Testing strategy

Unit and integration tests

We use unit tests to test all the parts of the system. Each component has its own test suite. To run tests:
* Server: Install mocha.js and run `npm test`
* Client:

For the server we also do an integration test as a part of the test suite. It mocks two simultaneous connections and verifies the server can correctly exchange packets between them. For details see `server/test/test.coffee`

Acceptance tests

As a final part of our testing strategy we do acceptance tests. In short that means we take the whole system and follow a testing protocol and test if every step product the expected outcome.

Acceptance test protocol

1. Select 3 arbitrary supported devices - Mac/PC/iOS/Android
2. Open the app on all 3 of them.
Expected result: the app successfully launches and is showing the login screen
3. Enter your name into the login field and press login
Expected result: You should be logged in and see the 'make a call' screen
4. Make a call by pressing the 'Start a Call' button
Expected result: The call should be connected and you should see a screen showing portraits of all call participants and "End the Call" button
5. Listen and speak to other participants
Expected result: You should hear others clearly and others should hear you
6. Move other call participants around the virtual 2D space and observe whether their voice seems to come from a different angle
Expected result: Other sound sources should come from the same angle as represented by the virtual 2D space
7. End the call with the 'End the Call' button
Expected result: The call should end and the application should return to 'make a call' screen

Testing in the Unity application:

Unity provides, its testing software that allows users to proceed fragmentation and unit tests of their application made. This testing software is downloaded from the unity assets store and added to the project library.

We first carried out unit test on the application. The unity unit testing uses nUnit as the basis of the unit testing framework. We used the built in test files to test the internal logic of our TCP client. Testing different alternatives such as what happens when we have to re-send the packets in case of errors. Apart from this we have performed fragmentation test on the TCP Client. We used a real bare bone TCP server (We used our node.js server). We were able to see if our client received and send messages correctly. As well as, we were able to see how it connects and disconnects from the server. Also we used exception testing(part of the unit test) to see if correctly threw exceptions when the client were not able to connect to the server.We were able to see if it showed the error messages correctly and recovered from the errors and continue running the application.