COMPUTER SCIENCE NEA

Quantum Computation Simulator



Tom Piercy (5514)

City of Norwich School (18133)

***Contents***

[***Analysis*** 4](#_Toc153327243)

[**Background to the problem** 4](#_Toc153327244)

[**Current solution** 4](#_Toc153327245)

[**Client interview** 5](#_Toc153327246)

[**Intended User and Prerequisite knowledge:** 6](#_Toc153327247)

[**User needs and project limitations:** 6](#_Toc153327248)

[**Data sources and destinations** 7](#_Toc153327249)

[**Data dictionary** 8](#_Toc153327250)

[**Data flow diagram** 9](#_Toc153327251)

[**Entity relationship diagram** 11](#_Toc153327252)

[**Object orientation plan** 12](#_Toc153327253)

[**Project goals, objectives and KPIs** 12](#_Toc153327254)

[**Potential solutions** 12](#_Toc153327255)

[**Chosen solution** 12](#_Toc153327256)

[***Design*** 13](#_Toc153327257)

[**Database design** 13](#_Toc153327258)

[**Initial solution** 13](#_Toc153327259)

[**Final solution** 13](#_Toc153327260)

[**IPSO chart** 13](#_Toc153327261)

[**Sample of planned SQL queries** 13](#_Toc153327262)

[**Validating user input** 14](#_Toc153327263)

[**Regular expressions** 14](#_Toc153327264)

[**SQL injection and system security** 16](#_Toc153327265)

[**Error handling** 16](#_Toc153327266)

[**Algorithm research** 17](#_Toc153327267)

[**Quantum algorithms** 17](#_Toc153327268)

[**Classical algorithms** 17](#_Toc153327269)

[**Algorithm Design** 19](#_Toc153327270)

[**Class definitions** 20](#_Toc153327271)

[**Project hierarchy chart** 20](#_Toc153327272)

[**UI/UX design** 20](#_Toc153327273)

[**Project directory layout** 24](#_Toc153327274)

[**Initial tree** 24](#_Toc153327275)

[**Final tree** 24](#_Toc153327276)

[***Technical solution*** 25](#_Toc153327277)

[**Database implementation** 25](#_Toc153327278)

[**UI implementation** 25](#_Toc153327279)

[**Simulator implementation** 25](#_Toc153327280)

[***Testing*** 26](#_Toc153327281)

[**Trace tables** 26](#_Toc153327282)

[**Test tables** 26](#_Toc153327283)

[**Code iterations** 26](#_Toc153327284)

[**UX testing** 29](#_Toc153327285)

[***Evaluation*** 30](#_Toc153327286)

[**Achieved solution versus project objectives** 30](#_Toc153327287)

[**Client feedback** 30](#_Toc153327288)

[**Independent feedback** 30](#_Toc153327289)

[**Scope for further development** 30](#_Toc153327290)

[***Appendix*** 31](#_Toc153327291)

# ***Analysis***

## **Background to the problem**

Mark, a first-year undergraduate student, is studying the optional module “Introduction to quantum computation” but is struggling to understand some of the key principles he is coming across. He would like a sandbox learning resource which balances teaching content with practical exercises so that he can get some experience working with the concepts he is learning. Additionally, he doesn’t understand when and where some of the knowledge he is gathering is used in real life and would like some exposure to the different applications of quantum computation.

Some definitions are provided to aid understanding of the project:

* A simulator is a piece of software that aims to model a physical system.
* Quantum computation refers to calculations performed using quantum states as a basis.
* A quantum computer is a computer that uses quantum states as opposed to standard classical bits to perform calculations.

Quantum computer simulators will naturally not perfectly reflect an actual quantum computer. Whilst this is fine for implementing most algorithms and explaining principles, it does mean that these simulators are often noiseless, stable and run over a large memory space.

## **Current solution**

Currently, Mark only has two places he can look for help: he can ask his lecturers, who are always very busy and inaccessible, or he can look on the internet. When Mark searches for the questions he wants answers to, he is greeted by a screen full of symbols and academic articles far too advanced for him to understand. Good introductory texts and videos can be found but they are spread across multiple sites and are hard to compile. Additionally, none of these resources are interactive so Mark has no real world understanding of what he is reading which is quite a mental block for him. An example of one such pre-existing resource is <https://quantum-computing.ibm.com/>



## **Client interview**

Before I began the project, I had the opportunity to talk to Mark about how he felt about his studies at the moment. The transcript is given below:

**Tom: “What issues are you facing at the moment with finding online resources?”**

**Mark:** “I think it's really hard to actually find stuff online as it is aimed towards people who have a deeper understanding than me. I really like videos because I understand them the best, but they don't really give me a chance to practice or consolidate anything. Loads of the websites I've found are from the early 90s and are running with the UI of a potato. I don't like this because I don't want to stay on the websites if they look bad.”

**Tom: “Are there any good parts to this process?”**

**Mark:** “There is loads of content online which is good in theory, but I am really overwhelmed by it, and I don't want to navigate it myself. There are probably some really good resources online but not for exactly what I need. I really want just one resource that can do everything.”

**Tom: “If you could improve any part of the current process of gathering resources, what would it be?”**

**Mark:** “As I mentioned earlier, there is stuff online but just way too much and I don't want to gather them, I just want them all available to me.”

**Learning points of the interview:**

1. Mark needs some form of resource collation as well as referring links so that if he finds a topic that particularly interests him, he can research it further.
2. Interactivity is a big thing for my client, he needs to be able to apply the concepts that he learns otherwise they won't stick in his head.
3. My client would like a teaching resource that can expose him to lots of different topics and what he needs to look for to explore them further.
4. The program solution should look nice since he will be on it for extended periods of time.
5. As an extension to 4, the program should be simple to learn to use and quick to pick up.
6. The program needs to be fun, otherwise my client won’t be inclined to learn from it.
7. Finally, the program needs to provide incentive to continue learning but not pressure the user in any way.

## **Intended User and Prerequisite knowledge:**

Whilst the project is being designed with Mark in mind (the primary client), the new system will be available to anyone who wants to increase their knowledge in the area of quantum computation. The program will not be bespoke or personalized to Mark in any way and will allow for multiple users on the same device to work through the lessons and play on the simulator independently of each other.

In terms of prerequisites, the only thing that will be required to use the software is a willingness to learn. The lesson design assumes no prior knowledge in the field. Some proficiency with technology would be helpful so as to not limit the software’s functionality; however, it is not required since the system will be designed to be as fluid and intuitive as possible. I will implement assertion statements and exception blocks to elegantly handle errors so that the user doesn’t get overwhelmed and can see clearly what error happened, why it happened and how the user should (if they can) fix it.

Despite their being one primary client, the concept of points and achievements create incentive for Mark to share the program with his friends and compete with them in a friendly manner - thus it is important to have isolated user “accounts” so that anyone using the software can easily see their own personal stats as well as how they rank up to others.

## **User needs and project limitations:**

**Client requirements:**

1. The program is fun, simple and quick to pick up.
2. The program incentivises learning and competing against others.
3. The program is aesthetically pleasing.
4. The program is fully interactive.
5. The program is a springboard to further studies.

**Project limitations:**

1. Time: The project is time bounded to February 2024
2. Knowledge: My programming experience is primarily in Python so this is the language that I will be using for my project. Whilst there may be more suitable languages for my solution, using python allows me to easily implement various programming paradigms, such as OOP.
3. The Python language: Python comes built-in with lots of useful modules that I can use to easily connect various parts of my program (such as modules for managing GUIs and databases) as well as a large community presence that can provide support with module implementation. However, Standalone Python is not very good at making web or mobile applications, so this does raise some limitations for me.

## **Data sources and destinations**

**Current system**

|  |  |  |
| --- | --- | --- |
| **Description** | **Source** | **Destination** |
| Academic papers | Internet/web archive from user input search query | Hard copy printout or digital file |
| Videos | Video archives on streaming sites (such as “YouTube”) from user query | Playback in browser |

**Proposed system**

|  |  |  |
| --- | --- | --- |
| **Description** | **Source** | **Destination** |
| User login information | Input (creating, logging into an account) | Users DB table |
| Player score/high score/ achievements | User completing an action inside the program | Scores DB table |
| Saved results | Saved csv file | History DB table |

## **Data dictionary**

**Database**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field name** | **Field purpose** | **Data type** | **Field size** | **Example** |
| Username | Uniquely identifies a user of the program (not a primary key though) | Unique string, varchar | <= 16 characters | “Markiplier” |
| Password | Hash of a user’s password that links to their username and grants access to features locked behind accounts | String - **[A-Z,a-z,0-9]{64}** | 64 characters | “2fd83795b2f270fc3be2d4bdf16674045ad5e2b1453998a020e305ba5c0c45ab” |
| UserID | Primary key of users table | Auto Integer primary key (>= 0) | 4 bytes | 1 |
| Highscore | Total score of user | Integer (>= 0) | 4 bytes | 10000 |
| Difficulty | Describes the difficulty of a level which is used to calculate points | integer (>= 0) | 0-99 (2-digit number) | 99 |
| ChallengeID | Uniquely identifies an achievement that the user can receive | Auto integer primary key (>=0) | 4 bytes | 3 |
| challengetext | Describes an achievement, the text that the user will see on completion | String, text | < 2^16 Characters | “Successfully entangle two qubits” |
| Reward | The number of points the user receives for completing a challenge | Integer (>=0) | < 6 characters | 555 |
| date | The current date as recorded when completing certain actions | String, datetime | 19 characters | “1000-01-01 00:00:00” |

**Program**

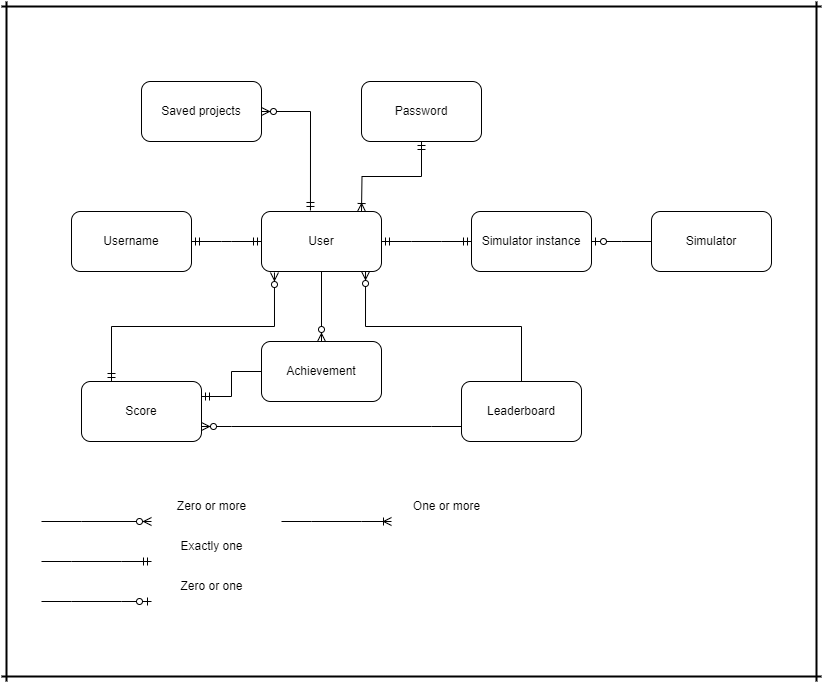
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Location** | **Variable name** | **Data type** | **Maximum size/value** | **Start value** | **Description** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## **Data flow diagram**

Towards the beginning of my project, I decided to make some basic data flow diagrams. These are heavily abstracted as rather than detailing everything I decided to just use them to gain a basic understanding of what I need to do, and how I could decompose the problem. They do not show all connections or even what happens in the subroutines. They don’t show what inputs are being taken or what decisions are being made: they are purely designed to show where these routines may be run and where inputs may be taken. I will use these diagrams to make, test and improve my final system.

## **Entity relationship diagram**

If I decide that databases are the right fit for my project, then the entities in my program could have the following relationships:

Every user has to have exactly one unique username, although multiple users can have the same password. A user is required to have a password. Users can, if they so wish, have a collection in the saved projects database linking to their files but they don’t have to - however if they choose to do this then every project or collection of projects must be attributed to one user id only. The leaderboard is set up when the first user signs up (as the score is a required property of every user this means that even when no points have been earned the leaderboard will show 0). Additionally, the user can get a predetermined number of points for completing achievements although there is the possibility that the user never completes any of these so this is a weak entity.

To implement this database, I would most likely use SQLite3 as it is built into python and provides a vast array of supporting documentation as well as functioning exactly as any other dedicated SQL controller. What this means is that using SQLite gives me the freedom to implement commands in a language that I am already familiar in (the database would be controlled and operated in python) but when it comes to querying the database, I can use the extensive and powerful instruction set of SQL - giving me the best of both worlds. Finally, there are lots of GUIs for viewing SQLite data, giving me the ability to see the database intuitively. This would rapidly speed up development as I wouldn’t need to write a structured command to debug or test a small part of the database and would also make tracking errors easier if there were any. Since SQLite3 is the most popular SQL python module alongside the fact that it is built in means that there doesn’t seem to be any worry of it becoming deprecated any time soon- which means that my program will still be able to function in the future.

## **Object orientation plan**

## **Project goals, objectives and KPIs**

## **Potential solutions**

## **Chosen solution**

# ***Design***

## **Database design**

### **Initial solution**

### **Final solution**

## **IPSO chart**

|  |  |
| --- | --- |
| INPUT   * Username * Password * Single line commands | PROCESS   * Verify user login details/register user * Store/Load progress * Simulate command * Recognise and award score and achievements * Update leaderboard * Draw diagram |
| STORAGE   * Leaderboard DB table * User credential DB table * User project DB table | OUTPUT   * Text/graphics * Achievement text/notifications * Leaderboard * Open project menu * Draw diagram |

## **Sample of planned SQL queries**

For testing and debugging purposes (as well as perhaps in the final program - see [“SQL injection and system security”](#_wfv8mqmrep4u)), all SQL queries that make changes to the database will be prefixed with the line “begin transaction” so that any changes can be rolled-back if the query was written incorrectly or made false changes to the database. Once the query has been verified to behave appropriately the changes can be committed.

\*Note that in SQL the “;” at the end of the line isn’t required so I will be omitting it for brevity

**Create**

**INSERT INTO** users (username, hash) **VALUES** (?, ?)

**INSERT INTO** leaderboard (userid,username,score) **VALUES** (?,?,?)

**Get**

**SELECT** userid **FROM** users **WHERE** (username = *“xyz”* and hash = “*xyz”)*

**Update**

**UPDATE** leaderboard **SET** (score) **WHERE** userid = “*xyz”*

**Delete**

**DELETE** **FROM** leaderboard **WHERE** (userid = *“xyz”)*

## **Validating user input**

### **Regular expressions**

A useful method of input validation is pattern matching. This is the process of pulling out patterns and general forms from an input string then comparing them to accepted and rejected states to determine if the input is valid. This is better than individually checking the string against others because it allows for easy adaptation as well as being a deterministic process (such that an accepted string will always remain an accepted string unless the underlying pattern matcher changes). For this reason, I decided to use the regex pattern matcher in my program (built-in: the “***re”*** module in python). Below are some regex tables containing the pattern match expressions that I could implement into my program as a method of validating user input.

|  |  |  |  |
| --- | --- | --- | --- |
| **Regex expression** | **Breakdown** | **Test strings** | **Pass/fail** |
| ^([A-Za-z0-9]|[A-Za-z0-9](([a-zA-Z0-9,=\.!\-#|\$%\^&\*\+/\?\_\{\}~]+)\*)[a-zA-Z0-9,=!\-#|\$%\^&\\*\+/\?\_\{\}~])@(?:[0-9a-zA-Z-]+\.)+[a-zA-Z]{2,9}$  (taken from [regex - How can I validate an email address using a regular expression? - Stack Overflow](https://stackoverflow.com/questions/201323/how-can-i-validate-an-email-address-using-a-regular-expression)) | **EMAIL VALIDATION**  Whilst the complete breakdown of this regex is far to complicated to fit into this table or indeed this project, the creator helpfully included a FSM diagram to illustrate how it works. You can find it [here](https://i.stack.imgur.com/YI6KR.png). | test  test@  test@test  test.com  test@test.com  test@test.co.uk  1test@test.t  test1@test.t  Test1@test.t  test!@test.t  \_test@test.t  !!!!@test.t  test@@test.com  @test@test.com | No (Pass)  No (Pass)  No (Pass)  No (Pass)  Yes (Pass)  Yes (Pass)  Yes (Pass)  Yes (Pass)  Yes (Pass)  Yes (Pass)  Yes (Pass)  Yes (Fail)  No (Pass)  Yes (Fail) |
| (?=.\*\d.\*)(?=.\*[\p{P}\p{S}].\*)(?=.\*[a-zA-Z].\*).{8,} | **PASSWORD VALIDATION**  **Positive Lookahead**  **(?=.\*\d.\*) -** Assert that the Regex below matches  **.** -matches any character (except for line terminators).  **\*** - matches the previous token between zero and unlimited times, as many times as possible.  **\d** - matches a digit.  **Positive Lookahead**  **(?=.\*[a-zA-Z].\*) -** Assert that the Regex below matches.  **[a-zA-Z] -** Match a single character present in the list  **{8,}** matches the previous token between 8 and unlimited times, as many times as possible.  **Global pattern flags**  **g** modifier: **g**lobal. All matches (don't return after first match)  **m** modifier: **m**ulti line. Causes ^ and $ to match the begin/end of each line (not only begin/end of string) | test  testtest  test1234  test123!  Test  Test1!  !test123  Test123!  test!!!! | No (Pass)  No (Pass)  No (Pass)  Yes (Pass)  No (Pass)  No (Pass)  Yes (Pass)  Yes (Pass)  No (Pass) |

### **SQL injection and system security**

In my login procedure, I will have to be particularly careful about sanitizing user input because it will be fed into a SQL query that gets sent to and executed on the users DB. This means that the user could theoretically abuse the login process to escape the query and execute their own query - this wouldn’t be especially difficult, due to the user actually being able to view exactly what query gets sent to the database since Python is an interpreted language. One safeguard around this is converting the program to an executable which I probably will anyway for user convenience: this doesn’t however fix the overall problem.

There are two main ways that I can attempt to fix this issue. The first is input sanitisation, which would mean controlling the users input to prevent them from injecting commands and also prevent users from accidentally setting their username or password to a banned string. This is probably the easiest of the two methods however, it depends entirely on having appropriate sanitisation which can be hard to do. Covering every single possible attack string as well as differentiating them from legitimate user credentials could be nearly impossible.

The second method would be much harder to implement but would function better (in fact, a combination of both methods would provide the best security but as I am limited by time, I'm not sure if I would have the resources to implement this). It revolves around using “Rollback” and “Commit” transaction commands in the SQL query itself. The program could read the database changes after executing the given command and compare these with the expected changes to decide whether or not to commit the command or rollback to before it was issue: for example, a select query should not produce any changes in the database, so if the program reads any updates when it supposedly sent a SELECT request then something has gone wrong.

### **Error handling**

My program takes full advantage of python’s call-stack based error handling by forcing certain interactions at the interpreter level – meaning that the user never has to fear of an unexpected crash or warning. The program makes full use of python’s built-in statements such ***try… except…*** for their optimised performance and ease of integration into large programs or projects that need to scale.

Another way of cleaning up console output for the user is by forced function calls. The program regularly uses statements such as ***assert***, ***raise*** and ***exit.*** These built-in functions can be used in conjunction with clever program design to produce clean program closing and human readable error messages when the user needs to respond to a problem with the interpreter. This solution is the most elegant because it includes functions that have existed in python for a long time and have become optimised to the interpreter rather than wasting time writing new code that wouldn’t function as well.

## **Algorithm research**

### **Quantum algorithms**

**Grover's algorithm:**

One of the algorithms that can be implemented into my system is Grover’s algorithm. This is a searching algorithm with order as opposed to its classical problem counterpart which cannot be solved in fewer than steps.

**Shor’s algorithm:**

**Entanglement:**

### **Classical algorithms**

**Diffusion process:**

To make my program more visual and appealing to users, I wanted to include a diffusion model for a particle so that the user could “see” the effects their commands were having on the simulation and to hopefully help them retain that knowledge.

In doing this I went through many different methods before settling on the final one. Initially I tried using a combination of a SoftMax algorithm to generate proportions and then a normalisation function to get it into an appropriate range. The thing that was difficult about this method was that there were lots of variables that needed to be considered and lots of temporary lists and changes that needed to be saved. Not only was this harmful due to the excessive memory use but also because of how python is interpreted as a language – having a large base of variables before proper testing of function application and scope means that data can get overwritten or stored unnecessarily throughout the program’s runtime.

****

A screenshot of a computer program

Description automatically generated

In the end I decided to use Gaussian noise to create an element of randomness proportional to the step of the loop. This meant that my random noise function could smooth out over time to be more representative of a diffusion function. It is also much more efficient, as no diffusion calculations need to take place: we can mimic them with a large amount of accuracy without having to spend the computational cost of full simulation.

A blue and red line graph

Description automatically generated

**Tree traversal algorithms:**

For parsing data and commands taken via user input, the standard method is to use the regular development pipeline of ***“lexer -> abstract syntax tree -> action tree -> interpretation”.*** Whilst not strictly necessary when you are not building an entire language from the ground up, it is always beneficial to adopt standard or conventional methods as this makes development significantly easy as well as increasing the options for adaption. Since AST’s and AT’s both make use of the tree data structure, they rely on traversal algorithms to function.



## **Algorithm Design**

In order to properly plan my project’s python implementation, I designed a set of simple algorithms in pseudocode that I can use to prepare for application in my program.

**IN-ORDER TRAVERSAL:**

**SUB** in\_order(root)

**IF** root **exists** **THEN**

in\_order(root.left)

**OUTPUT** root.value

in\_order(root.right)

**ENDIF**

**POST-ORDER TRAVERSAL:**

**SUB** post\_order(root)

A diagram of a network

Description automatically generated with medium confidence **IF** root **exists** **THEN**

post\_order(root.right)

post\_order(root.left)

**OUTPUT** root.value

**ENDIF**

**PRE-ORDER TRAVERSAL:**

**SUB** pre\_order(root)

**IF** root **exists** **THEN**

**OUTPUT** root.value

pre\_order(root.left)

pre\_order(root.right)

**ENDIF**

**BREADTH FIRST SEARCH:**

**SUB** BFS(step)

        visited **->** [**False**] **\*** (**MAX**(self.graph) **+** 1)

        queue **->** **LIST**()

**ENQUEUE** step, queue

        visited[s] **->** **True**

**WHILE** queue

            step **->** **POP** 0, queue

**OUTPUT** s

**FOR** I **IN** self.graph[step]

**IF** visited[i] **=** **False** **THEN**

**ENQUEUE** i, queue

                    visited[i] **->** **True**

**ENDIF**

**ENDFOR**

**ENDWHILE**

**DEPTH FIRST SEARCH:**

visited -> **SET**()

**SUB** DFS(vertex, visited)

**ADD** vertex, visited

**OUTPUT** vertex

**FOR** neighbour **IN** self.graph[v]

**IF** neighbour **NOT IN** visited **THEN**

                DFS(neighbour, visited)

**ENDIF**

**ENDFOR**

## **Class definitions**

#THIS SECTION CONTAINS CLASS PSEUDOCODE

## **Project hierarchy chart**

#THIS SECTION CONTAINS A DIAGRAM OF THE PROGRAMS HIERACHY, AS ONE PROBLEM SPLIT INTO SUBPROBLEMS AS WELL AS SHOWING INHERITENCE

## **UI/UX design**

**Title design:**

The first part of the program that the user interacts with once will be a console interface for the login system (which could also be implemented as a GUI for more accessibility later down the development cycle for the program). As of such, I wanted this to look interesting to leave a positive impression on the user: this is entirely cosmetic, it has no functionality other than making the program appear more friendly to new users or those less familiar with a CLI environment. I went through a few designs for the splash screen which I have included below:

A screenshot of a computer screen

Description automatically generated

In the end I settled on the third design as I felt it suited the rest of my design much better as well as being easiest to read and most compatible with different machines (as it is comprised entirely of 7-bit ascii symbols)

**GUI design:**

Since a large portion of my project requires visual feedback to the user it was important to design a GUI that looks and feels satisfying. Additionally, the objective of making the program as aesthetic and simple as possible fed into my design process. I ended up doing some digital drafts of the generic layout I was looking for in my program before implementing these in tkinter and tweaking as required.

A black rectangular object with white text

Description automatically generatedA drawing of a whiteboard

Description automatically generated

A screen shot of a computer

Description automatically generatedA drawing of a grid

Description automatically generated

**A drawing of a square object

Description automatically generated**A screenshot of a computer screen

Description automatically generated

A computer screen with a white screen

Description automatically generatedA screenshot of a computer

Description automatically generatedAfter this I went into figma to properly plan how the user might react with the program as well as to curate an effective and aesthetic program flow. The designs I produced are shown below.

A screenshot of a computer

Description automatically generated

Using figma helped me properly plan the interactions that the user would have with the system so that I could effectively program an implementation in tkinter.

**CLI design:**

## **Project directory layout**

### **Initial tree**

**>nea**

**---->pycache**

---->cbit.py

---->interface.py

---->login.py

---->main.py

---->qbit.py

---->vector.py

>users.db

### **Final tree**

# ***Technical solution***

## **Database implementation**

## **UI implementation**

## **Simulator implementation**

# ***Testing***

## **Trace tables**

## **Test tables**

My code was fully tested at two main points in my project: roughly halfway through as well as at the end. This was to ensure that no errors slipped through the smaller testing that went on during the development process. This was important to do because it gave me the opportunity to step back from my project for a bit and just focus on fixing any bugs or issues that might have come up before continuing to improve and adapt the project.

**Vector.py initial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| 1 | \_\_init\_\_ | Incorrect size parameter type | Error Message + exit | Pass |  |
| 2 | setElement | Valid data | Return True | Pass |  |
| 3 | setElement | Index larger than list | Return False | Pass |  |
| 4 | setElement | Wrong index type | Return False | Pass |  |
| 5 | getElement | Valid query | Return correct data | Pass | Data returned is from the right place |
| 6 | getElement | Wrong index type | Return False | Pass |  |
| 7 | getElement | Index larger than list | Return False | Pass |  |
| 8 | scalarMul | Valid query | Return multiplied vector | Fail | Returns (0,0) always |
| 9 | scalarMul | Bad type for ‘num’ | Return False | Fail | No error checking |
| 10 | \_\_mul\_\_ | Valid query | Return multiplied vector | Fail | Fails because of scalarMul |
| 11 | setElements | Valid entry | Return True | Pass |  |
| 12 | setElements | Invalid entries | Return False | Fail | No error handling |
| 13 | setN | Valid float entry | Return True | Pass |  |
| 14 | setN | String entry | Return False | Fail | Allows string entry |
| 15 | allZeros | Zeroed vector | Return True | Pass |  |
| 16 | allZeros | Non-zero vector | Return False | Fail | Works when any element other than the 0th index is not 0 |
| 17 | magnitude | Valid vector object | Return correct magnitude | Pass |  |
| 18 | isUnit | Unit vector input | Return True | Pass |  |
| 19 | isUnit | Non-unit vector input | Return False | Pass |  |
| 20 | unit | Valid vector object | Return correct unit vector | Pass |  |
| 21 | tensor | 2 vector objects – same length | Return correct tensor | Pass |  |
| 22 | tensor | 2 vector objects – different length | Return correct tensor | Pass |  |
| 23 | tensor | 1 vector object – one other type | Return False | Fail | No exception handling |
| 24 | \_\_repr\_\_ | Valid vector object | Return correct string representation | Pass |  |

**Cbit.py initial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| 1 | \_\_init\_\_ | Incorrect dirac type | Stops execution | Pass | Needs better exception handling |
| 2 | \_\_init\_\_ | Bad sub value | Stops execution | Pass | Needs better exception handling |
| 3 | \_\_init\_\_ | Bad sub type | Stops execution | Pass | Needs better exception handling |
| 4 | \_\_init\_\_ | Large dirac – no sub | Return correct tensor | Pass |  |
| 5 | \_\_init\_\_ | Large dirac – suitable sub | Return correct tensor | Pass |  |
| 6 | setElement | Index larger than list | Return False | Pass |  |
| 7 | setElement | Bad index type | Return False | Pass |  |
| 8 | setElement | Bad value | Return False | Pass | Cbits can only have 1’s or 0’s as elements |
| 9 | setElement | Valid data | Return True | Pass |  |
| 10 | measure | Valid vector | Return correct measurement | Pass |  |
| 11 | measure | Vector larger than 2 | Return False | Pass | Use probcollapse instead |

## **Code iterations**

Over the course of my project, each function and class had its own micro development cycle. This meant that some functions were initially written with no regard to speed, efficiency or readability to produce an MVP that would solve the problem. This gave me lots of scope for development as I could produce a working section of code and then go back through it to refine and improve it. This meant that sometimes code was scrapped completely because there existed a much better solution that was found later in the project. Some examples of code and how they changed between the start of the project and the present are given below to highlight this. Each function and class followed its own small spiral development, which helped me add all of the features my program needed in a timely manner avoiding a feature crunch towards the end of the project, in which functionality is shoe-horned in at the last minute without a stable or strong supporting code base. This is obviously not ideal. A black and white screen with a circular pattern

Description automatically generated with medium confidence

**Initial login:**



A white background with text

Description automatically generated

**Final login:**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Graph plotting and threads:**

The graph plotting code went through many different variations until I found the one that I wanted to settle with. This is because I had to run multiple main loops simultaneously and independently: which forced me to teach myself some basic asynchronous programming and threading in python. Ultimately, I decided to use the built-in function ***matshow*** from matplotlib – this was because it was just more optimised than anything I could have possible written by myself as well as being built with 2d arrays in mind, which was a base type that I am already very familiar with rather than having to learn how use NumPy arrays or another custom type.

**A computer screen shot of a program

Description automatically generated**

## **UX testing**

# THIS SECTION SHOWS USER FEEDBACK ON UI FLOW AND EXPERIENCE

# ***Evaluation***

## **Achieved solution versus project objectives**

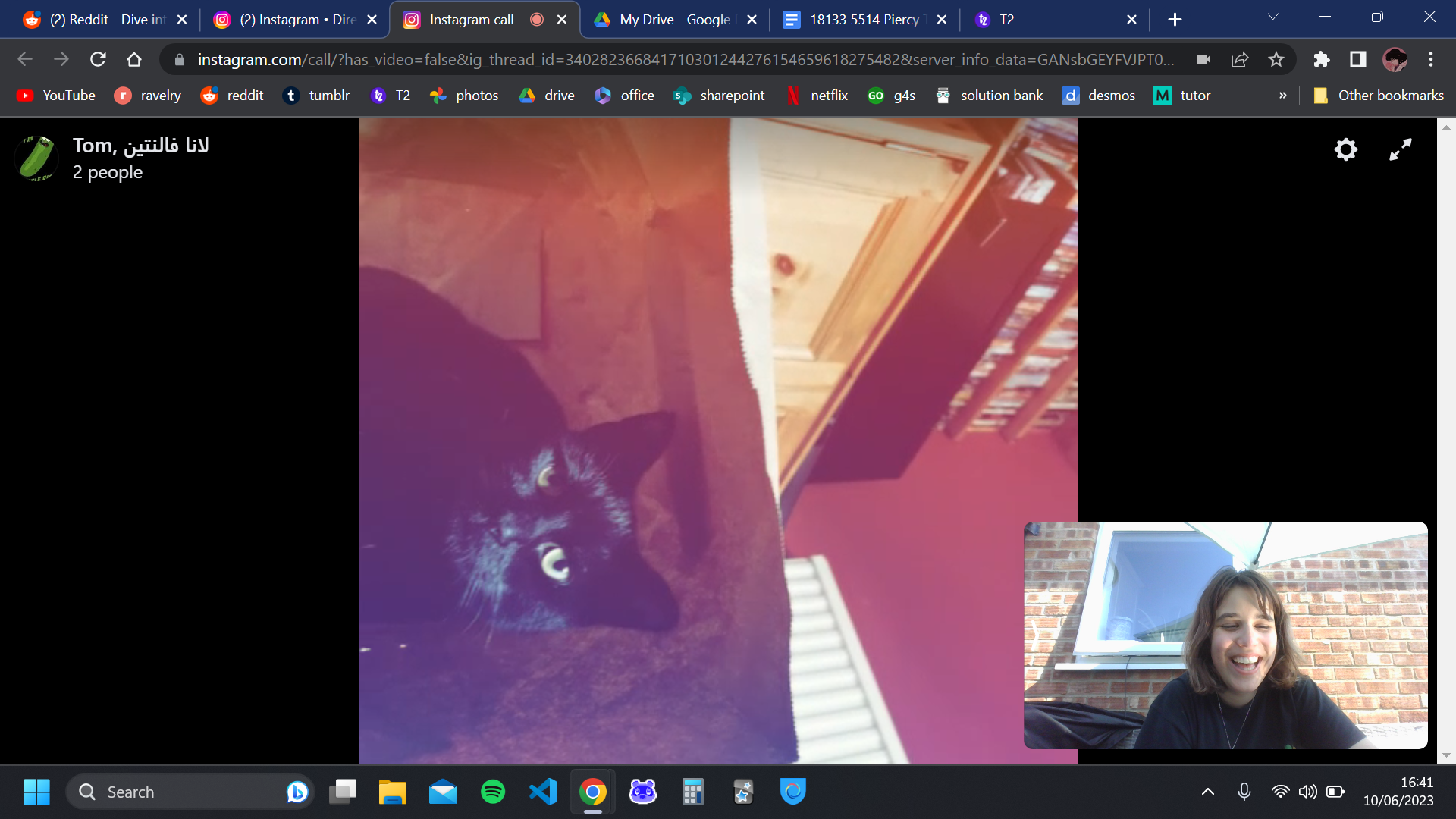
## **Client feedback**

#THIS SECTION CONTAINS MADE UP FEEDBACK FROM THE CLIENT

## **Independent feedback**

#THIS SECTION CONTAINS FEEDBACK FROM PEOPLE OTHER THAN THE INTENDED CLIENT

## **Scope for further development**



# ***Appendix***

