**Goal -** Simulate algorithm and run through code structure

**Does proposer send to every other node in the network? Or will it have its own quorum?**

**Array of logs, Reduced file (filename and dictionary inside the log)**

**CLI file that will ask PRM for log array**

**Classes:**

1. **Node**
2. **Log**
3. **PRM** 
   1. **BallotNum Tuple <0,0>**
   2. **AcceptNum Tuple <0,0>**
   3. **AcceptVal Null**
   4. **Log object**

**Algorithm**

**Messages – Prepare, Acknowledge**

1. **Node 1 sends (“Prepare”, <1,1> ). (msg, ballot) to ALL nodes**
2. **Node 2 receives (“Prepare”, bal) from site I** 
   1. **Check if new prepare bal is >= BallotNum** 
      1. **BallotNum = bal**
      2. **Send back (“ack”, bal, AcceptNum, AcceptVal)**

**Goal - Simulate algorithm and run through code structure**

**Does proposer send to every other node in the network? Or will it have its own quorum?**

**Reduced file (filename and dictionary inside the log)**

**Steps to Paxos**

**1) Connect with CLI to listen to first command**

**2) Connect to all the other PRMs**

**3) If replicate, start Paxos by sending everyone a Proposal and increase internal Ballot number**

**Message types: (Wait 400ms to make sure it didn't actually get majority accept or ack messages)**

**1) All the CLI commands**

**2) Prepare, Ballot Number, index in log array**

**3) Acknowledge, Ballot Number, Accept Number, Accept value, index in log array**

**4) Accept, Ballot Number, Accept value, index in log array**

**5) Decide, index in log array, Accept value**

**Accept value will be filename**

**Check to see if the filename is actually there. if it is, begin to replicate**

**Check to see if it's actually a command on there**

**CLI commands:**

**replicate filename, stop/resume... At any point client queries- total, print, merge**

**Classes:**

**1) Log Object**

**2) PRM**

**a. BallotNum Tuple <0,0> <Ballot #, ProcessID> Include index of array**

**b. AcceptNum Tuple <0,0> <AcceptNum #, ProcessID> from ballot**

**c. AcceptVal Null**

**d. Array of logs with Log objects**

**e. id of prm/ first 2 digits of port number**

**f. ballot number it's on right now**

**g. how many accept values it has so far**

**Algorithm**

**Messages –**

**1) Node 1 sends (“Prepare”, <1,1> ). (msg, ballot) to ALL nodes**

**2) Node 2 receives (“Prepare”, bal) from site 1**

**if(bal >= node.BallotNum):**

**node.BallotNum = bal**

**send("ack", node.BallotNum, node.AcceptNum, AcceptVal) #to node 1**

**3) Node 1 receives ack from Node 2 and now updates counter of joins to 2. KEEP in mind to store all the ack messages,**

**to know all the highest ballot numbered value and use that in your acceptVal message instead.**

**@node1**

**if(counter > total\_nodes/2)**

**if(all ack values have AcceptVal as null):**

**myVal = initial value**

**else:**

**myVal = AcceptVal from ack message that contains highest ballot number.**

**Change your current AcceptVal to 3 and AcceptNum to the one you proposed.**

**send("accept", BallotNum, myVal) to all the nodes #Still a proposal**

**4) Upon accept, check edge case and respond with "success' or "reject"**

**5) If majority accept, send everyone "decide" on certain value. Timeout if not majority accept propose it again with higher ballot #**

**6) Once its original node receives majority of accepts from other nodes, send decide to everyone periodically**

**Edge cases:**

**Every time you send accept message, check to see if ("accept", tempBallNum, AcceptVal) tempBallNum > BallotNum. Ignore**

**if it's less than**

**If you didn't receive ack from majority, wait for a random number and send another prepare with higher ballot. timeout**

**to see if it did not receive majority acks from the other nodes,**

**If node proposes something with index that's already filled, other node send that index onward. Global current\_index**

**"""**