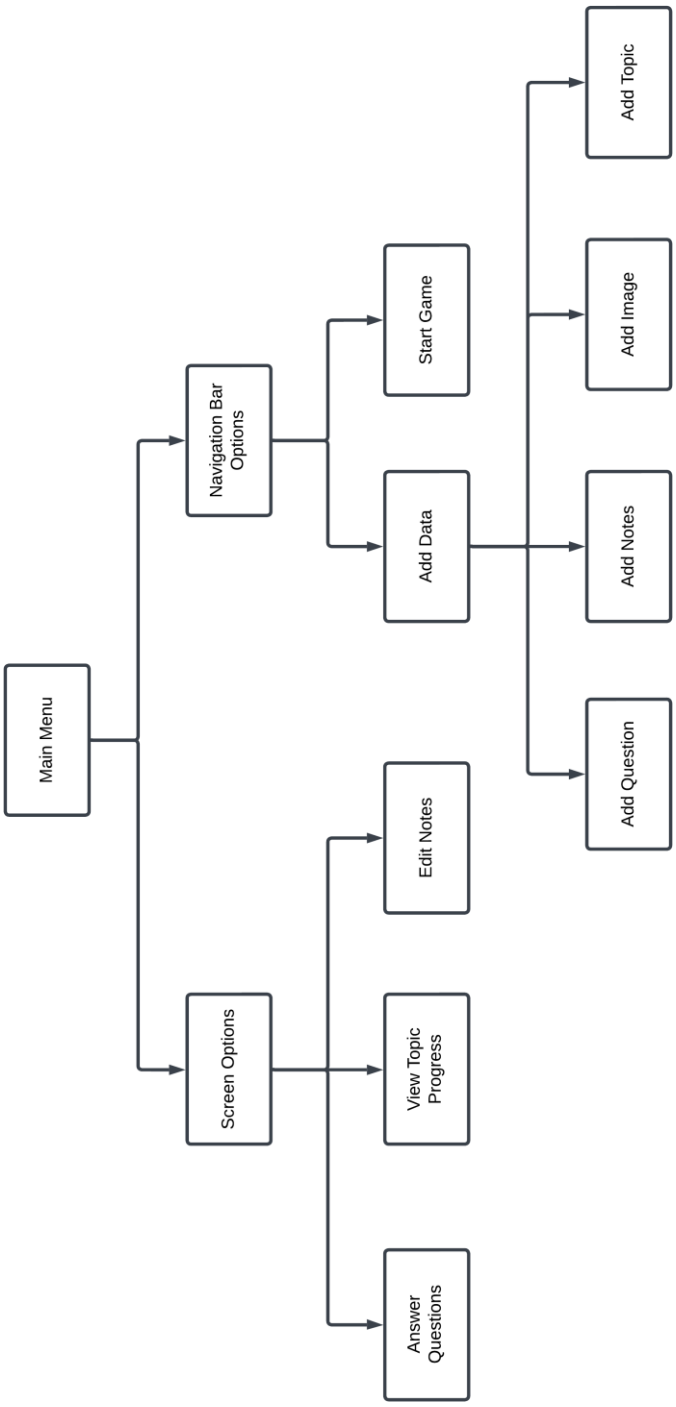
The leaderboard fetches the top 8 best times and lists them in ascending order. This is achieved through a fairly similar function (get_scores). The scores are fetched via a GET request, when the scores are returned. The JSON body is converted to a string which is formatted into a dictionary. This dictionary has the top scores for each leaderboard. The scores from whichever leaderboard is being fetched from are saved to a seperate leaderboards dictionary. This is then appended through and each of the 8 elements is added with the player name, and time, to make the final leaderboard.

Keep in mind that I didn't write some of the code for this functionality, I just utilised the function in the SilentWolf API based on my needs and I'm explaining how it works.

Implementation

System Overview

This is a structural breakdown of the system. The program starts with launching the main menu. The main menu contains screen options, to either answer questions, view topic progress, or view and edit notes. Additionally, there's a navigation bar, with options to add data (File sub-bar) or start the game (game sub-bar). Within the file sub-bar, there are specific processes to add questions, notes, images, or topics.



Techniques Utilised within Code

Queue Implementation and Operations: (Line 177, 292-295, 399-402)

A queue is used to track the last 5 questions answered by the user. When a question is answered, some of the main question details (question name, question number, result) are enqueued as an array to the back of the queue. When a question is added and the queue is full (5 questions in queue), the first element of the queue is dequeued, to make room for the new question.

Complex Client-Server Model with Server-Side Scripting: (In the Jimmy Game)

I have utilised a client-server model in the game. The server stores the best time of each player-name when a complete run of the game is achieved. This is achieved first by fetching the best time of the player-name with a GET request (via the SilentWolf API) and then checking if the time is lower. If it's lower, it sends a POST request to save the time. The leaderboard is generated by fetching the best times from the server and listing top 8 times in descending order. This is also achieved via a GET request through the SilentWolf API. I have 2 leaderboards for 2 separate worlds that I've made, so it saves to a different file depending on the level that has been completed at the end of a run (These processes are linked to the code explained much more in-depth within the Game Implementation section). **Keep in mind** that I didn't write the code for this functionality, I just utilised the function in the SilentWolf API based on my needs.

Parameterised Cross-Table SQL + Aggregate SQL Functions:

The menu that is used to add questions uses parameterised SQL. The question menu takes inputs for all the question details, and these inputs are used as parameters for an SQL INSERT INTO command which adds a record for the new question to tblQuestion. After checking the result of question, the tblTopic progress is updated for the specific topic of the question from tblQuestion, using the qTopic parameter depending on whether the user got the question right or wrong (Line 269-295). Within the Answer Questions menu, only the first question is randomly generated. The number of questions is fetched with the COUNT() function used on the tblQuestion (Line 188) and a random number is generated with the range of 1 to numberOfQuestions. Additionally, the MAX() function is used to get the largest question number so that in the Delete Question menu, qNumbers that don't exist aren't attempted to be deleted. (Line 745)

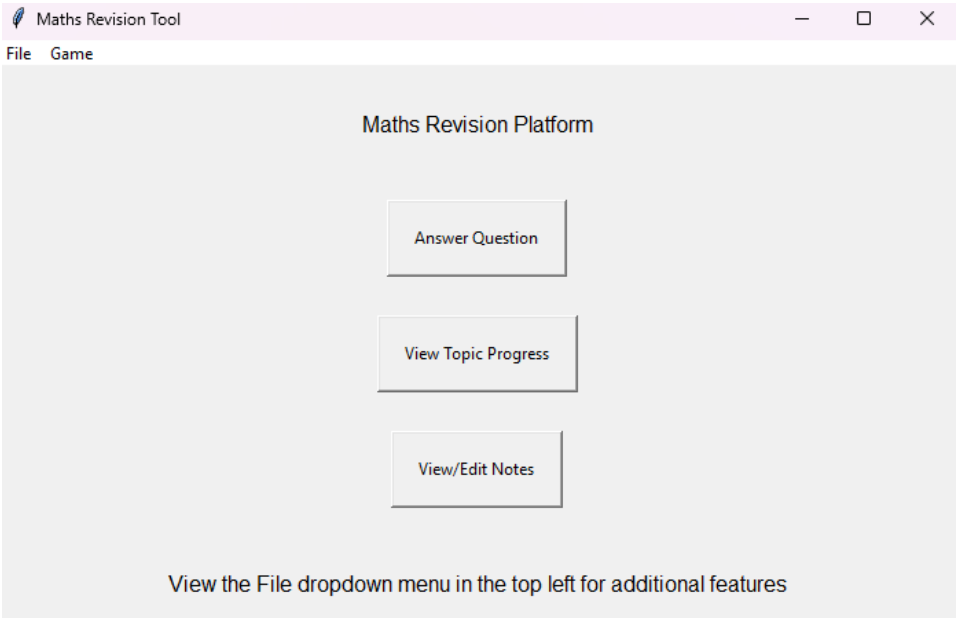
Recursive Algorithms: (Line 252, Line 317)

The functionality of the menu used to answer questions uses 2 functions that run each other, this is an example of mutual recursion. When the Show Answer button is pressed, the function show_answer() is run, the user's input is checked, and the button is changed to say Next Question. When this button is pressed, the function next_question() is run and the new question details are generated and displayed. The

button goes back to a Show Answer button, and this process repeats every time a user answers a question, checks the answer, and moves onto the next question.

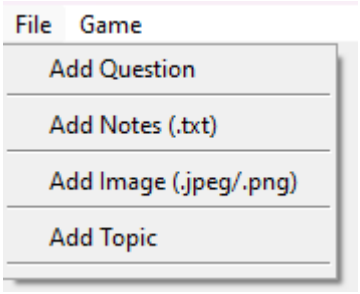
Code Listings with User Interface

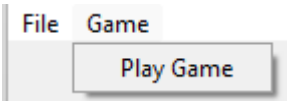
Main Menu



The buttons and dropdown menus open new popup windows, specifically the Answer Question, View Progress, and Edit Notes menus.

File Dropdown, Game Dropdown



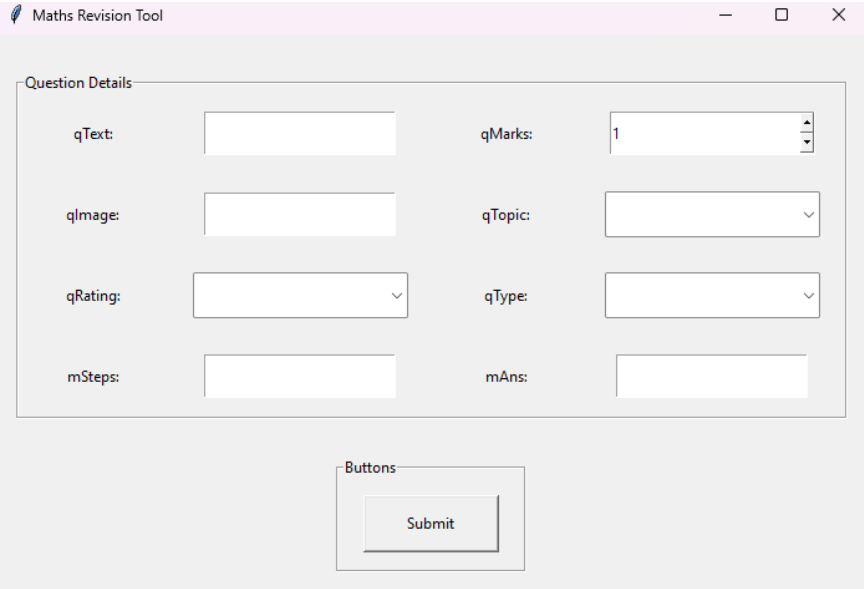


This allows the user to access the Add Question, Notes, Image, and Topic menus. The user can also play the game as part of the system.

```
window = tk.Tk()
window.geometry("700x500")
window.title("Maths Revision Tool")

menubar = tk.Menu(window)
window.config(menu=menubar)
operationMenu = tk.Menu(menubar, tearoff="off")
gameMenu = tk.Menu(menubar, tearoff="off")

menubar.add_cascade(label="File", menu=operationMenu)
menubar.add_cascade(label="Game", menu=gameMenu)

operationMenu.add_command(label="Add Question", command=add_question_menu)
operationMenu.add_separator()
operationMenu.add_command(label="Add Notes (.txt)", command=add_note)
operationMenu.add_separator()
operationMenu.add_command(label="Add Image (.jpeg/.png)", command=add_image)
operationMenu.add_separator()
operationMenu.add_command(label="Add Topic", command=add_topic_menu)
operationMenu.add_separator()

gameMenu.add_command(label="Play Game", command=play_game)

main_frame = tk.Frame(window)
main_frame.pack()

title = tk.Label(main_frame, text="Maths Revision Platform", font=40)
title.grid(row=0, column=0)

info_label = tk.Label(
    main_frame,
    text="View the File dropdown menu in the top left for additional features",
    font=25,
)
info_label.grid(row=4, column=0)

answer_question_button = tk.Button(
    main_frame, text="Answer Question", command=answer_question_menu
)
answer_question_button.grid(row=1, column=0)

view_progress_button = tk.Button(main_frame, text="View Topic Progress", command=view_progress_menu)
view_progress_button.grid(row=2, column=0)

view_notes_button = tk.Button(main_frame, text="View/Edit Notes", command=view_notes_menu)
view_notes_button.grid(row=3, column=0)

for widget in main_frame.winfo_children():
    widget.grid_configure(padx=20, pady=14, ipadx=15, ipady=15)

window.mainloop()
```

This generates the structure of the user interface for the main menu. First, the window is generated, and a menu bar is added to the window. Then, the frame is added to the window. Within the frame, the buttons to the View Progress, Answer Question, and Edit Notes menus are added. Within the menu bar, there are submenus for File and Game. Within the File submenu, you can access the Add Question, Image, and Topic menus.

Add Question



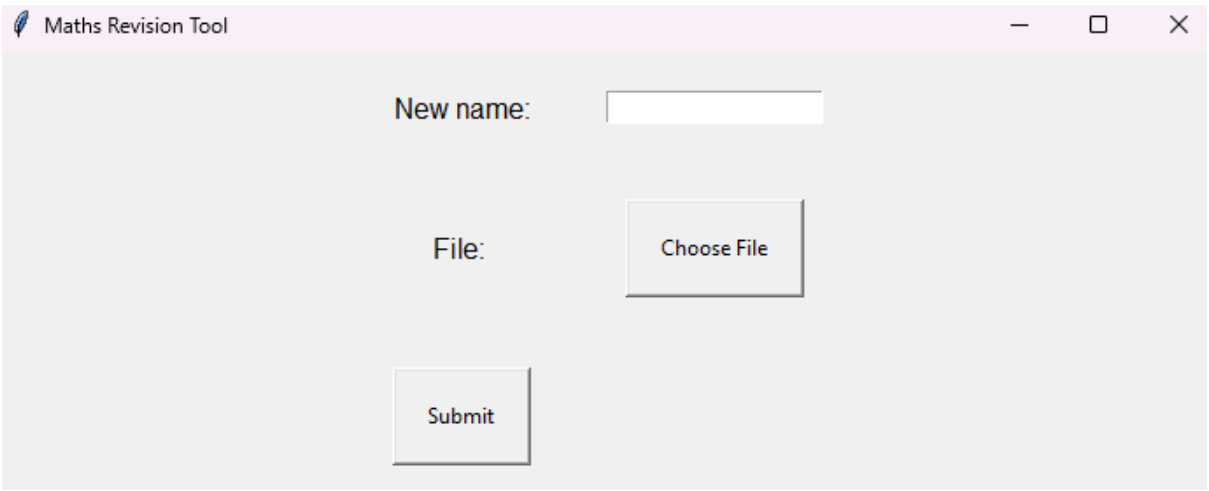
The menu allows the user to add question details, provides a list of options for setting question rating: 1 – Very Easy, 2 – Easy, 3 – Medium, 4 – Hard, 5 – Very Hard, and provides a list for setting question topic with the current topics added in tblTopic. The submit button runs an SQL statement that takes values from the inputs, and adds a question record to tblQuestion. This fulfils all parts of the Objective 7 – display text labels for each of the inputs for question details, display input boxes for each question detail, display add-question button, and run SQL command to add question to tblQuestion in the database when button pressed.

```
def add_question():
    #Assign appropriate variables for each input of question details
    qText = entries[0].get()
    qImage = entries[2].get()
    qRating = entries[1].get()
    qMarks = entries[3].get()
    qTopic = entries[4].get()
    qType = entries[5].get()
    mSteps = entries[6].get()
    mAns = entries[7].get()

    conn = sqlite3.connect("MathsPlatformStorage.db")
    cursor = conn.cursor()

    count_query = "SELECT COUNT(*) FROM tblQuestion" #Get number of questions
    cursor.execute(count_query)
    result = cursor.fetchone()
    totalQs = result[0]
    qNo = totalQs + 1 #Adds new question as next question number

    data_insert_query = """INSERT INTO tblQuestion (qNo, qText, qImage,
    qRating, qMarks, qTopic, qType, mSteps, mAns) VALUES (?,?,?,?,?,?,?,?,?)"""
    data_insert_tuple = (
        qNo,
        qText,
        qImage,
        qRating,
        qMarks,
        qTopic,
        qType,
        mSteps,
        mAns,
    )

    cursor.execute(data_insert_query, data_insert_tuple) #Inserts each associated value into the question record
    conn.commit()

    conn.close()
```

```
def add_question_menu():
    global entries
    conn = sqlite3.connect('MathsPlatformStorage.db')
    cursor = conn.cursor()
    cursor.row_factory = lambda cursor, row: row[0]
    topics = cursor.execute('SELECT topicName FROM tblTopic').fetchall() #Gets all the topic names

    window2 = tk.Toplevel()
    window2.focus()

    window2.geometry("700x500")
    window2.title("Maths Revision Tool")

    frame = tk.Frame(window2) #Configure user interface elements
    frame.pack()

    question_details_frame = tk.LabelFrame(frame, text="Question Details")
    question_details_frame.grid(row=0, column=0, pady=30)

    buttons_frame = tk.LabelFrame(frame, text="Buttons")
    buttons_frame.grid(row=2, column=0)

    qText_label = tk.Label(question_details_frame, text="qText:")
    qText_label.grid(row=0, column=0)

    qText_entry = tk.Entry(question_details_frame)
    qText_entry.grid(row=0, column=1)

    qImage_label = tk.Label(question_details_frame, text="qImage:")
    qImage_label.grid(row=1, column=0)

    qImage_entry = tk.Entry(question_details_frame)
    qImage_entry.grid(row=1, column=1)

    qRating_label = tk.Label(question_details_frame, text="qRating:")
    qRating_label.grid(row=2, column=0)

    qRating_entry = ttk.Combobox(
        question_details_frame,
        values=[
            "",
            "1 - Very Easy",
            "2 - Easy",
            "3 - Medium",
            "4 - Hard",
            "5 - Challenge",
        ],
    )
    qRating_entry.grid(row=2, column=1)

    qMarks_label = tk.Label(question_details_frame, text="qMarks:")
    qMarks_label.grid(row=0, column=2)

    qMarks_entry = tk.Spinbox(question_details_frame, from_=1, to=100)
    qMarks_entry.grid(row=0, column=3)

    qTopic_label = tk.Label(question_details_frame, text="qTopic:")
    qTopic_label.grid(row=1, column=2)
```

```
qTopic_entry = ttk.Combobox(question_details_frame, values=topics)
qTopic_entry.grid(row=1, column=3)

qType_label = tk.Label(question_details_frame, text="qType:")
qType_label.grid(row=2, column=2)

qType_entry = ttk.Combobox(
    question_details_frame,
    values=["", "Automark - num answer", "Manual - string answer"],
)
qType_entry.grid(row=2, column=3)

mSteps_label = tk.Label(question_details_frame, text="mSteps:")
mSteps_label.grid(row=3, column=0)

mSteps_entry = tk.Entry(question_details_frame)
mSteps_entry.grid(row=3, column=1)

mAns_label = tk.Label(question_details_frame, text="mAns:")
mAns_label.grid(row=3, column=2)

mAns_entry = tk.Entry(question_details_frame)
mAns_entry.grid(row=3, column=3)

submit_button = tk.Button(buttons_frame, text="Submit", command=add_question)
submit_button.grid(row=0, column=1, ipadx=30, ipady=10)

for widget in question_details_frame.winfo_children():
    widget.grid_configure(padx=20, pady=14, ipadx=15, ipady=8) #Add spacing between elements in question frame

for widget in buttons_frame.winfo_children():
    widget.grid_configure(padx=20, pady=14) #Add spacing between elements in button frame

entries = [qText_entry, qRating_entry, qImage_entry, qMarks_entry, qTopic_entry, qType_entry, mSteps_entry, mAns_entry]
```

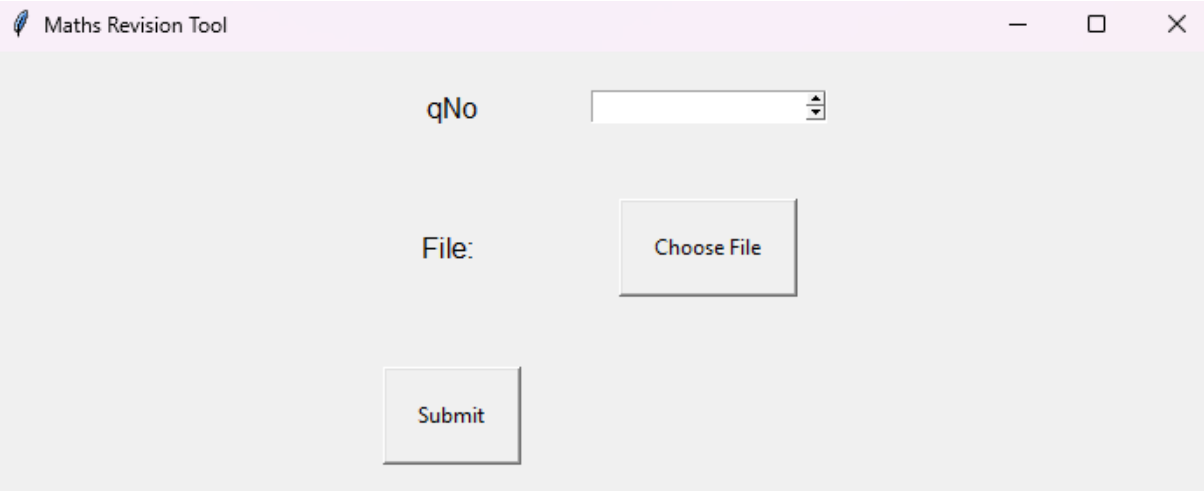
Add Note



This menu allows users to create new note files, while also allowing them to select already existing note files, rename them, and add them to the notes folder, so they can be accessed within the program, fulfilling objective 9.5 – allow users to add current note files that already have text for use within the program.

```
def add_note():
    global choose_file_note_text, file_note_name, note_frame
    window = tk.Toplevel()
    window.geometry("700x500")
    window.focus()
    note_frame = tk.Frame(window)
    note_frame.pack()

    file_note_label = tk.Label(note_frame, text="New name:", font=30)
    file_note_label.grid(row=0, column=0, padx=20, pady=20)

    file_note_name = tk.Entry(note_frame)
    file_note_name.grid(row=0, column=1, padx=20, pady=20)

    choose_file_note_text = tk.Label(note_frame, text="File: ", font=30)
    choose_file_note_text.grid(row=1, column=0, padx=20, pady=20)

    choose_file_note_button = tk.Button(
        note_frame, text="Choose File", command=get_file_path_note
    )
    choose_file_note_button.grid(row=1, column=1, padx=20, pady=20, ipady=15, ipadx=15)

    submit_note_button = tk.Button(note_frame, text="Submit", command=submit_add_note)
    submit_note_button.grid(row=2, column=0, padx=20, pady=20, ipady=15, ipadx=15)


def get_file_path_note():
    global original_note_filepath, new_note_filepath
    original_note_filepath = askopenfilename(defaultextension=".txt")
    text_file_name = file_note_name.get()
    newText = "File: " + original_note_filepath
    choose_file_note_text.config(text=newText)
    new_note_filepath = (
        "notes/" + text_file_name + ".txt"
    )


def submit_add_note():
    os.rename(original_note_filepath, new_note_filepath)
    choose_file_note_text.config(text="File: ")
    file_note_name.delete(0, END)
    submitted_text = "Received notes."
    submitted = tk.Label(note_frame, text=submitted_text)
    submitted.grid(row=2, column=1)
```

Add Image

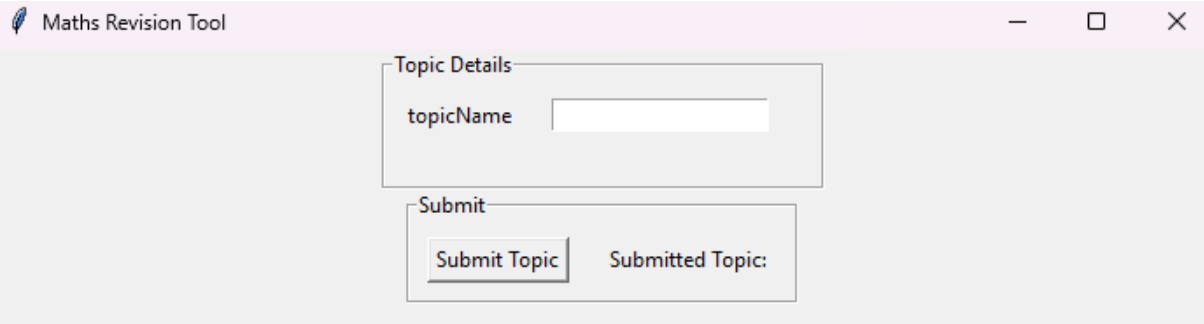


This menu allows the user to input an image file through the file browser. When the submit button is pressed, It takes the image selected and changes it's file name to the question number inputted to the user, and adds that filename to the qImage field for that specific tblQuestion record, making it part of the associated question. This allows the user to add images to questions later if they don't choose to add the image during the initial question adding process.

This fulfils objective 1.2 – store question images.

```
def add_image():
    global file_question_no, choose_file_image_text, image_frame
    window = tk.Toplevel()
    window.geometry("700x500")
    window.focus()
    image_frame = tk.Frame(window)
    image_frame.pack()

    file_question_label = tk.Label(image_frame, text="qNo", font=30)
    file_question_label.grid(row=0, column=0, padx=20, pady=20)

    file_question_no = tk.Spinbox(image_frame)
    file_question_no.grid(row=0, column=1, padx=20, pady=20)

    choose_file_image_text = tk.Label(image_frame, text="File: ", font=30)
    choose_file_image_text.grid(row=1, column=0, padx=20, pady=20)

    choose_file_image_button = tk.Button(
        image_frame, text="Choose File", command=get_file_path_image
    )
    choose_file_image_button.grid(row=1, column=1, padx=20, pady=20, ipady=15, ipadx=15)

    submit_image_button = tk.Button(
        image_frame, text="Submit", command=submit_add_image
    )
    submit_image_button.grid(row=2, column=0, padx=20, pady=20, ipady=15, ipadx=15)


def get_file_path_image():
    global questionNo, original_image_filepath, new_image_filepath
    original_image_filepath = askopenfilename()
    questionNo = file_question_no.get()
    newText = "File: " + original_image_filepath
    choose_file_image_text.config(text=newText)
    new_image_filepath = (
        "images/" + questionNo + ".png"
    )


def submit_add_image():
    os.rename(original_image_filepath, new_image_filepath)
    choose_file_image_text.config(text="File: ")
    file_question_no.delete(0, END)
    submitted_text = "Received qImg for Q" + questionNo
    submitted = tk.Label(image_frame, text=submitted_text)
    submitted.grid(row=2, column=1)
```

Add Topic



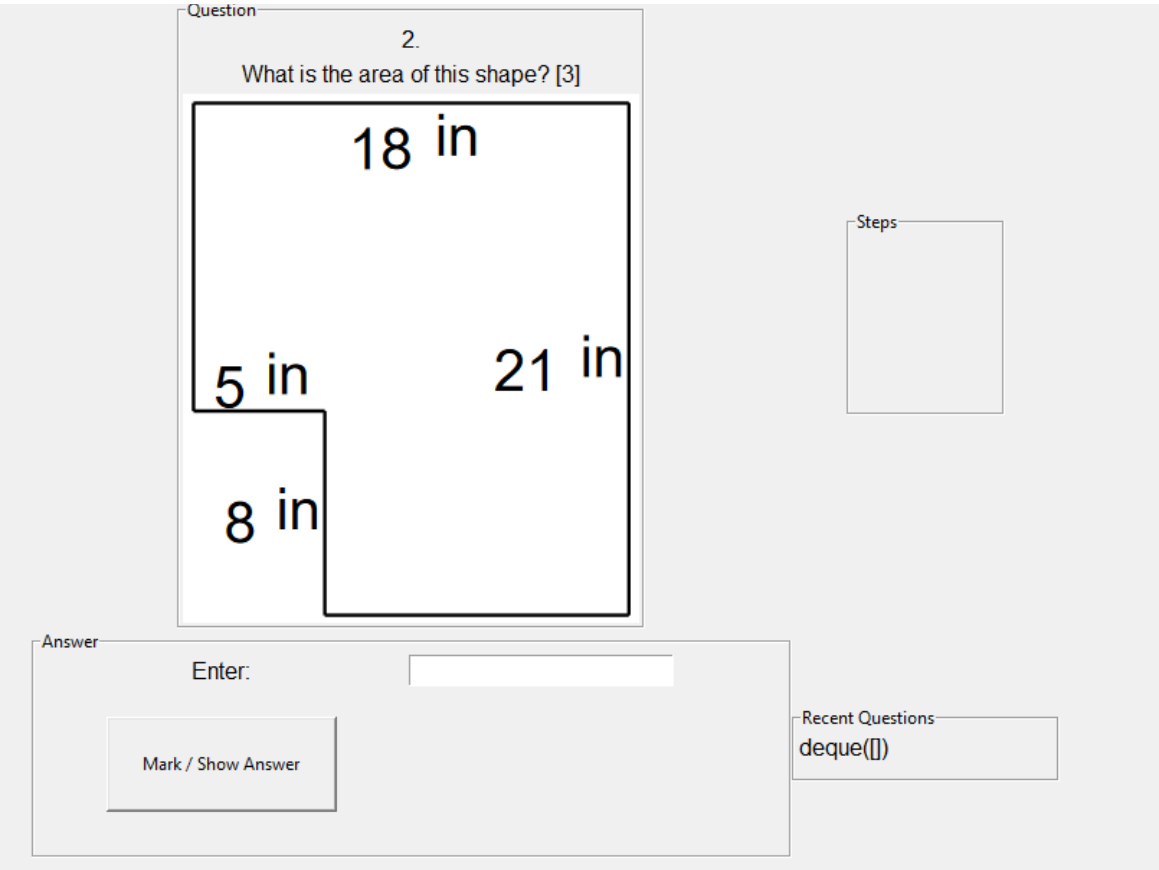
This menu takes the user input for topicName, running an SQL statement to add the topic to tblTopic if the topic does not already exist. It adds the number of correctQs as 1 and wrongQs as 1, so that there isn't a division by zero error when calculating proportions based on the progress in each topic, finding the topic to be selected for the next questions to be answered.

This fulfils objective 8.6 – allow the user to add new topics.

```
def add_topic_menu():
    global topic_name_entry, submitted_label
    window = tk.Toplevel()
    window.geometry("700x500")
    window.focus()
    topic_frame = tk.Frame(window)
    topic_frame.pack()

    topic_details_frame = tk.LabelFrame(topic_frame, text="Topic Details")
    topic_details_frame.grid(row=0, column=0, ipady=10, ipadx=10)

    topic_name_label = tk.Label(topic_details_frame, text="topicName")
    topic_name_label.grid(row=0, column=0, pady=10, padx=10)

    topic_name_entry = tk.Entry(topic_details_frame)
    topic_name_entry.grid(row=0, column=1, pady=10, padx=10)

    submit_frame = tk.LabelFrame(topic_frame, text="Submit")
    submit_frame.grid(row=1, column=0)

    topic_submit_button = tk.Button(
        submit_frame, text="Submit Topic", command=add_topic
    )
    topic_submit_button.grid(row=0, column=0, pady=10, padx=10)
    submitted_label = tk.Label(submit_frame, text='Submitted Topic: ')
    submitted_label.grid(row=0, column=1, pady=10, padx=10)
```

```
def add_topic():
    topicName = topic_name_entry.get()
    submitted_text = 'Submitted Topic: ' + topicName
    submitted_label.config(text=submitted_text)
    conn = sqlite3.connect("MathsPlatformStorage.db")
    cursor = conn.cursor()

    count_query = "SELECT COUNT(*) FROM tblTopic"
    cursor.execute(count_query)
    result = cursor.fetchone()
    totalTopics = result[0]
    topicNo = totalTopics + 1

    insert_topic_query = """INSERT INTO tblTopic(topicNo, topicName, correctQs, wrongQs, score)
     VALUES (?, ?, ?, ?, ?)"""
    insert_topic_tuple = (topicNo, topicName, 1, 1, 1)
    cursor.execute(insert_topic_query, insert_topic_tuple)
    conn.commit()
    conn.close()
```

Answer Question

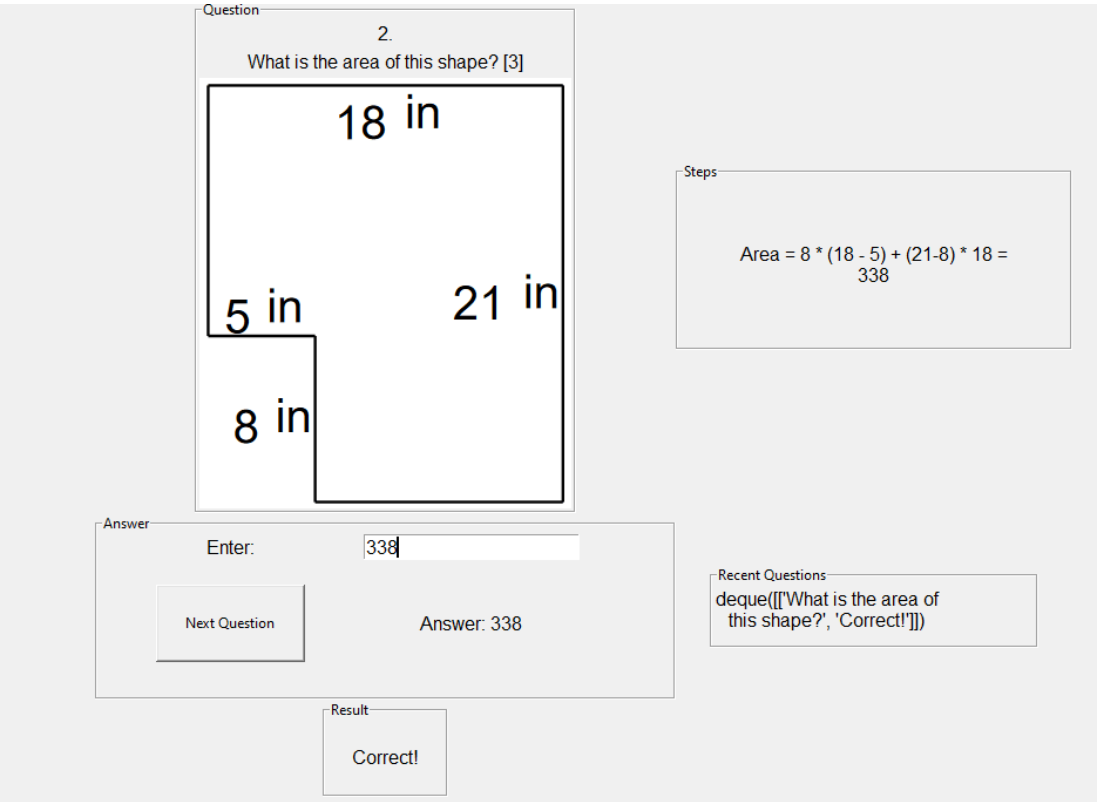


In the Answer Questions menu, the user is prompted with a question, showing the initial question details including the question number, text, marks and image. There is also an input box for the user to enter their answer, and then select the Mark / Show Answer button once complete.

This fulfils all of Objective 2 – display question details in box format, display question image, display input field for entering answer, display mark/show answer button, and display answer result box.

Additionally, the questions generated are chosen by topic (apart from the first question which is randomly generated from all the questions). The topic is chosen by calculating ratios using the correctQs and wrongQs values for the topic. The topic name and ratio are put into a dictionary, and using random choice with the weights being the dictionary, the questions will be generated at a rate that is proportional to the user's performance in those topics. This fulfils all of objective 6 – give more questions for topics that the user is shown to have struggled with based on their performance.

```
def answer_question_menu():
    global questionFrame, first_q, enterAnswer, label_img

    first_q = True
    recent_questions = Queue(maxsize = 5) #Queue for 5 most recently answered questions

    window = tk.Toplevel()
    window.focus()
    window.geometry("1200x700")
    window.title("Maths Revision Tool")

    frame = tk.Frame(window)
    frame.pack()

    conn = sqlite3.connect("MathsPlatformStorage.db")
    cursor = conn.cursor()

    count_query = "SELECT COUNT(*) FROM tblQuestion" #Get number of questions
    cursor.execute(count_query)
    result = cursor.fetchone()
    totalQs = result[0]
    currentQ = random.randint(1, totalQs) #Get random question for first question

    search_query = (
        "SELECT qText, qRating, qImage, qMarks, qTopic, qType, mSteps, mAns FROM tblQuestion WHERE qNo="
        + str(currentQ)
    )
    cursor.execute(search_query)
    question_info = cursor.fetchone() #Get question details

    questionFrame = tk.LabelFrame(frame, text="Question")
    questionFrame.grid(row=0, column=0)

    questionNoText = str(currentQ) + "."

    questionNo = tk.Label(questionFrame, text=questionNoText, font=30)
    questionNo.pack()

    questionText = question_info[0] + " [" + str(question_info[3]) + "]"

    questionTextLabel = tk.Label(questionFrame, text=questionText, font=30)
    questionTextLabel.pack()

    shown_image_path = "images/" + question_info[2]
    image = Image.open(shown_image_path)
    image_obj = ImageTk.PhotoImage(image)
    label_img = tk.Label(questionFrame, image=image_obj)
    label_img.image = image_obj
    label_img.pack()

answerFrame = tk.LabelFrame(frame, text="Answer")
answerFrame.grid(row=1, column=0, ipadx=40, ipady=5)
recentQuestionFrame = tk.LabelFrame(frame, text="Recent Questions")
recentQuestionFrame.grid(row=1, column=1, ipadx=40, ipady=5)
recentQuestionsLabel = tk.Label(recentQuestionFrame, text=recent_questions.queue, font=("Arial", 12), wraplength=200)
recentQuestionsLabel.grid(row=0, column=0)

if question_info[5] == 'Automark - num answer':

    enterLabel = tk.Label(answerFrame, text="Enter:", font=25)
    enterLabel.grid(row=0, column=0)

    enterAnswer = tk.Entry(answerFrame, font=20)
    enterAnswer.grid(row=0, column=1)

stepsFrame = tk.LabelFrame(frame, text="Steps")
stepsFrame.grid(row=0, column=1)
steps = tk.Label(stepsFrame, font=30, wraplength=250)
steps.grid(row=0, column=0, pady=50, padx=50)
```



Once the input is submitted, the program checks if the answer is correct, displaying the answer and if the input is correct or wrong. It also shows the working steps to answer the question, and displays a queue of the 5 most recent questions answered, and whether they got the question correct or wrong. When the queue is full, the front question gets removed from the queue. When the next question button is pressed, the content is cleared, and the new question details are fetched via SQL and displayed.

This fulfils all of Objective 3, 4, and 5 – mark questions after the user has answered them, update the question details displayed when going to the next question, and checking if the answer is correct/wrong and store progress for different topics based on that.

```
def show_answer(question_info):
    global resultCheck, answer, choiceBox, choiceButton, enterLabel, enterAnswer
    submitAnswerButton.config(text="Next Question", command=lambda: next_question(question_info))
    steps.config(text=question_info[6])
    conn = sqlite3.connect("MathsPlatformStorage.db")
    cursor = conn.cursor()
    if question_info[5] == 'Automark - num answer':
        answerText = "Answer: " + str(question_info[7])
        userAnswer = enterAnswer.get() #Get user input for answer
    else:
        answerText = ''
    answer = tk.Label(answerFrame, text=answerText, font=30)
    answer.grid(row=1, column=1)

    resultFrame = tk.LabelFrame(frame, text="Result")
    resultFrame.grid(row=2, column=0)
    resultCheck = tk.Label(resultFrame, font=40)
    resultCheck.grid(row=0, column=0, padx=20, pady=20)

    if question_info[5] == 'Automark - num answer': #Automark question case
        if userAnswer == str(question_info[7]): #If the answer is correct, increase correctQs by 1
            cursor.execute(
                "UPDATE tblTopic SET correctQs = correctQs + 1 WHERE tblTopic.topicName = '"
                + str(question_info[4])
                + "'"
            )
            cursor.execute( #Increase score by the question marks
                "UPDATE tblTopic SET score = score + "
                + str(question_info[3])
                + " WHERE tblTopic.topicName = '"
                + str(question_info[4])
                + "'"
            )
            resultText = 'Correct!'
        else: #If wrong, increase wrongQs by 1
            cursor.execute(
                "UPDATE tblTopic SET wrongQs = wrongQs + 1 WHERE tblTopic.topicName = '"
                + str(question_info[4])
                + "'"
            )
            resultText = 'Wrong!'
        resultCheck.config(text=resultText)
        if recent_questions.full(): #If queue is full, remove the front question
            recent_questions.get()
        recent_questions.put([question_info[0], resultText]) #Add current question to the rear
        recentQuestionsLabel.config(text=recent_questions.queue)

    if question_info[5] == 'Manual - string answer': #If question is manually marked, run alternate marking function
        choiceBox = ttk.Combobox(answerFrame, values=['correct','wrong'])
        choiceBox.grid(row=1, column=2)
        choiceButton = tk.Button(answerFrame, text='Submit Choice', command=check_choice)
        choiceButton.grid(row=1, column=1)

    print(recent_questions.queue)

    resultCheck.grid(row=2, column=0)
    conn.commit()
    conn.close()
```

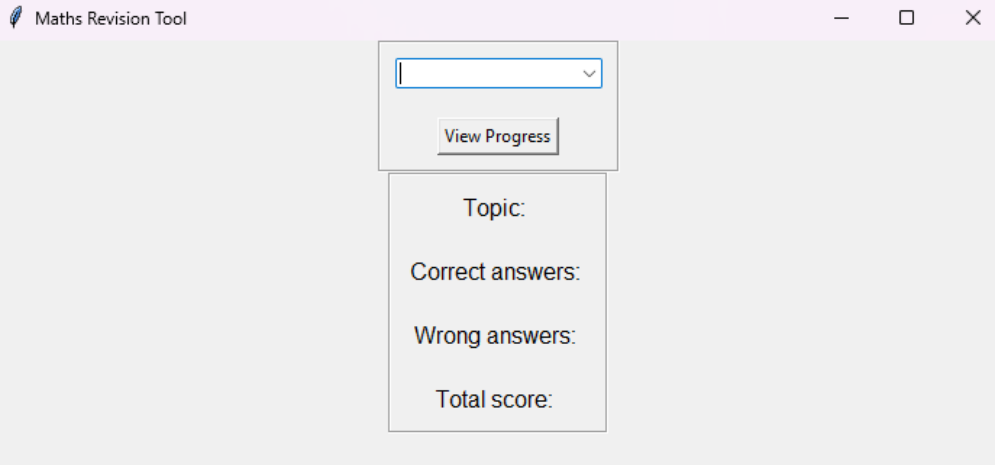
The answering questions system uses mutual recursion between 2 functions, by running the function to generate the next question (next_question), and then mark the answer (show_answer), looping back to the next question, repeating this while the user answers questions. The next question function is shown below.

This fulfils objectives 3, and 4 – mark the questions as the user progresses, and update the question details when going to the next question.

```
def next_question(question_info):
    global first_q, enterLabel, enterAnswer, label_img
    label_img.destroy()

    if question_info[5] == 'Manual - string answer' and first_q == True:
        choiceBox.destroy()
        choiceButton.destroy()

    submitAnswerButton.config(
        text="Mark / Show Answer", command=lambda: show_answer(question_info)
    )
    conn = sqlite3.connect("MathsPlatformStorage.db")
    cursor = conn.cursor()

    resultCheck.config(text="")
    answer.config(text="")
    steps.config(text="")

    topic_query = "SELECT topicName, correctQs, wrongQs FROM tblTopic" #Get all topic names
    cursor.execute(topic_query)
    topic_info = cursor.fetchall()
    topic_dict = {}
    ratio_sum = 0
    for x in topic_info:
        ratio = round((1/(x[1] / (x[1] + x[2]))),2)
        ratio_sum += ratio
    for x in topic_info:
        topic_name = x[0]
        ratio = round((1/(x[1] / (x[1] + x[2]))),2)
        topic_dict[topic_name] = ratio / ratio_sum #calculate weighted ratios for each topic based on performance in the topic
    topic = (random.choices(list(topic_dict.keys()), weights=topic_dict.values(), k=1))[0]
```

```
print(search_query)
cursor.execute(search_query) #Get next question details
question_info = cursor.fetchone()
if question_info[5] == 'Automark - num answer':

    enterLabel = tk.Label(answerFrame, text="Enter:", font=25)
    enterLabel.grid(row=0, column=0)

    enterAnswer = tk.Entry(answerFrame, font=20)
    enterAnswer.grid(row=0, column=1)

questionNoText = str(question_info[8]) + "."
questionText = question_info[0] + " [" + str(question_info[3]) + "]"

questionTextLabel.config(text=questionText)
questionNo.config(text=questionNoText)

shown_image_path = "images/" + question_info[2]
image = Image.open(shown_image_path)
image_obj = ImageTk.PhotoImage(image)
label_img = tk.Label(questionFrame, image=image_obj)
label_img.image = image_obj
label_img.pack()
```

View Topic Progress

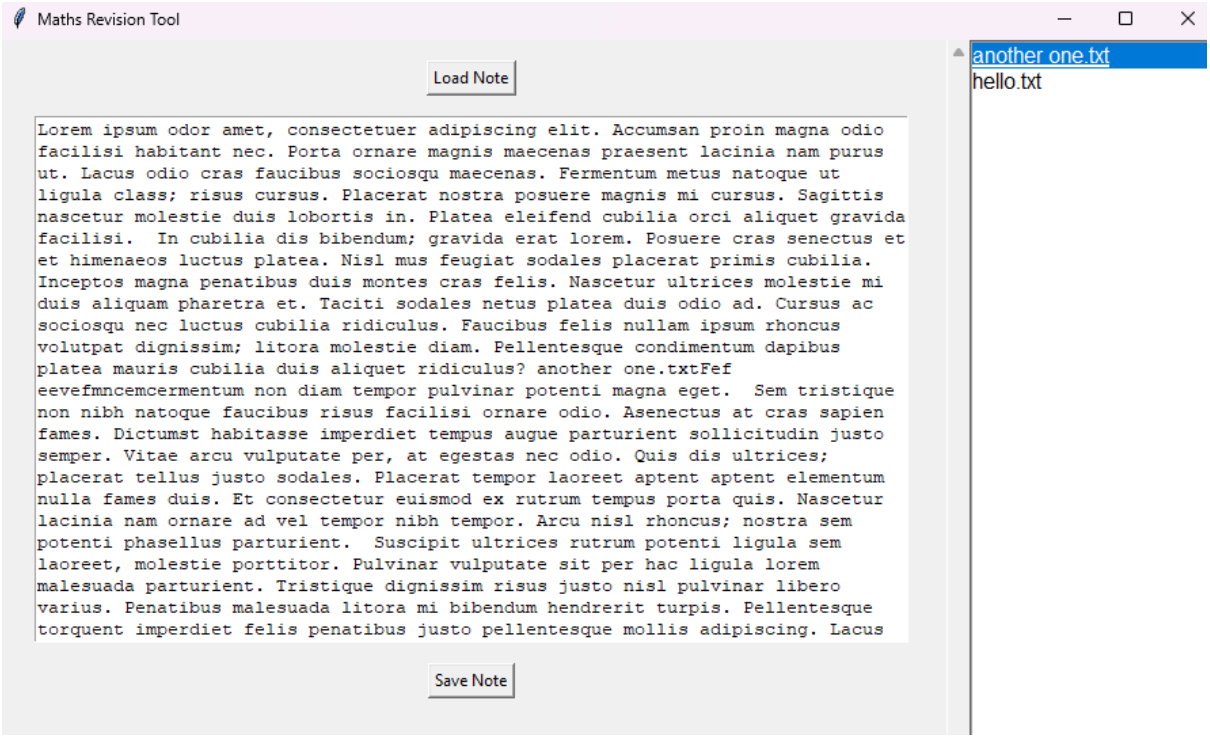


When opened, the View Topic Progress menu executes an SQL statement, the program gets the names of topics from tblTopic, and allows the user to view the progress within each topic, specifically the correctQs, wrongQs, and score for that topic by using SQL select statements, fulfilling objective 8 – view the topic progress and display each statistic, update the statistics depending on the topic selected.

```
def view_progress_menu():
    global topic_options, correct_label, topic_label, wrong_label, score_label
    conn = sqlite3.connect('MathsPlatformStorage.db')
    cursor = conn.cursor()
    cursor.row_factory = lambda cursor, row: row[0]
    topics = cursor.execute('SELECT topicName FROM tblTopic').fetchall()
    conn.close()

    window = tk.Toplevel()
    window.geometry("700x500")
    window.focus()
    progress_frame = tk.Frame(window)
    progress_frame.pack()

    choose_topic_frame = tk.LabelFrame(progress_frame)
    choose_topic_frame.grid(row=0, column=0)

    topic_progress_frame = tk.LabelFrame(progress_frame)
    topic_progress_frame.grid(row=1, column=0)

    topic_options = ttk.Combobox(choose_topic_frame, values=topics)
    topic_options.grid(row=0, column=0, padx=10, pady=10)

    view_button = tk.Button(choose_topic_frame, text='View Progress', command=get_progress)
    view_button.grid(row=1, column=0, padx=10, pady=10)

    topic_label = tk.Label(topic_progress_frame, text='Topic: ',font=16)
    topic_label.grid(row=0, column=0, padx=10, pady=10)
    correct_label = tk.Label(topic_progress_frame, text='Correct answers: ',font=16)
    correct_label.grid(row=1, column=0, padx=10, pady=10)
    wrong_label = tk.Label(topic_progress_frame, text='Wrong answers: ',font=16)
    wrong_label.grid(row=2, column=0, padx=10, pady=10)
    score_label = tk.Label(topic_progress_frame, text='Total score: ',font=16)
    score_label.grid(row=3, column=0, padx=10, pady=10)
```

```
def get_progress():
    topic = topic_options.get()

    conn = sqlite3.connect('MathsPlatformStorage.db')
    cursor = conn.cursor()
    topic_scores_statement = 'SELECT correctQs, wrongQs, score FROM tblTopic WHERE topicName = "' + topic + '"'
    cursor.execute(topic_scores_statement)
    topic_scores = cursor.fetchone()
    conn.close()

    topic_label.config(text=('Topic: '+topic))
    correct_label.config(text=('Correct answers: '+str(topic_scores[0])))
    wrong_label.config(text=('Wrong answers: '+str(topic_scores[1])))
    score_label.config(text=('Total score: '+str(topic_scores[2])))
```

View/Edit Notes



Within the Edit Notes menu, the .txt files stored in the notes folder are listed on the right. They can be selected and then loaded via the load note button, which fetches the text from the file and appends it to the input box. The files can also be edited by typing in the textbox and then selecting save note, fulfilling objective 9 – view notes to study from.

```
def view_notes_menu():
    global notes_list, note_textbox, notes
    window = tk.Toplevel()
    window.geometry("900x600")
    window.focus()

    notes_list = tk.Listbox(window, selectmode=SINGLE, font=20)
    notes = os.listdir('notes')
    print(notes)
    for x in notes:
        notes_list.insert(END, x)
    notes_list.pack(side = RIGHT, fill=BOTH)

    scrollbar = tk.Scrollbar(window)
    scrollbar.pack(side=RIGHT, fill=BOTH)
    notes_list.config(yscrollcommand=scrollbar.set)
    scrollbar.config(command=notes_list.yview)

    view = tk.Button(window, text='Load Note', command=view_note)
    view.pack(pady=15)
    note_textbox = tk.Text(window)
    note_textbox.pack()
    save = tk.Button(window, text='Save Note', command=save_note)
    save.pack(pady=15)

def view_note():
    global directory, note_textbox
    index_selected = int((notes_list.curselection())[0])
    note_selected = notes[index_selected]
    directory = "notes/" + note_selected
    x = open(directory, "r")

    text = x.read()
    x.close()
    text = textwrap.fill(text,80)
    note_textbox.delete(1.0, END)
    note_textbox.insert(END, text)

def save_note():
    new_text = note_textbox.get("1.0", "end-1c")
    x = open(directory, "w")
    x.write(new_text)
    x.close()
```

Database Initialization

```
conn = sqlite3.connect("MathsPlatformStorage.db") #Connect to db and configure cursor
cursor = conn.cursor()
question_table_query = """CREATE TABLE IF NOT EXISTS tblQuestion (qNo INT, qText TEXT, qImage TEXT,
                        qRating TEXT, qMarks INT, qTopic TEXT, qType TEXT, mSteps TEXT,
                        mAns TEXT, PRIMARY KEY (qNo)) FOREIGN KEY (qTopic) REFERENCES tblTopic(topicName)"""
                        #Create question table

topic_table_query = """CREATE TABLE IF NOT EXISTS tblTopic
 (topicNo INT, topicName TEXT, correctQs INT, wrongQs INT
 , score INT, PRIMARY KEY(topicNo))""" #Create topic table

cursor.execute(question_table_query)
cursor.execute(topic_table_query)

conn.commit() #Commit changes to db
conn.close() #Close db connection
```

The system utilizes a database with 2 tables, tblQuestion for storing question records with all the question details, and tblTopic for storing the different and the user's performance within the questions for those topics, fulfilling objective 1 – store all question details in a table (tblQuestion) in a database (MathsPlatformStorage.db).

Final Testing Plan

Test No.	Description	NEB	Expected Outcome	Actual Outcome	Comments
		N:		Same as expected.	No comments.
		E:		Same as expected.	No comments.
		B:		Same as expected.	No comments.

This is the testing plan I have finalised on. It has the following columns:

"Test No." – the test number, "Description" – a short summary of what specific menu/feature I am testing, "NEB" – the types of data I'm using for my tests, normal, erroneous, and boundary data. Normal data is valid inputs, erroneous data is invalid inputs, and boundary data is valid inputs that are on the boundary of the accepted range. "Expected Outcome" – a description of the result I expect when running the test, "Actual Outcome" – a description of what I observed happening when I ran the rest, and "Comments" – Any notable parts of the tests that I need to talk about, for example, if the actual outcome was different to the expected outcome, the comment would be what I did to fix the system so that the actual outcome matched the expected outcome.

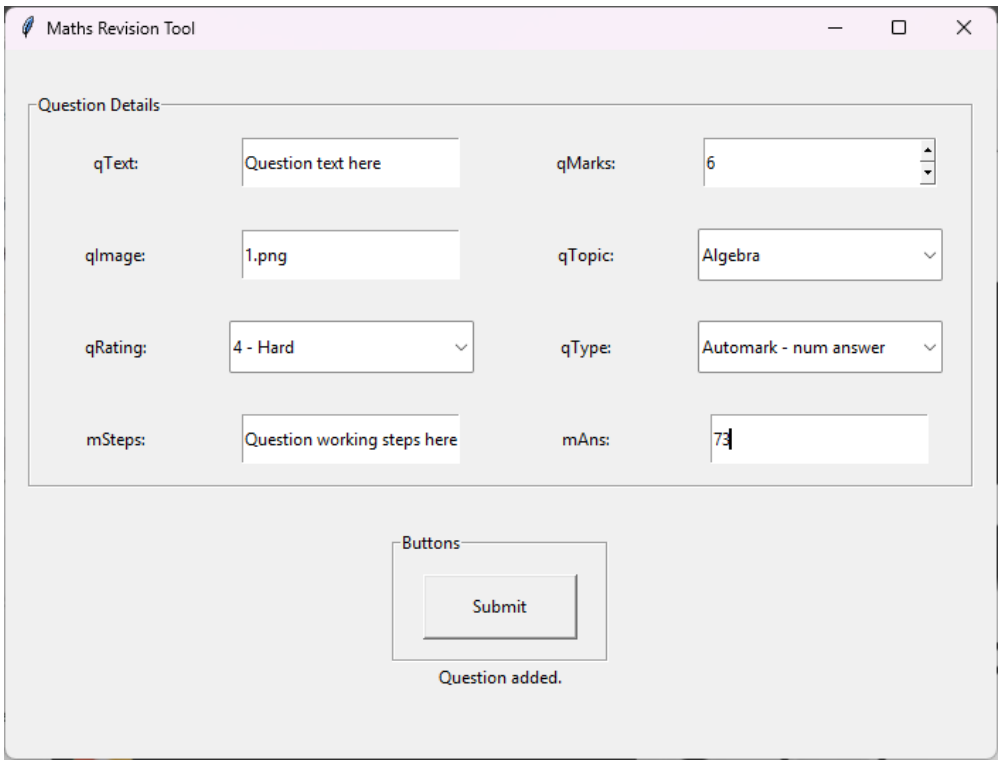
Testing

Input and Output Testing

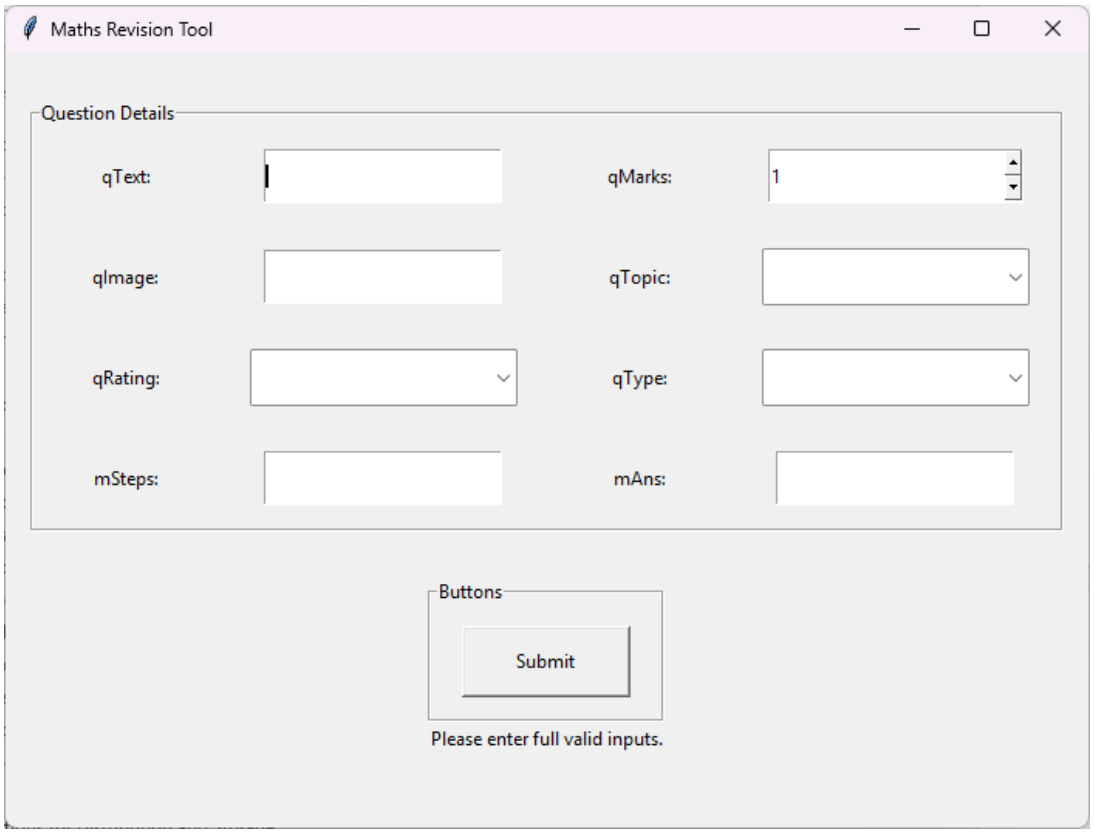
NEB – Normal (N), Erroneous (E), Boundary (B)
If an example of a boundary data input is not present, it's because there isn't a range of valid inputs, or there's only a few select inputs for that test case, so there's no boundary conditions.

Test No.	Description	NEB	Expected Outcome	Actual Outcome	Comments
1	Adding a question to the database – **Objectives 1 and 7**	N: All fields are inputted	'Question added.' outputted, SQL executed, and record added to tblQuestion.	Success - Same as expected. (Figure 1)	No comments.
		E: None of the fields have inputs	'Please enter valid inputs' outputted.	Success - Same as expected. (Figure 2)	No comments.
		B: All fields except default inputs.	Same as normal outcome.	Fail – Resulted in errors when trying to answer these questions, as data was missing. (Figure 3)	I fixed it so that some of the fields can be left blank as they have default values, such as mSteps, and qImage. (Figure 4)

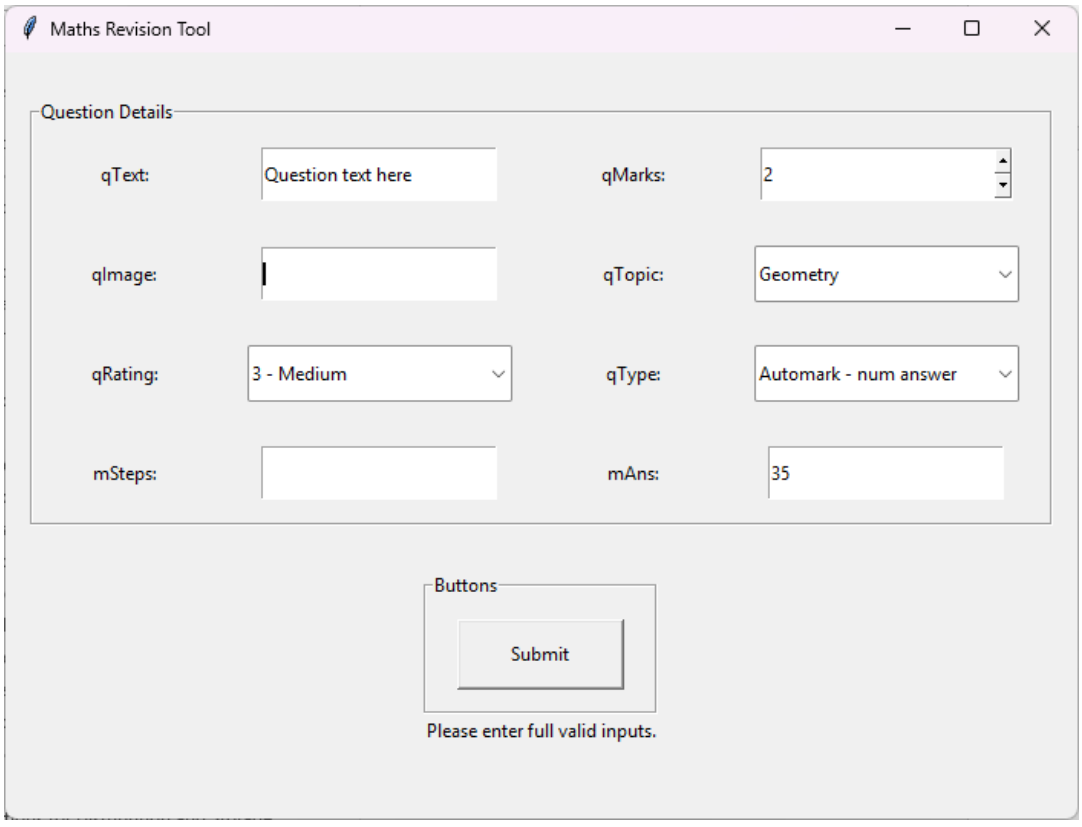
Achieved expected outcome for all fields inputted. (Figure 1)



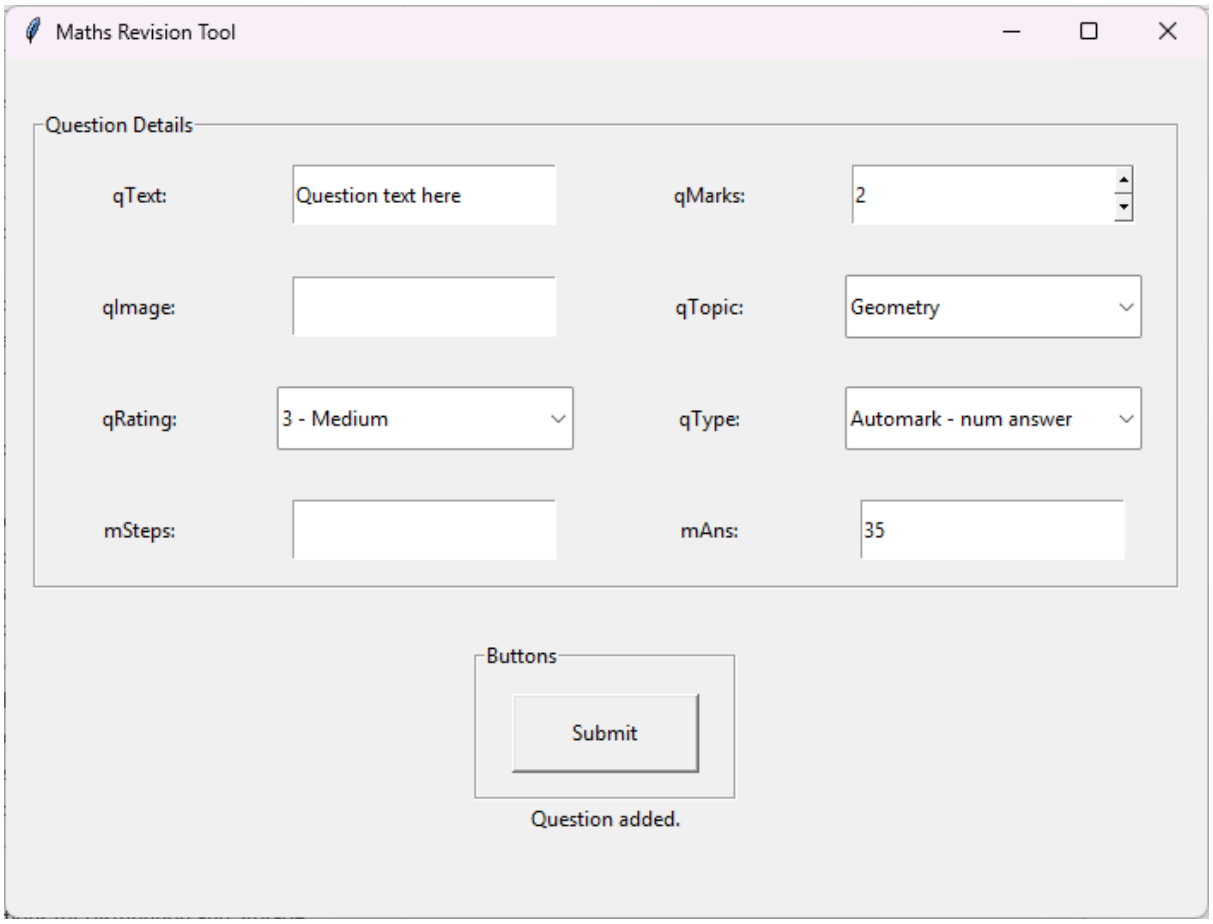
Achieved expected outcome for all inputs left blank. (Figure 2)



Error outcome for only default inputs left blank. (Figure 3)

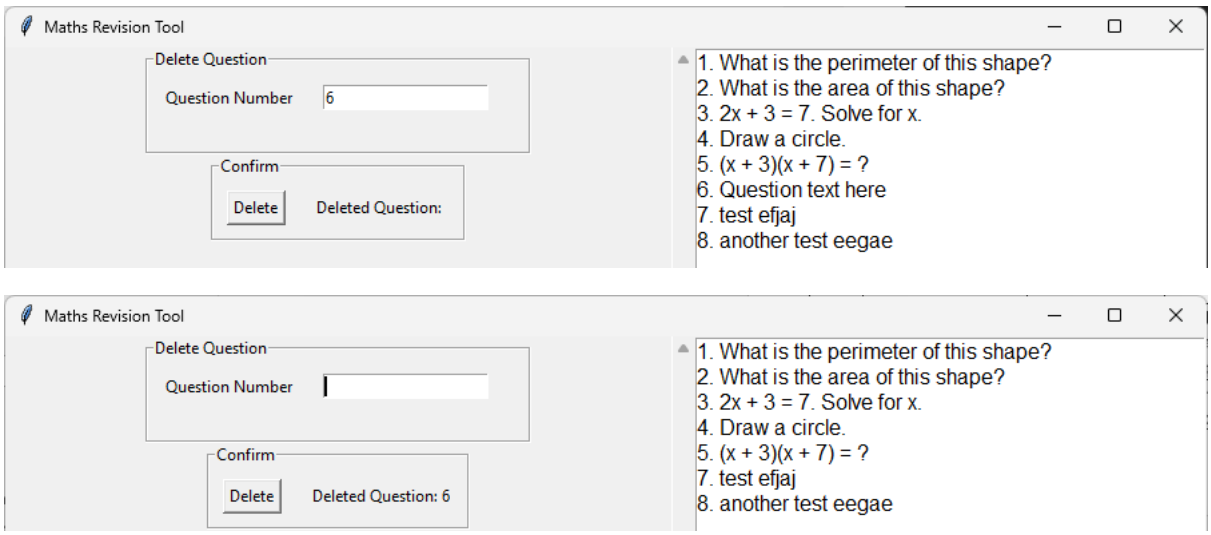


Expected outcome for default inputs left blank after fixing the error. (Figure 4)

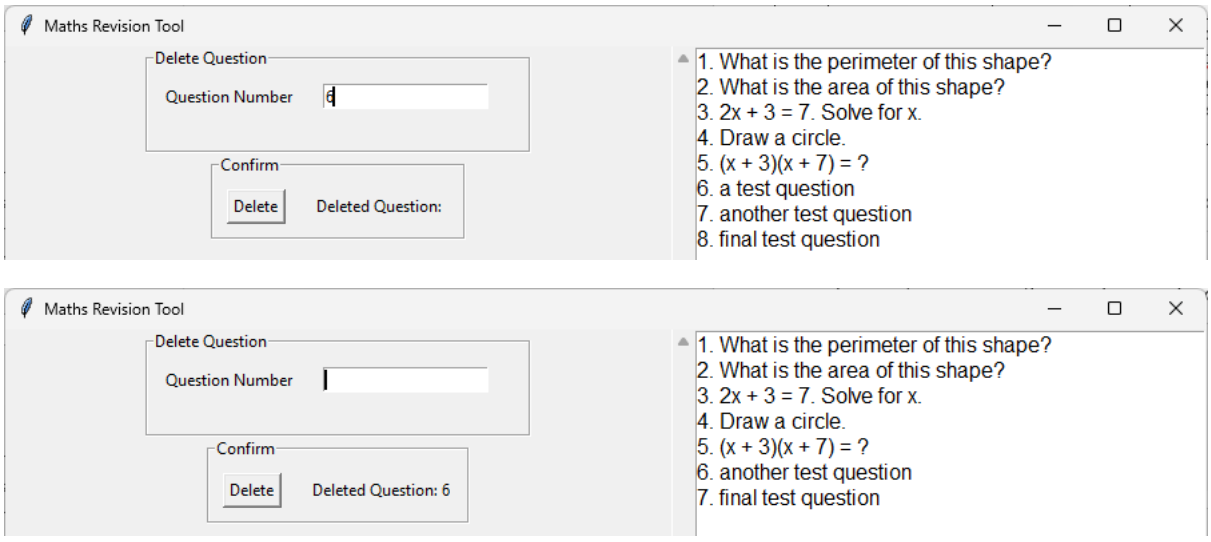


Test No.	Description	NEB	Expected Outcome	Actual Outcome	Comments
2	Deleting a question from the database – **Objective 1.11**	N: Valid question number inputted	'Deleted Question: (question number)' outputted, SQL executed, and record deleted from tblQuestion.	Fail - Resulted in errors when generating questions, as question numbers weren't adjusted. (Figure 5)	I fixed it so that question numbers are also shifted by 1 when a question is deleted, and the question list is refreshed. (Figure 6)
		E: Invalid qNo inputted	'Deleted Question: Invalid qNo' outputted.	Success - Same as expected. (Figure 7)	No comments.
		B: Boundary qNo inputted	Same as normal outcome.	Success - Same as expected. (Figure 8)	No comments.

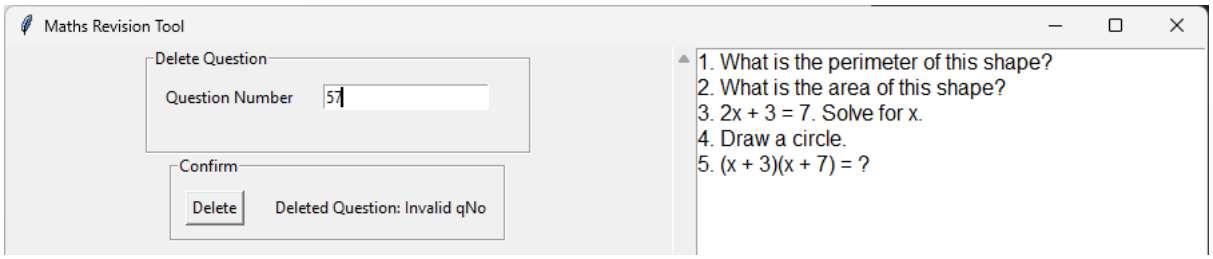
Error outcome for valid question number – question numbers are inconsistent (Figure 5)



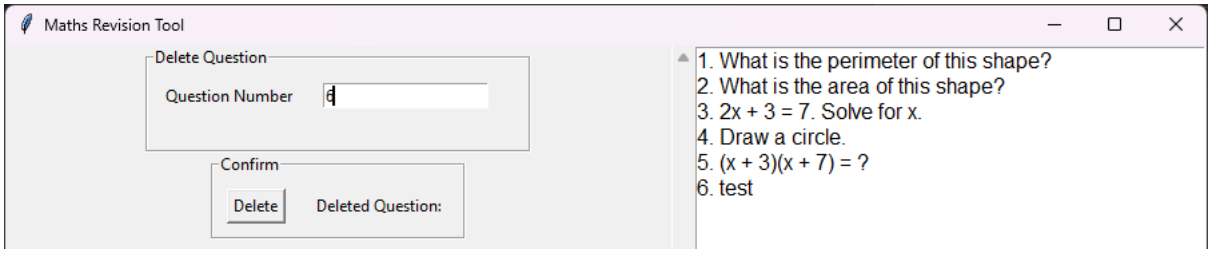
Fixed outcome for valid question number – question numbers update (Figure 6)

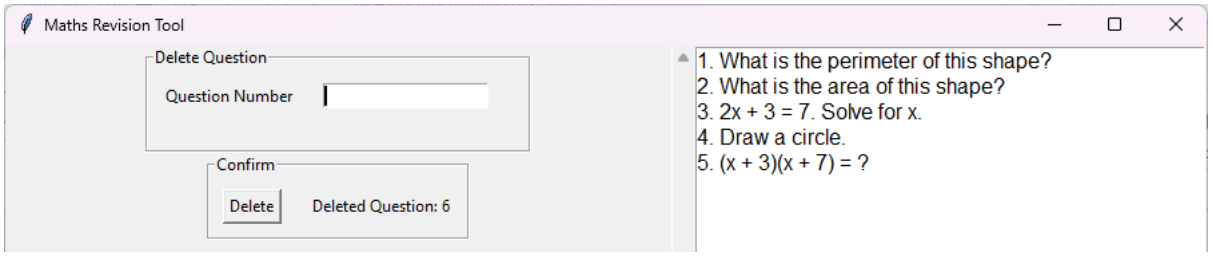


Expected outcome for invalid question number input. (Figure 7)



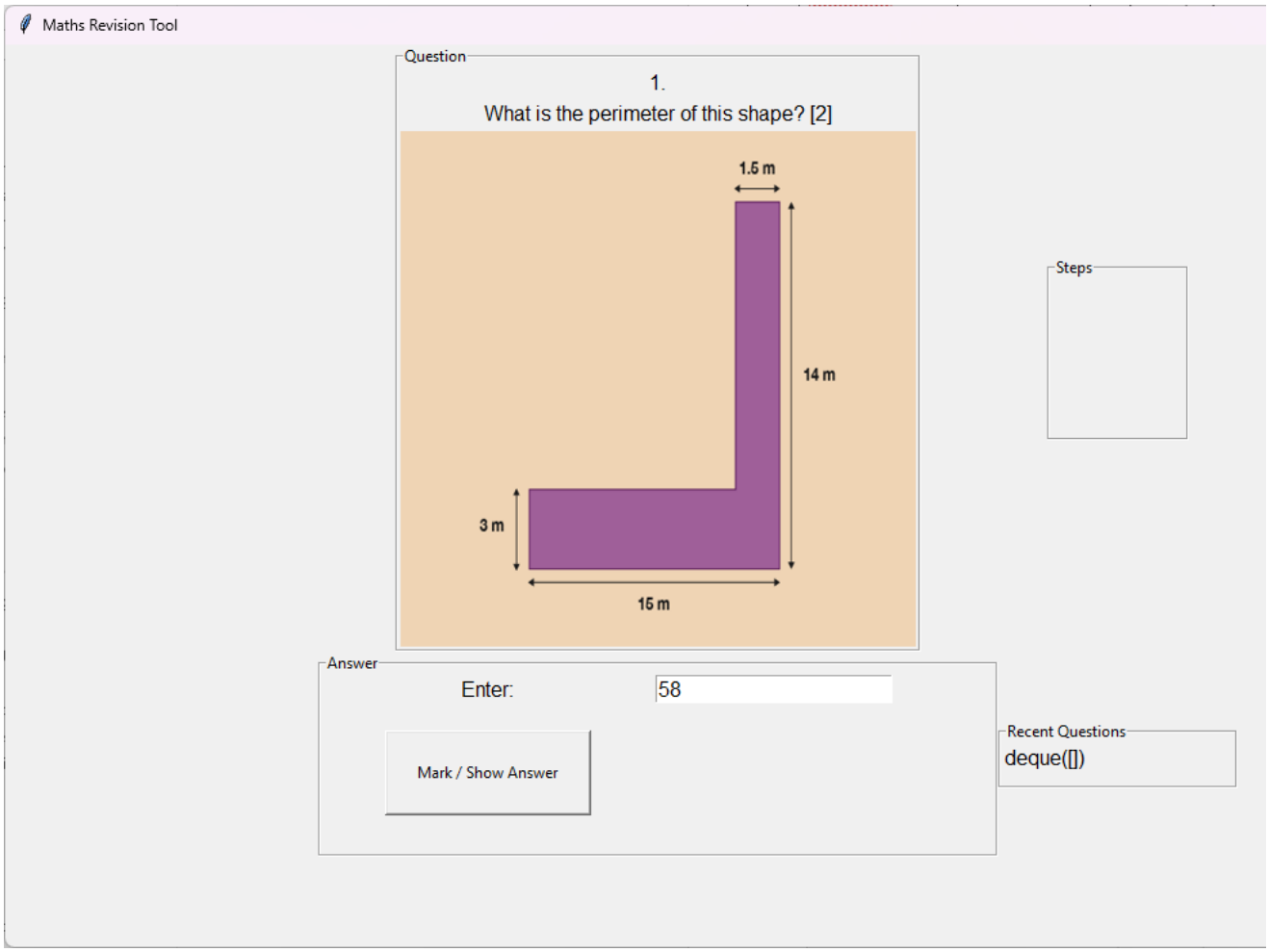
Expected outcome for boundary question number input. (Figure 8)

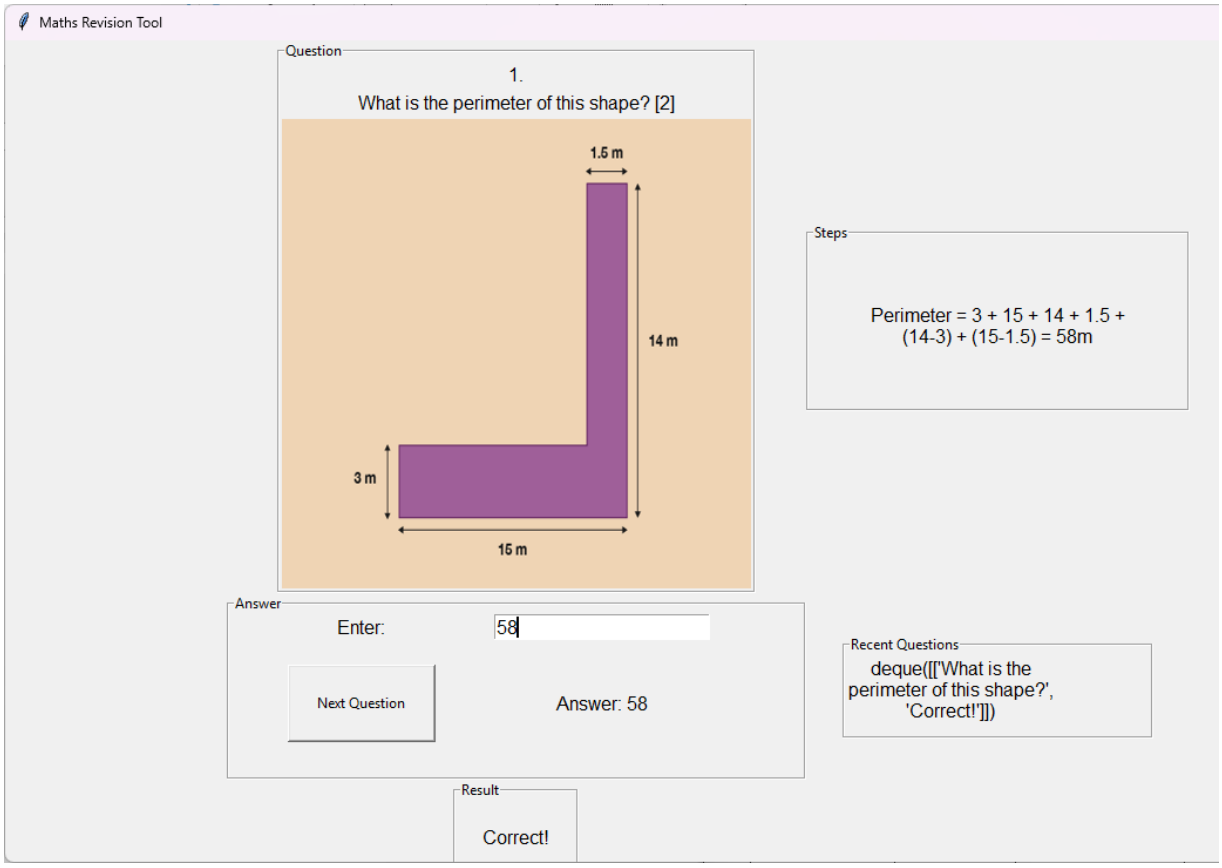




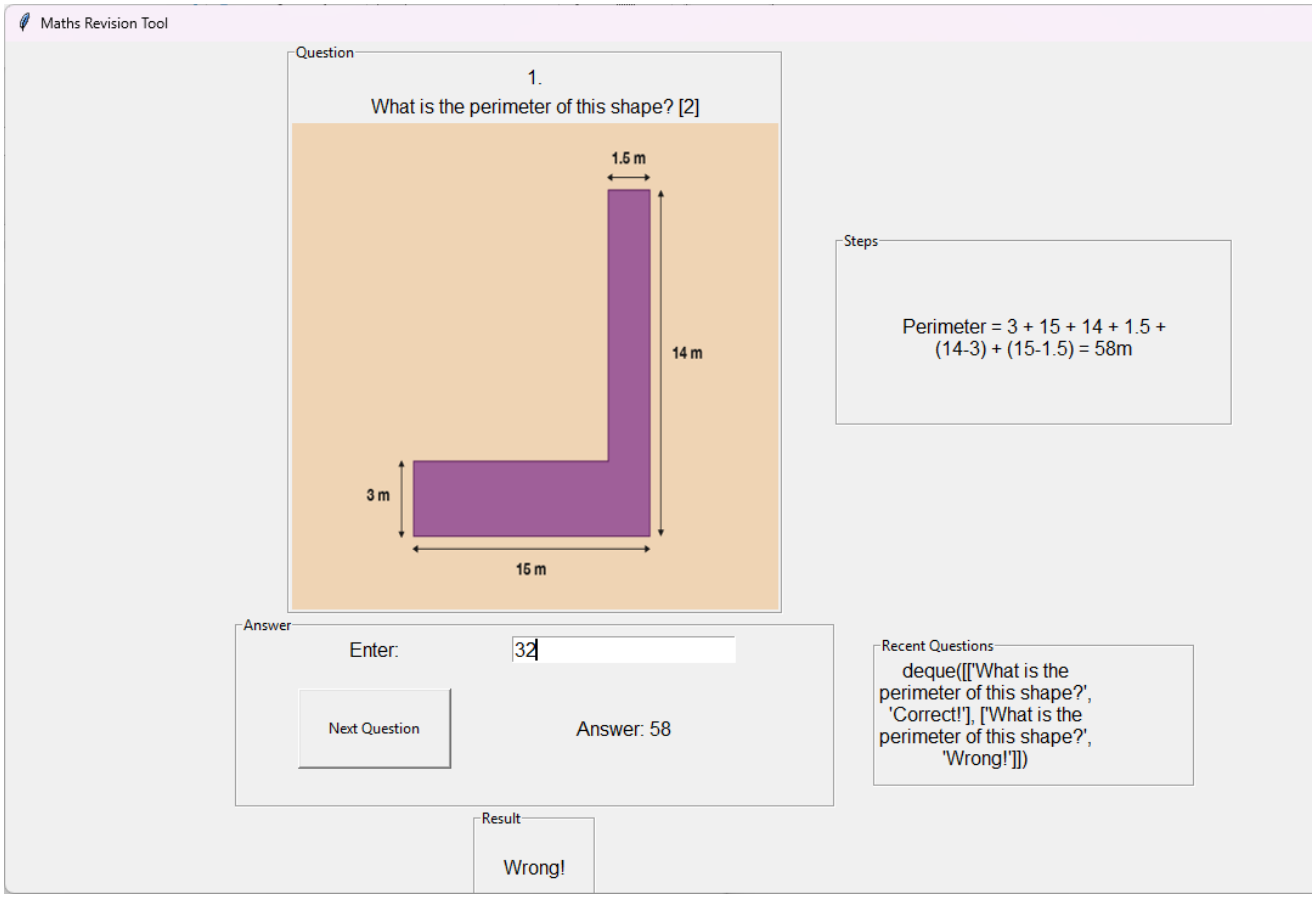
Test No.	Description	NEB	Expected Outcome	Actual Outcome	Comments
3	Answer an automark question – **Objectives 2, 3, 4, and 5**	N: Correct answer inputted	'Result: Correct!' outputted, topic score incremented, correctQs incremented, queue updated	Success - Same as expected. (Figure 9)	No comments.
		N: Incorrect answer inputted	'Result: Wrong!' outputted, wrongQs incremented, correct answer displayed, queue updated	Success - Same as expected. (Figure 10)	No comments.

Expected outcome for correct answer input. (Figure 9)



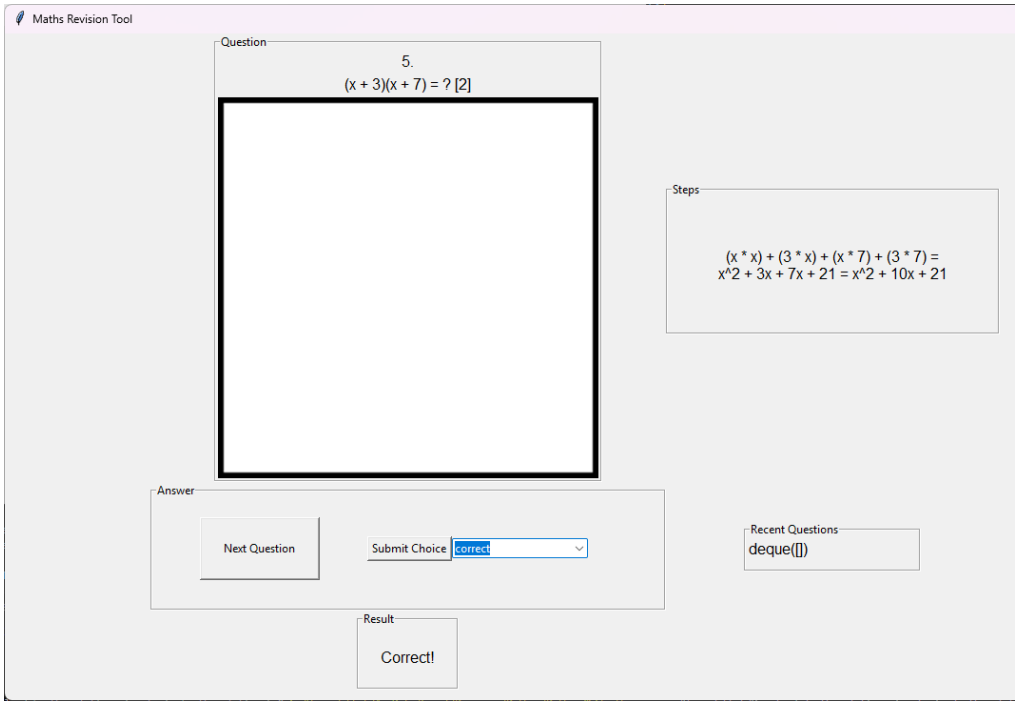


Expected outcome for incorrect answer input. (Figure 10)

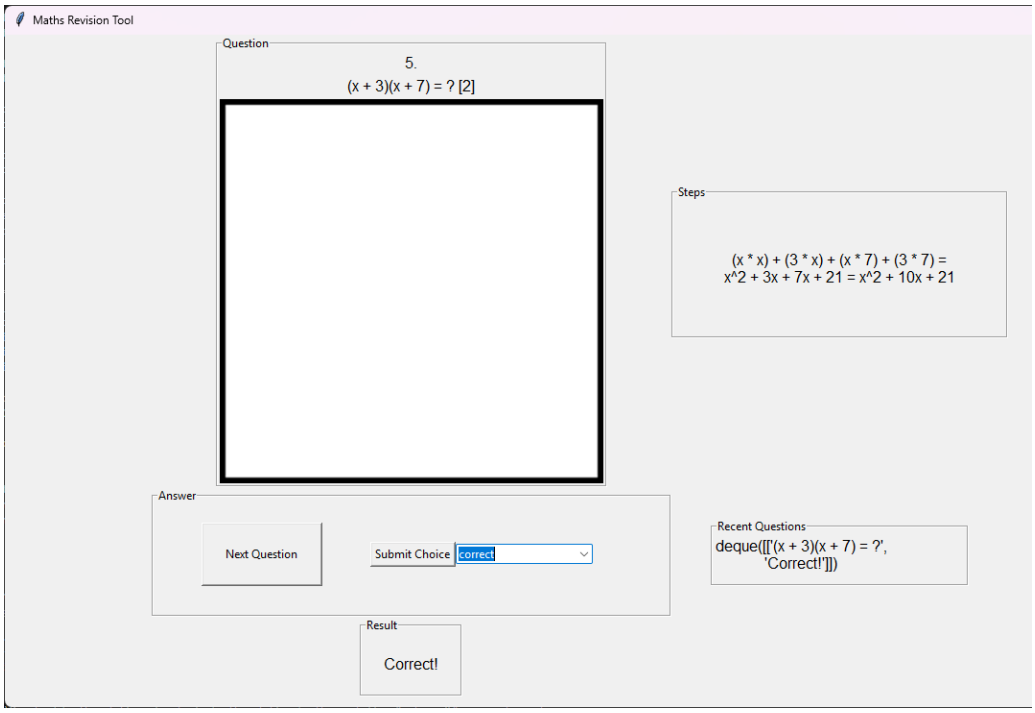


Test No.	Description	NEB	Expected Outcome	Actual Outcome	Comments
4	Answer a manually marked question – **Objectives 2, 3, 4, and 5**	N: Correct option selected	'Result: Correct!' outputted, topic score incremented, correctQs incremented, queue updated.	Fail - The question queue did not update as I only set it for automark questions, because the function is separate (Figure 11)	I fixed the queue not updating. For manual questions, the correct answer is shown after question completion, and the user selects whether they got it right. (Figure 12)
		N: Wrong option selected	'Result: Wrong!' outputted, wrongQs incremented, correct answer displayed, queue updated	Success - Same as expected. (Figure 13)	No comments.
		E: Input skipped	'Select if you got the answer correct or wrong before moving on' outputted.	Fail - Output didn't work as next question is generated instantly after skip. (Figure 14)	Fixed so that next question doesn't generate if trying to go to it when the result for the answer hasn't been selected yet. (Figure 15)

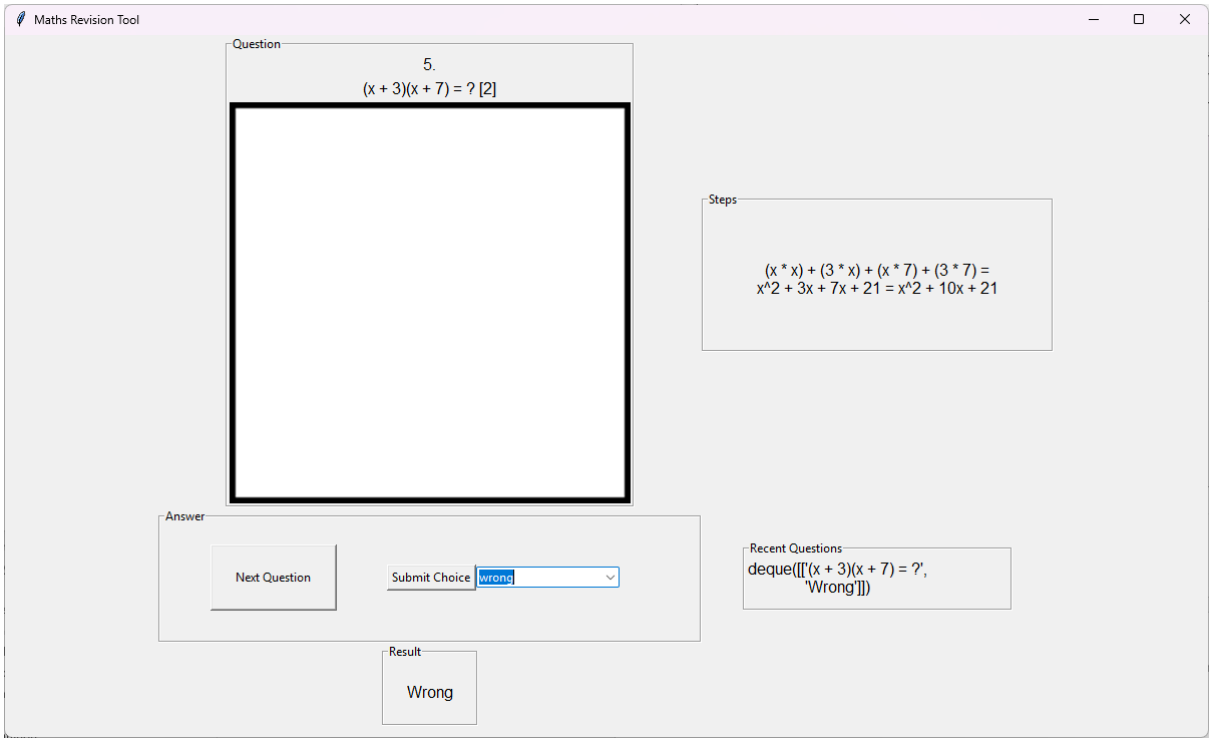
Error outcome for correct option – recent questions queue didn't update (Figure 11)



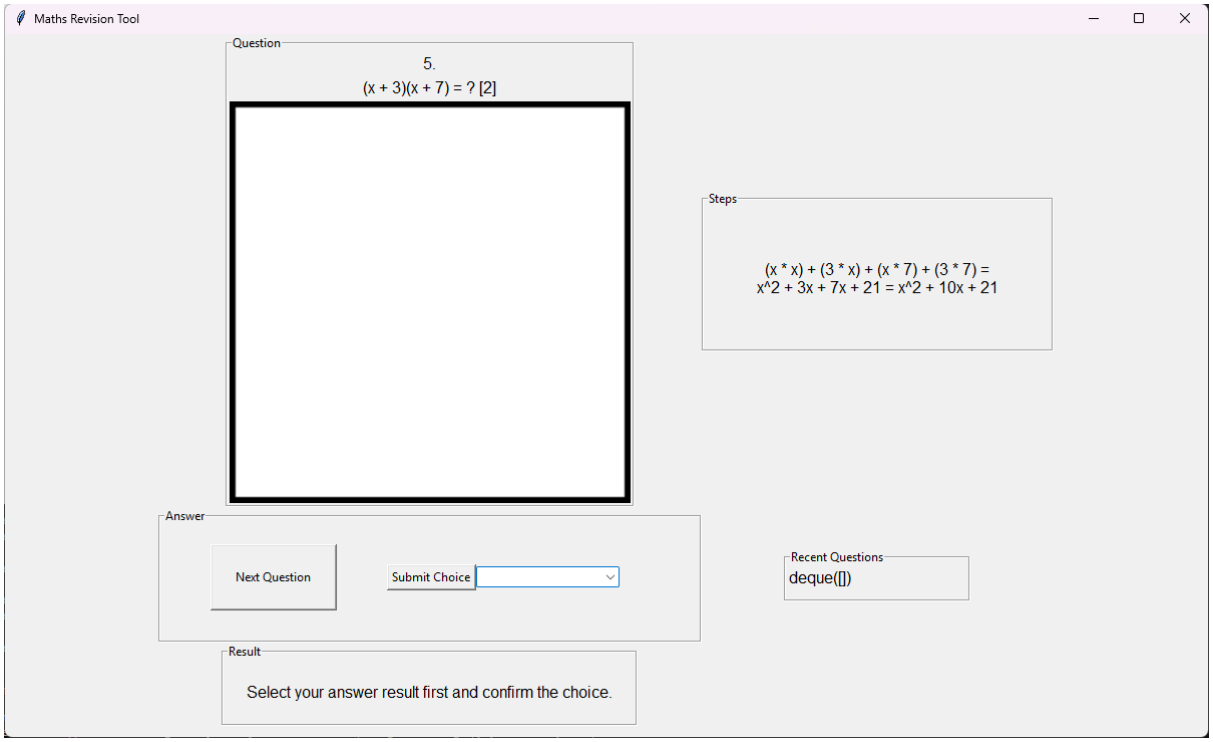
Fixed outcome for correct option selected. (Figure 12)



Expected outcome for wrong option selected. (Figure 13)

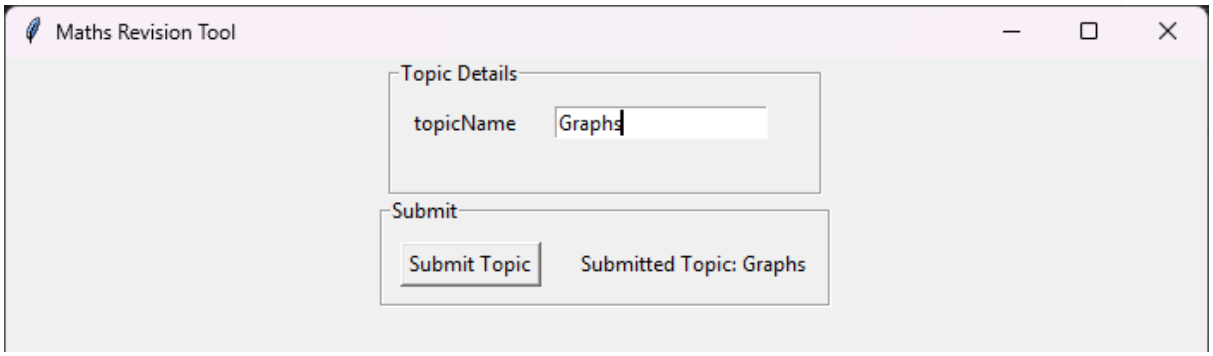


Fixed outcome for skipping input, the error outcome just generated a new question (Figure 14)

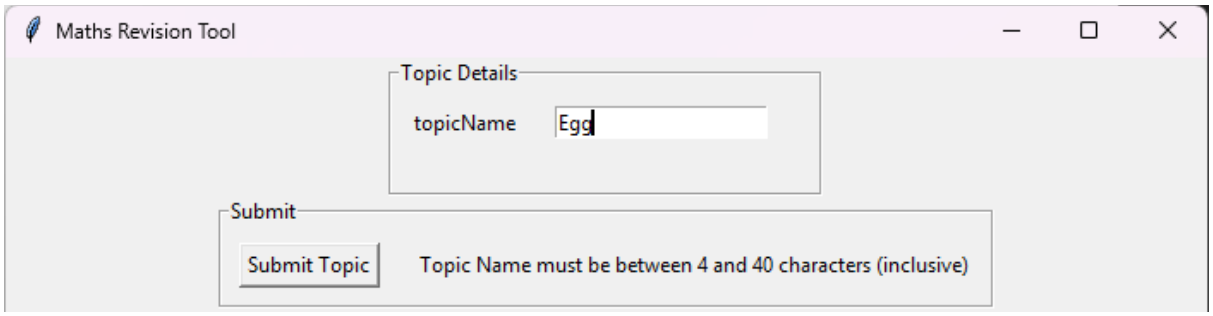


Test No.	Description	NEB	Expected Outcome	Actual Outcome	Comments
5	Adding a topic to the database – **Objective 8.6**	N: Standard topic name inputted	Input taken, SQL executed, and topic added to tblTopic within database.	Success - Same as expected. (Figure 15)	Topics have fields for how many correct questions and wrong questions have been answered, by default they should be 0, but I have set them to 1 as it causes a division error with the topic weighting formula otherwise. (Figure 16)
		E: Topic name outside length range	'Topic name must be between 4 and 40 characters (inclusive)' outputted.	Success - Same as expected. (Figure 17)	No comments.
		B: Topic name on boundary of length range	Same as normal outcome.	Success - Same as expected. (Figure 18, Figure 19)	No comments.

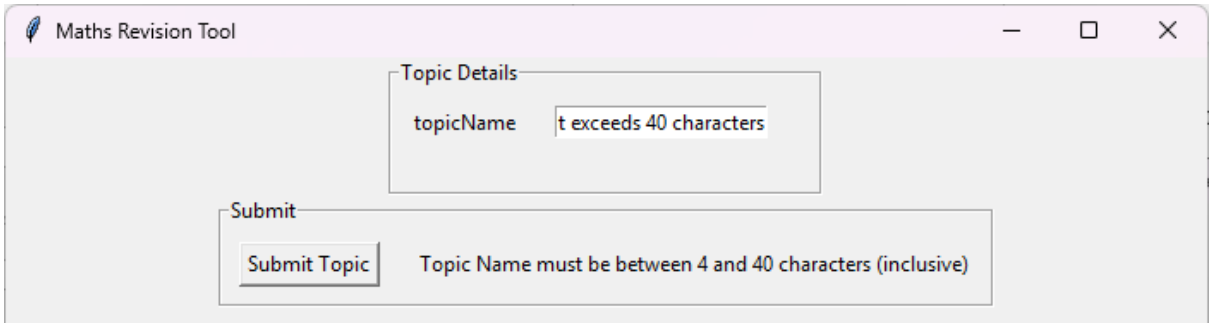
Expected outcome for normal data input. (Figure 15)



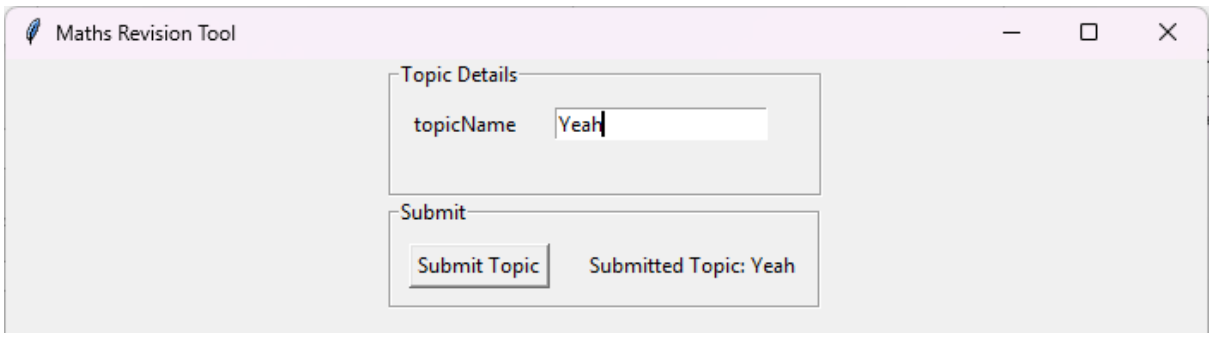
Expected outcome for invalid data input less than 4 characters (Figure 16)



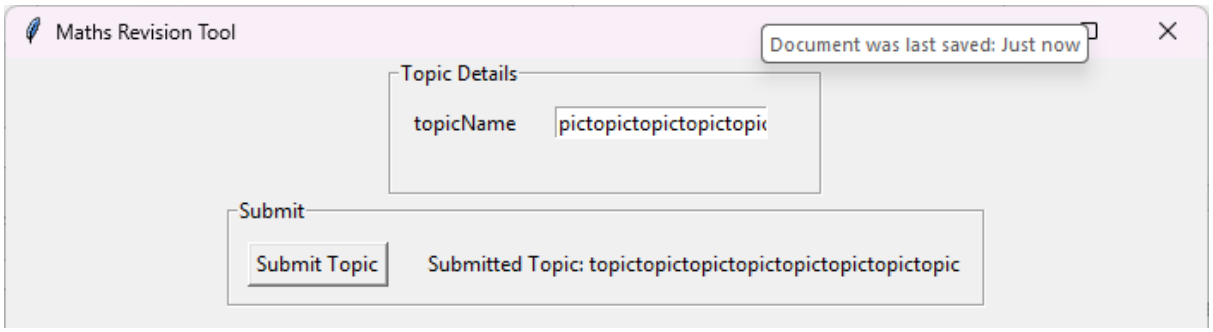
Expected outcome for invalid data input more than 40 characters (Figure 17)



Expected outcome for boundary data input of 4 characters. (Figure 18)

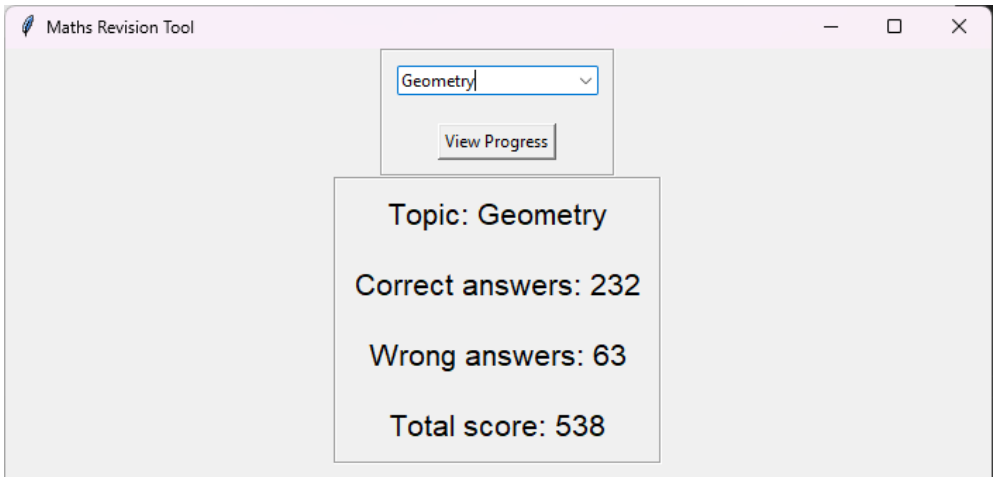


Expected outcome for boundary data input of 40 characters. (Figure 19)



Test No.	Description	NEB	Expected Outcome	Actual Outcome	Comments
7	View topic progress – **Objective 8**	N: Standard topic input from list options	SQL executed, correctQs, wrongQs, and score values fetched for topic inputted, labels updated.	Success - Same as expected. (Figure 20)	Viewing the topic progress involves selecting from a list of the topics, and a button, but it also lets you type in a topic name.
		E: No input	'Error: 'Topic doesn't exist.' outputted.	Fail – NoneType error, as the topic details can't be fetched because the topic doesn't exist. (Figure 21)	Fixed it by recognizing no input as an error trigger, so it outputs that accordingly. (Figure 22)
		B: Topic in the list but manually typed in	Same as normal outcome.	Success - Same as expected. (Figure 23)	No comments.

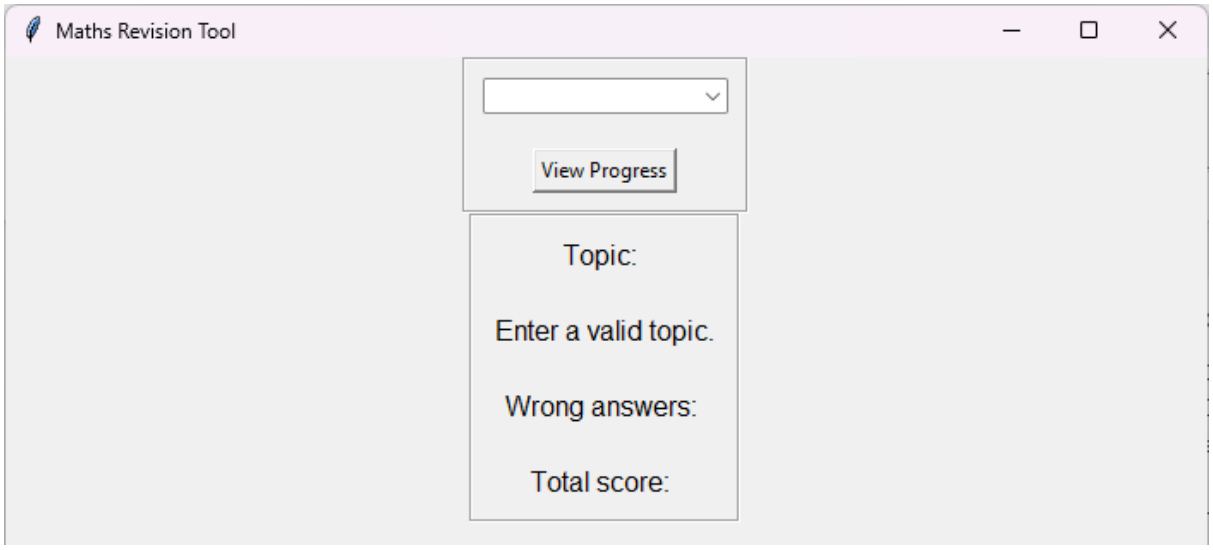
Expected outcome for normal data input. (Figure 20)

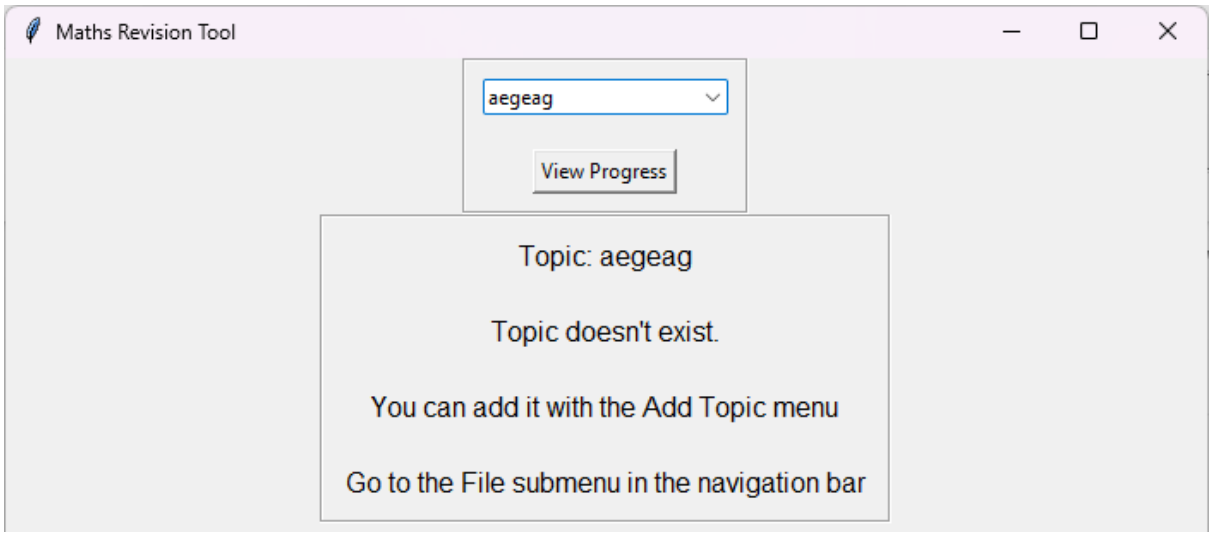


Error outcome for invalid data input: (Figure 21)

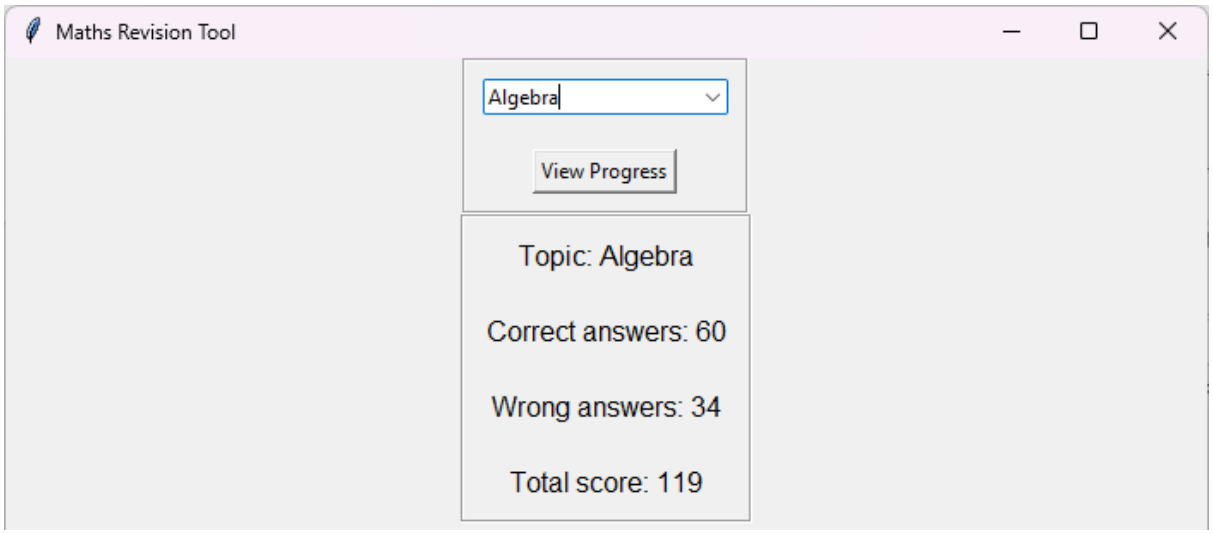
```
  File "c:\Users\abhik\Downloads\maths-platform-firstq-fix\maths-platform\mathexamstuff.py", line 613, in get_progress
    correct_label.config(text=('Correct answers: '+str(topic_scores[0])))
TypeError: 'NoneType' object is not subscriptable
```

Fixed outcome for invalid data input. (Figure 22)



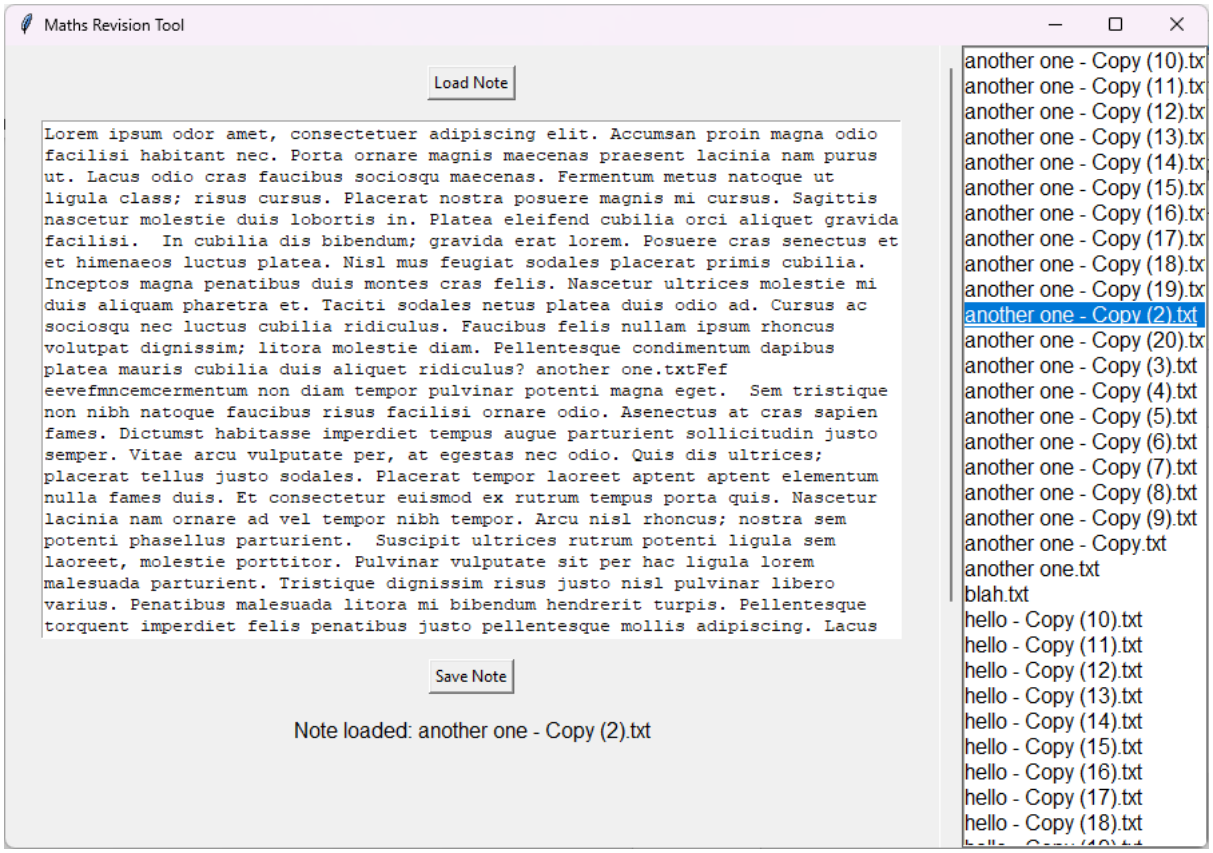


Expected outcome for boundary data input. (Figure 23)

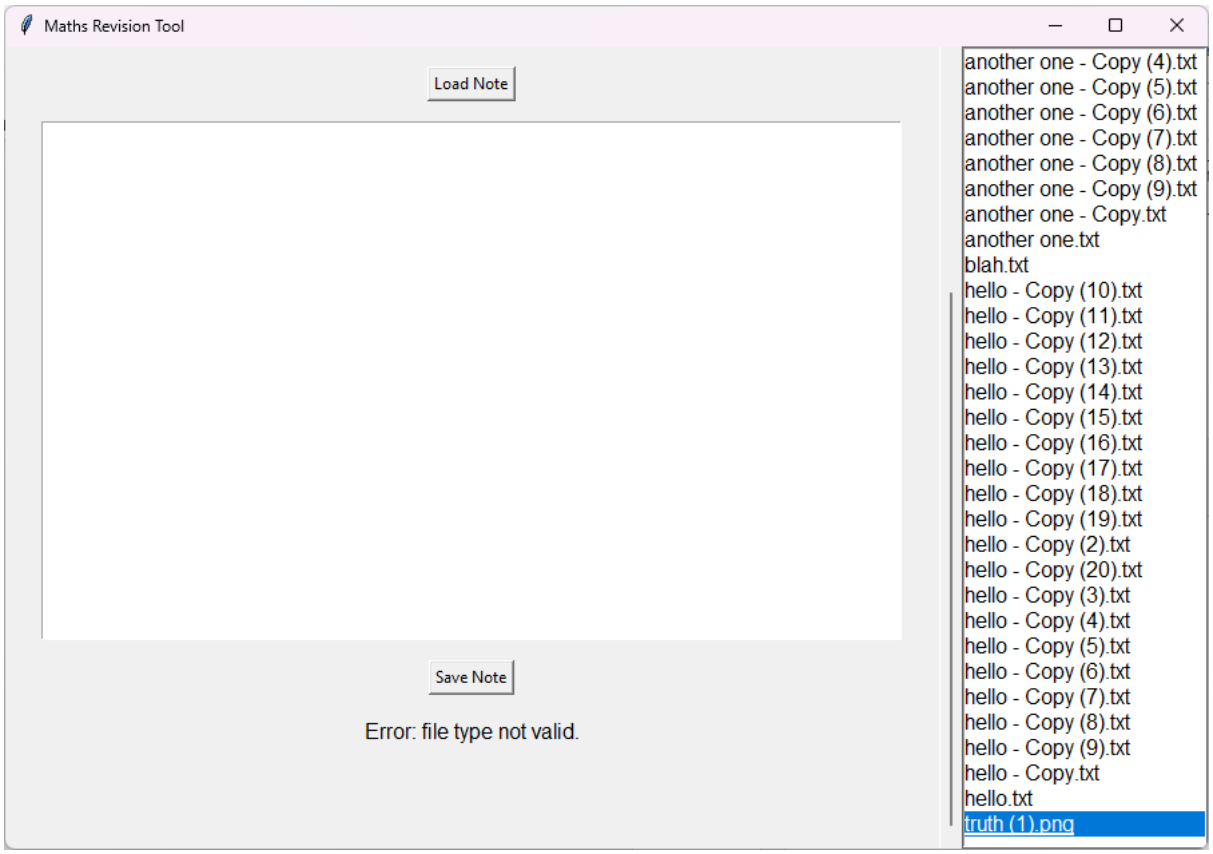


Test No.	Description	NEB	Expected Outcome	Actual Outcome	Comments
8	View notes – **Objective 9**	N: .txt file inputted	Note opened, text is read and copied to editable textbox	Success - Same as expected. (Figure 24)	No comments.
		E: Alternative file type inputted	'Error: file type not valid' outputted.	Success - Same as expected. (Figure 25)	No comments.
		E: No input	'Error: no file input' outputted.	Success - Same as expected. (Figure 26)	No comments.

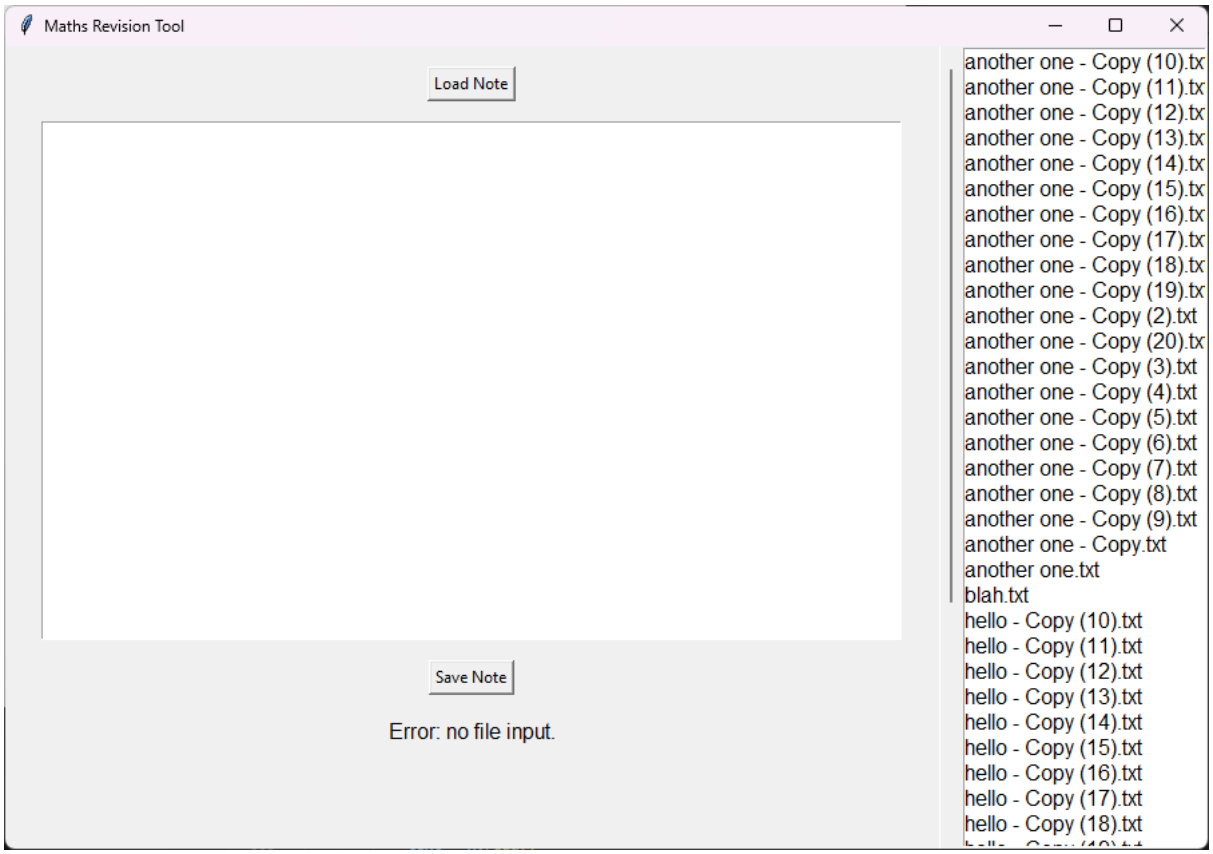
Expected outcome for valid .txt file input. (Figure 24)



Expected outcome for invalid file type input. (Figure 25)

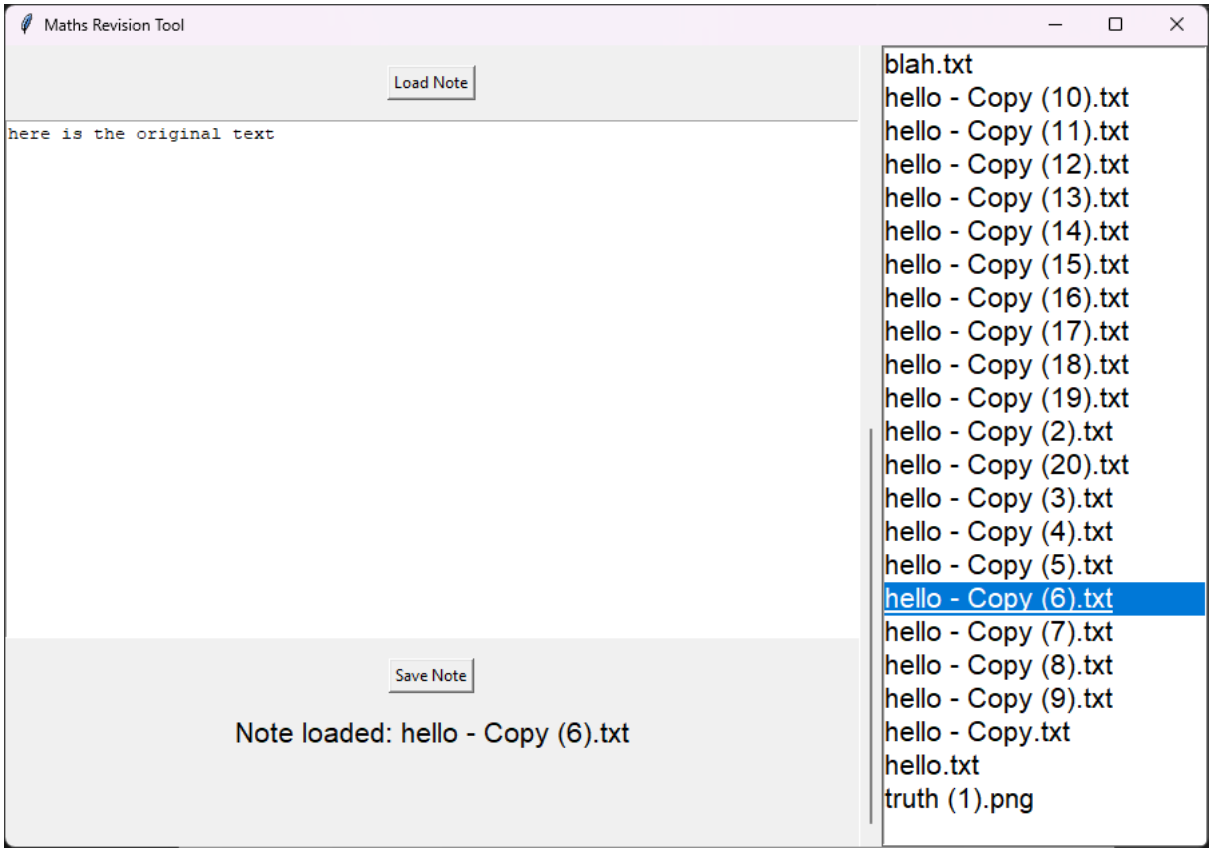


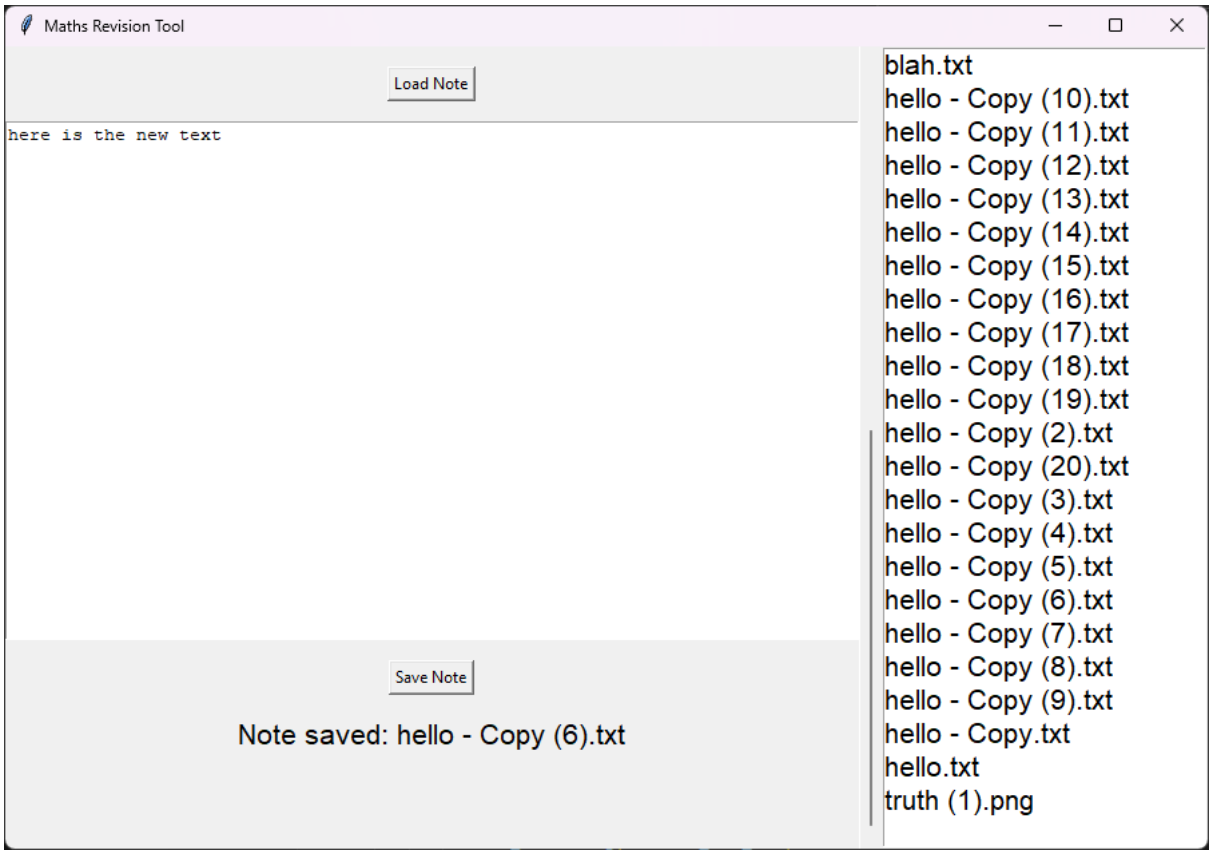
Expected outcome for invalid blank input. (Figure 26)



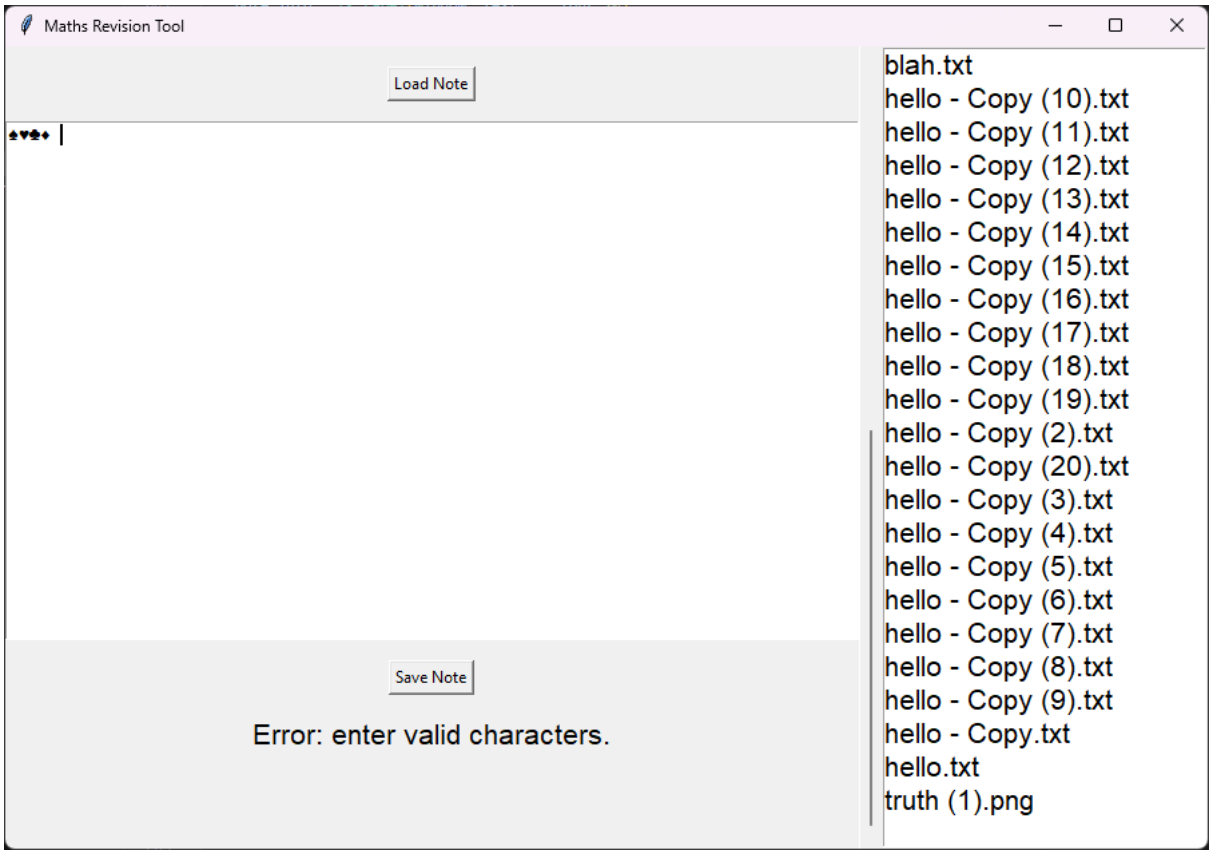
Test No.	Description	NEB	Expected Outcome	Actual Outcome	Comments
9	Edit and save notes – **Objective 9**	N: Standard text input	Text from textbox read and copied to current file loaded, file saved, 'Note saved: (note filepath)' outputted.	Success - Same as expected. (Figure 27)	No comments.
		E: Special character input	'Error: enter valid character' outputted.	Success - Same as expected. (Figure 28)	No comments.
		B: Exceedingly large text input	Same as normal outcome.	Success - Same as expected. (Figure 29)	Save time was considerably slower.

Expected outcome for normal input – standard text. (Figure 27)

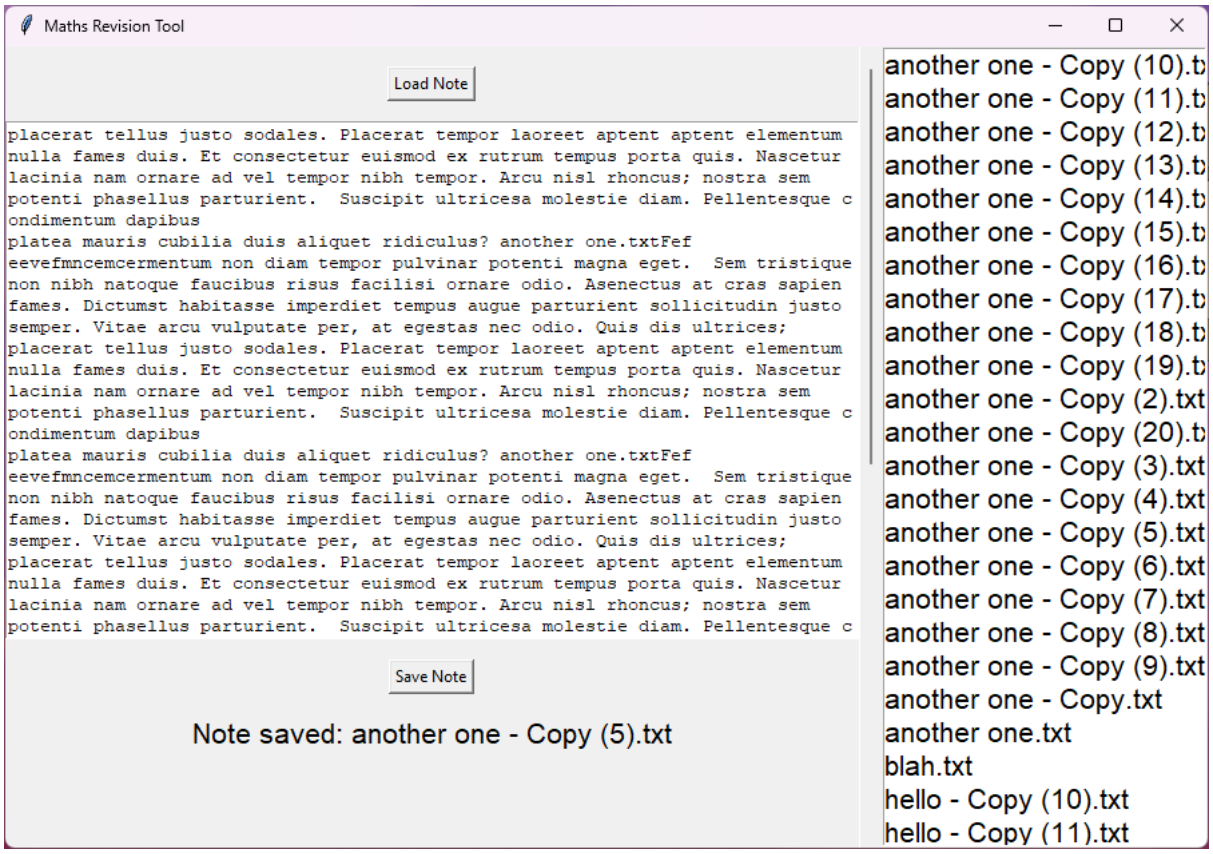




Expected outcome for invalid input – special characters. (Figure 28)

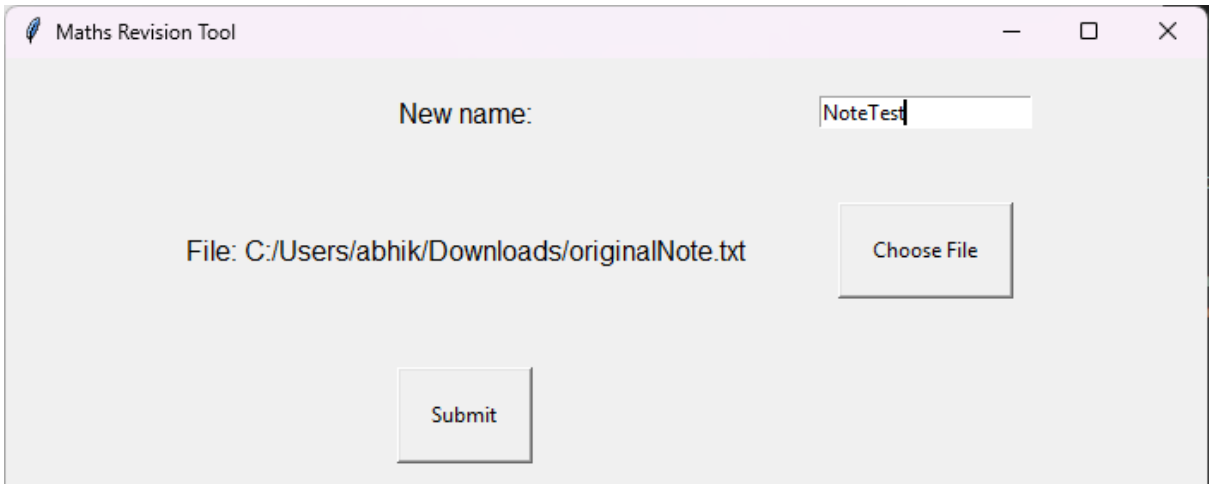


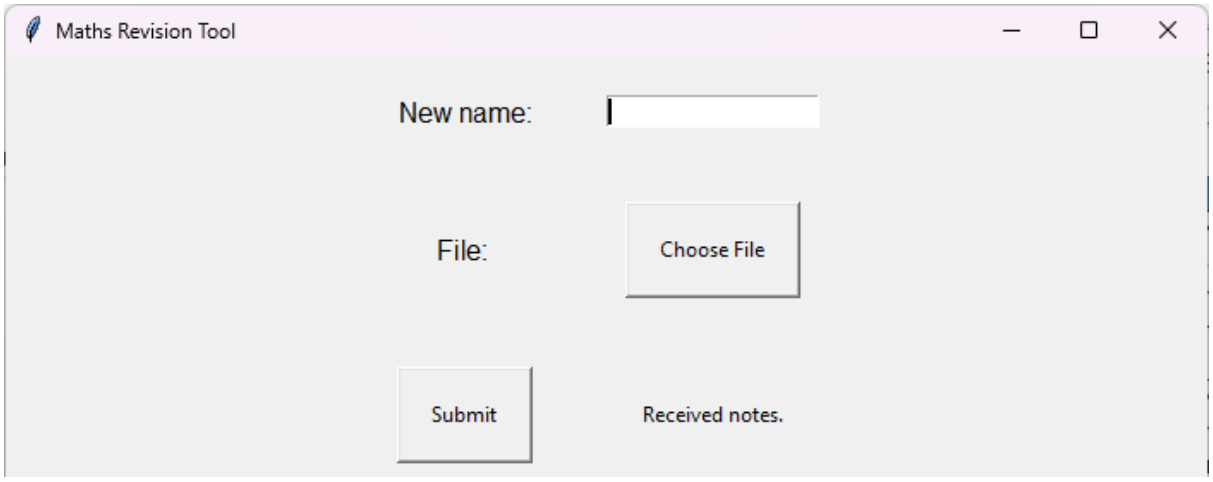
Expected outcome for boundary input – exceedingly large text quantity. (Figure 29)



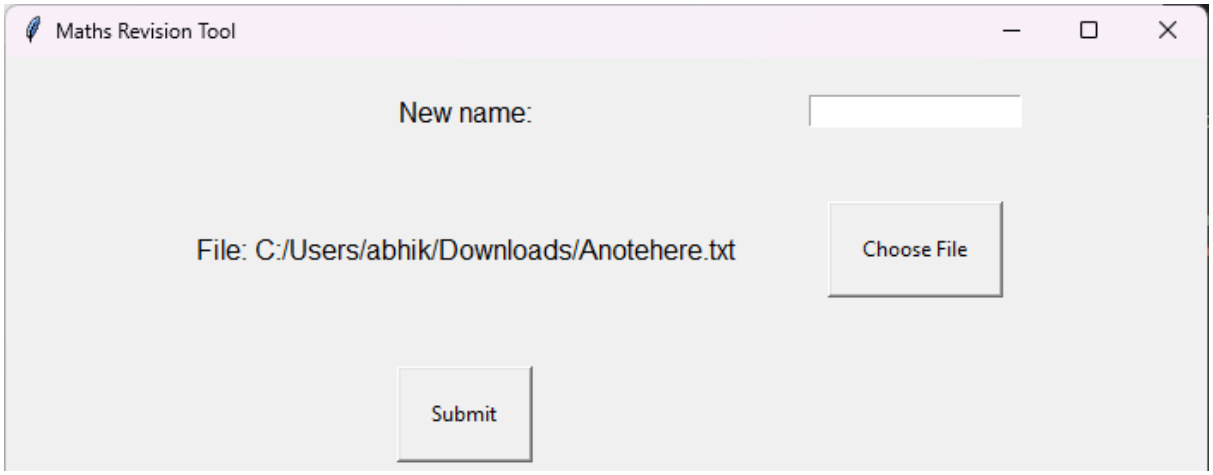
Test No.	Description	NEB	Expected Outcome	Actual Outcome	Comments
10	Add note menu – **Objective 9**	N: Normal text input for new note name	'Received notes' outputted. File gets moved to notes folder with the input name.	Success - Same as expected. (Figure 30)	No comments.
		N: No new note name input	'Received notes' outputted. File gets moved to notes folder with its original name.	Success - Same as expected. (Figure 31)	No comments.
		E: Invalid file type (non-txt)	'Error: invalid file type, use a text file" outputted.	Success - Same as expected. (Figure 32)	The file I attempted to input was a .jpg image file.

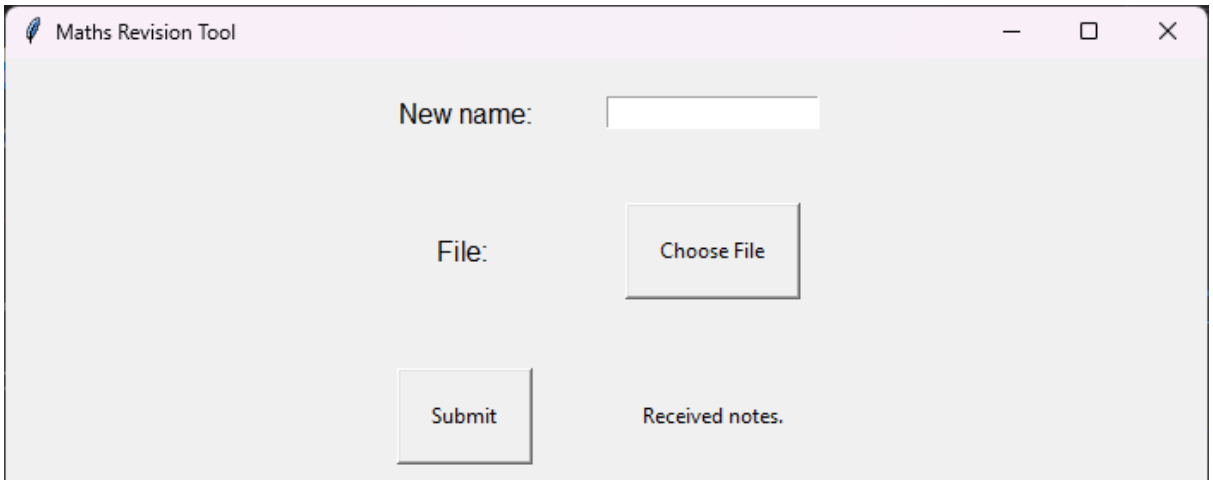
Achieved expected outcome for valid text input. (Figure 30)



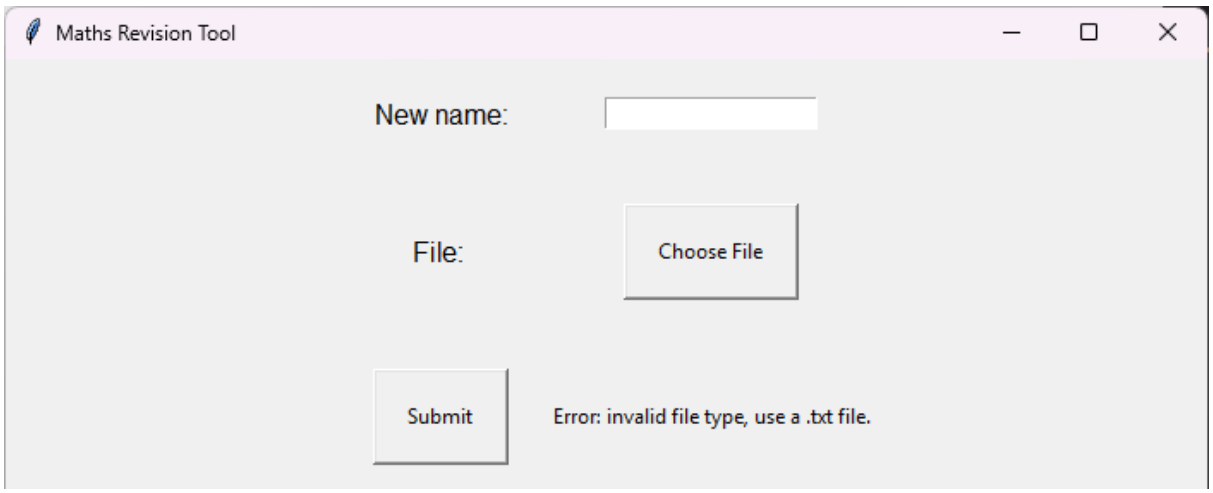


Achieved expected outcome for valid input – no note-name input. (Figure 31)





Achieved expected outcome for invalid file-type input. (Figure 32)



Video Testing

To show the tests in further detail, I have recorded videos of my testing. The videos allow me to show further the extent of use of the system through a user's point of view. Additionally, I have referenced the video tests within my modular testing where relevant, through the use of timestamps.

The tests I've shown include:

- Answer Questions menu
  - Initial question generation
  - Question being answered and checked
  - Next question being generated

- View Topic Progress menu
  - Fetch topics
  - Display topic statistics

- View/Edit Notes menu
  - Fetch notes list
  - Load and display note
  - Edit and save note

- Add Question menu
  - Accept question inputs accordingly
  - Add question to database

- Add Notes menu
  - Create new note file
  - Add existing note to notes folder

- Add Image menu
  - Add image to images folder
  - Change image name to question number

- Add Topic menu
  - Accept topic name input
  - Add new topic to database

- Play Game

Video Links: AQA A Level Computer Science NEA Testing Video

Client Testing

My testing was to make sure that the system operates in the way that I wish. In addition to that, I've had the client test the system too. This is more of a user experience test to check that the full system works well together and it's appropriate for use in revision. I've gone much more into detail about the client's personal experience within the Client Evaluation subsection in my Evaluation section. The client used the system for 2 of his Maths revision sessions to get an understanding of the features.

Positives from Client:

- Allows him to answer questions easily
  - The need for finding questions is completely eliminated since they're all stored via the database, and they can be fetched and answered immediately

- Can take notes digitally
  - Taking notes by hand is slower for the client, as he is fairly versed in typing and it allows him to save time

- Very central way to track topic progress
  - The client is able to tell much more easily if he's weaker in any specific area through the menu, this isn't as simple using traditional revision methods

- The game is very fun

Negatives from Client:

- No working area within the system
  - The client used paper to work out his answers for some of the questions, which makes sense but somewhat defeats the purpose of the system, as it should offer everything which is needed, so a working area would help

- Limited number of questions
  - The reliance on the database means that the question pool is fairly limited, and the client will need to add their own questions at some point, which is time consuming

- Harder to use when in fullscreen
  - The system is designed for use in windowed mode, the elements look too small when the window is maximised to fullscreen, it should be adaptable to any viewing size for the best accessibility

Evaluation

Comparison of Results against Objectives

No	Objective	Requirements	Evaluation
1	Store all question details	Store each individual question detail so they can be used for generating questions	This was completed fully. I used a database and within that database I had a table called tblQuestion. That table had several columns for each question detail, those being: qText, qImage, qNo, qAns, qTopic, qMarks, qType, qSteps. This allowed me to fetch and display questions, and mark them based on the user's input, or alternatively let the user self-check if the question can't be checked via an auto-mark, which is decided based on the qType value.
2	Answer questions	Allow user to see a question being generated and displayed	This was completed fully. I made a window, which initially runs an SQL command to fetch question details of a random question. The text areas to display question details are created and arranged in a user-friendly format. Their text contents are updated to the question details fetched, along with the question image being shown (if there is one), and an input answer field and show/mark answer button being added.
3	Mark questions as they go	Allow user to mark their questions and see results immediately after attempting the question	This was completed fully. When the show/mark answer button is pressed, the correct answer is displayed and the question working steps are displayed. If the question type is automatic, the correct answer is compared to the user's input. If the question type is automatic, a choicebox is generated to select if the user's answer was right or not. The show-answer button is updated to a next-question button.
4	Update question details	Fetch a new question and update the display contents to show the new question details	This was completed fully. The question details (except for the first question) are now fetched based on an algorithm (see objective 6). These question details are updated to the text content displayed in the window. The question steps and answer result boxes are cleared. The choicebox is removed or added depending on the type of the previous question and the new question being generated. The button functionality is changed back to the show/mark answer button.
5	Store marks for answers	Update performance values for topics based on question results	This was completed fully. Based on whether the result was correct or wrong, the question result is displayed, and an SQL command is run to increment the score and correctQs values by 1 if they got it right or just increment the wrongQs by 1 if they got it wrong. The values are updated within the tblTopic for the topicName which matches the qTopic of the question that was just answered by the user.

6	Give questions for harder topics	Generate questions of specific topics based on the user's performance within those topics	This was completed fully. I used a formula to calculate relative proportion of correctQs against totalQs, and making a dictionary, with the key-value pairs being the topics. It can then use random.choice with keys set as the dictionary, which will therefore generate questions at a rate based on the user's performance within each topic. Of course, this is random, so it is totally possible that it keeps generating easy questions by chance.
7	Input their own questions	Allow user to input question details in order to create a new question	This was completed fully. I made a window, with input boxes and labels showing what each input is, and a submit button, which runs an SQL command taking the question detail inputs as parameters, to update the tblQuestion in the db.
8	View topic progress	Display statistics based on the user's performance in questions for each topic	This was completed fully. The program fetches the topics list, and each of the values for these topics using an SQL command. Then, a window with a choicebox is generated, allowing the user to select the topic they want to see their performance for. When a topic is selected, the values shown are updated accordingly.
9	View/edit notes	Load and view notes to study from, edit those notes and save the changes	This was mostly completed. I created a notes folder to store the notes that can be used within the program. To add notes to this folder, the Add Notes menu can be used to change the name of existing notes files and add them to the system's notes folder. Within the View Notes menu, there's a list of text files on the right that can be selected and loaded. The text is loaded into a large input box, and this text can be read or edited and the changes made can be saved. The only feature missing is creating new blank notes. To do this, the user must create a text file anywhere through their operating system's file explorer and then use the Add Notes menu to add it the notes folder or transfer it themselves. This isn't difficult by any means and doesn't take long at all but counteracts ease of use.
10	Level system with points and rewards	Increase level XP when the user gets questions right, in order to progress and unlock items	This was not completed. I ended up not including this feature and added the platformer game that I created instead. The game is a platformer where you use advanced movement to complete different levels, where the goal is to clear all the levels in a world. There are 2 worlds, each with 6 levels. There are two main goals, to complete the game within the shortest time possible and obtain the most coins possible, by collecting coins within the level, and using the operation power-ups to increase your coin count.

Client Evaluation

No	Objective	Requirements	Evaluation
1	The user must be able to create questions	Allow the user to add their own questions so they can practice those questions later	"I was able to make questions fairly easily. I just used the menu and inputted the details in the Add Question menu to add the questions I wanted. The only difficulty was if I wanted to use images within the program, I would have to download them and then add them to the image folder either through the file explorer or the Add Image menu. Additionally, the whole process of adding questions can be somewhat tedious." The problem with using image address links is that it removes the ability to run the program even when offline, which is the intention of the software. The generation of questions automatically would be useful and save a lot of time instead of adding a lot of questions manually. This is feasible for simple arithmetic questions but would probably require an API to fetch longer worded questions or questions with more steps, this also requires online functionality.
2	The user must be able to answer questions	Allow the user to view questions and input an answer to the question	"The way the questions were displayed was fine. The layout of the elements was practical, and it was easy to view all the details. The input box was fine, but I can only see part of what I have typed when I'm typing a much longer answer." The input box probably should expand when the user is typing something beyond the length of the input box itself, so that the user can see the whole sequence of characters that they have typed.
3	The user must be able to get results	Mark the user's answers to the questions and show the result	"The fact that it shows working steps for the question after I check my answer is very useful. I like the instant feedback after each question. I think the manual marking system is quite useful too, allowing me to answer basically any question. However, sometimes the working steps are confused, and I don't understand why they are doing the calculations that are being performed." It makes sense that the working steps can sometimes be unclear. To make up for this, a separate question detail variable (e.g. qExplanation) could be added that gives a more in-depth explanation into why the question is answered in that way and shows exactly what they did. This could be directly shown along with the working steps, or shown if the user chooses to see it.

4	The user must be able to view their progress	Allow user to view how well they have been answering the questions for each topic	"The View Progress Menu is good; it provides statistics which are helpful. It kind of helps me to know if I am struggling in certain areas and need to focus on specific topics more. I think the layout could be improved, instead of just listing each progress value one by one." The layout could definitely be improved upon. It is very basic and could be presented in a way that more directly informs the user on what they're doing well in, and what they need to work on, instead of just values listed as the client has stated.
5	The user must be able to view and edit notes	Allow user to load notes, read those notes, and edit them, making any changes, and saving those changes	"I think the notes app is not that good to be honest. The interface has a weird structure. I can't easily add new blank notes or rename my notes. The way to do it through the Add Note menu is so long. I think the actual viewer/editor is decent, and the save feature is alright, but save times are quite slow for very large files. It is nice to have note-taking as part of the system, but there are alternative applications that work much better than this." The notes app could be improved, it is fairly basic. I think having a submenu where it shows the notes first, and then the user can select a note, which opens an editor for that note in a new menu. The way to add new notes is inefficient as I previously mentioned, and could be fixed with a Create New Note function via a button.
6	The user must be able to play the game provided	Allow user to play a platformer game, the game is somewhat Maths related and intended for user enjoyment as a reward for time spent studying	"I really like the game, it's genuinely addictive. The movement mechanics are well polished, and the animation makes it look quite nice, there's animations for practically every action and motion. The speedrun aspect is very engaging, it's the same levels but I still enjoy replaying them multiple times to see if I can improve my time further. The leaderboard also helps with this, as I can see other people's times and aim to beat them and improve upon them. The coin goals with the operation power-ups is a bit simple. The level design in World 1 is impressive, and there's a lot of variety. World 2 is also fun, but world 1 is definitely better. The overall user interface before starting the game looks alright, could be improved. The only problem is sometimes I get lost between the menus, it is possible to navigate between all of them, but it can get a bit confusing."

			I'm glad to hear the client enjoys the game. I think I understand the problem he means with the menus. For example, going into a submenu like the Worlds menu or the Leaderboards menu, there's no way to go back to the original menu, apart from going into one of those menus and then closing it, taking you back to the start. This should be improved upon with a simpler menu navigation layout. Additionally, the operation power-ups are a bit simple, they don't relate that strongly to Maths as a whole. I added them so that the overall system relates to the game more. Instead, I think a more direct link with the system would work better, such as – if you study for a certain amount of time, you can play the game for a proportionate amount of time.

Potential System Improvements

Looking back, there are some parts and functions of the system that can be improved upon.

I think the User Interface of the main system is somewhat unappealing for these reasons:

- The font looks pixelated when the text is small
- The structure is blocky and bulky
- Lack of animations or smooth features
- Colour scheme is basic (black text on white background)
- The windows look ugly when resized to fullscreen, too much blank space around the main elements of the layout

Improvements that could be made for the main system:

- Use smoother modern font-face
- Make elements with unique shapes, borders, sizes, drop shadows
- Add window transitions and button animations
- Have multiple customizable colour themes (e.g. dark mode, visibility mode)
- Make elements resize based on the dimensions of the window + expanding textboxes

Functionality I think could be added/improved for the main system:

- Automatically generated questions for simpler arithmetic questions
- Question working explanations in addition to the working steps
- Ability to easily create new blank notes within the system

Potential Game Improvements

I like how the game turned out, the movement mechanics, level structure, animations, and timer system worked together quite well to make a well-rounded game.

The main improvements/additions I'd make are:

- Make coin counter fixed value across all levels, instead of tracking the number of coins at the end of each level and finding a total
- Make the speedrun timer a constant timer that doesn't reset when completing a level, instead of tracking the time at the end of each level and finding a total
- Add some additional buttons to parts of the menu so that the user can immediately get to the main screen, so that navigation from one menu to any other menu is possible within 3 scene changes
- Make it more directly linked to the Maths platform; a feasible idea is that the system tracks the amount of time that the user studies for, and allows the user to play the game for a proportionate amount of time by closing the game after a certain point.
- This is probably the hardest idea, but it would be the best by far if it was implemented successfully. This was something I wanted to do initially, but definitely didn't have the time to fulfil so I ended up leaving it out – Allow the user to make their own levels, using a set of pre-defined resources, such as the blocks, spikes, and star. The user would set their spawn point too. Once the user has completed the level as a test run, to show that it is possible, they can upload it to the cloud platform. Once uploaded, it's shown within the game in the Custom Levels menu, and anyone else can play the level. There will be an automatically generated leaderboard for each level, and players can compete to get the fastest times on not just the game's levels made by me but also made by other players. Levels can be ranked by difficulty/skill based on weighted completion rates and average time taken, and players can obtain ranking points based on the difficulty of levels they clear, and the number of levels they clear. The ranking points of each player are used to determine an overall leaderboard showing the best players. This concept takes a lot of influence from the game Super Mario Maker, where the whole game is based around players making their own custom levels for other players to attempt.

```
import tkinter as tk
from tkinter import ttk
from tkinter import *
from tkinter.filedialog import askopenfilename
import sqlite3
import random
import os
import textwrap
import subprocess
from PIL import Image, ImageTk
from queue import Queue

conn = sqlite3.connect("MathsPlatformStorage.db") #Connect to db and configure cursor
cursor = conn.cursor()
question_table_query = """CREATE TABLE IF NOT EXISTS tblQuestion (qNo INT, qText
TEXT, qImage TEXT,
                        qRating TEXT, qMarks INT, qTopic TEXT, qType TEXT, mSteps TEXT,
                        mAns TEXT, PRIMARY KEY (qNo))""" #Create question table

topic_table_query = """CREATE TABLE IF NOT EXISTS tblTopic (topicNo INT, topicName
TEXT, correctQs INT, wrongQs INT, score INT, PRIMARY KEY(topicNo))""" #Create topic
table

cursor.execute(question_table_query)
cursor.execute(topic_table_query)

conn.commit() #Commit changes to db
conn.close() #Close db connection

def add_question():
    #Assign appropriate variables for each input of question details
    qText = entries[0].get()
    qImage = entries[2].get()
    qRating = entries[1].get()
    qMarks = entries[3].get()
    qTopic = entries[4].get()
    qType = entries[5].get()
    mSteps = entries[6].get()
    mAns = entries[7].get()

    if mSteps == "":
        mSteps = "No working steps provided."

    if "" in [qText, qRating, qType, qTopic, mAns]:
        question_added_label.config(text="Please enter full valid inputs.")
    else:
        conn = sqlite3.connect("MathsPlatformStorage.db")
        cursor = conn.cursor()

        count_query = "SELECT COUNT(*) FROM tblQuestion" #Get number of questions
        cursor.execute(count_query)
        result = cursor.fetchone()
        totalQs = result[0]
        qNo = totalQs + 1 #Adds new question as next question number

        data_insert_query = """INSERT INTO tblQuestion (qNo, qText, qImage,
        qRating, qMarks, qTopic, qType, mSteps, mAns) VALUES (?,?,?,?,?,?,?,?,?)"""
        data_insert_tuple = (
            qNo,
            qText,
            qImage,
            qRating,
            qMarks,
            qTopic,
            qType,
            mSteps,
            mAns,
        )

        cursor.execute(data_insert_query, data_insert_tuple) #Inserts each associated
        value into the question record
        conn.commit()
```

```
69
70              conn.close()
71              question_added_label.config(text="Question added.")
72
73
74
75      def add_question_menu():
76          global entries, question_added_label
77          conn = sqlite3.connect('MathsPlatformStorage.db')
78          cursor = conn.cursor()
79          cursor.row_factory = lambda cursor, row: row[0]
80          topics = cursor.execute('SELECT topicName FROM tblTopic').fetchall() #Gets all
            the topic names
81
82          window2 = tk.Toplevel()
83          window2.focus()
84
85          window2.geometry("700x500")
86          window2.title("Maths Revision Tool")
87
88          frame = tk.Frame(window2) #Configure user interface elements
89          frame.pack()
90
91          question_details_frame = tk.LabelFrame(frame, text="Question Details")
92          question_details_frame.grid(row=0, column=0, pady=30)
93
94          buttons_frame = tk.LabelFrame(frame, text="Buttons")
95          buttons_frame.grid(row=2, column=0)
96
97          question_added_label = tk.Label(frame, text="")
98          question_added_label.grid(row=3, column=0)
99
100         qText_label = tk.Label(question_details_frame, text="qText:")
101         qText_label.grid(row=0, column=0)
102
103         qText_entry = tk.Entry(question_details_frame)
104         qText_entry.grid(row=0, column=1)
105
106         qImage_label = tk.Label(question_details_frame, text="qImage:")
107         qImage_label.grid(row=1, column=0)
108
109         qImage_entry = tk.Entry(question_details_frame)
110         qImage_entry.grid(row=1, column=1)
111
112         qRating_label = tk.Label(question_details_frame, text="qRating:")
113         qRating_label.grid(row=2, column=0)
114
115         qRating_entry = ttk.Combobox(
116             question_details_frame,
117             values=[
118                 "",
119                 "1 - Very Easy",
120                 "2 - Easy",
121                 "3 - Medium",
122                 "4 - Hard",
123                 "5 - Challenge",
124             ],
125         )
126         qRating_entry.grid(row=2, column=1)
127
128         qMarks_label = tk.Label(question_details_frame, text="qMarks:")
129         qMarks_label.grid(row=0, column=2)
130
131         qMarks_entry = tk.Spinbox(question_details_frame, from_=1, to=100)
132         qMarks_entry.grid(row=0, column=3)
133
134         qTopic_label = tk.Label(question_details_frame, text="qTopic:")
135         qTopic_label.grid(row=1, column=2)
136
137         qTopic_entry = ttk.Combobox(question_details_frame, values=topics)
138         qTopic_entry.grid(row=1, column=3)
139
```

```
140         qType_label = tk.Label(question_details_frame, text="qType:")
141         qType_label.grid(row=2, column=2)
142 
143         qType_entry = ttk.Combobox(
144             question_details_frame,
145             values=["", "Automark - num answer", "Manual - string answer"],
146         )
147         qType_entry.grid(row=2, column=3)
148 
149         mSteps_label = tk.Label(question_details_frame, text="mSteps:")
150         mSteps_label.grid(row=3, column=0)
151 
152         mSteps_entry = tk.Entry(question_details_frame)
153         mSteps_entry.grid(row=3, column=1)
154 
155         mAns_label = tk.Label(question_details_frame, text="mAns:")
156         mAns_label.grid(row=3, column=2)
157 
158         mAns_entry = tk.Entry(question_details_frame)
159         mAns_entry.grid(row=3, column=3)
160 
161         submit_button = tk.Button(buttons_frame, text="Submit", command=add_question)
162         submit_button.grid(row=0, column=1, ipadx=30, ipady=10)
163 
164         for widget in question_details_frame.winfo_children():
165             widget.grid_configure(padx=20, pady=14, ipadx=15, ipady=8) #Add spacing
            between elements in question frame
166 
167         for widget in buttons_frame.winfo_children():
168             widget.grid_configure(padx=20, pady=14) #Add spacing between elements in
            button frame
169 
170         entries = [qText_entry, qRating_entry, qImage_entry, qMarks_entry, qTopic_entry,
        qType_entry, mSteps_entry, mAns_entry]
171 
172 
173     def answer_question_menu():
174         global questionFrame, first_q, enterAnswer, label_img
175 
176         first_q = True
177         recent_questions = Queue(maxsize = 5) #Queue for 5 most recently answered
        questions
178 
179         window = tk.Toplevel()
180         window.focus()
181         window.geometry("1200x800")
182         window.title("Maths Revision Tool")
183 
184         frame = tk.Frame(window)
185         frame.pack()
186 
187         conn = sqlite3.connect("MathsPlatformStorage.db")
188         cursor = conn.cursor()
189 
190         count_query = "SELECT COUNT(*) FROM tblQuestion" #Get number of questions
191         cursor.execute(count_query)
192         result = cursor.fetchone()
193         totalQs = result[0]
194         currentQ = random.randint(1, totalQs) #Get random question for first question
195 
196         search_query = (
197             "SELECT qText, qRating, qImage, qMarks, qTopic, qType, mSteps, mAns FROM
            tblQuestion WHERE qNo="
198             + str(currentQ)
199         )
200         cursor.execute(search_query)
201         question_info = cursor.fetchone() #Get question details
202 
203         questionFrame = tk.LabelFrame(frame, text="Question")
204         questionFrame.grid(row=0, column=0)
205 
206         questionNoText = str(currentQ) + "."
```

```
207
208         questionNo = tk.Label(questionFrame, text=questionNoText, font=30)
209         questionNo.pack()
210
211         questionText = question_info[0] + " [" + str(question_info[3]) + "]"
212
213         questionTextLabel = tk.Label(questionFrame, text=questionText, font=30)
214         questionTextLabel.pack()
215
216         if question_info[2] != "":
217             shown_image_path = "images/" + question_info[2]
218         else:
219             shown_image_path = "images/filler.png"
220
221         image = Image.open(shown_image_path)
222         if shown_image_path == "images/filler.png":
223             resize_image = image.resize((400,30))
224         else:
225             resize_image = image.resize((400,400))
226         image_obj = ImageTk.PhotoImage(resize_image)
227         label_img = tk.Label(questionFrame, image=image_obj)
228         label_img.image = image_obj
229         label_img.pack()
230
231         answerFrame = tk.LabelFrame(frame, text="Answer")
232         answerFrame.grid(row=1, column=0, ipadx=40, ipady=5)
233         recentQuestionFrame = tk.LabelFrame(frame, text="Recent Questions")
234         recentQuestionFrame.grid(row=1, column=1, ipadx=40, ipady=5)
235         recentQuestionsLabel = tk.Label(recentQuestionFrame, text=recent_questions.queue,
            font=("Arial", 12), wraplength=200)
236         recentQuestionsLabel.grid(row=0, column=0)
237
238         if question_info[5] == 'Automark - num answer':
239
240             enterLabel = tk.Label(answerFrame, text="Enter:", font=25)
241             enterLabel.grid(row=0, column=0)
242
243             enterAnswer = tk.Entry(answerFrame, font=20)
244             enterAnswer.grid(row=0, column=1)
245
246         stepsFrame = tk.LabelFrame(frame, text="Steps")
247         stepsFrame.grid(row=0, column=1)
248         steps = tk.Label(stepsFrame, font=30, wraplength=250)
249         steps.grid(row=0, column=0, pady=50, padx=50)
250
251         def show_answer(question_info):
252             global resultCheck, answer, choiceBox, choiceButton, enterLabel, enterAnswer
253             submitAnswerButton.config(text="Next Question", command=lambda: next_question(
                question_info)) #Mutual recursion - as the user continues it runs two
                alternating functions throughout
254             steps.config(text=question_info[6])
255             conn = sqlite3.connect("MathsPlatformStorage.db")
256             cursor = conn.cursor()
257             if question_info[5] == 'Automark - num answer':
258                 answerText = "Answer: " + str(question_info[7])
259                 userAnswer = enterAnswer.get() #Get user input for answer
260             else:
261                 answerText = ''
262             answer = tk.Label(answerFrame, text=answerText, font=30)
263             answer.grid(row=1, column=1)
264
265             resultFrame = tk.LabelFrame(frame, text="Result")
266             resultFrame.grid(row=2, column=0)
267             resultCheck = tk.Label(resultFrame, font=40)
268             resultCheck.grid(row=0, column=0, padx=20, pady=20)
269
270             if question_info[5] == 'Automark - num answer': #Automark question case
271                 if userAnswer == str(question_info[7]): #If the answer is correct,
                    increase correctQs by 1
272                     cursor.execute(
273                         "UPDATE tblTopic SET correctQs = correctQs + 1 WHERE
                            tblTopic.topicName = '"
```

```
                    + str(question_info[4])
                    + "'"
                )
                cursor.execute( #Increase score by the question marks
                    "UPDATE tblTopic SET score = score + "
                    + str(question_info[3])
                    + " WHERE tblTopic.topicName = '"
                    + str(question_info[4])
                    + "'"
                )
                resultText = 'Correct!'
            else: #If wrong, increase wrongQs by 1
                cursor.execute(
                    "UPDATE tblTopic SET wrongQs = wrongQs + 1 WHERE
                    tblTopic.topicName = '"
                    + str(question_info[4])
                    + "'"
                )
                resultText = 'Wrong!'
            resultCheck.config(text=resultText)
            if recent_questions.full(): #If queue is full, remove the front question
                recent_questions.get()
            recent_questions.put([question_info[0], resultText]) #Add current
            question to the rear
            recentQuestionsLabel.config(text=recent_questions.queue)

        if question_info[5] == 'Manual - string answer': #If question is manually
        marked, run alternate marking function
            choiceBox = ttk.Combobox(answerFrame, values=['correct','wrong'])
            choiceBox.grid(row=1, column=2)
            choiceButton = tk.Button(answerFrame, text='Submit Choice', command=
            check_choice)
            submitAnswerButton.config(command=wait_for_choice)
            choiceButton.grid(row=1, column=1)

        resultCheck.grid(row=2, column=0)
        conn.commit()
        conn.close()

    def next_question(question_info):
        global first_q, enterLabel, enterAnswer, label_img
        label_img.destroy()

        submitAnswerButton.config(
            text="Mark / Show Answer", command=lambda: show_answer(question_info)
        )
        conn = sqlite3.connect("MathsPlatformStorage.db")
        cursor = conn.cursor()

        resultCheck.config(text="")
        answer.config(text="")
        steps.config(text="")

        topic_query = "SELECT topicName, correctQs, wrongQs FROM tblTopic" #Get all
        topic names
        cursor.execute(topic_query)
        topic_info = cursor.fetchall()
        topic_dict = {}
        ratio_sum = 0
        for x in topic_info:
            ratio = round((1/(x[1] / (x[1] + x[2]))),2)
            ratio_sum += ratio
        for x in topic_info:
            topic_name = x[0]
            ratio = round((1/(x[1] / (x[1] + x[2]))),2)
            topic_dict[topic_name] = ratio / ratio_sum #calculate weighted ratios for
            each topic based on performance in the topic
        topic = (random.choices(list(topic_dict.keys()), weights=topic_dict.values(),
        k=1))[0]
        #Select a topic for next question using weighted keys in random choice
        search_query = ("SELECT qText, qRating, qImage, qMarks, qTopic, qType,
        mSteps, mAns, qNo FROM tblQuestion WHERE qTopic=" + "'" + topic + "' ORDER BY
```

```
            RANDOM() LIMIT 1")
            cursor.execute(search_query) #Get next question details
            question_info = cursor.fetchone()
            if question_info[5] == 'Automark - num answer':

                enterLabel = tk.Label(answerFrame, text="Enter:", font=25)
                enterLabel.grid(row=0, column=0)

                enterAnswer = tk.Entry(answerFrame, font=20)
                enterAnswer.grid(row=0, column=1)

            questionNoText = str(question_info[8]) + "."
            questionText = question_info[0] + " [" + str(question_info[3]) + "]"

            questionTextLabel.config(text=questionText)
            questionNo.config(text=questionNoText)

            print(question_info[2])
            if question_info[2] != "":
                shown_image_path = "images/" + question_info[2]
            else:
                shown_image_path = "images/filler.png"

            image = Image.open(shown_image_path)
            if shown_image_path == "images/filler.png":
                resize_image = image.resize((400,30))
            else:
                resize_image = image.resize((400,400))
            image_obj = ImageTk.PhotoImage(resize_image)
            label_img = tk.Label(questionFrame, image=image_obj)
            label_img.image = image_obj
            label_img.pack()

        def wait_for_choice():
            resultCheck.config(text="Select your answer result first and confirm the
            choice.")

        def check_choice():
            conn = sqlite3.connect('MathsPlatformStorage.db')
            cursor = conn.cursor()
            correctChoice = choiceBox.get()
            if correctChoice == 'correct': #if user selects correct, increase correctQs
            by 1 and score by marks
                cursor.execute(
                    "UPDATE tblTopic SET correctQs = correctQs + 1 WHERE
                    tblTopic.topicName = '"
                    + str(question_info[4])
                    + "'"
                )
                cursor.execute(
                    "UPDATE tblTopic SET score = score + "
                    + str(question_info[3])
                    + " WHERE tblTopic.topicName = '"
                    + str(question_info[4])
                    + "'"
                )
                resultText = 'Correct!'
            else: #if user selects wrong, increase wrongQs by 1
                cursor.execute(
                    "UPDATE tblTopic SET wrongQs = wrongQs + 1 WHERE tblTopic.topicName =
                    '"
                    + str(question_info[4])
                    + "'"
                )
                resultText = 'Wrong'
            resultCheck.config(text=resultText)
            if recent_questions.full(): #if queue full, remove front question
                recent_questions.get()
            recent_questions.put([question_info[0], resultText]) #Add current question to
            rear of queue
            recentQuestionsLabel.config(text=recent_questions.queue)

```

```
404             conn.commit()
405             conn.close()
406             choiceBox.destroy()
407             choiceButton.destroy()
408 
409             submitAnswerButton.config(command=lambda: next_question(question_info))
410 
411         submitAnswerButton = tk.Button(
412             answerFrame,
413             text="Mark / Show Answer",
414             command=lambda: show_answer(question_info),
415         )
416         submitAnswerButton.grid(row=1, column=0, pady=20, padx=50, ipadx=20, ipady=20)
417 
418 
419 
420 
421 
422     def add_image():
423         global file_question_no, choose_file_image_text, image_frame
424         window = tk.Toplevel()
425         window.geometry("700x500")
426         window.focus()
427         image_frame = tk.Frame(window)
428         image_frame.pack()
429 
430         file_question_label = tk.Label(image_frame, text="qNo", font=30)
431         file_question_label.grid(row=0, column=0, padx=20, pady=20)
432 
433         file_question_no = tk.Spinbox(image_frame)
434         file_question_no.grid(row=0, column=1, padx=20, pady=20)
435 
436         choose_file_image_text = tk.Label(image_frame, text="File: ", font=30)
437         choose_file_image_text.grid(row=1, column=0, padx=20, pady=20)
438 
439         choose_file_image_button = tk.Button(
440             image_frame, text="Choose File", command=get_file_path_image
441         )
442         choose_file_image_button.grid(row=1, column=1, padx=20, pady=20, ipady=15, ipadx=
        15)
443 
444         submit_image_button = tk.Button(
445             image_frame, text="Submit", command=submit_add_image
446         )
447         submit_image_button.grid(row=2, column=0, padx=20, pady=20, ipady=15, ipadx=15)
448 
449 
450     def get_file_path_image():
451         global questionNo, original_image_filepath, new_image_filepath
452         original_image_filepath = askopenfilename() #Get filepath for image and display
        name
453         questionNo = file_question_no.get()
454         newText = "File: " + original_image_filepath
455         choose_file_image_text.config(text=newText)
456         new_image_filepath = (
457             "images/" + questionNo + ".png"
458         )
459 
460 
461     def submit_add_image():
462         os.rename(original_image_filepath, new_image_filepath) #Change name and location
        of image file to local images folder
463         choose_file_image_text.config(text="File: ")
464         file_question_no.delete(0, END)
465         submitted_text = "Received qImg for Q" + questionNo
466         submitted = tk.Label(image_frame, text=submitted_text)
467         submitted.grid(row=2, column=1)
468 
469 
470     def add_note():
471         global choose_file_note_text, file_note_name, note_frame, submitted
472         window = tk.Toplevel()
```

```
473         window.geometry("700x500")
474         window.focus()
475         note_frame = tk.Frame(window)
476         note_frame.pack()
477 
478         file_note_label = tk.Label(note_frame, text="New name:", font=30)
479         file_note_label.grid(row=0, column=0, padx=20, pady=20)
480 
481         file_note_name = tk.Entry(note_frame)
482         file_note_name.grid(row=0, column=1, padx=20, pady=20)
483 
484         choose_file_note_text = tk.Label(note_frame, text="File: ", font=30)
485         choose_file_note_text.grid(row=1, column=0, padx=20, pady=20)
486 
487         choose_file_note_button = tk.Button(
488             note_frame, text="Choose File", command=get_file_path_note
489         )
490         choose_file_note_button.grid(row=1, column=1, padx=20, pady=20, ipady=15, ipadx=15
            )
491 
492         submit_note_button = tk.Button(note_frame, text="Submit", command=submit_add_note)
493         submit_note_button.grid(row=2, column=0, padx=20, pady=20, ipady=15, ipadx=15)
494 
495         submitted = tk.Label(note_frame, text='')
496         submitted.grid(row=2, column=1)
497 
498 
499     def get_file_path_note():
500         global original_note_filepath, new_note_filepath
501         original_note_filepath = askopenfilename(defaultextension=".txt") #get filepath
            for text file and display it
502         if original_note_filepath == "":
503             pass
504         elif original_note_filepath[original_note_filepath.index('.'):] != ".txt":
505             submitted.config(text="Error: invalid file type, use a .txt file.")
506             original_note_filepath = ""
507         else:
508             submitted.config(text="")
509             text_file_name = file_note_name.get()
510             newText = "File: " + original_note_filepath
511             choose_file_note_text.config(text=newText)
512             new_note_filepath = (
513                 "notes/" + text_file_name + ".txt"
514             )
515 
516 
517     def submit_add_note():
518         global original_note_filepath, new_note_filepath
519         if file_note_name.get() == "":
520             new_note_filepath = "notes/" + os.path.basename(original_note_filepath)
521         os.rename(original_note_filepath, new_note_filepath) #Change name and location of
            text file to local notes folder
522         choose_file_note_text.config(text="File: ")
523         file_note_name.delete(0, END)
524         submitted.config(text="Received notes.")
525         original_note_filepath = ""
526 
527 
528 
529     def add_topic_menu():
530         global topic_name_entry, submitted_label
531         window = tk.Toplevel()
532         window.geometry("700x500")
533         window.focus()
534         topic_frame = tk.Frame(window)
535         topic_frame.pack()
536 
537         topic_details_frame = tk.LabelFrame(topic_frame, text="Topic Details")
538         topic_details_frame.grid(row=0, column=0, ipady=10, ipadx=10)
539 
540         topic_name_label = tk.Label(topic_details_frame, text="topicName")
541         topic_name_label.grid(row=0, column=0, pady=10, padx=10)
```

```
542
543         topic_name_entry = tk.Entry(topic_details_frame)
544         topic_name_entry.grid(row=0, column=1, pady=10, padx=10)
545
546         submit_frame = tk.LabelFrame(topic_frame, text="Submit")
547         submit_frame.grid(row=1, column=0)
548
549         topic_submit_button = tk.Button(
550             submit_frame, text="Submit Topic", command=add_topic
551         )
552         topic_submit_button.grid(row=0, column=0, pady=10, padx=10)
553         submitted_label = tk.Label(submit_frame, text='Submitted Topic: ')
554         submitted_label.grid(row=0, column=1, pady=10, padx=10)
555
556
557     def add_topic():
558         topicName = topic_name_entry.get()
559         if len(topicName) < 4 or len(topicName) > 40: #Make sure topic name is between 4
            and 40 characters (inclusive)
560             submitted_label.config(text="Topic Name must be between 4 and 40 characters
                (inclusive)")
561         else:
562             submitted_text = 'Submitted Topic: ' + topicName
563             submitted_label.config(text=submitted_text)
564             conn = sqlite3.connect("MathsPlatformStorage.db")
565             cursor = conn.cursor()
566
567             count_query = "SELECT COUNT(*) FROM tblTopic"
568             cursor.execute(count_query)
569             result = cursor.fetchone()
570             totalTopics = result[0] #Get the number of topics in tblTopic
571             topicNo = totalTopics + 1
572
573             insert_topic_query = """INSERT INTO tblTopic(topicNo, topicName, correctQs,
                wrongQs, score)
574             VALUES (?, ?, ?, ?, ?)"""
575             insert_topic_tuple = (topicNo, topicName, 1, 1, 1)
576             cursor.execute(insert_topic_query, insert_topic_tuple) #Insert new topic with
                performance values into tblTopic
577             conn.commit()
578             conn.close()
579
580
581     def view_progress_menu():
582         global topic_options, correct_label, topic_label, wrong_label, score_label, topics
583         conn = sqlite3.connect('MathsPlatformStorage.db')
584         cursor = conn.cursor()
585         cursor.row_factory = lambda cursor, row: row[0]
586         topics = cursor.execute('SELECT topicName FROM tblTopic').fetchall() #get topics
            list
587         conn.close()
588
589         window = tk.Toplevel()
590         window.geometry("700x500")
591         window.focus()
592         progress_frame = tk.Frame(window)
593         progress_frame.pack()
594
595         choose_topic_frame = tk.LabelFrame(progress_frame)
596         choose_topic_frame.grid(row=0, column=0)
597
598         topic_progress_frame = tk.LabelFrame(progress_frame)
599         topic_progress_frame.grid(row=1, column=0)
600
601         topic_options = ttk.Combobox(choose_topic_frame, values=topics)
602         topic_options.grid(row=0, column=0, padx=10, pady=10)
603
604         view_button = tk.Button(choose_topic_frame, text='View Progress', command=
            get_progress)
605         view_button.grid(row=1, column=0, padx=10, pady=10)
606
607         topic_label = tk.Label(topic_progress_frame, text='Topic: ',font=16)
```

```
608         topic_label.grid(row=0, column=0, padx=10, pady=10)
609         correct_label = tk.Label(topic_progress_frame, text='Correct answers: ',font=16)
610         correct_label.grid(row=1, column=0, padx=10, pady=10)
611         wrong_label = tk.Label(topic_progress_frame, text='Wrong answers: ',font=16)
612         wrong_label.grid(row=2, column=0, padx=10, pady=10)
613         score_label = tk.Label(topic_progress_frame, text='Total score: ',font=16)
614         score_label.grid(row=3, column=0, padx=10, pady=10)
615 
616 
617 
618     def get_progress():
619         topic = topic_options.get()
620         topic_label.config(text=('Topic: '+topic))
621         if topic == "": #Account for the blank input
622             correct_label.config(text="Enter a valid topic.")
623         elif topic not in topics: #Make sure topic inputted exists
624             correct_label.config(text="Topic doesn't exist.")
625             wrong_label.config(text="You can add it with the Add Topic menu")
626             score_label.config(text="Go to the File submenu in the navigation bar")
627 
628         else:
629             conn = sqlite3.connect('MathsPlatformStorage.db')
630             cursor = conn.cursor()
631             topic_scores_statement = 'SELECT correctQs, wrongQs, score FROM tblTopic
                WHERE topicName = "' + topic + '"' #Get performance stats for the topic
                inputted
632             cursor.execute(topic_scores_statement)
633             topic_scores = cursor.fetchone()
634             conn.close()
635 
636             correct_label.config(text=('Correct answers: '+str(topic_scores[0])))
637             wrong_label.config(text=('Wrong answers: '+str(topic_scores[1])))
638             score_label.config(text=('Total score: '+str(topic_scores[2])))
639 
640     def view_notes_menu():
641         global notes_list, note_textbox, notes, note_info
642         window = tk.Toplevel()
643         window.geometry("900x600")
644         window.focus()
645 
646         notes_list = tk.Listbox(window, selectmode=SINGLE, font=20)
647         notes = os.listdir('notes') #Get all the text file names from notes folder
648         for x in notes:
649             notes_list.insert(END, x)
650         notes_list.pack(side = RIGHT, fill=BOTH)
651 
652         scrollbar = tk.Scrollbar(window)
653         scrollbar.pack(side=RIGHT, fill=BOTH)
654         notes_list.config(yscrollcommand=scrollbar.set)
655         scrollbar.config(command=notes_list.yview)
656 
657         view = tk.Button(window, text='Load Note', command=view_note)
658         view.pack(pady=15)
659         note_textbox = tk.Text(window)
660         note_textbox.pack()
661         save = tk.Button(window, text='Save Note', command=save_note)
662         save.pack(pady=15)
663         note_info = tk.Label(window, text='', font=20)
664         note_info.pack()
665 
666     def view_note():
667         global directory, note_textbox
668         if notes_list.curselection() == ():
669             note_info.config(text="Error: no file input.") #Make sure a file is selected
                when loading a note
670         else:
671             index_selected = int((notes_list.curselection())[0])
672             note_selected = notes[index_selected]
673             if note_selected[note_selected.index('.'):] != '.txt':
674                 note_info.config(text="Error: file type not valid.")
675             else:
676                 directory = "notes/" + note_selected
```

```
677                 x = open(directory, "r") #Open the text file
678 
679                 text = x.read() #Get the file contents
680                 x.close()
681                 text = textwrap.fill(text,80)
682                 note_textbox.delete(1.0, END)
683                 note_textbox.insert(END, text) #Write the text to the input box
684                 note_info_text = "Note loaded: " + note_selected
685                 note_info.config(text=note_info_text)
686 
687     def save_note():
688         new_text = note_textbox.get("1.0", "end-1c")
689         x = open(directory, "w")
690         try:
691             x.write(new_text) #Write inputted text to the file and save it
692             x.close()
693             note_info_text = "Note saved: " + directory[6:]
694             note_info.config(text=note_info_text)
695         except:
696             note_info.config(text="Error: enter valid characters.") #Account for invalid
                character inputs
697 
698     def delete_question_menu():
699         global question_number_entry, deleted_label, questions_list
700         window = tk.Toplevel()
701         window.geometry("900x600")
702         window.focus()
703 
704         conn = sqlite3.connect('MathsPlatformStorage.db')
705         cursor = conn.cursor()
706 
707         questions_list = tk.Listbox(window, selectmode=SINGLE, font=20)
708         cursor.execute('SELECT qNo, qText FROM tblQuestion')
709         questions = cursor.fetchall()
710         for x in range(len(questions)):
711             questions_list.insert(END, str(questions[x][0]) + ". " + questions[x][1])
712         questions_list.pack(side=RIGHT, fill=BOTH, ipadx=100)
713 
714         conn.close()
715 
716         scrollbar = tk.Scrollbar(window)
717         scrollbar.pack(side=RIGHT, fill=BOTH)
718         questions_list.config(yscrollcommand=scrollbar.set)
719         scrollbar.config(command=questions_list.yview)
720 
721         question_frame = tk.Frame(window)
722         question_frame.pack()
723 
724         question_delete_frame = tk.LabelFrame(question_frame, text="Delete Question")
725         question_delete_frame.grid(row=0, column=0, ipady=10, ipadx=10)
726 
727         question_number_label = tk.Label(question_delete_frame, text="Question Number")
728         question_number_label.grid(row=0, column=0, pady=10, padx=10)
729 
730         question_number_entry = tk.Entry(question_delete_frame)
731         question_number_entry.grid(row=0, column=1, pady=10, padx=10)
732 
733         delete_frame = tk.LabelFrame(question_frame, text="Confirm")
734         delete_frame.grid(row=1, column=0)
735 
736         question_delete_button = tk.Button(
737             delete_frame, text="Delete", command=delete_question
738         )
739         question_delete_button.grid(row=0, column=0, pady=10, padx=10)
740         deleted_label = tk.Label(delete_frame, text='Deleted Question: ')
741         deleted_label.grid(row=0, column=1, pady=10, padx=10)
742 
743     def delete_question():
744         question_num_delete = int(question_number_entry.get())
745         conn = sqlite3.connect('MathsPlatformStorage.db')
746         cursor = conn.cursor()
747 
```

```
748         count_query = "SELECT MAX(qNo) FROM tblQuestion" #Get the largest question number
749         cursor.execute(count_query)
750         result = cursor.fetchone()
751         largest_qNo = result[0]
752 
753         if question_num_delete > 0 and question_num_delete <= largest_qNo:
754             cursor.execute('DELETE FROM tblQuestion WHERE qNo='+str(question_num_delete))
755             deleted_label.config(text=deleted_label_text)
756             deleted_label_text = "Deleted Question: " + str(question_num_delete)
757             questions_list.delete(0, tk.END)
758             conn.commit()
759 
760             if question_num_delete != largest_qNo: #When a question is deleted, adjust
                the questions with a higher qNo to be accurate
761                 for x in range(question_num_delete+1, largest_qNo+1):
762                     cursor.execute('UPDATE tblQuestion SET qNo='+str(x-1)+' WHERE qNo='+
                        str(x))
763             conn.commit()
764 
765             cursor.execute('SELECT qNo, qText FROM tblQuestion')
766             questions = cursor.fetchall()
767             for x in range(len(questions)):
768                 questions_list.insert(END, str(questions[x][0]) + ". " + questions[x][1])
                    #Display the new set of questions
769 
770         else:
771             deleted_label.config(text='Deleted Question: Invalid qNo')
772         conn.close()
773         question_number_entry.delete(0, tk.END)
774 
775 
776 
777     def play_game():
778         exe_path = 'jimmy.exe'
779         subprocess.run([exe_path])
780 
781     window = tk.Tk()
782     window.geometry("700x500")
783     window.title("Maths Revision Tool")
784 
785     menubar = tk.Menu(window)
786     window.config(menu=menubar)
787     operationMenu = tk.Menu(menubar, tearoff="off")
788     gameMenu = tk.Menu(menubar, tearoff="off")
789 
790     menubar.add_cascade(label="File", menu=operationMenu)
791     menubar.add_cascade(label="Game", menu=gameMenu)
792 
793     operationMenu.add_command(label="Add Question", command=add_question_menu)
794     operationMenu.add_separator()
795     operationMenu.add_command(label="Add Notes (.txt)", command=add_note)
796     operationMenu.add_separator()
797     operationMenu.add_command(label="Add Image (.jpeg/.png)", command=add_image)
798     operationMenu.add_separator()
799     operationMenu.add_command(label="Add Topic", command=add_topic_menu)
800     operationMenu.add_separator()
801     operationMenu.add_command(label="Delete Question", command=delete_question_menu)
802 
803     gameMenu.add_command(label="Play Game", command=play_game)
804 
805     main_frame = tk.Frame(window)
806     main_frame.pack()
807 
808     title = tk.Label(main_frame, text="Maths Revision Platform", font=40)
809     title.grid(row=0, column=0)
810 
811     info_label = tk.Label(
812         main_frame,
813         text="View the File dropdown menu in the top left for additional features",
814         font=25,
815     )
816     info_label.grid(row=4, column=0)
```

```
817
818 answer_question_button = tk.Button(
819     main_frame, text="Answer Question", command=answer_question_menu
820 )
821 answer_question_button.grid(row=1, column=0)
822
823 view_progress_button = tk.Button(main_frame, text="View Topic Progress", command=
    view_progress_menu)
824 view_progress_button.grid(row=2, column=0)
825
826 view_notes_button = tk.Button(main_frame, text="View/Edit Notes", command=
    view_notes_menu)
827 view_notes_button.grid(row=3, column=0)
828
829 for widget in main_frame.winfo_children():
830     widget.grid_configure(padx=20, pady=14, ipadx=15, ipady=15)
831
832 window.mainloop()
833
```

```gdscript
extends AudioStreamPlayer #audio_stream_player_2d script

var jump_sound_file = preload("res://sfx/jump.mp3")
var next_level_sound_file = preload("res://sfx/next_level.mp3")
@onready var sfx = $"."

# Called when the node enters the scene tree for the first time.
func _ready():
    pass # Replace with function body.


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(_delta):
    pass

func jump_sound(): #the function is called when the user jumps (presses spacebar)
    sfx.stream = jump_sound_file
    sfx.play() #Play the jump sound effect

extends Area2D #bubble powerup script
@onready var bubble_text = $RichTextLabel
@onready var coin_manager = $"../../coin_manager"
@onready var coin_count = $"../../CanvasLayer/coin_count"
@onready var sprite = $Sprite2D

func generate_values():
    var operation = (['+','-','*','/']).pick_random()
    var rng = RandomNumberGenerator.new()
    var value = rng.randi_range(1,20)
    var calculation = operation + str(value)
    bubble_text.text = calculation
    if operation == '-' or operation == '/':
        sprite.modulate = Color.from_hsv(345, 100, 45)



func _on_body_entered(body):
    var calculation = bubble_text.text
    var expression = Expression.new()
    expression.parse(str(coin_manager.coins) + calculation)
    var result = expression.execute()
    coin_manager.coins = result
    coin_count.text = str(coin_manager.coins)
    queue_free()



func _on_ready():
    generate_values()

extends Button #button script

@onready var line_edit = $"../CenterContainer/LineEdit"
@onready var label_2 = $"../Label2"

# Called when the node enters the scene tree for the first time.
func _ready():
    pass # Replace with function body.


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(_delta):
    pass


func _on_pressed():
    if len(line_edit.text) > 2 and len(line_edit.text) < 17:
        button_manager.player_name = line_edit.text
        get_tree().change_scene_to_file("res://scenes/menu2.tscn")
    else:
        label_2.text = "Name too short/long (3 char min, 16 char max)"

extends Area2D #coin script
```

```
@onready var coin_count = $"../../CanvasLayer/coin_count"
@onready var coin_manager = $"../../coin_manager"

func _on_body_entered(_body):
    coin_manager.coins += 1
    coin_count.text = str(coin_manager.coins)
    queue_free()


extends Node2D #coin_manager script

var coins = 0

extends Area2D #enemy script

var run_speed = 33
@onready var jimmy = $"../jimmy"
@onready var global_timer = %GlobalTimer

func _process(delta):
    position += position.direction_to(jimmy.position) * run_speed


func _on_body_entered(body):
    if body.name == "jimmy":
        global_timer.get_time_formatted()
        get_tree().call_deferred("reload_current_scene")

extends Node #global script


func _ready():
    SilentWolf.configure({
        "api_key": "5qu1itZArVaVxhULVq7Gi1krhFUWOIvy9PWFD4ru",
        "game_id": "jimmy",
        "log_level": 1
    })

    SilentWolf.configure_scores({
        "open_scene_on_close": "res://scenes/menu.tscn"
    })

func save_score(name, time, lb):
    SilentWolf.Scores.save_score(str(name), 1+(1 / time), lb)

extends Label #high_scores script
@onready var high_scores = $"."
@onready var leaderboard = $".."


# Called when the node enters the scene tree for the first time.
func _ready():
    var sw_result: Dictionary = await SilentWolf.Scores.get_scores(0, "1").
    sw_get_scores_complete
    high_scores.text = "Scores: " + str(sw_result.scores)


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
    pass

extends CharacterBody2D #jimmy (player) script

var SPEED = 1200.0 * time_manager.multiplier
var JUMP_VELOCITY = -1500.0 * time_manager.multiplier
@onready var sprite = $AnimatedSprite2D
@export var coyote_time = 0.1
var jump_available = true
var jump_buffer = false
var jump_buffer_time = 0.1
@onready var sfx = $"../sfx"
var accel = 20
@onready var global_timer = %GlobalTimer
```

```
146
147 func _physics_process(delta):
148     if Input.is_action_pressed("menu"):
149         time_manager.time = 0
150         time_manager.multiplier = 1
151         get_tree().change_scene_to_file("res://scenes/menu2.tscn")
152     #if Input.is_action_pressed("reset"):
153         #global_timer.get_time_formatted()
154         #get_tree().call_deferred("reload_current_scene")
155     if not is_on_floor() and not is_on_wall():
156         if jump_available:
157             get_tree().create_timer(coyote_time).timeout.connect(coyote_timeout)
158         sprite.play("jump")
159         velocity += get_gravity() * delta * 3
160         if velocity.y > 0:
161             velocity += get_gravity() * delta * 1.5
162         elif velocity.y < 0 and not Input.is_action_pressed("jump"):
163             velocity += get_gravity() * delta * 1
164     else:
165         jump_available = true
166         if jump_buffer == true:
167             jump()
168             jump_buffer = false
169     if is_on_wall_only():
170         velocity.y += 220 * delta * 3
171     if is_on_wall():
172         if Input.is_action_pressed("move_right"):
173             sprite.flip_h = false
174             sprite.play("climb")
175         elif Input.is_action_pressed("move_left"):
176             sprite.flip_h = true
177             sprite.play("climb")
178
179
180     if Input.is_action_pressed("dash") and not is_on_wall():
181         SPEED = 1600.0 * time_manager.multiplier
182         if Input.is_action_pressed("move_left"):
183             sprite.play("run")
184             sprite.flip_h = true
185         elif Input.is_action_pressed("move_right"):
186             sprite.play("run")
187             sprite.flip_h = false
188     if Input.is_action_just_released("dash"):
189         SPEED = 1200.0 * time_manager.multiplier
190
191     if Input.is_action_pressed("move_left") and is_on_floor() and not Input.
    is_action_pressed("dash"):
192         sprite.play("walk")
193         sprite.flip_h = true
194     if Input.is_action_pressed("move_right") and is_on_floor() and not Input.
    is_action_pressed("dash"):
195         sprite.play("walk")
196         sprite.flip_h = false
197     if not Input.is_action_pressed("move_right") and not Input.is_action_pressed(
    "move_left") and is_on_floor():
198         sprite.play("idle")
199     if Input.is_action_just_pressed("jump"):
200         if jump_available:
201             if not sfx.playing:
202                 sfx.stop()
203                 sfx.jump_sound()
204             jump()
205         else:
206             jump_buffer = true
207             get_tree().create_timer(jump_buffer_time).timeout.connect(
                on_jump_buffer_timeout)
208
209     var direction = Input.get_axis("move_left", "move_right")
210     if direction:
211         velocity.x = move_toward(velocity.x, direction * SPEED, 150.0)
212     else:
213         #if sprite.flip_h == false and velocity.x > 0:
214             #velocity.x -= accel
```

```
215             #if sprite.flip_h == true and velocity.x < 0:
216                 #velocity.x += accel
217             velocity.x = move_toward(velocity.x, 0, SPEED)
218         move_and_slide()
219 
220     func jump()->void:
221         if is_on_wall():
222             velocity.y = JUMP_VELOCITY * 1.3
223             velocity.x += 900 * get_wall_normal().x
224         else:
225             velocity.y = JUMP_VELOCITY
226         jump_available = false
227 
228     func coyote_timeout():
229         jump_available = false
230 
231     func on_jump_buffer_timeout()->void:
232         jump_buffer = false
233 
234     extends Area2D #killzone script
235     @onready var global_timer = %GlobalTimer
236 
237     func _on_body_entered(_body):
238         global_timer.get_time_formatted()
239         get_tree().call_deferred("reload_current_scene")
240 
241     extends Panel #panel for leaderboards script
242 
243     func _on_lb_1_pressed():
244         var sw_result: Dictionary = await SilentWolf.Scores.get_scores(8).
        sw_get_scores_complete
245         get_tree().change_scene_to_file(
        "res://addons/silent_wolf/examples/CustomLeaderboards/ReverseLeaderboard.tscn")
246 
247 
248     func _on_lb_2_pressed():
249         var sw_result: Dictionary = await SilentWolf.Scores.get_scores(8, "world2").
        sw_get_scores_complete
250         get_tree().change_scene_to_file(
        "res://addons/silent_wolf/examples/CustomLeaderboards/W2ReverseLeaderboard.tscn")
251 
252     extends Label #most_recent_time script
253     @onready var label_2 = $"."
254 
255     # Called when the node enters the scene tree for the first time.
256     func _ready():
257         label_2.text = "Most Recent Time: " + str("%.3f" % time_manager.most_recent_time)
258 
259 
260     # Called every frame. 'delta' is the elapsed time since the previous frame.
261     func _process(_delta):
262         pass
263 
264     extends Area2D #next_level script
265     @onready var coin_manager = $"../coin_manager"
266     @onready var coin_count = $"../CanvasLayer/coin_count"
267     @onready var sfx = $"../sfx"
268     @onready var global_timer = %GlobalTimer
269 
270     func _on_body_entered(_body):
271         var current_scene = get_tree().current_scene.name
272         var current_level_number = int(current_scene.get_slice("_", 1))
273         var new_level_number = current_level_number + 1
274         if new_level_number == 12:
275             time_manager.multiplier = 1.5
276         else:
277             time_manager.multiplier = 1
278         global_timer.get_time_formatted()
279         if new_level_number == 7:
280             Global.save_score(button_manager.player_name, time_manager.time, "main")
281         elif new_level_number == 13:
282             Global.save_score(button_manager.player_name, time_manager.time, "world2")
283         if new_level_number != 7 and new_level_number != 13:
```

```
            var new_level_path = "res://scenes/level_" + str(new_level_number) + ".tscn"
            get_tree().call_deferred("change_scene_to_file",new_level_path)
        else:
            time_manager.most_recent_time = time_manager.time
            time_manager.time = 0
            time_manager.multiplier = 1
            get_tree().call_deferred("change_scene_to_file","res://scenes/menu2.tscn")
        coin_manager.coins = 0
        coin_count.text = str(coin_manager.coins)

extends Panel #panel for main menu script

# Called when the node enters the scene tree for the first time.
func _ready():
    pass


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(_delta):
    pass


func _on_start_pressed():
    get_tree().change_scene_to_file("res://scenes/worlds.tscn")


func _on_leaderboards_pressed():
    get_tree().change_scene_to_file("res://scenes/leaderboards.tscn")


func _on_player_name_menu_pressed():
    get_tree().change_scene_to_file("res://scenes/menu.tscn")

extends Button #refresh_leaderboard script

# Called when the node enters the scene tree for the first time.
func _ready():
    pass # Replace with function body.


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
    pass


func _on_pressed():
    var current_scene = get_tree().current_scene.name
    if current_scene == "ReverseLeaderboard":
        var sw_result: Dictionary = await SilentWolf.Scores.get_scores(8).
        sw_get_scores_complete
        get_tree().change_scene_to_file(
        "res://addons/silent_wolf/examples/CustomLeaderboards/ReverseLeaderboard.tscn"
        )
    elif current_scene == "W2ReverseLeaderboard":
        var sw_result: Dictionary = await SilentWolf.Scores.get_scores(8, "world2").
        sw_get_scores_complete
        get_tree().change_scene_to_file(
        "res://addons/silent_wolf/examples/CustomLeaderboards/W2ReverseLeaderboard.tsc
        n")

extends Panel #timer script

var time = 0
var minutes = 0
var seconds = 0
var milliseconds = 0

@onready var tminutes = %Minutes
@onready var tseconds = %Seconds
@onready var tmilliseconds = %Milliseconds
@onready var star = $"../../star"

# Called when the node enters the scene tree for the first time.
```

```
func _ready():
    pass # Replace with function body

# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
    time+= delta
    milliseconds = fmod(time,1) * 100
    seconds = fmod(time, 60)
    minutes = fmod(time, 3600) / 60
    tminutes.text = "%02d:" % minutes
    tseconds.text = "%02d." % seconds
    tmilliseconds.text = "%02d" % milliseconds

func stop() -> void:
    set_process(false)

func get_time_formatted() -> String:
    time_manager.time += time
    return "%02d:%02d.%02d" % [minutes, seconds, milliseconds]

extends Node2D #time_manager script

var time = 0

var most_recent_time = 0

var stack_level_time = 0

var multiplier = 1

extends Panel #panel for worlds menu script


# Called when the node enters the scene tree for the first time.
func _ready():
    pass # Replace with function body.


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
    pass



func _on_world_1_pressed():
    get_tree().change_scene_to_file("res://scenes/level_1.tscn")




func _on_world_2_pressed():
    get_tree().change_scene_to_file("res://scenes/level_7.tscn")




func _on_world_3_pressed():
    pass

extends Label #your_best_time script
@onready var label_3 = $"."


# Called when the node enters the scene tree for the first time.
func _ready():
    pass


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(_delta):
    pass

func load_best_time():
```

```
424         var sw_result = await SilentWolf.Scores.get_top_score_by_player(button_manager.
            player_name,0,"main").sw_top_player_score_complete
425         var time = 1 / (sw_result.top_score.score - 1)
426         var time_formatted = "%.3f" % time
427         label_3.text = button_manager.player_name + "'s Best Time: " + str(time_formatted)
428
429
430     func _on_refresh_pressed():
431         load_best_time()
432
```