DuQhess Details:

I probably should have planned the meetings so they didn't overlap with rush but here we are. In this write up I will attempt to legibly explain what was discussed in the past few meetings for those interested. The structure will be as follows:

- **Overview**
  - **Classical Position Mapping**
  - **Position value function**
  - **Quantum implementation & its advantages**
  - **Challenges and Potential Solutions**
- **My mini-version**
  - **I built a very weak, but functioning quantum chess engine before, and the goal is to expand off of it. I will be going over how it works. For those wanting a concrete example this is important to go over.**

<u>Overview: The goal is to take a given chess position, find the next best move</u>

Classical Position Mapping:

- The first step is to take in any given classical position, and format it so that it can be read by the code. This will most likely be in the format of 2D arrays, where we have the elements of the array represent squares on the chess board, as well as chess pieces.
- We also want to take the given position, and following the rules of chess, determine all possible next configurations.

Position Value Function:

- Separately, we want a function that can take a given position, and return its 'value'. This value correlates with how advantageous a position is for us. For example, in a position where white is much better than black, we would have a high positive value, say around 5, whereas if white is worse than black, we would have around 0. The larger the magnitude of the number, the more advantageous the position.
- There are different types of value functions, and while I had initially intended on using a neural net to perform this function, I realized it is extremely complicated and hard to implement into a quantum circuit. Instead, we will be using AI, to train weights on a linear function. For example, if our output has inputs x1 and x2, our function could be:
  - f(x1, x2) = a*(b+c)*x1 + a*c*x2
- Where the constants a,b, and c, are determined via training in a neural net, or another optimization method. The data used to do the training will be collected by inputting random positions into a current chess engine such as stockfish, and recording the output. Some of you might have realized that stockfish outputs

advantageous black positions with a negative sign. We will try to scale it so the perfect black position is the value 0.

Quantum Implementation & its Advantages

- The quantum part of this algorithm lies in the (hopefully possible) ability to evaluate the position value function for multiple positions at the same time. To do this, we would input a superposition of all possible next positions, with equal probability. So for instance if we have 3 next possible positions with probability A, we would have:
    - A * (Pos 1) + A * (Pos 2) + A * (Pos 3) (Ignore normalization, this is just conceptual)
- We would then run all 3 positions ‘simultaneously’ through the position value function, and ‘mark’ the position with highest value. This would then do a phase kickback, which adjusts the initial probabilities, so that our ‘marked’ position would have the highest probability. So for instance if our position values are Pos 2 > Pos 1 > Pos 3, our new superposition could be.
    - A*(Pos 1) + 1.5A * (Pos 2) + 0.5A * (Pos 3).
- Hopefully over the course of several iterations, our Pos 2 probability gets so high our superposition approaches 100% probability of measuring Pos 2, which means we would have found our best next move.

  A deeper look can be understood by looking at the grover's algorithm here:
  https://qiskit.org/textbook/ch-algorithms/grover.html
  Or by working further in the project.

Challenges and Solutions

- There are some challenges when it comes to implementing the quantum part of the algorithm. Primarily, coding a way to evaluate the cost function, as it involves some addition which can be difficult to implement in a quantum system. Secondly, marking and performing the phase kickback will be a bit tricky. These are the primary problems that will be needing solutions in this project.

Others have asked for a more in depth example, and so I will go over my previous quantum chess engine, although much more simplified. A python notebook file is attached to the email with comments in it. Those interested can look through the comments to see a walk through of how my previous engine worked.