TDQC

Zergmap

Jack Spence

26 October 2018

1. Write-Up

1.1 Requirements

Requirements were to read in any amount of .pcap files and find zerg units and their locations. If health is found it should be added as well. Once the locations have been discovered, we were expected to verify there are two disjoint node paths between them and eliminate any that disrupted this network.

1.2 Suggested Features

Suggested feature attempted in my submission is to read packets that have 6in4 tunneling included, reading Big Endian .pcaps, as well as having the -n option to change the limit on how many zergs can be removed.

1.3 Syntax

Compile:

1. Make – Make alone will compile the default binary of Zergmap

2. make profile – Will produce the executable with the profile flags set to produce gmon.out

3. make debug – Produces executable with debug flags set

4. make clean – Cleans the executable as well as any .a, .o. and .outs in the current directory

Usage: ./zergmap <filename> [-h <threshold>] [-n <limit>]

1.4 Known Problems

There is 1 valgrind error on a couple of test cases that I could not figure out how to fix with my implementation, at the trade off of no memory leaks.

Dijkstra's will not properly find disjointed node paths. My solution is a band-aid.

2. Project Design Plans

2.1 Initial Design Plans

Being a hybrid between codec and maze, I relied pretty heavily on those two projects source code and modified it to suit the task at hand. Between the pcap reading from codec and the Dijkstra's from maze, I had most of the project written. There was still plenty left to do after importing that source, the largest being the disjointed node paths.

2.2 What didn't work

I completely overlooked the disjointed "node" paths, and had a working implementation with disjointed edge paths using Dijkstra's alone. Then upon the realization that they are completely different, which was well into the project, I did not feel comfortable enough with graph theory and path finding in general to scratch it all, which probably hindered my solution significantly. My current solution still uses Dijkstra's at most 3 times per check and makes a pseudo disjointed node path, and makes some assumptions for deletion.

2.3 What went well

Not a whole lot of modifications had to be made to codec to allow reading directly into the zergUnit structures. I left a lot of the code exactly the way it was for the rest of the fields on the packets, so if a message gets sent it will be printed to the screen, would have been re written for the most part with more time.

2.4 Conclusion

Graph theory is easily the hardest concept we have been taught at our time at TDQC in my opinion. The complex test cases as well as the sheer amount of them makes everything needing to be extremely robust and almost foolproof. I don't think I'd prefer to do many more problems like these.

3. Test Procedures

3.1 Test files

I created some bash scripts, located in the tests directory, to run my program against everything located in the mapper/pcaps/ directory. Most of the folders in the pcap directory have the expected output in the folder name which I visually checked once ran.

3.2 Testing Procedure

Test script for running with no options:

~/jspence-superawesomeproject/zergmap/tests/tesh.sh

Test script for running with -n:

~/jspence-superawesomeproject/zergmap/tests/ntest.sh

Test script for running with -h:

~/jspence-superawesomeproject/zergmap/tests/htesh.sh

Test script for running with valgrind:

~/jspence-superawesomeproject/zergmap/tests/valg

Test script for running with -n and valgrind:

~/jspence-superawesomeproject/zergmap/tests/nvalg

Test script for running with -h and valgrind:

~/jspence-superawesomeproject/zergmap/tests/hvalg