COMPUTER SCIENCE NEA

Quantum Computation Simulator



Tom Piercy (5514)

City of Norwich School (18133)

***Contents***

[***Analysis*** 4](#_Toc157694123)

[Background to the problem 4](#_Toc157694124)

[Current solution 4](#_Toc157694125)

[Client interview 5](#_Toc157694126)

[Intended User and Prerequisite knowledge: 6](#_Toc157694127)

[User needs and project limitations: 6](#_Toc157694128)

[Data sources and destinations 7](#_Toc157694129)

[Data dictionary 8](#_Toc157694130)

[Data flow diagram 9](#_Toc157694131)

[Entity relationship diagram 12](#_Toc157694132)

[Project goals, objectives and KPIs 13](#_Toc157694133)

[Potential solutions 14](#_Toc157694134)

[Chosen solution 15](#_Toc157694135)

[***Design*** 17](#_Toc157694136)

[Database design 17](#_Toc157694137)

[**Initial solution** 17](#_Toc157694138)

[**Final solution** 19](#_Toc157694139)

[IPSO chart 19](#_Toc157694140)

[Sample of planned SQL queries 20](#_Toc157694141)

[Validating user input 21](#_Toc157694142)

[**Regular expressions** 21](#_Toc157694143)

[**SQL injection and system security** 23](#_Toc157694144)

[**Error handling** 23](#_Toc157694145)

[Algorithm research 24](#_Toc157694146)

[**Quantum algorithms** 24](#_Toc157694147)

[**Classical algorithms** 25](#_Toc157694148)

[Algorithm Design 29](#_Toc157694149)

[Class diagram 33](#_Toc157694150)

[Class definitions 34](#_Toc157694151)

[Object orientation plan 51](#_Toc157694152)

[Project hierarchy chart 75](#_Toc157694153)

[UI/UX design 79](#_Toc157694154)

[Project directory layout 85](#_Toc157694155)

[**Initial tree** 85](#_Toc157694156)

[**Final tree** 85](#_Toc157694157)

[***Technical solution*** 87](#_Toc157694158)

[Database implementation 87](#_Toc157694159)

[UI implementation 87](#_Toc157694160)

[Simulator implementation 87](#_Toc157694161)

[Runtime screenshots 87](#_Toc157694162)

[***Testing*** 90](#_Toc157694163)

[Trace tables 90](#_Toc157694164)

[Test tables 90](#_Toc157694165)

[Code iterations 109](#_Toc157694166)

[UX testing 113](#_Toc157694167)

[***Evaluation*** 117](#_Toc157694168)

[Achieved solution versus project objectives 117](#_Toc157694169)

[Client feedback 119](#_Toc157694170)

[Independent feedback 120](#_Toc157694171)

[Scope for further development 121](#_Toc157694172)

[***Appendix*** 123](#_Toc157694173)

[Testing screenshots 123](#_Toc157694174)

[Full code listing 123](#_Toc157694175)

# ***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 does not 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 particularly 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 achievements | User completing an action inside the program | DB table/JSON object |
| Saved results | User saving a file | Recorded into a custom file, in the users folder |

## **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.

A diagram of a computer program

Description automatically generatedA final data-flow illustration has been provided to aid any other developers. It shows the path taken through the whole program on initialisation and is used to give an example of which functions should return to where in the call-stack. The hope is that it aids understanding of the overall program flow:

## **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.

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

Using the learning points of the interview and marks requirements for the solution I have devised a table listing the goals of the project and how I aim to meet them.

|  |  |  |  |
| --- | --- | --- | --- |
| **Goal** | **Priority** | **Testing** | **Key performance indicator** |
| The program is intuitive, fluid, and easy to use | High | Client and Independent UX testing during development | Positive feedback from testers |
| The program incentivises learning in a fun environment | High | Independent testing at end of project | Positive feedback from testers |
| The program should be aesthetic and easy to be on for extended periods of time | Moderate | Developer testing and client interviews | Positive feedback from interviews |
| The program should be fully interactive | High | Developer testing | User freedom |
| The program should be timeless and not quickly outdated | Moderate | Developer testing | N/A |
| The program should be feature rich | Moderate | Developer testing | Client feedback |
| The program should accurately reflect the system it is emulating | High | Developer testing | Client interviews |
| The program should contain examples and relevant links. | Low | Independent testing | Positive feedback from testers |
| The program should not be able to break, crash or be exploited | High | Developer testing | Positive feedback during testing period |

## **Potential solutions**

1. An interactive website learning resource that provides exercises, questions, and additional links

- This solution is quite similar to the current solution, with the addition of extra interactive elements to benefit mark. This is potentially a good thing because it increases familiarity with the program format – by extension increasing fluidity.

- An advantage of having a web implementation is that no programs need to be installed natively, freeing memory and potentially increasing program reach and access by having a very vast distribution network. This would mean that it would be extremely easy for other people to find and use the program.  
- A disadvantage of web distribution is that using the system is reliant on a stable internet connection. This potentially limits the amount of people that could use the system and the times that they could use it. This may be inconvenient and put people off using the system  
- A central server would have to be introduced which would both require both more complex code and considerations towards balancing usage and load. The server would need to go through strength and resilience testing which would take up more time and be more labour intensive.

2. A cross-device mobile game that teaches the basics of quantum computing through play.

- This solution would require considerations about distribution, as it would be important to know how the product would be sent out to potential users. Both Apple and Google have their own APK distribution platforms with large reach and additional ease of accessibility for less technical users, however the drawback to these is getting approval from the companies to list the application on their servers.  
- An advantage of this system would be that once distribution was complete, the program would be very easy to navigate through and intuitive for end users. This would be due to the familiarity and fluency of users for mobile applications.

- The system would be installed locally, so there would be no need for complex server code, whilst still maintaining an effective and efficient way to push updates to the program through the app store.

- A disadvantage of this solution would be the complexity of the code required to produce it. Tools such as Android App Development Studio and mitLab exist to speed up the process although both of these still rely on mobile optimised languages such as Kotlin, which I am unfamiliar with. Another issue would be that to avoid alienated part of the user base, the app would ideally need to be compatible with IOS, for which most apps are coding in a language such as Swift as well as having different coding requirements to be accepted onto the apple distribution system.

3. A sandbox desktop application that teaches quantum computing by encouraging exploration and providing visual feedback

- This solution gives a good compromise with the problem of distribution: Getting the program out to users is slightly more difficult and updates would be infrequent or non-existent, but this comes at the benefit of not needing to code to a FAANGs company specifications but rather being free to take the project in my own direction.

- To address the issue of program repetitively and interest stagnation - on a desktop we have the resources at hand to effectively design a less restrictive, more sandbox environment allowing the user to decide for themselves what they want to do in any given session. Additionally, coding in for desktop environment allows for a more advanced I/O system. We have more screen space to play with, better input control and increased flow for an extended range of characters that can be represented with a physical keyboard.  
- The code complexity is also a good compromise. The code required for the solution wouldn’t be as complex as the other ones as well as being in a language I am familiar with, whilst retaining a certain degree of program intricacy.  
- Code distribution could be done through free application hosting sites on the internet, such as sourceforge or majorgeeks and cross-OS compatibility – whilst still an issue – will be much easier to address as well as affecting significantly less people than with the mobile application due to the widespread use of windows in professional environments.

## **Chosen solution**

After talking with my client and explaining all of the different solutions, we have decided to go ahead with solution 3. Together we feel that solution 3 strikes a good balance between being a complex and complete answer to the problem whilst maintaining achievability. We have decided not to go with solution 1 due to my inexperience with server and network coding as well as the limited time frame of the project. We have decided not to go with solution 2 due to issues with learning the required development language as well as issues with authorship and the distribution of the system. After a detailed discussion of solution 3, mark is aware of the limitations of the project and how we can mitigate them. A table is provided below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Problem** | **Impact** | **Significance/risk** | **Mitigation** |
| Declining interest due to rigid, repetitive, or reused lesson structures | Users will stop using the program quickly, opting for other solutions | 3/10 – Low harm | Freeing the user from restrictive lesson structures by creating an open sandbox experience |
| Difficulty distributing the system to users | Users won’t be able to find the program | 2/10 – Easily mitigated | Using free distribution systems that already exist on the web |
| Difficulty porting the solution to other platforms | Mac and Linux users won’t be able to user the program | 5/10 – high probability, Low harm | Linux users can build from source. Mac users will be able to run from the python interpreter until development can be completed. Community porting is also an option |
| Program modification by 3rd parties | Due to the python interpreter code could be modified for malicious purposes | 6/10 – moderate risk, low impact | Coding best practices and secure development limiting access. Secure hashing on sensitive data. Installed locally with no network connection so no risk of large data leaks. |
| Device compatibility issues | Not all devices or displays will be compatible with display libraries used | 4/10 – minimal impact | Program can be shipped as executable with the python interpreter and all necessary libraries to minimise user setup. Python handles screen resolution natively |
| Accessibility for non-native English users | Users might struggle to use program. Developers might struggle to read code | 2/10 – Low harm | Translation resources exist on the web as well as coding for a desktop environment providing a greater range of printable Unicode characters and keyboard setups. Automatic OS translation |
| System misuse | Database dumping, SQL injecting and other program exploitation | 6/10 – moderate risk, low harm | Local install nature of program limits impact of database attacking. Saving sensitive information securely limits attack vector. Program corruption through SQL injection can be reset through a program reinstall with negligible impact on user. |

# ***Design***

## **Database design**

### **Initial solution**

**0-NF**

The following table is its non-normalised initial state. This is because in its current state it is unclear which field should be the primary key, and the context of the rest of the data depends on this. For example, picking a primary key of “Challenge ID” means repetition in the userID field, creating issues with retrieving, updating, and removing data.

**1-NF**

The following data is in 1st normal form. We have moved data into more appropriate locations as well as unifying styling across all fields. The primary key has been specified to be UserID. The problem with using this form of the database is that there are still pieces of unrelated or dependant data being stored in the same location. We have also checked that all data is atomic.

**2-NF**

The data has now been put into 2nd normal form, separating data into its appropriate linking tables. However, in the process of putting the database into 2nd normal form we have had to reintroduce some redundancy in our table which we will fix in 3rd normal form. “ChallengeID” has been chosen as the primary key for the new table Challenges.

**3-NF**

The database has now been put into 3rd normal form, which is the final form of the data. It was decided that difficult to handle pieces of data (in particular the date achieved section being reliant on the users device to provide information: having the potential to break the database) would be handled by a JSON file linked to the other tables via the use of foreign keys. This also helps us remove our redundant userID field in the challenges table and perform a successful inference test on the rest of the data. Scoring has also been placed into a new table to eliminate redundancy.



### **Final solution**

I had the opportunity to speak to mark about the final database design and it was agreed that we would go ahead with the 3rd normal form design. In order of priority of implementation- mark commented that, whilst to best meet the project objectives we need the inclusion of every table, the highscores table was less important to him than the core functionality that the Users table (and associated login system) provides. We settled on the priority order: *Users, Challenges, JSON, Highscores*.

## **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 is not 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 too complicated to fit into this table or indeed this project, the creator helpfully included an 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) |
| ^([a-zA-Z]+:)|(:[a-zA-Z]+:) | **SUPPLIMENT SYNTAX**  **^** - asserts position at start of a line  **:** - matches the character “:”  **+** - matches the previous token between one and unlimited times | test  test:  :test:  :  ::  test:  :test: | No (Pass)  Yes (Pass) Yes (Pass) No (Pass) No (Pass) No (Pass) Yes (Pass) |
| [a-zA-Z\_]\w\*\([a-zA-Z\_]\w\*\) | **OBJECT SYNTAX**  **\w** - matches a word character. | test  test()  test(a)  test(a )  test (a)  test(a)  test(\_)  test(%) test(a a) | No (Pass) No (Fail) Yes (Pass) No (Pass) No (Pass) Yes (Pass)  Yes (Pass)  No (Pass) No (Pass) |
| [0-9]+(\.[0-9]+)? | **DIGIT SYNTAX  ?  -** matches the previous token between zero and one times | 0  10  10.1  10.  10.0.10 | Yes (Pass) Yes (Pass)  Yes (Pass) Yes (Pass) Yes (Fail) |

### **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**

Algorithm research was vital to producing a solution that fit mark as well as being the most effective tool for learning. During my research, I drew on knowledge of algorithms that I had studied in education as well as those that go beyond the curriculum to help me create an efficient project. I explored both quantum and classical algorithms to set my project apart.

### **Quantum algorithms**

To fully understand the intricacies of a quantum computer, I had to effectively research various quantum algorithms. This provided me a platform to make sure that my program worked as accurately as possible. It also helped me gain the necessary base knowledge to build my system appropriately. In order to research this topic, I used resources I found online as well as a few series of lecture notes – I then collated these into a large binder for ease of research. This provided me an extensive array of resources that I could turn to when I needed guidance with my project.

**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 for an unsorted array. It does this by solving multiple oracles with the intent of raising the probability of finding the correct answer and lowering the probability of measuring an incorrect answer. The probability states are then measured to resolve which solution is most likely to be correct.

A black background with a black square

Description automatically generated with medium confidence

**Superposition:**

Not an algorithm but still a crucial element of quantum computers and one that fits well to talk about here. Superposition is the name given to the state of probability that an entity exists in because of the non-deterministic nature of quantum mechanics. It is an important concept in quantum computing as it is the mechanism that allows exponential speed up of subroutines by solving multiple oracles at the same time. It is akin to parallel processing or threading, but it does not take up additional resources in the same way because it is an inherent property of nature as opposed to a human construction. Superposition is a far more complicated topic than has just been described however the hope is that this small paragraph sufficiently aids understanding for the other algorithms discussed below.

**Shor’s algorithm:**

A black background with a black square

Description automatically generated with medium confidenceShor’s algorithm is a complex quantum algorithm for factorising large numbers. It was developed in 1994 by Peter Shor. On a quantum computer it runs in polynomial time: and has important implications for cybersecurity: undermining the current security standard of the intractability of factoring large numbers. At this moment in time, creating a stable system with enough qubits is impossible.

**Entanglement:**

Entanglement is the process of equally splitting the probability of finding an entity in any given state with no leaning. This means that the object has no preference to be in one state versus any other: it is achieved with the Hadamard gate and is very important in quantum computing for setting/resetting qubits as it is symmetrical (it is its own inverse), cyclical and deterministic. This means that applying the algorithm a set number of times from a set state will always result in the same result.

### **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

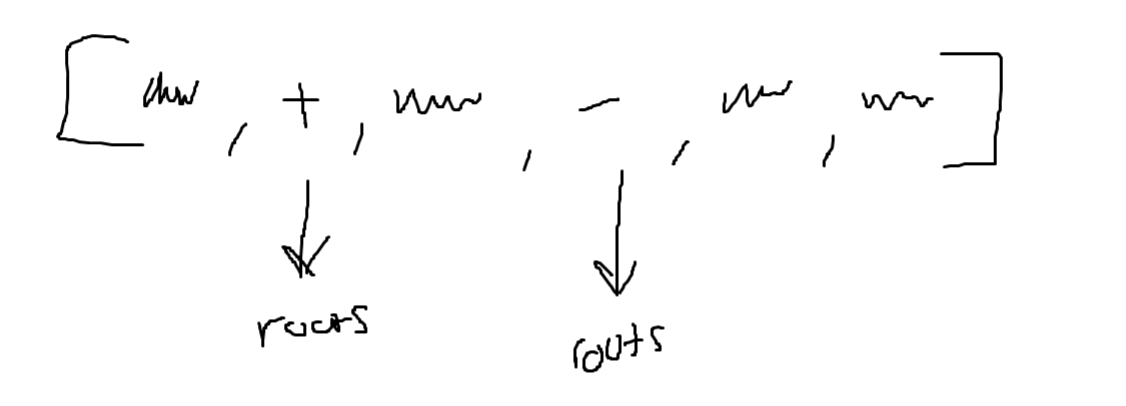
**Tree traversal algorithms:**

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.

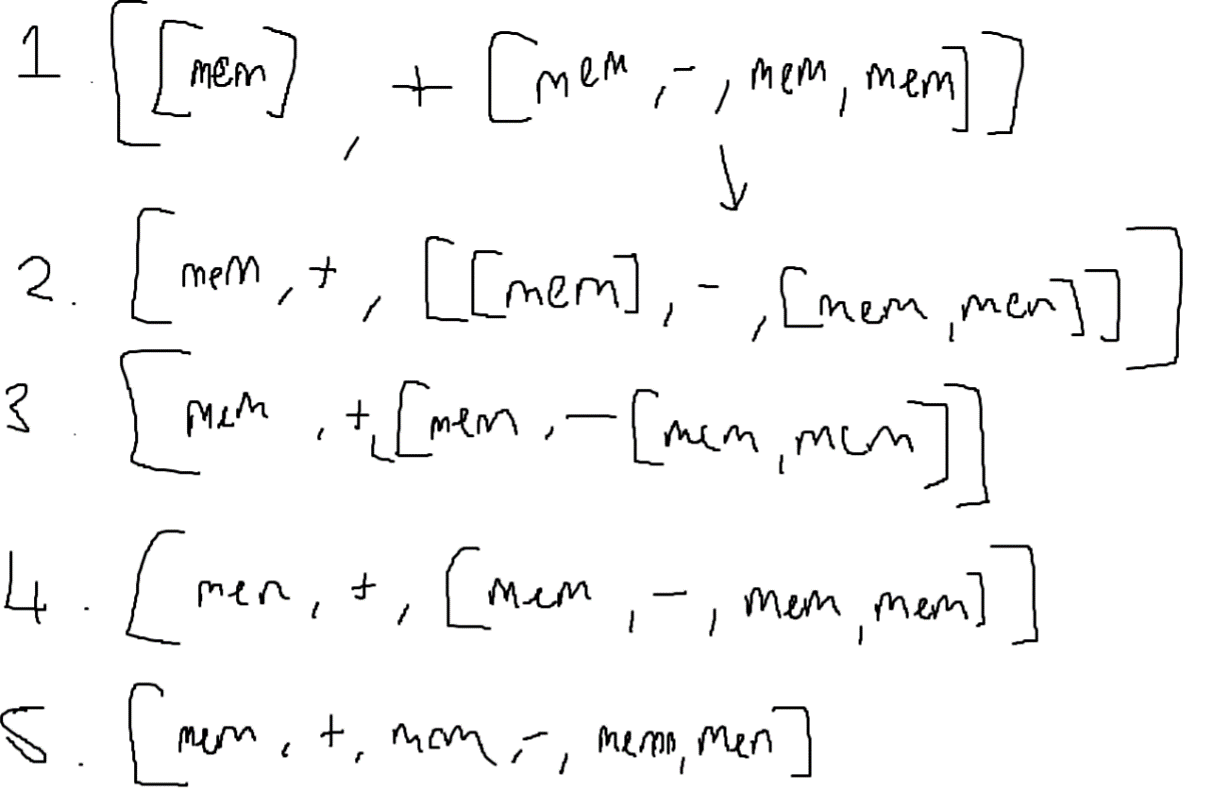


As of such, I had to research various different traversal algorithms including the most common in/pre/post order methods.

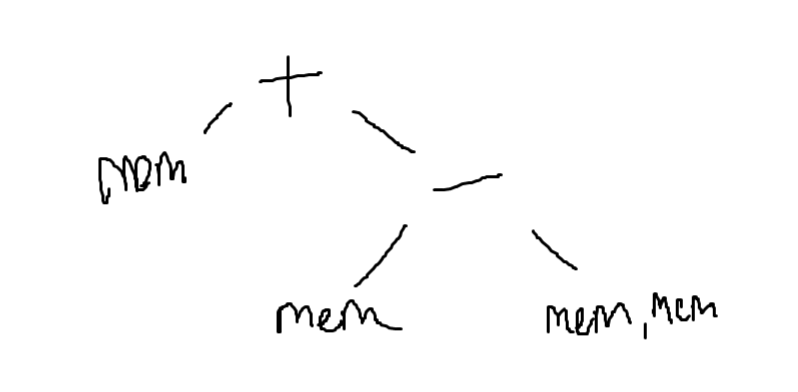
Designing the AST for my project was a long process as I had to explore various methods to find one which fit with the scale and type of my project. I had to settle between using generic and binary trees depending on how I would handle object parameters. In the end I settled on using a binary tree and later balancing it to reduce the search time. To explore how to structure the tree I went into Microsoft paint to sketch out some designs and brainstorm my ideas. I also spoke to a peer which helped my plan out my thoughts.



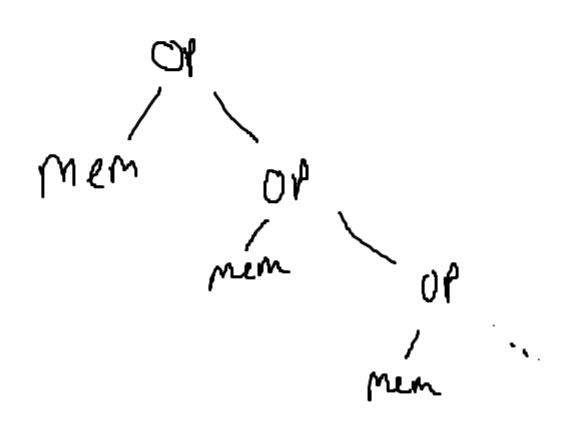
I had a list containing the various different tokenised and parsed elements, which I intended to place into a tree. I decided it made the most sense to place the root nodes as operators and the leaves be memory addresses or data. Then I could use different traversal algorithms to restructure the list which I can pipe into the exec function in python.



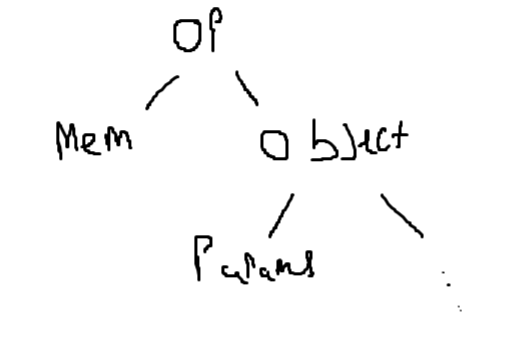
The steps 1-3 would be for placing into a tree and then 4 and 5 could be used to get the tree back into list form. This got me considering the use of a [left,data,right] structure for my tree. This then encouraged me to shift my view of tree and explore different layouts for my tree: whilst I was looking at these, I found a [memory, operator, next operator] structure which seemed to suit my project very well.



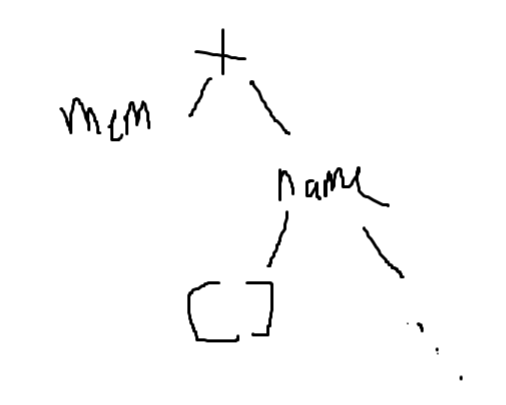
Expanding upon this gave me the idea to loop through the list saving the indices of all the operators then placing them into a binary tree in the following format - Allowing easy traversal.



This format also suited handling of objects surprisingly efficiently as I could treat parameters as data and function names as operators (since all operators are fundamentally objects in python). Like this:



Or (a more concrete example):



## **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**

**GROVERS ALGORITHM:**

**SUB** oracle(x, target)  
 **IF** x = target **THEN**

**RETURN TRUE**

**ELSE RETURN FALSE**

**ENDIF**

N -> **LENGTH** list

**FOR** qubit **IN** list

**HADAMARD** qubit

**ENDFOR**

**FOR** k = 1 **TO** **SQRT** N

oracle(list[k], target)

**FOR** qubit **IN** list

**HADAMARD** qubit  
 **X** qubit  
 **IF** **FOR** **EVERY** element **IN** list, element = |1> **THEN**

**Z** qubit  
 **ENDIF  
 HADAMARD** qubit

**ENDFOR**

**ENDFOR**

result -> **MEASURE** list

**OUTPUT** result

**SOFTMAX**

**SUB** softmax(vector)

e **->** **LIST**()

**FOR** element **IN** vector

**APPEND** **EXP** element, e

**ENDFOR**

normal **->** **LIST**()

**FOR** item **IN** e

**APPEND** item **/** **SUM** e, normal

**ENDFOR**

**RETURN** normal

**NORMALISE**

**SUB** normalise(vector, new\_max)

normal **-> LIST**()

**FOR** value **IN** vector

**APPEND** (new\_max \* (value – **MAX** vector) + new\_max), normal

**ENDFOR**

**RETURN** normal

**GAUSSIAN NOISE**

**SUB** gauss2d(array, mean, stddv)

numcol **-> LENGTH** array

numrow **-> LENGTH** array[0]

size **-> TUPLE** (numcol, numrow)

noise **-> RANDOM NORMAL** (loc**->***mean*, scale**->***stddv*, size**->***size*)

**LIST** noise

**FOR** i **->** 0 **TO** numcol

**FOR** j **->** 0 **TO** numrow

array[i][j] **->** array[i][j] **+** noise[i][j]

**ENDFOR**

**ENDFOR**

**RETURN** array

## **Class diagram**

A screenshot of a computer screen

Description automatically generated

## **Class definitions**

**Vector**

**CLASS** Vector

**SUB** \_\_init\_\_ (self, size)

**PUBLIC** vector <- [0] \* size

**SUB** getElement (self, index)

**RETURN** vector[index]

**SUB** setElement (self, index, value)

vector[index] <- value

**RETURN TRUE**

**SUB** scalarMul (self, num)

mulvec <- Vector (**LENGTH** vector)

**FOR** count, element **IN** vector

mulvec.setElement(count, num\*element)

**RETURN** mulvec

**SUB** setElements (self)

**OUTPUT NEWLINE**

**FOR** i <- 0 **TO** **LENGTH** vector

number <- **USERINPUT**()

setElement(i, number)

**OUTPUT NEWLINE**

**RETURN TRUE**

**SUB** setN (self, n)

size <- **LENGTH** vector

vector <- [n] \* size

**RETURN TRUE**

**SUB** allZeros (self)

**RETURN** **NOT ALL** vector

**SUB** magnitude (self)

total <- 0

**FOR** i <- 0 **TO** **LENGTH** vector

total += (vector[i])\*\*2

**RETURN** **SQRT** total

**SUB** isUnit (self)

**RETURN** magnitude() = 1

**SUB** unit (self)

size <- **LENGTH** vector

unitvec <- Vector (size)

mag <- magnitude()

**FOR** count, ele **IN** vector

unitvec.setElement (count, ele/mag)

**RETURN** unitvec

**SUB** tensor (self, other)

**IF** **NOT** **ISINSTANCE** other, object **THEN**

**RETURN FALSE**

newsize <- **LENGTH** vector \* **LENGTH** other.vector

tensorproduct <- Vector(newsize)

i <- -1

**FOR** count, element **IN** vector

**FOR** count2, element2 **IN** other.vector

i +=1

tensorproduct.setElement(i, element\*element2)

**RETURN** tensorproduct

**SUB** \_\_repr\_\_(self)

**RETURN** (vector)

**Cbit**

**CLASS** Cbit (Vector)

**SUB** \_\_init\_\_(self, dirac, sub)

**PUBLIC** Cbit **<-** **NONE**

**PRIVATE** sub **<-** sub

**IF** sub **=** **NONE THEN**

            sub **<-** **LENGTH** **FORMAT** **BINARY** dirac

        dirac **<-** (sub - **LENGTH** **FORMAT** **BINARY** dirac \* "0" + **FORMAT** **BINARY** dirac)

**IF** sub **=** 1

            Cbit **<-** Vector(2)

            Cbit.setElement(**ABS** (0 – **INTEGER** dirac), 1)

**ELSE THEN**

**SPLIT** dirac

            tensorprod **<-** **NONE**

**FOR** count, ele **IN** dirac

                ele **<-** **INTEGER** ele

**IF** count **=** 0 **THEN**

**CONTINUE**

                element **<-** Vector(2)

                element.setElement(**ABS** (0-ele), 1)

**IF** tensorprod **=** **NONE**

                    lastelement **<-** Vector(2)

                    lastelement.setElement(**ABS** (0 - **INTEGER** dirac[count-1]), 1)

**ELSE THEN**

                    lastelement **<-** tensorprod

                tensorprod **<-** element.tensor(lastelement)

            Cbit **<-** tensorprod

**@override**

**SUB** setElement(self, index, value)

**IF** **LENGTH** Cbit.vector **=** 2

        Cbit.vector[index] **<-** value

**RETURN** **TRUE**

**SUB** measure(self)

**IF** **LENGTH** Cbit.vector **<>** 2

**RETURN** **FALSE**

**ELSE THEN**

**RETURN** Cbit.vector[1]

**SUB** probcollapse(self):

        size **<-** **LOG** **LENGTH** Cbit.vector, 2

**OUTPUT “**Probability of collapse:”

**FOR** count, element **IN** Cbit.vector

            state **<-** **INTEGER** (size - **LENGTH** **FORMAT** **BINARY** count \* "0" + **FORMAT** **BINARY** count)

            percent **<-** element \*\* 2

**OUTPUT** “|state>, percent\*100 %"

**RETURN**

**SUB** \_\_repr\_\_(self)

**RETURN** (Cbit.vector)

**Qbit**

**CLASS** Qbit (Cbit)

**SUB** \_\_init\_\_(self,dirac, sub**<-NONE**):

**SUPER**.\_\_init\_\_(dirac, sub)

**PUBLIC** Qbit **<-** Cbit

**PUBLIC** probability **<-** [[0 **FOR** i **<- 0 TO** 10] **FOR** j **<- 0 TO 10**]

**PUBLICS** x, y **<- RANDOM INTEGER** (0,10), **RANDOM INTEGER** (0,10)

        probability[x][y] **<-** 1

**PRIVATE** changed **<-** [[x,y]]

**SUB** measure(self)

**IF** **LENGTH** Qbit.vector <> 2

**RETURN** **FALSE**

**ELSE THEN**

            bits **<-** [0,1]

            collapse **<-** **INTEGER** **RANDOM CHOICES** (bits, weights**<-**(**POW** Qbit.vector[eles], 2), k**<-**1)[0]

**RETURN** collapse

**SUB** \_softmax(self,vector)

        e **<-** [**EXP** ele **FOR** ele **IN** vector]

**RETURN** [item/**SUM** e **FOR** item **IN** e]

**SUB** \_normalise(self,vector, maxprime)

**IF** **NOT** **ISINSTANCE** (maxprime, **FLOAT**) **OR ISINSTANCE** (maxprime, **INTEGER**)

**RETURN** **FALSE**

        min, max **<-** 0, 1

**RETURN** [(maxprime\*(value-max)+maxprime) **FOR** value **IN** vector]

**SUB** \_applygauss2d(self,array, n)

        mean **<-** 0

        stddv **<-** 100/(2\*\*n)

        noise **<-** **LIST** **RANDOM NORMAL** (loc**<-**mean,scale**<-**stddv,size**<-**(**LENGTH** array, **LENGTH** array[0]))

**FOR** i **<- 0 TO LENGTH** array

**FOR** j **<- TO LENGTH** array[0]

                array[i][j] +**=** noise[i][j]

**RETURN** array

**SUB** probprint(self)

**FOR** i <- 0 **TO** **LENGTH** probability -1

**FOR** j <- 0 **TO LENGTH** probability - 1

**OUTPUT ROUND ROUND** (probability[i][j], 3) \* 100, 3

**OUTPUT NEWLINE**

**SUB** diffuse(self, step)

**IF** **NOT ISINSTANCE** (step, **INTEGER**)

**RETURN** **FALSE**

        newarray **<-** **LIST**

        probability **<-** applygauss2d(probability,step)

**FOR** i **<- 0 TO** **LENGTH** probability

**FOR** j **<- 0 TO LENGTH** probability[0]

**APPEND** probability[i][j], newarray

        newarray **<-** \_softmax(newarray)

        n **<-** **LENGTH** self.probability[0]

        self.probability **<-** [newarray[idx:idx+n] **FOR** idx **<- 0 TO LENGTH** newarray, n)]

**SUB** setElement(self, index, value)

**IF** **LENGTH** Qbit.vector **=** 2 **THEN**

**TRY ASSERT** value <**=** 1 **AND** value >**=** -1

**EXCEPT**

**OUTPUT** "Value can only take the range [-1,1]"

**RETURN** **FALSE**

        Qbit.vector[index] **<-** value

**RETURN** **TRUE**

**Tree**

**CLASS** Tree

**SUB** \_\_init\_\_ (self, name**<-**'root', children**<-NONE**)

**PUBLIC** name **<-** name

**PUBLIC** children **<-** **LIST**

**IF** children **IS NOT NONE THEN**

**FOR** child **IN** children:

                add\_child(child)

**SUB** add\_child(self, node)

**ASSERT** **ISINSTANCE** node, Tree

**APPEND** node children

**SUB** \_\_repr\_\_(self)

**RETURN** name

**Draggable**

**CLASS** Draggable

    lock **<-** **NONE**

**SUB** \_\_init\_\_ (self, point, update, object)

**PUBLIC** point **<-** point

**PUBLIC** press **<-** **NONE**

**PUBLIC** background **<-** **NONE**

**PUBLIC** update **<-** update

**PUBLIC** object **<-** object

**SUB** connect(self)

        cidpress **<-** point.figure.canvas.mpl\_connect('button\_press\_event', on\_press)

        cidrelease **<-** point.figure.canvas.mpl\_connect('button\_release\_event', on\_release)

        cidmotion **<-** point.figure.canvas.mpl\_connect('motion\_notify\_event', on\_motion)

**SUB** on\_press (self, event)

**IF** event.inaxes <> point.axes:

**RETURN** **NONE**

**IF** Draggable.lock is **NOT** **NONE**

**RETURN** **NONE**

        contains, attrd **<-** point.contains(event)

**IF** **NOT** contains:

**RETURN** **NONE**

        press **<-** (point.center), event.xdata, event.ydata

        Draggable.lock **<-** self

        canvas **<-** point.figure.canvas

        axes **<-** point.axes

        point.set\_animated(**TRUE**)

        canvas.draw()

        background **<-** canvas.copy\_from\_bbox(self.point.axes.bbox)

        axes.draw\_artist(point)

        canvas.blit(axes.bbox)

**SUB** on\_motion(self, event)

**IF** Draggable.lock **IS** **NOT** self:

**RETURN** **NONE**

**IF** event.inaxes <> point.axes:

**RETURN** **NONE**

        point.center, xpress, ypress **<-** press

        dx **<-** event.xdata - xpress

        dy **<-** event.ydata - ypress

        point.center **<-** (point.center[0]+dx, point.center[1]+dy)

        canvas **<-** point.figure.canvas

        axes **<-** point.axes

        canvas.restore\_region(background)

        axes.draw\_artist(point)

        canvas.blit(axes.bbox)

        object.x **<-** point.center[0]

        object.y **<-** point.center[1]

**SUB** on\_release(self, event)

**IF** Draggable.lock **IS** **NOT** self

**RETURN** **NONE**

        press **<-** **NONE**

        Draggable.lock **<-** **NONE**

        point.set\_animated(**FALSE**)

        background **<-** **NONE**

**UPDATE**()

        point.figure.canvas.draw()

**SUB** disconnect(self)

        point.figure.canvas.mpl\_disconnect(cidpress)

        point.figure.canvas.mpl\_disconnect(cidrelease)

        point.figure.canvas.mpl\_disconnect(cidmotion)

**CodeEditor**

**CLASS** CodeEditor(tk.Tk)

**SUB** \_\_init\_\_(self,interpreter, file\_open**<-NONE**)

**SUPER**.\_\_init\_\_()

**PUBLIC** title <- "Code Editor"

        darkstyle()

**PRIVATE** thisMenuBar **<-** Menu(self)

**PRIVATE** thisFileMenu **<-** Menu(thisMenuBar, tearoff**<-**0)

**PRIVATE** thisEditMenu **<-** Menu(thisMenuBar, tearoff**<-**0)

**PRIVATE** file **<-** **NONE**

**PRIVATE** interpreter **<-** interpreter

**PUBLIC** text\_widget **<-** tk.Text(self, wrap**<-**"word", undo**<-TRUE**, font**<-**("Calibri",16))

        text\_widget.pack(expand**<-TRUE**, fill**<-**"both")

**PRIVATE** thisScrollBar **<-** Scrollbar(text\_widget)

**PUBLIC** entry **<-** tk.Entry(self, width**<-**50,font**<-**("Calibri",16))

        entry.pack(side**<-**"bottom", fill**<-**"x")

        entry.bind("<Return>", handle\_enter)

        thisFileMenu.add\_command(label**<-**"New",command**<-**newFile)

        thisFileMenu.add\_command(label**<-**"Open",command**<-**openFile)

        thisFileMenu.add\_command(label**<-**"Save",command**<-**saveFile)

        thisFileMenu.add\_separator()

        thisFileMenu.add\_command(label**<-**"Exit",command**<-**quitApplication)

        thisMenuBar.add\_cascade(label**<-**"File",menu**<-**thisFileMenu)

        thisEditMenu.add\_command(label**<-**"Cut",command**<-** cut)

        thisEditMenu.add\_command(label**<-**"Copy",command**<-**copy)

        thisEditMenu.add\_command(label**<-**"Paste",command**<-**paste)

        thisMenuBar.add\_cascade(label**<-**"Edit",menu**<-**thisEditMenu)

        config(menu**<-**thisMenuBar)

        thisScrollBar.pack(side**<-**RIGHT,fill**<-**Y)

        thisScrollBar.config(command**<-**text\_widget.yview)

        text\_widget.config(yscrollcommand**<-**self.\_\_thisScrollBar.set)

        TAGDEFS   **<-** {   'COMMENT'    : {'foreground': "orange"  , 'background': **NONE**},

                'TYPES'      : {'foreground': "orange"   , 'background': **NONE**},

                'NUMBER'     : {'foreground': "orange"    , 'background': **NONE**},

                'BUILTIN'    : {'foreground': "orange"  , 'background': **NONE**},

                'STRING'     : {'foreground': "orange"   , 'background': **NONE**},

                'DEFINITION' : {'foreground': "orange"    , 'background': **NONE**},

                'INSTANCE'   : {'foreground': "orange"     , 'background': **NONE**},

                'KEYWORD'    : {'foreground': "orange", 'background': **NONE**},

            }

        cd         **<-** ic.ColorDelegator()

        cd.prog    **<-** re.compile(PROG, re.S | re.M)

        cd.idprog  **<-** re.compile(IDPROG, re.S)

        cd.tagdefs **<-** {\*\*cd.tagdefs, \*\*TAGDEFS}

        ip.Percolator(self.text\_widget).insertfilter(cd)

**IF** file\_open <> **NONE** **AND** **TYPE** file\_open **=** **STRING THEN**

            openFile(file\_name**<-**file\_open)

**SUB** handle\_enter(self, event)

        code\_line **<-** entry.get()

        code\_line **<-** **STRING** code\_line

        text\_widget.insert("end", code\_line + "\n")

        entry.delete(0, "end")

**INTERPRET** code\_line

**SUB** \_\_newFile(self):

        Title <- "Untitled - Code editor"

file **<-** **NONE**

        text\_widget.delete(1.0,END)

**SUB** darkstyle(self):

        style **<-** ttk.Style(self)

        tk.call('source', r"C:\Users\OSINT\OneDrive\Documents\GitHub\NEA\azuredark\azuredark.tcl")

        style.theme\_use('azure')

**RETURN** style

**SUB** \_\_openFile(self, file\_name**<-NONE**)

**IF** file\_name **=** **NONE**

           file **<-** askopenfilename(defaultextension**<-**".txt",filetypes**<-**[("All Files","\*.\*"),("Text Documents","\*.txt")])

**ELSE THEN**

            file **<-** file\_name

**IF** file **=** ""

            file **<-** **NONE**

**ELSE THEN**

**TRY** title <- **PATH** file + " - Code editor"

**EXCEPT** **RETURN** **FALSE**

            text\_widget.delete(1.0,END)

            file **<-** **OPEN** file, read

            text\_widget.insert(1.0,file.read())

**CLOSE** file

        recents **<-** open(r"C:\Users\OSINT\OneDrive\Documents\GitHub\NEA\recents.txt","a")

**APPEND STRING PATH** file, recents

**CLOSE** recents

**SUB** \_\_cut(self)

        text\_widget.event\_generate("<<Cut>>")

**SUB** \_\_copy(self)

        self.text\_widget.event\_generate("<<Copy>>")

**SUB** \_\_paste(self):

        text\_widget.event\_generate("<<Paste>>")

**SUB** \_\_saveFile(self)

**IF** file **=** **NONE** **THEN**

            file **<-** asksaveasfilename(initialfile**<-**'Untitled.txt',defaultextension**<-**".txt",filetypes**<-**[("All Files","\*.\*"),("Text Documents","\*.txt")])

**IF** file **=** ""

                file **<-** **NONE**

**ELSE THEN**

                file **<-** **OPEN** file, write

**WRITE** text\_widget.get(1.0,END), file

**CLOSE** file

                Title <- **PATH** file + " - Code editor"

**ELSE THEN**

            file **<-** **OPEN** file, write

**WRITE** text\_widget.get(1.0,END), file

**CLOSE** file

        recents **<-** open(r"C:\Users\OSINT\OneDrive\Documents\GitHub\NEA\recents.txt","a")

**APPEND STRING PATH** file, recents

**CLOSE** recents

**SUB** \_\_quitApplication(self)

**EXIT**

**Interpreter**

**CLASS** Interpreter

**SUB** \_\_init\_\_(self, graphqbit)

**PUBLIC** lex **<-** Lexer(rules)

**PUBLIC** command\_list **<-** **LIST**

**PUBLIC** user\_vars **<-** **DICT**

**PRIVATE** temp\_vars **<-** **DICT**

**SUB** interpret(self, line)

**FOR** token **IN** lex.scan(line):

**APPEND** token command\_list

**FOR** count, element **IN** command\_list

**SWITCH** element[0]

**CASE** "SUPPLIMENT"|"END\_STMNT":

**REMOVE** element, command\_list

**CASE** "OBJECT":

parameters <- **SPLIT STRIP SPLIT** element[1]

                    function **<-** **SPLIT** element[1], 0

**FOR** element **IN** parameters

                        identifier **<-** **JOIN** “ ” **RANDOM LETTER** **FOR** i **<-** 0 **TO** 19

**WHILE** identifier **IN** temp\_vars

                                identifier **<-** **JOIN** “ ” **RANDOM LETTER** **FOR** i **<-** 0 **TO** 19                        temp\_vars[identifier] **<-** **ID** identifier

                        parameters[element] **<-** **ID** identifier

                    command\_list[count] **<-** (function, parameters)

**CASE** \_:

**PASS**

        setvars()

        self.ast **<-** AbstractSyntaxTree(command\_list)

**SUB** \_\_setvars(self)

**FOR** element **IN** command\_list

**IF** command\_list[element][0] **=** "LITERAL" **OR** command\_list[element][0] **=** "DIGIT":

            identifier **<-** **JOIN** “ ” **RANDOM LETTER** **FOR** i **<-** 0 **TO** 19

**WHILE** identifier **IN** temp\_vars

                identifier **<-** **JOIN** “ ” **RANDOM LETTER** **FOR** i **<-** 0 **TO** 19            temp\_vars[identifier] **<-** **ID** identifier

                command\_list[element] **<-** **ID** identifier

**Lexer**

**CLASS** Lexer

**SUB** \_\_init\_\_(self, rules, case\_sensitive**<-TRUE**, omit\_whitespace**<-TRUE**)

**PROTECTED** callbacks **<-** **DICT**

**PUBLIC** omit\_whitespace **<-** omit\_whitespace

**PUBLIC** case\_sensitive **<-** case\_sensitive

        parts **<-** **LIST**

**IF** **NOT** **ISINSTANCE** rules, **LIST**

**RAISE** TypeError("'Rules' needs to be an iterable")

**IF NOT** **ISINSTANCE** case\_sensitive, **BOOL**

**RAISE** TypeError("Case flag needs to be a Boolean value due to python interpretation of strings")womp womp

**FOR** name, rule **IN** rules

**IF** **NOT** **ISINSTANCE** rule, **STRING**:

                rule, callback **<-** rule

                callbacks[name] **<-** callback

**APPEND** ("(?P<%s>%s)" % (name, rule)), parts

**IF** case\_sensitive **THEN**

            flags **<-** re.M

**ELSE THEN**

            flags **<-** re.M | re.I

        regexc **<-** re.compile("|".join(parts), flags)

        ws\_regexc **<-** re.compile("\s\*", re.MULTILINE)

**SUB** scan(self, inp)

**IF** **TYPE** inp **= STRING**

**RETURN** \_InputScanner(self, inp)

**ELSE THEN**

            inp **<-** **STRING** inp

**RETURN** \_InputScanner(self, inp)

**\_InputScanner**

**CLASS** \_InputScanner(object)

**SUB** \_\_init\_\_(self, lexer, inp)

**PRIVATE** position **<-** 0

**PUBLIC** lexer **<-** lexer

**PUBLIC** input **<-** inp

**SUB** \_\_iter\_\_(self)

**RETURN** self

**SUB** \_\_next\_\_(self)

**IF** **NOT** done\_scanning()

**RETURN** scan\_next()

**RAISE** StopIteration

**SUB** done\_scanning(self)

**RETURN** position >**=** **LENGTH** input

**SUB** scan\_next(self)

**IF** done\_scanning()

**RETURN** **NONE**

**IF** lexer.omit\_whitespace

            match **<-** lexer.ws\_regexc.match(input, position)

**IF** match

                position **<-** match.end()

        match **<-** lexer.regexc.match(input, position)

**IF** match **IS** **NONE**:

            lineno **<-** input[:position].count("\n") + 1

**RAISE** UnknownTokenError(input[position], lineno)

        position **<-** match.end()

        value **<-** match.group(match.lastgroup)

**IF** match.lastgroup **IN** lexer.\_callbacks

            value **<-** lexer.\_callbacks[match.lastgroup](self, value)

**RETURN** match.lastgroup, value

**UnknownTokenError**

**CLASS** UnknownTokenError(Exception)

**SUB** \_\_init\_\_(self, token, lineno)

**PUBLIC** token **<-** token

**PUBLIC** lineno **<-** lineno

**SUB** \_\_repr\_\_(self)

**RETURN** "Line #%s, Found token: %s" % (lineno, token)

**Login**

**CLASS** Login

**SUB** \_\_init\_\_(self,username**<-NONE**, password**<-NONE**)

**PRIVATE** username **<-** username

**PRIVATE** password **<-** password

**PRIVATE** userid **<-** **NONE**

**PRIAVTE** authenticated **<-** **FALSE**

        first **<-** **FALSE**

**TRY** conn **<-** connect("file:master.db?mode**<-**rw", uri**<-TRUE**)

**EXCEPT** conn, first **<-** connect("master.db"), **TRUE**

        c **<-** conn.cursor()

**IF** first **=** **TRUE**

            firsttime()

**ELSE IF** **NOT**((username **IS** **NONE**) **AND** (password **IS** **NONE**))

**IF** login()

                authenticated **<-** **TRUE**

**ELSE THEN**

**OUTPUT** "Invalid username/password combination"

**EXIT** 1

**ELSE THEN**

            register()

        conn.close()

**SUB** \_\_firsttime(self):

        c.execute('''CREATE TABLE users (userid INTEGER PRIMARY KEY, username UNIQUE NOT NULL, hash TEXT)''')

        conn.commit()

        Setup()

        register()

**SUB** \_\_userlookup(self)

**RETURN** **BOOL STRING FETCHALL** c.execute(f"SELECT username FROM users WHERE username**<-**'{username}'") **=** "[(1,)]"

**SUB** \_\_login(self)

        password **<-** **HEXDIGEST SHA256 ENCODE** password

        userid **<-** **STRING FETCHALL** c.execute(f"SELECT userid FROM users WHERE username **<-**'{username}' AND hash **<-**'{password}'")

**IF** userid **=** "[(1,)]"

            userid **<-** userid

**RETURN** userid

**ELSE THEN**:

**RETURN** **FALSE**

**SUB** loggedin(self)

**RETURN** authenticated

**SUB** \_\_validPassword(self)

**RETURN** **FULLMATCH** (r"/(?**<-**.\*\d.\*)(?**<-**.\*[a-zA-Z].\*).{8,}/gm", password) **=** **NONE**

**SUB** register(self)

        self.\_\_username **<-** **USERINPUT**("\nEnter a username to register: ")

        usernamecheck **<-** **STRING FETCHALL** c.execute(f"SELECT COUNT(\*) FROM users WHERE username**<-**'{username}'")

**WHILE** usernamecheck **<>** "[(1,)]"

**OUTPUT** "Username in use. Please pick another")

            self.\_\_username **<-** **USERINPUT**("Enter a different username to register: ")

            usernamecheck **<-** **STRING FETCHALL** c.execute(f"SELECT COUNT(\*) FROM users WHERE username**<-**'{username}'")

        self.\_\_password **<-** **GETPASS**("Enter the password you want to use for this account: ")

**WHILE** validPassword() <> **TRUE**

            self.\_\_password **<-** **GETPASS**("Please enter a different password, Check it meets all the requirments (8 characters, at least one uppercase letter, lowercase letter and number must be present): ")

hash <- **HEXDIGEST SHA256 ENOCDE** password

        c.execute(f'''INSERT INTO users(username,hash) VALUES(?,?)''',(username,hash))

        conn.commit()

**RETURN** **TRUE**

**SUB** getuserid(self)

**RETURN** userid

**Point**

**@dataclass**

**CLASS** Point

**PUBLIC FLOAT** x

**PUBLIC FLOAT** y

**PUBLIC FLOAT** size

**PUBLIC FLOAT** tens

**Renderer**

**CLASS** Renderer

**SUB** \_\_init\_\_ (self, system, XMAX, YMAX, density, rx, ry)

**PUBLIC** system **<-** system

**PUBLICS** XMAX, YMAX **<-** XMAX, YMAX

**PUBLIC** density **<-** density

**PUBLICS** rx, ry **<-** rx, ry

**SUB** launch(self)

        figure, ax **<-** subplots()

**UPDATE**

        ax.set\_xlabel('$x$')

        ax.set\_ylabel('$y$')

        ax.set\_xlim(-XMAX, XMAX)

        ax.set\_ylim(-YMAX, YMAX)

        ax.set\_aspect('equal')

**SHOW**

**SUB** update(self)

**CLEAR**

**DFIELD**

**DPOINTS**

**DWALLS**

**SUB** clear(self):

        ax.cla()

**SUB** dfield(self)

        x **<-** **LINSPACE** (-XMAX, XMAX, rx)

        y **<-** **LINSPACE** (-YMAX, YMAX, ry)

        X, Y **<-** **MESHGRID** (x, y)

        V **<-** system.field(X, Y)

        [Ex, Ey] **<-** **GRADIENT** (V, rx, ry)

**IF** **LENGTH** Ex **AND** **LENGTH** Ey

            ax.streamplot(x, y, Ey, Ex, color**<-**(2\***LOG** **HYPOT**(Ex, Ey))), linewidth**<-**1, cmap**<-**plt.cm.inferno, density**<-**density, arrowstyle**<-**'->', arrowsize**<-**1.5)

            ax.matshow(V, interpolation**<-**'nearest', alpha**<-**1, cmap**<-**plt.cm.plasma, extent**<-**(-XMAX, XMAX, YMAX, -YMAX))

**SUB** dpoints(self):

        draggables **<-** **LIST**

**FOR** point **IN** system.points:

            circle **<-** **CIRCLE** ((point.x, point.y), point.size, color**<-**plt.cm.RdBu(mpl.colors.Normalize(vmin**<-**-10, vmax**<-**10)(-point.tens)), zorder**<-**100)

            ax.add\_patch(circle)

            draggable **<-** **DRAGGABLE** (circle, update, point)

**CONNECT** draggable

**APPEND** draggable, draggables

**SUB** dwalls(self)

**FOR** wall **IN** system.walls:

            ax.plot([wall.x1, wall.y1], [wall.x2, wall.y2], marker **<-** 'o')

**System**

**CLASS** System

**SUB** \_\_init\_\_ (self, epsilon, gamma):

**PUBLIC** points **<-** **LIST**

**PUBLIC** walls **<-** **LIST**

**PUBLIC** epsilon **<-** epsilon

**PUBLIC** gamma **<-** gamma

**SUB** addPoint(self, point)

**APPEND** point, points

**SUB** addWall(self, wall):

**APPEND** wall, walls

**SUB** compute (self, i, X, Y, R, U)

        I **<-** (U \* gamma \* 2 \* Pi)

        Sigma **<-** (I \* epsilon) / (gamma \* 4 \* Pi \* **POW** R, 2)

        dist **<-** **SQRT** (**POW**(Y[i], 2) + **POW**(X[i], 2))

**IF** dist < R

**RETURN** ((Sigma \* **POW**(R, 2))/epsilon)\*(1/R)

**ELSE**

**RETURN** ((Sigma \* **POW**(R, 2))/epsilon)\*(1/dist)

**SUB** field(self, X, Y)

        u, v **<-** **SHAPE** X

        size **<-** **SIZE** X

        X.shape **<-** (size)

        Y.shape **<-** (size)

        V **<-** **ZEROS** ((u, v))

**FOR** point **IN** points:

            tX **<-** [X[i]-point.x **FOR** i <- 0 **TO** size]

            tY **<-** [Y[i]-point.y **FOR** i <-0 **TO** size]

            E **<-** **ARRAY** [compute(i, tX, tY, size, tens) **FOR** i <- 0 **TO** size]

            E.shape **<-** (u, v)

            V **<-** V + E

**RETURN** V

**Wall**

**@dataclass**

**CLASS** Wall

**PUBLIC FLOAT** x1

**PUBLIC FLOAT** y1

**PUBLIC FLOAT** x2

**PUBLIC FLOAT** y2

## **Object orientation plan**

**Tree**

The tree class contains an implementation for a generic tree data structure. It is based on the principle of Nodes and recursive generation – this means that each node is implemented as a tree without any children. Generating the tree structure as a whole is as simple as adding existing trees to the current root as children. This recursive structure makes the code much simpler and shorter than other tree implementations. This is an example of a generic tree, because each node can have as many children as it would like (although it wouldn’t be hard to transform it into a binary tree using dunder methods such as \_\_setattr\_\_)

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.name | public | Any | ‘root’ | The data stored inside each node is saved here |
| self.children | public | list | [] | contains the list of children objects belonging to the current instance |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, name, children | None | Initialises the Tree class, with the given parameters |
| add\_child | public | self, node | None | Adds a child node to the current referenced Tree object |
| \_\_repr\_\_ | public | self | self.name: Any | Returns a machine-readable description of the current instance |

**Node**

The node class is a simple recursive generating binary tree data structure, with appropriate implementations of different tree traversal algorithms. Recursive tree data structures are the most popular because they are inherently simple to implement. Once you can implement one node, then you can implement an exponentially larger tree by recursively generating node objects and linking them. Ultimately, I decided that this wasn’t the right fit for my project due to the difficulty with reading an expression into the tree (this is a complex problem to solve and has many different proposed solutions, all of which are beyond the scope of my project, the aim isn’t to create a new programming language after all). This combined with the added restriction that the input data must be an int or float value means that this particular binary tree implementation fell out of favour.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.left | public | integer | None | None | Sets up the Node object with values to the left of the current instance |
| self.right | public | integer | None | None | Sets up the Node object with values to the right of the current instance |
| self.data | public | integer | None | Parameter | Contains the data used for comparison in tree placement and traversal |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, data | None | Initialises the Node object with the provided parameters |
| insert | public | self, data | None | Adds a new Node object to the current root instance in the appropriate place |
| PreorderTraversal | public | self, root | res: list | Performs preorder traversal on the provided root Node object |
| InorderTraversal | public | self, root | res: list | Performs inorder traversal on the provided root Node |
| postorderTraversal | public | self, root | res: list | Performs postorder traversal on the provided root Node |

**AbstractSyntaxTree**

The AbstractSyntaxTree class is used for the main parsing of inputted commands. This is useful because the implementation uses linked lists: one of the most efficient ways of solving this problem. Using this list-built structure allowed me much greater control over the internals as there is a wider array of documenting that exists for lists on the python website and in the PEP styling guides. What this meant for my project is that I could use more of the thousands of pre-implemented functions that can be used on iterables in python, rather than having to implement them myself in a fresh class. This saved me a lot of time as was ultimately more memory efficient as python’s lists are very well optimised by this point in time.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.\_\_commandtree | private | list | [] | The linked-list data structure that will hold the final tree implementation |
| self.\_\_parsed | private | list | parameter | the current list of tokenised user commands |
| self.\_\_commandlist | private | list | [] | The list that temporarily holds the parsed commands |
| self.var\_lookup | public | dictionary | parameter | holds the dictionary of user variables, so that they can be easily retrieved and dereferenced when not in use |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, parsed, variables | None | Initialises the tree object and performs variable casting to convert from reference to identifier, then places these into the tree using standard linked-list rules. |
| in\_order\_traversal | public | self, current\_index, datastream | datastream: list | returns a list representation of the tree after performing inorder traversal |

**Cbit**

The Cbit class is used to represent classical bits with a quantum system in mind. This forms the base class of the Qbits that are used throughout the program and by the user. They are functionally equivalent to bits on a regular computer with different format styling to allow for method overriding in the qbit class that will provide our program the key functionality that separates it as a quantum computation simulator. The structure of the cbit is set up to make conversion into qbits as painless as possible. Most of the time the cbit class is used purely like a function prototyping system for the qbit class (almost like a middleman for connecting the properties of a vector object, with the functionality of a quantum state). Whilst being publicly available, it will be rarely touched by the user.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.Cbit | public | Vector | None | None | The variable that holds the underlying vector class instance that powers the c and qbit system |
| self.\_\_sub | private | integer | parameter | The variable that holds the subscript that would be present in traditional bra-ket notation of quantum states – loosely speaking, holds the maximum length before a tensor product is required |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, dirac, sub | None | Initialises the cbit object by formatting parameters then initialising vector object |
| setElement | public | self, index, value | True | False | Sets the element at the index specified to the value specified |
| measure | public | self | False | self.Cbit.vector[1] | Checks the second position in a 1x2 vector to evaluate the classical value – this is because in vector form this bit can be used to identify the value |
| probcollapse | public | self | None | displays a printout to the user showing them the probability breakdown of each state (in this case, classically) |
| \_\_repr\_\_ | public | self | string form vector | Returns a machine-readable representation of the cbit object |

**Setup**

The setup class is used to aid the first-time program run. It aids the setup of all of the various database and JSON objects as well as adding in any default data or required fields. Some extra commands have to be executed here so that sqlite3 understands the structure of our database: for example, *“PRAGMA foreign\_keys = 1”* needsto be run because sqlite3 doesn’t automatically search for foreign key references and won’t understand our table without it. This is just one of many minor inconveniences present in sqlite3 (another example of something you would expect it to do that it doesn’t, would be type enforcement, sqlite3 really doesn’t care what type you set a field up as). Despite these complications, they never became too large to be more of a hassle then learning a new database connection system and then installing it into python.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.\_\_conn | private | connection object | N/A | sqlite3 database connection object, initialised so we can interact with our database file |
| self.\_\_c | private | cursor object | self.\_\_conn.cursor | sqlite3 cursor object so we can execute commands in our database |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self | None | Used to create our database file and create each table with the appropriate fields |
| \_\_populateDB | private | self | None | Used to fill our challenges database with input as this won’t be done by the user (in this iteration at least) |

**Draggable**

The draggable class is used to create an object that can be moved around in a matplotlib window. This class is utilised by the renderer to create an interactive field simulator. This class controls the logic for connecting, moving and disconnecting an object.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.point | public | object | parameter | The point variable holds point object data |
| self.press | public | tuple | None | None | The press variable holds the data from the user click interaction |
| self.background | public | callable | None | None | The background variable holds background information.[[1]](#footnote-2) |
| self.update | public | callable | parameter | This variable holds the reference to the renderer clear function |
| self.object\_selected | public | object | parameter | This variable holds the reference to the object that is being made into a draggable object |
| self.cidpress | public | callable | N/A | This variable holds the reference to the connection of the press function |
| self.cidrelease | public | callable | N/A | This variable holds the reference to the connection of the release function |
| self.cidmotion | public | callable | N/A | This variable holds the reference to the connection of the motion function |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, point, update, object\_selected | None | Initialises the draggable object |
| connect | public | self | None | Connects the motion functions to the draggable object |
| on\_press | public | self, event | None | Sets up the selected object for being moved |
| on\_motion | public | self, event | None | Draws the motion of the selected object |
| on\_release | public | self, event | None | Places the selected object in its new location and updates its stored position |
| disconnect | public | self | None | Disconnects the motion functions from the draggable object |

**Gates**

The Gates class is an Enumerator object that hold constant values in python. It is a special kind of dataclass that can be used to store collections of immutable related items: in this case it is storing the matrix representation of quantum logic gates. Using the matrix forms is beneficial because matrix multiplication is a much easier operation that some of the other forms that could be used. It is also clearer to other computer scientists and is specifically designed for the vector notation of qbits, which we use extensively as it is the most applicable to our system. Vector form qbits are a great solution for simulators and similar programs as they can utilise a larger memory pool but transfer the same information: They are the standard representation of quantum states in a classical device. As a dataclass derivative, the Gates class has no methods.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| HADAMARD | public | list | [[,],  [,-]] | Contains the Hadamard gate in matrix form. This gate puts a qbit into a superposition state and is used frequently |
| PAULI\_X | public | list | [[0,1],  [1,0]] | Contains the Pauli X gate, also known as the Not gate. Functions the same as the classical not gate |
| PAULI\_Y | public | list | [[0, -1j],  [1j, 0]] | Contains the Pauli Y gate, which shifts the qbit without affecting its magnitude |
| PAULI\_Z | public | list | [[1, 0],  [0, -1]] | Contains the Pauli Z gate, which acts as a reflection of the qbit |
| PHASE | public | list | [[1,0],  [0,1j]] | Contains the phase gate |
| T | public | list | [[1,0],  [0,]] | contains the T gate, a special shift gate |
| CNOT | public | list | [[1,0,0,0],  [0,1,0,0],  [0,0,0,1],  [0,0,1,0]] | contains the controlled not gate, which applies a not operation on a second state only if the first state is 1 |
| CZ | public | list | [[1, 0, 0, 0],  [0, 1, 0, 0],  [0, 0, 1, 0],  [0, 0, 0, -1]] | contains the controlled Z gate which applies a Z gate on a state if the first state is a 1 |
| SWAP | public | list | [[1,0,0,0],  [0,0,1,0],  [0,1,0,0],  [0,0,0,1]] | swaps the states of two qbits |
| TOFFOLI | public | list | [[1,0,0,0,0,0,0,0], [0,1,0,0,0,0,0,0], [0,0,1,0,0,0,0,0], [0,0,0,1,0,0,0,0], [0,0,0,0,1,0,0,0], [0,0,0,0,0,1,0,0], [0,0,0,0,0,0,0,1], [0,0,0,0,0,0,1,0]] | A controlled controlled not gate, CCNOT. Used so commonly it has its own name. Also, can form a basis of a universal set of quantum logic gates. |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
|  |  |  |  |  |

**CodeEditor**

The CodeEditor class controls one of the main GUI elements that the user interacts with. It is responsible for accepting commands from the user and then handling them and piping them to the interpreter. The CodeEditor is a large class because it has to handle any meta operations that the user may want, outside of standard code execution (such as the copy and paste clipboard being connected to the program). It also has to handle the saving and opening of files. The interface file that it exists in is one of the most important files as - aside from the interpreter - it connects the most parts of the program together.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.recents | public | string | URL | Contains the URL for the recents text file, which holds information about which code files have been edited recently |
| self.title | public | string | “Code Editor” | This variable holds the tkinter title attribute information |
| self.\_\_thisMenuBar | private | object | N/A | This variable holds the reference to the menu bar object |
| self.\_\_thisFileMenu | private | object | N/A | This variable holds the reference to the file menu object |
| self.\_\_thisEditMenu | private | object | N/A | This variable holds the reference to the edit menu object |
| self.\_\_file | private | string | None | This variable holds the reference to the current open file |
| self.\_\_interpreter | private | object | parameter | This variable holds the reference to the supplied interpreter instance. This means that on a technicality, multiple editor instances can be run with multiple interpreter instances simultaneously |
| self.text\_widget | public | object | N/A | This variable holds the reference to the text widget object |
| self.\_\_thisScrollBar | private | object | N/A | This variable holds the reference to the scroll bar object |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, interpreter, file\_open | None | Initialises the CodeEditor class and sets up the variables as well as initialising custom syntax highlighting for tkinter windows |
| handle\_enter | public | self, event | None | This function handles input taken from the user and then pushes it into the interpreter to decode and execute instructions |
| \_\_newfile | private | self | None | This function wipes the workspace and makes a new file |
| darkstyle | public | self | style | This function changes the current workspace into dark mode |
| \_\_openfile | private | self, file\_name | None | This function opens a file |
| \_\_cut | private | self | None | This function manages cutting |
| \_\_copy | private | self | None | This function manages copying |
| \_\_paste | private | self | None | This function manages pasting |
| \_\_savefile | private | self | None | This function saves a file |
| \_\_quitApplication | private | self | None | This file quits the tkinter interface |

**Interpreter**

The interpreter class is responsible for running the commands given to it by the user through the interface. To do this it uses the lexer class and the AbstractSyntaxTree class as well as some regex pattern matching. The interpreter class is also responsible for some other functionality, such as searching the challenge database and pattern matching to determine when achievements should be given to the user.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.lex | public | object | N/A | Creates the lexer object and links it to the project’s interpreter, then stores the reference to this object in the variable lex |
| self.command\_list | public | list | [] | This variable stores the tokenised list of commands before parsing |
| self.user\_vars | public | dictionary | {} | This variable stores long-term user variables that may need to be accessed more than once |
| self.\_\_temp\_vars | private | dictionary | {} | This variable stores the identifiers and addresses of temporary variables so that these can be cleared from the memory after each cycle, thus reducing overall memory usage |
| self.\_\_command\_string | private | string | “” | This variable stores the unparsed command list in string form for ease of regex matching. |
| self.graph\_instance | public | object | parameter | This variable stores the reference to the qbit object that is being used as the basis of graphic program elements |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, graphqbit | None | This function initialises the interpreter class and sets up all of the different variables that are needed in the process of interpreting a line of code |
| interpret | public | self, line | None | This function is responsible for interpreting and executing a line of code. It runs the second stage of input decoding as well as executing. It also uses the built-in garbage collector to deallocate memory and free up space that is no longer used |
| \_\_giveaward | private | self | False | None | This function is responsible for scanning the users input to identify achievement completions |
| \_\_setvars | private | self | None | This function is used to set up the temp\_vars dictionary to enable the memory efficient interpreting |

**UnknownTokenError**

The UnknownTokenError class inherits from the built-in Exception class which allows us to raise it as an exception during python runtime. This gives us greater precision over the degree and nature of the error messages presented to the user, allowing for a finer level of interaction with the user.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.token | public | string | parameter | Refers to the offending token |
| self.lineno | public | integer | parameter | Refers to the line number with the bad token |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, token, lineno | None | Initialises the new exception type |
| \_\_repr\_\_ | public | self | Line and token | Returns a machine-readable representation of the exception object |

**\_InputScanner**

The InputScanner class is used to introduce the lexer to the input string in a safe environment so that the input is handled with care and doesn’t cause any unexpected behaviour on runtime. The InputScanner facilitates a complete encapsulation of the actual parsing process and runs in conjunction with the other processes in the lexer file such that the system is safe and protected from any bad input that may be given to – minimising and containing damage to a small part of the program.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.\_position | protected | integer | 0 | Holds the current position in the line |
| self.lexer | public | object | parameter | This variable holds the reference to the lexer object |
| self.input | public | string | parameter | This variable holds the input line |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, lexer, inp | None | Initialises the InputScanner class and sets up the references to the lexer object |
| \_\_iter\_\_ | public | self | self | Syntactic sugar so that python treats the lexer as a generator/iterable as well as an object |
| \_\_next\_\_ | public | self | tuple | None | Syntactic sugar |
| done\_scanning | public | self | True | False | Returns whether or not the InputScanner has reached the end of the program input |
| scan\_next | public | self | None | tuple[match, integer] | Facilitates the regex scanning of the input string and raises an error if it encounters an error parsing the input. |

**Lexer**

The lexer class is responsible for tokenising the user input. It is useful for parsing the commands such that they can be stored efficiently and retrieved for execution. It takes in an input and a set of rules based on regular expressions. It then scans the input and returns the tokens one-by-one. It is meant to be used through iterating.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.\_callback | protected | dictionary | {} | This variable stores exceptions, before being sent to traceback or handled with a separate function |
| self.omit\_whitespace | public | boolean | parameter | This variable determines whether or not regex checks whitespace |
| self.case\_sensitive | public | boolean | parameter | This variable determines whether or not the regex searches are case sensitive |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, rules, case\_sensitive, omit\_whitespace | None | Initialises the lexical scanner and sets up the regex searcher |
| scan | public | self, inp | \_Inputscanner object | Returns an instance of the input scanner |

**Point**

The Point class is a dataclass responsible for storing information about points overlayed onto a matplotlib window. These are the objects that we move around interactively in the renderer frame – they get passed into classes such as draggable.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.x | public | float | parameter | This variable stores x position values |
| self.y | public | float | parameter | this variable stores y position values |
| self.size | public | float | parameter | This variable stores ray information |
| self.tens | public | float | parameter | This variable stores tension information |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
|  |  |  |  |  |

**Login**

The login class is used to control user authentication in my program. It also helps to set up the required database and other files on the programs first-time run. It uses regex to validate user login information. It is the first interactive class that the user comes across when running the program.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.\_\_username | private | string | parameter | None | The current user’s username information |
| self.\_\_password | private | string | parameter | None | The current user’s password information |
| self.\_\_userid | private | None | integer | None | The userid of the current user |
| self.\_\_authenticated | private | boolean | False | The login state of the current user |
| self.\_\_conn | private | object | N/A | Holds the reference to the database connection |
| self.\_\_c | private | object | N/A | Holds the reference to the database cursor object so we can interact with the database |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, username, password | None | Initialises the login system, and calls all relevant functions given the parameter input |
| \_\_firsttime | private | self | None | Performs some set up for the first-time run of the program, including initialising databases |
| \_\_userlookup | private | self | True | False | Checks to see if the provided username already exists in the database |
| \_\_login | private | self | False | userid | This function performs the actual login of the user |
| loggedin | public | self | True | False | This function gets whether or not the user is logged in |
| \_\_validpassword | private | self | True | False | This function performs regex scan and checks if the password meets the minimum requirements |
| register | public | self | True | Registers the user in the database |
| getuserid | private | self | userid | returns the userid |

**Qbit**

The qbit class is a critical class in my solution. It handles all of the functionality of the quantum states in my program and is built as a subclass of the classical bit vector implementation. It works from a general vector object bundled with the logic of quantum bits for control.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.Qbit | public | object | Cbit | The basic Cbit object that our Qbit derives from |
| self.probability | public | list | [[0,0,0,0,0,0,0,0,0,0],  [0,0,0,0,0,0,0,0,0,0],  [0,0,0,0,0,0,0,0,0,0],  [0,0,0,0,0,0,0,0,0,0],  [0,0,0,0,0,0,0,0,0,0],  [0,0,0,0,0,0,0,0,0,0],  [0,0,0,0,0,0,0,0,0,0],  [0,0,0,0,0,0,0,0,0,0],  [0,0,0,0,0,0,0,0,0,0],  [0,0,0,0,0,0,0,0,0,0]] | The probability grid that our matplotlib graph view is based on |
| self.\_\_changed | private | list | [[]] | A variable showing what values in our probability grid have been altered |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, dirac, sub | None | Qbits are the quantum extension of Cbits that can take values intermediate of 0/1 |
| measure | public | self | False | collapse | 'Measures' the Qbits state using the probabilistic definition of quantum bits |
| \_softmax | protected | self, vector | False | list | Performs the softmax normalisation on a list of values. |
| \_normalise | protected | self, vector, maxprime | False | list | Min-max normalisation of given list |
| \_applyguass2d | protected | self, array, n | array | False | Applies gaussian noise (centred on the standard normal Z distribution) to a 2d array |
| probprint | public | self | None | Pretty-prints the probability grid |
| diffuse | public | self, step | False | None | Diffuse the probability grid over time. A representation of uncertainty in our visualisation |
| setElement | public | self, index, value | True | False | Sets the value at one index in the vector to the given value |

**Renderer**

The renderer class is used to set up interactivity for our matplotlib window. It provides an overlay to a standard frame that we can draw on and move objects around.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.system | public | object | parameter | References the system object |
| self.XMAX | public | float | parameter | Stores the maximum x value for the matplotlib window |
| self.YMAX | public | float | parameter | Stores the maximum y for the matplotlib window |
| self.density | public | float | parameter | Stores the density information of the system |
| self.rx | public | float | parameter | Stores the x point density value |
| self.ry | public | float | parameter | stores the y point density value |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, system, XMAX, YMAX, density, rx, ry | None | Initialises the renderer object |
| launch | public | self | None | start the renderer |
| update | public | self | None | update the renderer with new placement information, as well as the matplotlib window |
| clear | public | self | None | clear the renderer information and the matplotlib window |
| dfield | public | self | None | create a new field object |
| dpoints | public | self | None | create new points object |
| dwalls | public | self | None | create new wall object |

**System**

The system class is responsible for all of the electromagnetic calculations being performed. It calculates the correct values to be displayed by the renderer. It is the core of the matplotlib particle configuration view.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.points | public | list | [] | Stores the reference to a list of points objects |
| self.walls | public | list | [] | Stores the reference to a list of wall objects |
| self.epsilon | public | float | parameter | Stores the permittivity information |
| self.gamma | public | float | parameter | stores the conductivity information |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, epsilon, gamma | None | Initialises the system class |
| addPoint | public | self, point | None | links a point object reference to the system class |
| addWall | public | self, wall | None | links a wall object reference to the system class |
| compute | public | self, I, X, Y, R, U | Computed force | Computes the correct EM equations for system |
| field | public | self, X, Y | V | calculates the effects of the EM field |

**Vector**

The vector class is the model class that both the Cbits and Qbits are built from. It is the essence of the entire program as both the Quantum and classical logic is derived from a sub-class of this. The vector class implements all of the standard vector operations as well as introducing a standard type that is used throughout the rest of the program.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.vector | public | list | [0] \* size | Main vector object, implemented as a static array |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
| \_\_init\_\_ | public | self, size | None | Initialises the vector as a zero array of given size |
| getElement | public | self, index | self.vector[index] | False | Returns the value stored in the given index of the vector |
| setElement | public | self, index, value | True | False | Sets the value at one index in the vector to a given value |
| scalarMul | public | self, num | False | mulvec | Performs scalar multiplication on a vector |
| \_\_mul\_\_ | public | self, num | scalarMul | Scalar multiplication shorthand |
| setElements | public | self | True | Provides console interface to set all of the elements of the vector |
| setN | public | self, n | True | False | Shorthand, sets every element of the vector to the same number |
| allZeros | public | self | True | False | Returns true if every element of the vector is 0 |
| magnitude | public | self | magnitude | Returns the size of the vector using standard analytic geometry formula sqrt(a^2 + b^2...) |
| isUnit | public | self | True | False | Returns true if the current vector instance is an example of a unit vector |
| unit | public | self | unitvec | Creates a new vector object that is the unit of the current instance |
| tensor | public | self, other | False | tensorprod | Returns tensor product of two vectors |
| \_\_repr\_\_ | public | self | String vector | Returns human friendly version of object using more traditional curved brackets |

**Wall**

The wall class is a dataclass holding information about boundary points for the matplotlib window.

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute name** | **Access type** | **Data type** | **Initial value** | **Description** |
| self.x1 | public | float | parameter | Variable holds the first x co-ordinate |
| self.y1 | public | float | parameter | Variable holds the first y co-ordinate |
| self.x2 | public | float | parameter | Variable holds the second x co-ordinate |
| self.y2 | public | float | parameter | Variable holds the second y co-ordinate |

**Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method name** | **Access type** | **Parameters** | **Return value(s)** | **Description** |
|  |  |  |  |  |

## **Project hierarchy chart**

This section demonstrates function hierarchy from when the user first runs the program. It doesn’t display any built-in function calls or calls outside of the scope of my code. As of such only functions that I have written, and that get called by the system during the initial runtime of the program (with minimal user interaction). This is to provide some clarity on functions that may not get called by through the users’ actions and thus are more obscure/unknown. Alongside this I have provided a chart detailing function calling – this is to provide a quick reference for other developers in-case they need help navigating or tracing the code.

A diagram of a computer system

Description automatically generated

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function** | | | **Called by** | | |
| **File** | **Class** | **Function** | **File** | **Class** | **Function** |
| login.py | login | getuserid |  |  |  |
| login.py | login | register | login.py | login | \_\_init\_\_ |
| login.py | login | \_\_firsttime |
| login.py | login | \_\_validpassword | login.py | login | register |
| login.py | login | loggedin |  |  |  |
| login.py | login | \_\_login | login.py | login | \_\_init\_\_ |
| login.py | login | \_\_userlookup | login.py | login | \_\_init\_\_ |
| login.py | login | \_\_firsttime | login.py | login | \_\_init\_\_ |
| login.py | login | \_\_init\_\_ | main.py |  | main |
| main.py |  | main |  |  |  |
| main.py |  | maingraphloop | main.py |  | main |
| main.py |  | drawgraph | main.py |  | maingraphloop |
| lexer.py | lexer | scan | interpreter.py | interperter | interpret |
| lexer.py | lexer | \_\_init\_\_ | interpreter.py | interpreter | \_\_init\_\_ |
| lexer.py | \_inputscanner | scan\_next | lexer.py | \_inputscanner | \_\_next\_\_ |
| lexer.py | \_inputscanner | done\_scanning | lexer.py | \_inputscanner | scan\_next |
| lexer.py | \_inputscanner | \_\_next\_\_ |
| lexer.py | \_inputscanner | \_\_init\_\_ | lexer.py | lexer | scan |
| lexer.py | unknowntokenerror | \_\_init\_\_ | lexer.py | \_inputscanner | scan\_next |
| interpreter.py | interpreter | \_\_setvars | interpreter.py | interpreter | \_\_init\_\_ |
| interpreter.py | interpreter | interpret | interface.py | codeeditor | handle\_enter |
| interpreter.py | interpreter | \_\_init\_\_ | main.py |  | main |
| interface.py |  | filemenu | main.py |  | main |
| interface.py | codeeditor | \_\_quitapplication |  |  |  |
| interface.py | codeeditor | \_\_savefile | interface.py | codeeditor | \_\_init\_\_ |
| interface.py | codeeditor | \_\_paste | interface.py | codeeditor | \_\_init\_\_ |
| interface.py | codeeditor | \_\_copy | interface.py | codeeditor | \_\_init\_\_ |
| interface.py | codeeditor | \_\_cut | interface.py | codeeditor | \_\_init\_\_ |
| interface.py | codeeditor | \_\_openfile | interface.py | codeeditor | \_\_init\_\_ |
| interface.py | codeeditor | darkstyle | interface.py | codeeditor | \_\_init\_\_ |
| interface.py | codeeditor | \_\_newfile | interface.py | codeeditor | \_\_init\_\_ |
| interface.py | codeeditor | handle\_enter | interface.py | codeeditor | \_\_init\_\_ |
| interface.py | codeeditor | \_\_init\_\_ | main.py |  | main |
| databasesetup.py | setup | \_\_populateDB | databasesetup.py | setup | \_\_init\_\_ |
| databasesetup.py | setup | \_\_init\_\_ | Login.py | login | \_\_firsttime |
| Abstract.py | AST | \_\_init\_\_ | Interpreter.py | interpreter | interpret |
| Abstract.py | tree | add\_child | Abstract.py | tree | \_\_init\_\_ |
| Abstract.py | tree | \_\_init\_\_ | Abstract.py | AST | \_\_init\_\_ |
| Wall.py | wall | \_\_init\_\_ |  |  |  |
| System.py | system | field |  |  |  |
| System.py | system | compute | System.py | system | field |
| System.py | system | addwall |  |  |  |
| System.py | system | addpoint | main.py |  | main |
| System.py | system | \_\_init\_\_ | main.py |  | main |
| Renderer.py | renderer | dwalls | Renderer.py | renderer | update |
| Renderer.py | renderer | dpoints | Renderer.py | renderer | update |
| Renderer.py | renderer | dfield | Renderer.py | renderer | update |
| Renderer.py | renderer | clear | Renderer.py | renderer | update |
| Renderer.py | renderer | update | Draggable.py | draggable | on\_release |
| Renderer.py | renderer | launch |
| Renderer.py | renderer | launch | main.py |  | main |
| Renderer.py | renderer | \_\_init\_\_ | main.py |  | main |
| Point.py | point | \_\_init\_\_ | main.py |  | main |
| Draggable.py | draggable | disconnect |  |  |  |
| Draggable.py | draggable | on\_release | Draggable.py | draggable | connect |
| Draggable.py | draggable | on\_motion | Draggable.py | draggable | connect |
| Draggable.py | draggable | on\_press | Draggable.py | draggable | connect |
| Draggable.py | draggable | connect | Renderer.py | Renderer | dpoints |
| Draggable.py | draggable | \_\_init\_\_ | Renderer.py | Renderer | dpoints |
| Qbit.py | qbit | setElement |  |  |  |
| Qbit.py | qbit | diffuse | Main.py |  | drawgraph |
| Qbit.py | qbit | probprint |  |  |  |
| Qbit.py | qbit | \_applygauss2d | Qbit.py | qbit | diffuse |
| Qbit.py | qbit | \_normalise | Qbit.py | qbit | diffuse |
| Qbit.py | qbit | \_softmax | Qbit.py | qbit | diffuse |
| Qbit.py | qbit | measure |  |  |  |
| Qbit.py | qbit | \_\_init\_\_ | Main.py |  | main |
| Cbit.py | cbit | probcollapse |  |  |  |
| Cbit.py | cbit | measure |  |  |  |
| Cbit.py | cbit | setElement | Cbit.py | cbit | \_\_init\_\_ |
| Cbit.py | cbit | \_\_init\_\_ | Qbit.py | qbit | \_\_init\_\_ |
| Vector.py | Vector | setElements |  |  |  |
| Vector.py | Vector | setElement | Vector.py | vector | setElements |
| Vector.py | Vector | getElement |  |  |  |
| Vector.py | Vector | scalarMul | Vector.py | vector | \_\_mul\_\_ |
| Vector.py | Vector | setN |  |  |  |
| Vector.py | Vector | allZeros |  |  |  |
| Vector.py | Vector | magnitude |  |  |  |
| Vector.py | Vector | isUnit |  |  |  |
| Vector.py | Vector | unit |  |  |  |
| Vector.py | Vector | tensor | Cbit.py | cbit | \_\_init\_\_ |
| Vector.py | Vector | \_\_init\_\_ | Vector.py | vector | scalarMul |
| Vector.py | vector | unit |
| Vector.py | vector | tensor |
| Cbit.py | cbit | \_\_init\_\_ |

Mapping out the function calls is not only useful for helping other developers examine the codebase but also aided me in finding some functions that may have been needed previously but are no longer called, and thus can be removed to improve the readability of the code. This is good for defensive programming because it means that there is no confusion in exactly what attributes and methods are used and why. Using this information, I went back through my code base and made quality of life improvements that included adding additional comments and documenting strings.

## **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 substantial 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 carefully 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:**

An integral part of my project was the CLI that the user would interact with throughout the runtime of the program. As of such, it was vital that the CLI was designed to be fluid, intuitive and visually appealing. I went back into figma to sketch some drafts of what I wanted the user to interact with and plot the general flow of how they would perform these interactions. This was useful because it allowed me to have a physical blueprint to work to when programming.

A screenshot of a computer

Description automatically generatedAt first, I was unsure of how I wanted the user to perform entry in my program. I had a wide selection to choose from and I didn’t want any key binds to interfere with each other due to the growing number of tabs in my program. I found a good module to use called getch which solved some of these problems. Getch works by directly scanning the standard output buffer for new input characters and sends them directly to the program. This has the effect of being able to take input in python without having the user have to use the enter key – reducing collisions with keybinds in matplotlib and tkinter. Since it is written in python it can also be called directly as a function whenever you as the developer want to check the buffer. This provides much greater control, especially in programs that may have multiple input points perhaps even simultaneously.

A number and number in black squares

Description automatically generated with medium confidence

As for drawing a quantum circuit: It has been a part of my project from the beginning, and I have always wanted to implement it into my solution. After further research however, designing my own ascii printer seems to be beyond the scope of my project: as of such I have decided to investigate using pre-existing tools[[2]](#footnote-3) as I imagine it will be easier to parse the users input into the format that the tool requires than it would be to rewrite the tool from scratch for my project. I will still consider writing my own, however it would be as a later iteration of the code, and I won’t worry about implementing it until I have finished everything else.

## **Project directory layout**

### **Initial tree**

**>nea**

**---->pycache**

---->cbit.py

---->interface.py

---->login.py

---->main.py

---->qbit.py

---->vector.py

>users.db

### **Final tree**

**>git**

**>mypy\_cache**

**>azuredark**

**>nea**

**---->pycache**

**---->azuredark**

---->abstract.py

---->cbit.py

---->databasesetup.py

---->draggable.py

---->gates.py

---->interface.py

---->interpreter.py

---->lexer.py

---->login.py

---->main.py

---->point.py

---->qbit.py

---->recents.txt

---->renderer.py

---->system.py

---->tempcoderunnerfile.py

---->unit\_tests.py

---->vector.py

---->wall.py

>18133 5514 Piercy Tom (1).docx

>achievements.json  
>master.db

>recents.txt

>gitignore

# ***Technical solution***

This section contains examples and highlights of complex programming techniques that have been used during

## **Database implementation**

## **UI implementation**

## **Simulator implementation**

## **Runtime screenshots**

This section gives a broad overview of how my project looks. More detailed information about each image is given later in the appendix, with links to relevant sections of test tables and UX design.

|  |  |
| --- | --- |
| A screen shot of a computer screen  Description automatically generated | A screen shot of a graph  Description automatically generated |
| A screenshot of a computer screen  Description automatically generated | A screenshot of a computer  Description automatically generated |
| A screenshot of a computer  Description automatically generated | A screen shot of a computer  Description automatically generated |
| A screenshot of a computer screen  Description automatically generated | A close up of a word  Description automatically generated |
| A screenshot of a computer  Description automatically generated |  |
|  |  |

# ***Testing***

## **Trace tables**

**Softmax**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vector** | **e** | **element** | **normal** | **item** | **sum**(e) | **OUTPUT** |
| [1,2,3,2,1] | [] | 1 |  |  | 0 |  |
|  | [] | 2 |  |  |  |  |
|  | [*,*] | 3 |  |  | 10.10 |  |
|  | [*,,*] | 2 |  |  | 30.19 |  |
|  | [*,,,*] | 1 |  |  | 37.58 |  |
|  | [*,,,,*] |  | [] |  | 40.3 |  |
|  |  |  | [0.06] |  |  |  |
|  |  |  | [0.06,0.18] |  |  |  |
|  |  |  | [0.06,0.18,0.49] |  |  |  |
|  |  |  | [0.06,0.18,0.49,0.18] |  |  |  |
|  |  |  | [0.06,0.18,0.49,0.18.0.06] |  |  |  |
|  |  |  |  |  |  | [0.06,0.18,0.49,0.18.0.06] |

**Normalise**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Vector** | **new\_max** | **max(vector)** | **value** | **normal** | **OUTPUT** |
| [1,2,3] | 5 | 3 |  | [] |  |
|  |  |  | 1 | [-5] |  |
|  |  |  | 2 | [-5,0] |  |
|  |  |  | 3 | [-5,0,5] |  |
|  |  |  |  |  | [-5,0,5] |

## **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. The tests are numbered and demonstration screenshots are provided in the Appendix.

**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** |
| 25 | \_\_init\_\_ | Incorrect dirac type | Stops execution | Pass | Needs better exception handling |
| 26 | \_\_init\_\_ | Bad sub value | Stops execution | Pass | Needs better exception handling |
| 27 | \_\_init\_\_ | Bad sub type | Stops execution | Pass | Needs better exception handling |
| 28 | \_\_init\_\_ | Large dirac – no sub | Return correct tensor | Pass |  |
| 29 | \_\_init\_\_ | Large dirac – suitable sub | Return correct tensor | Pass |  |
| 30 | setElement | Index larger than list | Return False | Pass |  |
| 31 | setElement | Bad index type | Return False | Pass |  |
| 32 | setElement | Bad value | Return False | Pass | Cbits can only have 1’s or 0’s as elements |
| 33 | setElement | Valid data | Return True | Pass |  |
| 34 | measure | Valid vector | Return correct measurement | Pass |  |
| 35 | measure | Vector larger than 2 | Return False | Pass | Use probcollapse instead |
| 36 | probcollapse | Standard cbit | Print correct probabilities | Pass | Can’t be used in conjunction with setElement |
| 37 | probcollapse | Larger Vector | Print correct probabilities | Pass |  |
| 38 | \_\_repr\_\_ | Valid cbit object | Print correct string | Pass |  |

**Qbit.py initial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| 39 | \_\_init\_\_ | Incorrect dirac type | Stops execution | Pass | Code repeated from Cbit.py |
| 40 | \_\_init\_\_ | Bad sub value | Stops execution | Pass | ‘’ |
| 41 | \_\_init\_\_ | Bad sub type | Stops execution | Pass | ‘’ |
| 42 | Measure | Basic Cbit configuration | Return Cbit value | Pass |  |
| 43 | Measure | Modified Cbit | Return collapsed odds cbit | Pass | Returns a value pulled from a list with magnitudes as weights |
| 44 | Measure | Larger than expected Qbit | Return False | Pass | Use probcollapse instead |
| 45 | \_softmax | Valid input | Return probabilities | Pass | Returns correct values |
| 46 | \_softmax | Non-list argument | Return False | Fail | Needs exception handling |
| 47 | \_softmax | Non integer array | Return False | Fail | ‘’ |
| 48 | \_normalise | Valid input | Return modified list | Pass |  |
| 49 | \_normalise | Non-list parameter | Return False | Fail | Needs exception handling |
| 50 | \_normalise | Non-float maxprime | Return False | Fail | Iterates over string, All operations performed are considered valid between str and list types as they are both iter. |
| 51 | \_normalise | Smaller than expected maxprime | Return modified list | Pass | Function should work for any integers |
| 52 | \_normalise | Negative maxprime | Return modified list | Pass | ‘’ |
| 53 | \_normalise | 0 maxprime | Return modified list | Fail | ‘’ |
| 54 | \_applygauss2d | Valid input | Return correct gauss array | Pass |  |
| 55 | \_applygauss2d | Wrong dim array | Return False | Fail | Internal function |
| 56 | \_applygauss2d | Wrong type array | Return False | Fail | ‘’ |
| 57 | \_applygauss2d | Wrong type n - string | Return False | Fail | ‘’ |
| 58 | \_applygauss2d | Wrong type iterable array | Return False | Fail | ‘’ |
| 59 | probprint | Valid object | Pretty prints probabilities | Pass |  |
| 60 | diffuse | Valid input | Returns correct diffusion | Pass |  |
| 61 | diffuse | Wrong step type | Return False | Fail |  |
| 62 | diffuse | Negative step | Return correct diffusion | Fail | Gives math range error from softmax function |
| 63 | setElement | Valid input | Return True | Pass |  |
| 64 | setElement | Invalid range input | Raise exception | Pass |  |
| 65 | setElement | Bad index range | Return False | Fail |  |
| 66 | setElement | Bad index type | Return False | Pass |  |
| 67 | setElement | Bad value type | Return False | Pass |  |

**Draggable.py initial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| 68 | \_\_init\_\_ | Valid input | Create object | Pass |  |
| 69 | connect | Valid instance | Connect objects | Pass |  |
| 70 | on\_press | Valid event | Set lock | Pass |  |
| 71 | on\_press | Bad event | Return None | Pass |  |
| 72 | on\_motion | Valid event | Set coordinates | Pass |  |
| 73 | on\_motion | Bad event | Return False | Fail | User can’t run into this error |
| 74 | on\_motion | Locked event | Return None | Pass |  |
| 75 | on\_release | Valid event | Update canvas | Pass |  |
| 76 | on\_release | Bad event | Return False | Fail | Event not referenced |
| 77 | on\_release | Locked instance | Return None | Pass |  |
| 78 | disconnect | Valid instance | Disconnect objects | Pass |  |

**Interface.py initial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| 79 | \_\_init\_\_ | Valid input | Create object | Pass |  |
| 80 | \_\_init\_\_ | Bad interpreter type | Raise exception | Fail | No checking until later function |
| 81 | \_\_init\_\_ | Bad file\_open value | Return False | Fail | ‘’ |
| 82 | \_\_init\_\_ | No file\_open value | Create object | Pass |  |
| 83 | handle\_input | Valid single-line input | Interpret line | Pass | Any one line is a valid input |
| 84 | handle\_input | multi-line input | Interpret lines | Pass | Splits input |
| 85 | \_\_newfile | Valid instance | Set new file | Pass |  |
| 86 | darkstyle | valid instance | Set dark mode | Pass | Breaks on computers with different URLs for file path. Needs widening |
| 87 | \_\_openfile | Valid instance | Open file | Pass |  |
| 88 | \_\_openfile | Bad file\_name | Return False | Fail | Needs exception handling |
| 89 | \_\_openfile | No file\_name | Open dialog | Pass |  |
| 90 | \_\_openfile | Append | Append to recents.txt | Fail | Can break on other devices |
| 91 | \_\_cut | valid instance | cuts string | Pass |  |
| 92 | \_\_copy | valid instance | copies string | Pass |  |
| 93 | \_\_paste | valid instance | pastes string | Pass |  |
| 94 | \_\_savefile | valid instance | Saves file | Pass |  |
| 95 | \_\_savefile | Append | Appends to recents.txt | Fail | Breaks on different devices |
| 96 | \_\_quitApplication | valid instance | Quits program | Pass |  |

**Interpreter.py initial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| 97 | \_\_init\_\_ | Valid instance | Create object | Pass |  |
| 98 | \_\_init\_\_ | Bad graphqbit type | Raise exception | Fail | Not implemented yet |
| 99 | interpret | Valid input | Lex object + create AST | Pass |  |
| 100 | interpret | Bad line value | Return False | Fail | Error not possible for user |
| 101 | interpret | Bad line type | Raise exception | Fail | Not implemented yet |

**Lexer.py initial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| ------ | UnkownTokenError | ------------ | -------------------- | -------- | --------------------------- |
| 102 | \_\_init\_\_ | Valid input | Create error | Pass |  |
| 103 | \_\_init\_\_ | Bad token type | Create error | Pass | Token has a conventional type, but this isn’t enforced and doesn’t affect user |
| 104 | \_\_init\_\_ | Bad lineno type | Create error | Pass | Lineno has a conventional type, but this isn’t enforced and doesn’t affect user |
| 105 | \_\_init\_\_ | Bad lineno value | Create error | Pass | Can’t occur in normal use. Creates confusion for user but doesn’t affect runtime |
| 106 | \_\_init\_\_ | Bad token value | Create error | Pass | ‘’ |
| 107 | \_\_repr\_\_ | Valid instance | Print traceback | Pass |  |
| ------ | \_InputScanner | ------------ | -------------------- | -------- | --------------------------- |
| 108 | \_\_init\_\_ | Valid input | Create object | Pass |  |
| 109 | \_\_init\_\_ | Bad lexer type | Raise exception | Fail | Doesn’t fail until later function call |
| 110 | \_\_init\_\_ | Bad input type | Raise exception | Fail | Error can’t occur in normal use |
| 110 | \_\_iter\_\_ | Valid instance | Return instance | Pass |  |
| 111 | \_\_next\_\_ | End scan | Raise exception | Pass | Intended behaviour |
| 112 | \_\_next\_\_ | Mid scan | call scan\_next | Pass |  |
| 113 | done\_scanning | Mid scan | Return False | Pass |  |
| 114 | done\_scanning | End scan | Return True | Pass |  |
| 115 | done\_scanning | big position | Return True | Pass | >= equality in return statement |
| 116 | scan\_next | Done scanning | Return None | Pass |  |
| 117 | scan\_next | No match | Return UnknownTokenError | Pass |  |
| 118 | scan\_next | Valid input | Return matching group, value | Pass |  |
| ------ | Lexer | -------------- | ------------------ | ---------- | ------------------------------- |
| 119 | \_\_init\_\_ | Valid input | Compile regex | Pass |  |
| 120 | \_\_init\_\_ | Bad rules type – non iter | Raise Exception | Fail | Needs error handling |
| 121 | \_\_init\_\_ | Bad rules type - iter | Raise Exception | Fail | Can’t split string, needs error handling |
| 122 | \_\_init\_\_ | Bad rule type | Add to callback, pass over | Pass |  |
| 123 | \_\_init\_\_ | Bad case\_sensitive type | Raise exception | Fail | Allows any value due to Boolean logic in python |
| 124 | \_\_init\_\_ | No case\_sensitive value | Compile regex | Pass |  |
| 125 | \_\_init\_\_ | No omit\_whitespace | Compile regex | Pass |  |
| 126 | scan | Valid input | Return object | Pass |  |
| 127 | scan | bad input type | Raise exception | Fail | No error handling |
| 128 | stmnt\_callback | Valid input | Returns scanned string | Pass |  |
| 128 | stmnt\_callback | Bad input type | Raise exception | Pass | No prettyprint error. Deimplemented |

**Login.py initial**

Login has a weird bug in which file paths are breadth first searched in sqlite3, meaning that if you happen to have a database file of the same name in a super folder it will treat whichever file it finds first as its use database. This isn’t a problem most of the time although occasionally sqlite3 likes to make the master.db in an unusual place. This is due to discrepancies between pythons understanding of the current working directory and the file location. These errors can arise easily if you use an interpreter/IDE that exists system wide (such as VS code which works via windows explorer), as opposed to an instance in the CWD of the python file (such as IDLE). This shouldn’t be a problem to most users; however, it must be noted that you should be careful when moving files around.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| 129 | \_\_init\_\_ | Username not None - string | Create user | Pass | Converts to string in DB query so type is redundant here |
| 130 | \_\_init\_\_ | Username not None - bool | Create user | Pass |  |
| 131 | \_\_init\_\_ | Username not None – string | Create user | Pass | Intended behaviour |
| 132 | \_\_init\_\_ | Non-none type password | Create user | Pass |  |
| 133 | \_\_init\_\_ | No username | Register new user | Pass |  |
| 134 | \_\_init\_\_ | No password | Raise login exception | Pass | Treats “” as password value |
| 135 | \_\_init\_\_ | No username or password | Register new user | Pass |  |
| 136 | \_\_init\_\_ | first-time execution | Connect databases | Pass |  |
| 137 | \_\_firsttime | Valid instance | Create databases | Pass |  |
| 138 | \_\_userlookup | Invalid username | Return False | Pass |  |
| 139 | \_\_userlookup | Valid username | Return True | Pass |  |
| 140 | \_\_login | Valid username, valid password | Return userid | Pass |  |
| 141 | \_\_login | Valid username, invalid password | Return False | Pass |  |
| 142 | \_\_login | Invalid username | Return False | Pass |  |
| 143 | loggedin | Valid user | Return True | Pass |  |
| 144 | loggedin | Not signed in | Return False | Pass |  |
| 145 | \_\_validPassword | Invalid password | Return False | Pass |  |
| 146 | \_\_validPassword | Valid password | Return True | Pass |  |
| 147 | register | Valid username, invalid password | Loop till valid | Pass |  |
| 148 | register | Valid username, valid password | Return True | Pass |  |
| 149 | register | invalid username | Loop till valid | Pass |  |
| 150 | getuserid | valid instance | Return userid | Pass |  |

**Point.py Initial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| 151 | \_\_init\_\_ | Valid input | Create class | Pass |  |
| 152 | \_\_init\_\_ | Bad parameter type | Create class | Pass | Needs type conversion code |
| 153 | \_\_repr\_\_ | Valid instance | Return string object | Pass |  |

**Renderer.py initial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| 154 | \_\_init\_\_ | Valid input | Create class | Pass |  |
| 155 | \_\_init\_\_ | Bad parameter type | Create class | Fail | Needs type conversion, fails later in script |
| 156 | launch | Valid instance | Show plot | Pass |  |
| 157 | update | Valid instance | updates graph | Pass | Can be slow due to function calls |
| 158 | clear | Valid instance | Clears axis | Pass | Can cause traceback to the console due to the setter method not being implemented for patches objects, requiring the use of a built-in axis-clear function that isn’t really the intended method – this can introduce delay |
| 159 | dfield | Valid instance | Calculate field | Pass |  |
| 160 | dpoints | Valid instance | Add patch | Pass |  |
| 161 | dwalls | Valid instance | Plot wall | Pass |  |

**System.py initial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| 162 | \_\_init\_\_ | Valid input | Create class | Pass |  |
| 163 | \_\_init\_\_ | Bad parameter type | Create class | Fail | Needs type conversion |
| 164 | addPoint | Valid point object | Add to class | Pass |  |
| 165 | addPoint | Bad point object | Return False | Fail | Needs error handling |
| 166 | addWall | Valid wall object | Add to class | Pass |  |
| 167 | addWall | Bad wall object | Return False | Fail | Needs error handling |
| 168 | compute | Valid input | Return calculation | Pass |  |
| 169 | compute | Bad parameter[s] type | Return False | Fail | No error checking |
| 170 | compute | Bad parameter[s] value | Return False | Fail | Creates math and range errors |
| 171 | field | Valid input | Return iterable | Pass |  |
| 172 | field | Bad parameter[s] type | Return False | Fail | Index errors |

**Wall.py initial**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** |
| 173 | \_\_init\_\_ | Valid input | Create class | Pass |  |
| 174 | \_\_init\_\_ | Bad parameter type | Create class | Pass | Needs type conversion code |
| 175 | \_\_repr\_\_ | Valid instance | Return string object | Pass |  |

**Total number of tests: 175**

**Passed: 135** (77.14%)

**Failed: 40** (22.86%)

After testing all my code midway through the program, I could see clearly areas that needed improvements as well as ones that were fine in their current state. When testing at the end of my program I decided to add an extra column to the test tables representing which functions were internal or program controlled as opposed to user run. Internal functions were allowed to fail tests in situations where other program control meant that a specific input wasn’t possible. For instance, the \_softmax function is only called by the program itself which checks elsewhere that the arguments are the right variable type – thus failing the function for bad exception handling isn’t a fair assessment because in normal cases, it would never need to be implemented.

**Abstract.py final**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#** | **Location** | **Test** | **Intended termination behaviour** | **Pass/Fail** | **Comments** | **User-facing function?** |
|  |  |  |  |  |  |  |

## **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 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**

**Variables:**

Trying to work out how to implement variables in my program was a key decision as my implementation would massively affect the rest of the programs run style. In the end I decided to tack onto the end of pythons existing complex variable system. They way I could do this is by creating temporary variables for digits and literals in pythons built-in vars system and then using functions such as id() to convert into a memory address as well as ctypes to convert out to values again. This meant that my command queue went through two different iterations before being put into the AST. This was quite a slow process, as this portion of the interpreting algorithm ran in order.





## **UX testing**

Over the course of my project, I was constantly in touch with my client as well as a range of independent testers who helped me evaluate the strengths and weaknesses of my program. One such area that was especially important for me to get feedback on was the user experience. It was extremely beneficial to hear from potential users of the product and explore what they think needed to be improved with regards to the UX pipeline. This testing process was to ensure that objectives 1 and 3 were met. For ease of evaluation, we will keep these objectives separate when testing such that we test visuals and flow distinctly.

**Visuals**

Testers were shown a series of screenshots from runtime and told to make any comments they wished regarding the visual elements they could see. They were encouraged to make at least 1 criticism as well as asked some guided questions to further the development of the program.

|  |  |  |  |
| --- | --- | --- | --- |
| **Screenshot** | **Positive comments** | **Criticisms** | **Solutions** |
|  | The colours are very clear, and it is obvious that there is a difference in the two regions.  The colours are cool enough for long program use to be a strain on the eyes of the users | The particles would take up too much of the window if more were added | Dynamic sizing based on number of particles.  Smaller standard size |
|  | The orange/grey colour scheme compliments itself well and works with the colours of the rest of the program without clashing. | Some users may find it hard to look at for a long time. Needs to be more accessible to all users | Make an optional dark mode slider.  Make optional font resizer. |
|  | The arrows are really clear and the slow colour shift towards the particle looks very nice. The lit pixel outlines towards the shape help to blend the geometry into the rest of the figure without sticking out. | The scales on the side don’t really make any sense as there is no context for what they represent. | Remove or relabel the scales.  *(they don’t mean anything)* |
|  | The whole diffusion process is very satisfying to watch and having a large pixel block makes it the probabilities very easy to eyeball as the user. It also gives it a nice retro feel. The scale bar remaining consistent is a plus | There isn’t any context for the scales. The diffuser settles very quickly which means that it can be boring to look at if no code has been written in a while. | Add optional buttons onto the graph figure that pre-interpret common functions or algorithms.  *(Post-release feature)* |
|  | All titles have their own flair. #1 looks very futuristic and works well as a splash screen for a CLI. #2 looks best as a persistent header title (especially on a paper printout). #3 looks best as a persistent CLI header. | In a dark colour scheme console that most users would first experience the program in, #1 would be far too distracting to the user. #2 can suffer from resizing issues. | #3 splash title selected.  Minimize resizing issues as much as possible. |

**Flow, intuition, and functionality**

It was important for me to check that the program worked well for the users rather than solely being coded to a specification with no regard for the user experience. Consequently, lots of testing went into ensuring that the UX was the best it could possibly be. This resulted in development branches of the program being shared with independent testers and feedback being collected through surveys and interviews.

|  |  |  |
| --- | --- | --- |
| **Program element** | **Comments** | **Patch/Enhancement** |
| Login system | Works well, is foolproof for the end user.  Can be a bit jarring to go from a CLI login system to a GUI program. | Ultimately will be changed into a GUI login system using getpass however, this requires a rework of the current code base so perhaps post release feature |
| CLI project interaction screen | “I found this so profoundly easy to navigate that I believe a small child would succeed.”  Numbers need clearer spacing to differentiate the 3 different sections. Could also be implemented as a GUI | Fix number spacing and provide a suitable range of numbers to make mistyping virtually impossible. (e.g. 1-5 for key functions, 95-98 for recent projects, 99 for exit) |
| Diagram screen | Has some lag due to having to be passed through the requests package and called from an external server | Will ultimately be replaced from the ground up but a custom-built diagram maker. In the meantime, delay can be mitigated by caching common images or making less requests to the server |
| Probability density screen | Looks good but takes up a lot of screen space for something that isn’t directly interactable through the figure window. | Can be made interactive post-release. Can be integrated into the particle view using a tab system through tkinter. |
| Particle view window | Looks good but there is some delay when interacting with the particle.  Takes up more screen space and takes up resources even when not being used by the user.  Has the chance to output errors onto the screen due to the clearing function no longer being supported in python 3.12. This doesn’t affect anything visually but could be the cause of some delay. | This is partly a python bottleneck problem that can only be fixed by porting the program into a more efficient/modern language such as mojo. The other issue is with running multiple processes at the same time and keybinds conflicting. There is also an issue with the clearing.  Can be integrated with the probability density view via a tabbing system. |
| Overall GUI design | Can be a bit jarring (especially for new users), going from the simplistic and minimal CLI login and interaction screens to lots of windows.  Key binds can occasionally conflict with each other.  The windows can sometimes be created with weird dimensions and must be manually resized by the user. | Assimilate various windows to cut down on screen space and allow the user to only look at what they are interested in.  Look at using 3rd party modules such as getch to remove key binding problems. The various different windows key binds can also be manually manipulated because matplotlib is built off tkinter.  Use dynamic size setting using the os library to make sure that the windows start at suitable locations and dimensions. |

# ***Evaluation***

## **Achieved solution versus project objectives**

My client and I agreed that I would be creating solution 3 to make “*a sandbox desktop application that teaches quantum computing by encouraging exploration and providing visual feedback”.*

Overall, I feel that I have been able to implement such a solution to meet all of my client’s needs and requirements. In the process I have been able to improve my personal programming ability and experience with developing code systems. This project has been invaluable to help me perfect my python programming.

**Objective 1: The program is intuitive, fluid, and easy to use.**

This objective is about the user experience of my program and how simple it is to navigate. Testing this part of my program involved getting both client and independent feedback so I could iteratively improve the experience of the users and make my program the best it could be. After receiving positive comments about the UX design of my project I feel as if I have met this objective well. Additionally, the extra time that was invested into this portion of the project by using online design tools definitely made a noticeable impact on the final appearance of the project.

**Objective 2: The program incentivises learning in a fun environment.**

The goal of this objective was to effectively create a tool that encourages learning through exploration. This was checked by independent testers who reported feedback to me as part of the UX design and testing process. This is a difficult objective to directly test and improve on as the developer. It relies on the user reporting issues and post-release usage data. After the program has been released then further information can be taken regarding the uptime of the program and how often the program is used. This would give us an indication of how fun our solution is.

**Objective 3: The program should be aesthetic and easy to be on for extended periods of time.**

This objective is related to the user experience and requires testing in a similar way to objective 3 as it is difficult for the developer to properly determine if the client will find the program’s style to be appropriate for extended use. Despite this I believe the objective to be met well from the feedback I received during the UX testing process. The information collected during this time can be seen in a separate section.

**Objective 4: The program should be fully interactive.**

This objective concerns itself with how the user should run and experience the program. The intent was to make the program completely accessible to anyone that wants to use it. Not only can all parts of the program be interacted with and changed/adapted but if the user really wants then due to the solution being implemented in the python programming language, the user has control to override or change the underlying program if they desire. The inclusion of this documentation with the released code means that if any user wants to create an implementation in a novel language, then this is trivial. This objective has been met well.

**Objective 5: The program should be timeless and not quickly outdated.**

This object is related to the research process of the project. It specifies the importance of using correct information as well as using methods that will hopefully scale as computing power scale. It uses a simple (almost trivial) solution to the simulation problem, yet despite this simplicity, the program is still highly applicable to many different systems/platforms/devices. It is my belief that if Moore’s law holds then there will come a day that this implementation could be run on any e embedded system. This objective has been met.

**Objective 6: The program should be feature rich.**

This objective ties in to many others as it is about the usability of my program. The program has been constructed as to the client’s requirements and includes all of the major features that they requested. This objective has been met well. To improve upon the current solution new features could be added in a modular fashion, allowing the user to construct their ideal solution and modify according to their needs.

**Objective 7: The program should accurately reflect the system it is emulating.**

This objective is all about conducting proper research into the topic of quantum computing and communications such that I could produce a program that accurately describes the world of qubits in as much detail as possible, whilst simultaneously making the program accessible to everyone that may wish to use it. In doing my research for this project, I managed my time such that I could allocate blocks of program time to rigorously explore quantum computing and our modern understanding of it. Because of this, I feel like I have effectively met this objective.

**Objective 8: The program should contain examples and relevant links.**

Personally, I do not feel like I met this objective as well as I could have. I think that this objective would be better completed by including a user manual or operating guide of sorts alongside the program’s executable. As this was not in the project specifications, it was not factored into the time management guideline and as of such it has not been implemented in time. If I were to extend upon my project, this would perhaps be the first issue to be addressed.

**Objective 9: The program should not be able to break, crash or be exploited.**

This objective was a functional check to confirm that the program was working as intended and not feature any major bugs. After 3 rounds of testing (developer, client and independent), I feel like this objective has been adequately met. Some errors persist in my program, and this can be demonstrated through the test tables, however I am confident that I have enough test coverage that the program can be released with any remaining niche or specific bugs being removed via hotfixes and patches.

## **Client feedback**

To finalise my project, I had the opportunity to discuss with Mark about the solution that was crafted for him. I used this time to explore what he liked about the system and how he had found his experience.

**Tom: “[…] I am curious to know your initial thoughts. What aspects of the simulator stood out to you?”**

**Mark:** “I'm quite impressed with the simulator. The user interface is intuitive, and I appreciate the ease of navigation. The accuracy in replicating quantum states is commendable. It's been an invaluable tool for helping my studies.”

**Tom: “Was there anything specific that you particularly liked about the simulator?”**

**Mark:** “Definitely the versatility. I liked the freedom to experiment with different quantum algorithms and scenarios. It provided a hands-on experience without the need for specialized hardware. The detailed visualizations also helped in grasping complex quantum concepts.”

**Tom: “Now, on the flip side, did you come across anything you didn't like or found challenging?”**

**Mark:** “Well, not exactly a dislike, but having so much of my screen filled all of the time can get quite difficult to work with – especially if I have to keep moving windows to get to what I want”

**Tom: “Is there anything else you'd like to see incorporated into the simulator?”**

**Mark:** “I would like to see the project maintained and updated so when there are advancements in technology these are used to improve the program – I’d like to see how fast the simulator can run.”

My client feedback has clearly demonstrated achievement of the various objectives set during my project, whilst also giving valuable notice to some light issues. I can use this information to further improve my project to produce the best solution possible for Mark.

## **Independent feedback**

Throughout my project it was important to have a fair and representative feedback system so that I could appropriately adjust the project as needed. To do this, I found a small group of people who would regularly test my project through its development – this was invaluable in providing me a space where I could trial code impartially before finalising the implementation. An overview of the results from these testing periods is given below.

**Final findings:**

The testers were asked to rank the program across 4 main areas, this was to assess how well the program met the project objectives.

A graph of a review

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Jacob | Melissa | Rachel | Sam |
| “The simulator not only provided a user-friendly interface but also showcased the immense potential of quantum computing in solving complex problems.”  “The accuracy and speed with which the simulator executed quantum algorithms were impressive, allowing me to simulate real-world scenarios with ease.”  “The detailed visualizations and comprehensive analysis tools provided invaluable insights into the quantum states and operations, making it an excellent educational tool for understanding the nuances of quantum mechanics. “ | “The simulator not only surpassed my expectations but also proved to be an invaluable resource for understanding the intricacies of quantum computing.”  “The user interface was intuitive, allowing for seamless navigation and experimentation with various quantum algorithms.”  “The simulator's ability to accurately replicate quantum states and simulate quantum operations provided a comprehensive understanding of the underlying principles.” | “I am genuinely impressed with the depth and capabilities this tool offers.”  “The simulator's interface balances user-friendliness with advanced features, making it accessible for both novices and seasoned enthusiasts.” | “The simulator seamlessly merges user-friendly design with robust functionality, making it accessible for users across different proficiency levels.”  “The simulator showcased emulating intricate quantum processes with remarkable precision.”  “The simulator's technical capabilities also emphasized its reliability and stability.”    “I particularly appreciated the simulator's ability to simulate quantum algorithms with efficiency, providing a comprehensive and immersive experience.” |

## **Scope for further development**

Looking forward, there is tremendous scope for further development in my project, and several avenues stand out for enhancement.

One notable improvement could involve implementing a tabbing system for windows within the simulator interface, facilitating the concurrent exploration of multiple quantum algorithms or scenarios. This feature would streamline workflow and enhance user efficiency – this would be good because it would specifically target some of the key objectives Mark was looking for. I could implement this using advanced techniques in tkinter and this would allow the user the option of which screens they wished to look at. Whilst at my current programming experience level, this would be too challenging for my to do I think this would go a long way to improving program fluidity and aesthetics.

Additionally, the integration of supplementary viewing options, such as 3D visualizations or virtual reality interfaces, could offer users a more immersive and intuitive experience, deepening their understanding of quantum phenomena. This would work by implementing the core system in a modular way – with various different viewports/filters attached via modules which could be loaded into the system (either as a default or through the user). Whilst I attempted to develop my current system in this way – as more code got added to the program the different views became more inseparable. As a learning point going forwards, the option of that style of the functional paradigm may be the best method for producing a versatile simulator.

Accessibility options, catering to diverse user needs (as mentioned during my UX testing period), could also be a focal point for development, ensuring that the simulator is inclusive and user-friendly for individuals with varying levels of expertise. This was brought up to me by both Mark and some of the independent testers because getting the program running on their system came to be quite complicated, due to the nature of executable production from python code. Certain tools produce executables that can only be run on the same type of chip, due to differences in the implementation of assembly.

Furthermore, as advancements in classical computing power continue, leveraging these breakthroughs could significantly amplify the capabilities of the simulator, allowing for more intricate simulations and expanding its utility across scientific research, algorithm development, and educational applications.

# ***Appendix***

## **Testing screenshots**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## **Full code listing**

1. This is about as much as I can say about this variable. This project will be the death of me [↑](#footnote-ref-2)
2. <https://quantikz.krastanov.org/?circuit=> [↑](#footnote-ref-3)