Lab 10 README

ECE 13

Sean Manger

Prof: Steve McGuire

F24

<Author> Sean Manger (smanger@ucsc.edu)

Note: I took the Battle Boats Solo Option and partially implemented Message/MessageTest.c I also scratched out a skeleton code for Agent/AgentTest.c. Unfortunately, prior to this lab I had Pneumonia for 2 weeks which caused me to fall behind both in ECE 13 & ECE 141. As a result, I was unable to fully complete this lab. I will however talk about what I did implement and my ideas of the rest of Battleboats

Collaborators:

For this Lab, I again briefly worked with Donny Tang. Donny assisted me with correctly organizing my files within my Lab10 repository. Besides this due to my illness and trying to keep up with my other classes I worked solo. As always all my code is written by myself and is my own work.

General Approach To Lab

My first priority in this lab was to try and implement the Message.c module and its associated functions first since I thought this might be rather complicated due to the required string manipulation. To start off, as with the toaster lab, I printed out a copy of the Message FSM from the manual and kept it close on hand. In addition to this, I also printed out a copy of the Ascii Table from the Wikipedia Ascii page. I thought it would be a good idea to have an Ascii table close by since we were dealing with strings where each character has a different Ascii value. Therefore, string manipulation is much easier when you have a reference to quickly refer to. To start off with, I went ahead and implemented the Message_Calculate_Checksum() function. This seemed daunting at first but, ended up proving to be rather simple. I worked out a few payload examples myself on paper and realized that to get the check sum it was just a simple XOR operation with all the payload characters including the commas. So what I ended up doing was just using a simple for loop and a counter to calculate the checksum.

From here, I worked on the Parse_Message() function. The idea behind this function was that we wanted to create a message event in BattleBoats in response to a message. In other words, if we received a message, and it was correct we responded accordingly and appended the message_event struct with the appropriate event parameters. These parameters would be 1-3 pieces of data depending on which type of message was received. We had 5 possibilities: Result, Accept, Reveal, Challenge, and Shot. To deal with the errors I just used some if statements and returned the appropriate errors. Unfortunately, this is pretty much where my work ended due to running out of time. To if the strings were valid I would have likely used a strcmp and a for loop to compare and append the appropriate data as necessary. F the message state machine I traced through it and it seemed simple enough to implement when compared with the toaster. Since we weren't dealing with any hardware, I imagine the state machine could have been easily done with some if statements and switching states as necessary.

Conclusion

This was another excellent Lab that I probably spent about 25 hrs working on. State Machines I would say were my favorite part of this course and I pan to do my extra credit tutorial on them. I am sad this class has come to a close but I really grew as a programmer over 10 weeks and I feel way more confident than I did when I was starting out.