Background

Classical computers have limitations in the generation of a truly random number. In classical implementations, the numbers generated are predictable if it is generated sufficiently numerous incidences. A true random number can only be generated by utilizing the quantum nature which can be done by reading a qubit from a quantum circuit. A quantum circuit achieves true randomness by putting a Hadamard gate after each qubit. Putting the Hadamard gate has the effect of dividing the probability in exactly half when the output is collapsed into a binary number. To generate a range of random numbers between 0 and decimal of , a minimum of n-number of qubits is needed in the quantum circuit. For randomly generating four directions of movements, we implemented two qubits and two Hadamard gates in our quantum circuit.

Quantum circuit implementation using the Qiskit

Cell1: The Qiskit's user credentials are configured. Also, a provider and the list of currently available backends are found for running on an actual quantum computer.

Cell2: The backend is specified from which the user desires.

Cell3: The quantum circuit is initialized and set up as follows. Two registers are specified for setting up the two qubits. And the qubits are initialized using the classical bit of "0". The Hadamard gates are put after each qubit. The output of the quantum circuit is measured. Finally, the quantum circuit is plotted.

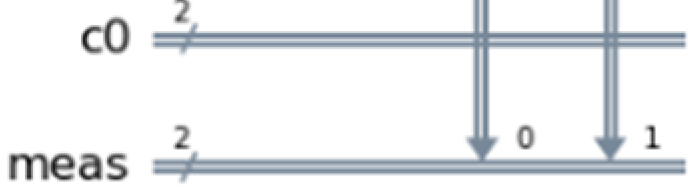


Figure 1. Quantum circuit implementing two qubits. The qubits are read after the Hadamard gates.

Cell4: Running the quantum circuit is repeated for Nshots of instances. For instance, our implementation initializes and reads the quantum circuits for Nshots = 10000 times. The job status message is printed as it gets updated.

Cell 5: Saved the quantum circuit readout results, printed and plotted the histogram of the circuit outputs in the decimal range of 0 ~ .

```
Job Status: job has successfully run
{'00 00': 2723, '01 00': 2649, '10 00': 2342, '11 00': 2286}
```

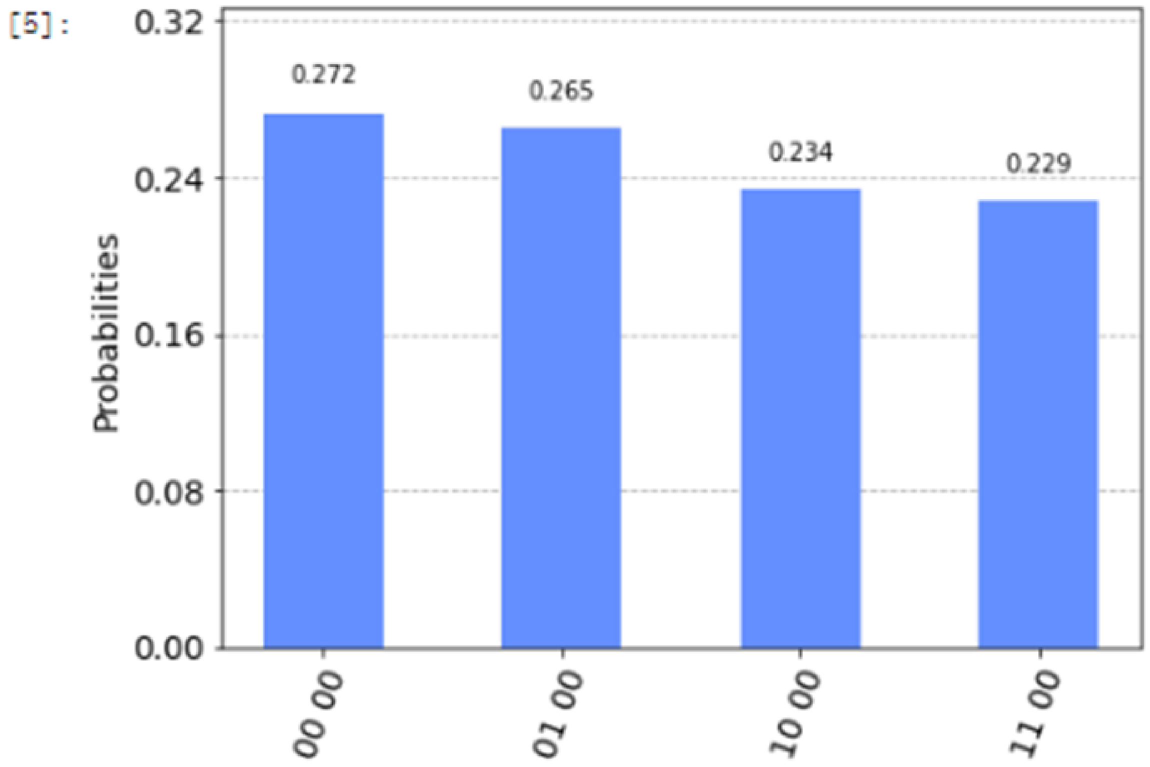


Figure 2. The probability histogram of readout from the two qubits. The bins in the x-axis represent “output number space input bit”.

Cell 6: Stored the generated list of random numbers in a variable called the “dir_list”. The saved list is in a binary representation which is a string of a combination of “output number space input bit” where the input bit is “00” in our case.

Cell 7: The output binary numbers are extracted from the output string form and transformed into a decimal number.

Cell8: The output of decimal numbers of 10000 instances is written to a text file.

Experience

This summer was my first involvement with the Quantum Game Club and my experience with the club was beyond my expectations. Being a Ph.D. student, I was compelled by how much tech-savvy young students are in programming nowadays and have an interest in quantum even if their background is not directly in physics but just in the general engineering discipline.

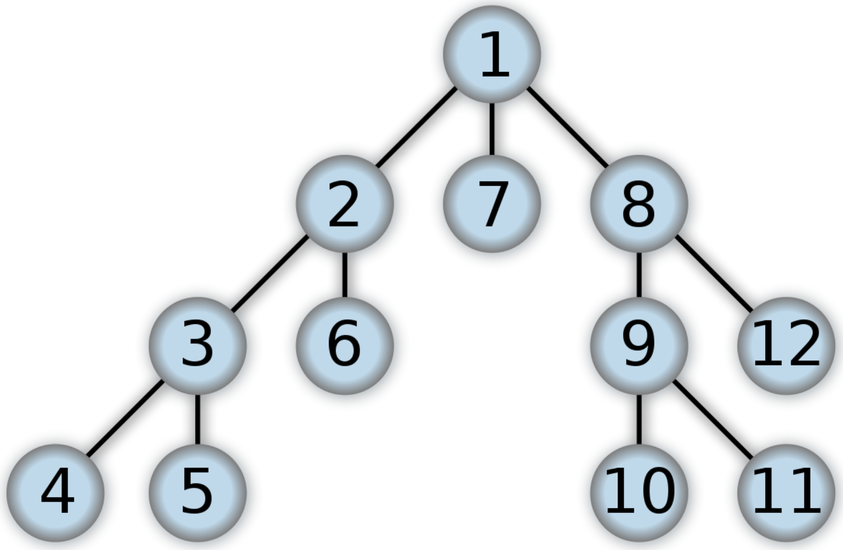
This summer, it was my first time getting involved in a quantum-related project as a hobbyist. I participated in a group project for finding the shortest path through a quantum maze. The problem was divided into two parts: namely, generating a random path through the maze using the randomness of a quantum measurement and solving a maze by implementing a quantum algorithm using the Qatalyst. My contribution to the project was to help generate a path through the quantum maze by using a random number generator that aids in determining the next course of movement in a 2D square gridded maze. Using the Qiskit, I programmed a quantum computer to generate random numbers between 0 to 3 such that each number corresponded to one of the four directions of movement as north, south, east, and west in a 2D gridded maze.

Most of the technical challenge for me was learning about the correct parameters for the configuration commands in the Qiskit. I spent about 20 to 30 hours in total searching online and finally getting the commands working in my code. Overall, I would recommend joining the Quantum Game Club and getting involved in a quantum project.

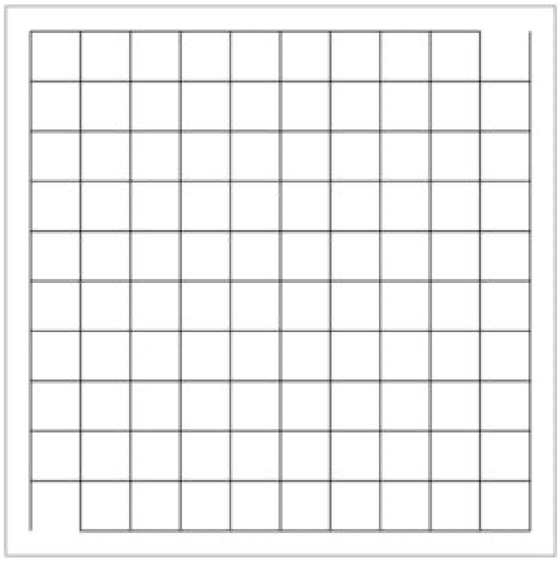
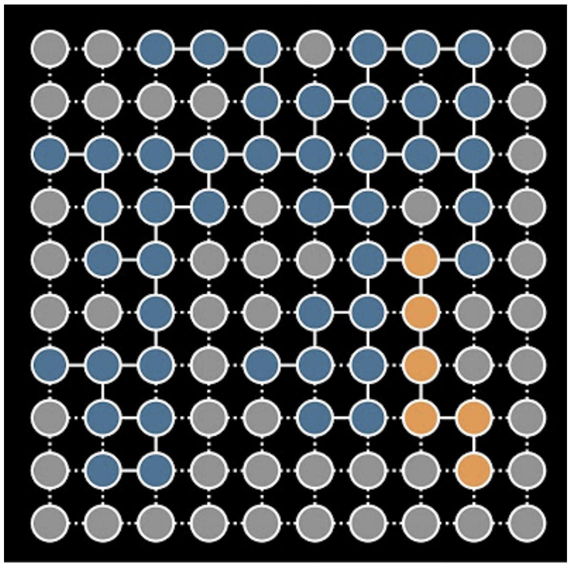
Introduction:

For the frontend part of the application we needed to generate a random maze. I investigated a number of algorithms such as the kruskal's algorithm and the prim's algorithm but eventually I went on with the recursive backtracking algorithm as it generated a random maze everytime and was more efficient than the other algorithms given our size constraint for the maze(We had a size constraint to limit the number of qbits used). I coded the entire frontend part of the game in python by utilizing the pygame library.

Details of the algorithm used:
The recursive backtracking algorithm is based on Depth First Search(DFS). DFS is a graph traversal algorithm with idea behind it being that it starts at the root node and then travels down deep and as far as possible into a particular branch of the graph until it encounters an end upon which it backtracks to the last unvisited node in the stack and repeats the same process, thus eventually traversing the entire graph. An extra stack memory is needed to keep track of the nodes that have been visited so far so that we can backtrack to an unvisited node upon traversing a particular branch.
Let us look at the following example.



Here 1 is the root node and therefore the graph traversal starts from 1. Next, each of its neighbours namely, 1, 2 and 7 are pushed to the stack and one of these nodes is selected at random. Let's say that 2 was selected next, therefore, 2 now becomes the current node and is popped off the stack. More importantly, 1 and 2 both are added to the list of visited nodes. Now this exact same process is performed for node 2, i.e., its neighbors 3 and 6 are added to the stack, one of them is selected randomly and the selected node becomes the new current node and is popped off the stack and added to the list of visited nodes.
Therefore, this is basically a recursion problem where at each step we are left with a smaller version of the original problem. An important detail here is that when we encounter a node that does not have any unvisited neighbors, the algorithm backtracks to the last unvisited node in the stack and repeats the same process again. This way we traverse the entire graph. Therefore, in the above example one of the possible traversals could be 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
The way we utilize this depth-first search approach to generate the graph is by first constructing a grid and then imagining the grid as a graph. As an example :-

We could first generate a grid as on the left and then imagine the grid to be a graph as on the right. Then the algorithm traverses through the graph using a depth first search approach.To perform this algorithm we use two variables visited and stack. Visited keeps track of all the cells that have been visited until then and stack is used to store the set of unvisited nodes and every time a node is visited that does not have any unvisited neighbours, it is popped off the stack. While looping we randomly select one of the neighbours of the current cell that has not yet been visited and upon which we update the visited list and make the adjacent unvisited cell to be the current node. In order to actually carve out the maze path using pygame, we call one of the push functions which fills up the appropriate cells on the screen with blue color thus, overriding the white wall between the cells which creates a continuous path. We simply repeat this process in a loop performing DFS over the graph until each cell is visited. And as mentioned, if any time it happens that the current cell does not have any unvisited neighbours, then it is popped off the stack and the algorithm backtracks to the last unvisited node on the stack.
This is how the entire maze is generated.

Overall Experience and challenges:
This was my first semester with the quantum game club and I would say that the time I have spent with the club and doing the summer project has been extremely productive and fruitful. In computer science, beyond high level coding, I have always been interested in knowing the behind the scenes of the hardware as well as the overall method of computation that allows algorithms to work. This is the reason that I was curious about quantum computing as it incorporates a change in the very basic methods of computation and hardware that the classical computers use. Besides the joy of brainstorming on the problem of the quantum shortest path algorithm and working on the frontend of the game, I feel the best part of the summer project was that the very topics we were given the opportunity to tackle, pushed me to spend hours self studying and exploring the field to understand things like what qbits actually are and how are qbits and quantum circuits actually physically made in the industry. This aided me in getting a better understanding of the fundamental difference between the classical and the quantum approach to computation which was in itself a rewarding feeling. More importantly, through the course of the project, I got to read various research papers on possible implementations of various quantum algorithms such as

the Grover’s algorithm and the quantum Dijkstra's algorithm which aim at solving the shortest path problem.

Quantum computing is still in its early stages and I believe that having the opportunity as an undergraduate to understand this field, experiment and learn quantum programming and write code that runs on actual quantum computers is really exciting and valuable.

Although I did not face any particular obstacles while working on the frontend part of the game, managing all aspects of the project and integrating the different parts together was a difficult task. For example, once the maze is generated, it needs to be converted into an undirected graph which can be fed to the quantum shortest path algorithm and in order to generate a randomized maze every time, a quantum random number generator algorithm is also needed. Therefore, we divided the tasks amongst the team members and made sure to have effective communication and collaboratively brainstorm on topics that any member was stuck on.

The biggest challenge that we faced as a team in this project was developing the quantum shortest path algorithm itself, as solving the shortest path problem using a quantum computer is new and does not have many implementations yet. But we had the advantage of having access to people from QCI with whom we shared our ideas about the constraint function for the quantum shortest path algorithm. Their guidance greatly assisted us during this stage of the project.

Overall, I had a great experience working on an interesting problem and building the project together with my teammates during this summer.