**Quantum Final Project Script:**

Hello; My name is Seixas Aldrich and today I will be telling you about my work for my final project as part of a quantum software development class at Tufts University. My task was to study a quantum algorithm of my choosing and implement it using quantum coding software. The algorithm I decided on is the twos complement algorithm, which changes any signed integer to its opposite sign in binary.

Before I go into the algorithm, I would like to give some context on what quantum computing is and how we can implement these algorithms with it. Quantum computing is a type of computing where instead of using bits that are represented by either a 0 or 1 to calculate things, we use something called a Qubit. Qubits are like a bit, but instead of being 0 or 1, they are in a superposition between both 0 and 1. For example, a Qubit can have a 50% chance of being a 0 and a 50% chance of being a 1. Another important aspect of quantum computing is you cannot use logic gates with qubits like in classical computing, as they do not have a set value. This is why in quantum computing we use transformations on the qubits to do all of the work. These transformations change the values of the components of the Qubits, allowing us to use them in interesting ways. For example, the only transformation I use in the two’s complement algorithm is the X transformation, that swaps the values for the X and Y components.

Now that we have some background on quantum computing, I can start describing the problem I set out to solve. I wanted to develop an algorithm that would take in an integer in a signed binary representation, represented by an array, or register, or Qubits, and flip the sign, leaving the same magnitude. Now, the normal classical computing algorithm for this is you flip every bit of the integer, and then add one to that to get the two’s complement. The flipping of every bit, or in my case Qubit, is easy as there is a quantum transformation for that, however there is no easy way to do addition in quantum with qubits, so it will be a challenge to implement that.

The first thing I did when trying to approach this problem was do more research on this algorithm in a quantum computing field, to see other peoples methods and to see if I could be inspired or recreate them. I went searching on the web for twos complement algorithms for quantum computing, but did not find any that were doing exactly what I wanted. Many of the papers I found were using two’s complement as a part of a larger project and made it in a way that fit to their case, but not very well as a general two’s complementor. However, after looking through these research papers, I thought I had a good handle on what I would have to do to implement this, so I started coding.  
I set up an environment on VS code to run the quantum software, and first made tests that would confirm if my quantum algorithm worked. I did this by creating registers of qubits of varying lengths, and using X gates, which flip the qubits 0 and 1 components, I made signed integers of values I wanted to test. Then I put my function for twos complement in the test code, and finally I flipped the output of the algorithms qubits that were 1’s to 0’s and ran the assert all zeros function on the register to make sure all of the qubits were zero. This confirmed that my algorithm would work because I knew what the correct output would be, and thus I knew which qubits would have to be flipped, so if the algorithm was incorrect the register would not be all zeros in the end, telling me that my algorithm was wrong, or it would be all zeros and my algorithm was working as expected.

Now that I had tests to make sure my code was correct, I moved on to implementing the algorithm. The first step of two’s complementing a number is flipping every Qubit in it. I did this by doing an not gate on every bit in the register for the number.

The next, and more difficult step of the algorithm is adding one to the entire number. To do this I thought about what effect adding one had on each qubit. For the least significant qubit, if it is zero it becomes a one, and nothing else happens to the number. If the least significant qubit is one, then it becomes zero and the next number is tested with the same process.

Now with quantum gates this setup cannot be done efficiently in Q# as there is no controlled gates when a qubit is 0. However, I found a way to do this starting from the most significant bit, where if every other bit is 1, then the bit would be changed as the addition of 1 would get carried all the way to that bit. Then, the second most significant qubit is done the same way, with every other bit lesser qubit being the control for that one. This is shown in the graphic here.

Now, why did I go through all of the trouble to make this. Well two’s complement is a basic type of numbering system that is needed for almost all arithmetic that will be used in quantum algorithms. If a number ever needs to be negative, or ever needs to be subtracted, twos complementing will take a part in it. That is why I believe learning to develop an algorithm like this is very important, as it will lead to success with much larger algorithms.

The main takeaway I took from this project has been that simple calculations in classical computing can take a lot more work than expected in quantum. I have seen this through examples on previous assignments, but having to research and build up this algorithm really made me realize how much time it must take to develop large scale algorithms like the Fast Fourier Transform.

In the future I would love to work more with quantum algorithms, real life quantum computers, and on resource estimation as I could not get that working for this project.