Documentation

Build instructions in the README.md file.

Components:

- **MainWindow**
  - Handles UI and user interactions.
- **UDPHandler**
  - Manages network communication between peers.
- **Test_P2P**
  - Unit tests for verifying message propagation works properly.

```
mkdir build && cd build && cmake .. && cmake --build .
```

```
install_name_tool -add_rpath /Users/devin/Qt/6.8.2/macos/lib
P2Pal_Devin_Liu.app/Contents/MacOS/P2Pal_Devin_Liu
```

Addressing requirements:

- Basic GUI Implementation:
  - The chat log is displayed in the receivedMessageBox widget at the top of the app.
  - The text input is located under the message box as editMessageBox. It is autofocused on launch, and you can submit messages by clicking the send button.
- Network Communication:
  - Majority of the network features are in UDPHandler.cpp.
  - Messages are sent over QUdpSocket in a QByteArray object, which can be deserialized to obtain a QVariantMap object holding:
    - "**clock**" – the local tick that the message was sent
    - "**message**" – the message detail
    - "**origin**" – the port the message originated from
    - "**type**" – between 'message', 'ack', 'history', 'request_history', to let the receiving peer know what actions to take regarding the received data.
    - "**sequenceNum**" – the local sequence that was sent. Note that this can be modified during history reconciliation based on local tick if there are conflicting messages for the same sequence number.
  - Local port discover:
    - The way the network is set up is as follows:

    - Each peer has a left and right node (except for starting and end peers). So peer 5002 has neighbors 5001 and 5003. It cannot communicate directly with other peers.
    - Peer starts at 5000.
  - The basic message format is discussed above.
- **Gossip Protocol:**
  - **Rumor mongering**
    - The message can be identified using Origin, sequence number, tick, and the message itself.
    - Messages are sent in order (3 before 4).
    - A peer will send a message and wait for an acknowledgement from the destination peer. It will wait for 2 seconds. If there isn't an acknowledgement, the peer will send the same message once again and stop.
    - Each peer has a copy of their own history in **UDPHandler->messageHistory.**
  - **Anti-Entropy**
    - Every 2 seconds, peers will compare their history with their left and right peer. It will reconcile any missing messages, as will as resolve conflicting sequence numbers. (For example, peer A has 5. Apple. Peer B has 5. Orange, it will check the clock tick the message was sent at and reorder the message history. See unit test for demonstration).
- **Peer discovery:**
  - Nodes will only have a left and right peer (except for 5000 and 5009 since they are the two ends). 5002 will only be able to communicate with 5001 and 5003.
  - Peers automatically search for available ports starting from 5000. If 5000 is taken, it checks 5001, etc.
  - Once it finds an available port, it will maintain connection until the app is closed.

Use Instruction:

- **Using the app:**
  - The largest box is the **receivedMessageBox**. It will hold the messages both sent by the peer and messages received from other peers.
  - To send a message, simply click send.

- There are two other buttons, antientropy and print history. They are there for test purposes. Clicking antientropy manually requests history from neighboring peers to reconcile instead of waiting for the timer (right now the timer is set to 2 seconds so this button isn't as useful). Print history does as the name says and prints the lift of local history to the console. This is used to check message ordering and content.

Test Cases:

The unit test spins up 4 instances of peers, from port 5000 to 5003.

There are 3 tests:

1. **Test1()** tests message propagation. It sends different messages from different peers and checks to see if every peer on the network got the same message.
2. **Test2()** tests peer discovery. In this model, peers only connect with their left and right neighbor (so port 5001 has neighbors 5000 and 5002, but port 5000 only has neighbor 5001 since it is at the starting end of the range). It checks that each peer only have their designated neighbors.
3. **Test3()** tests a couple of things. It tests peer connection and disconnection behavior, message updating on reconnect, and message history reconciliation. The idea is that since each peer can be cut off from half the system if one peer leaves, we can start sending separate messages from each disconnected side, that way the message orderings will be upon when the peer rejoins. First when the peer rejoins, it has to check the left and right peer, get both of their message history, determine which one is better (just pick the longer history for our purposes) and update their local history. At this point, it will only have one of the history from left or right. Next, when the antientropy timer hits (1 second), it starts comparing to a random left or right neighbor and checks the history difference, reconciling and adding messages if it is different. It does this by checking the clock ticks for when the messages were sent to determine message ordering. This propagates through the entire network, and eventually, all peers will have the same message ordered correctly. (Note that since antientropy picks either left or right neighbors to update message at random, there is a chance that the 10 second delay I have in the test won't be enough for it to pick all the neighbors and fully update, although this would be rare).