CSC 258 Project Proposal

Brandon Willett

March 21, 2018

I don't really play video games anymore, but back when I was in high school, it was the main way my friends and I would pass the time after classes let out. The one we played the most was called World of Warcraft, where you play as an Elf or an Orc (or something), and you'd complete quests to level up and get better gear for your character. Doing quests was fun, but my favorite part of the game came only at the end, when you reached the maximum level – then, you could start doing "raids", where groups of up to 40 players would team up and attempt to defeat massive boss enemies together. These bosses had abilities that could wipe out your entire team in one shot, so not only were individual players' skill and reaction times important, but also the overall group's composition: if you had too many high-damage-output members, the whole team would die before the boss could be killed, since those types tend to be fragile. Or, if you had too many healing characters, the entire raid might never end, since you might never kill a boss at all!

These exercises in teamwork and perseverance were a lot of fun for me at the time, and in fact they represent a pretty complicated distributed system. There are 40 players all rapidly sending commands to a central location which affect the boss's health and condition, and each of these players need there to be almost zero latency and perfect (guaranteed) delivery of their messages, which – when the "central location" is made up of many machines which might fail – is hard to accomplish; the continuous synchronization of these messages is a challenge. In addition, each client needs to not only send "damage the boss" messages to the central location (be a publisher), but also to listen for "you've been healed" messages back (be a subscriber). This sort of player-count-agnostic system is required, since at any one time there might be 4 or 40 players sending messages at once, and this too presents an interesting issue.

Thus, we reach the idea of my project, which is to implement a simplified version of this game as a command line app. At any time, a client could "join the fight" either as a healer or a damage character. Then, they'll have a set of commands they can send to the remote server, like querying the boss's current health or executing a move to heal a friend whose character is in danger of dying. I believe the "hard part" of this project will be making the central location synchronized, scalable to any number of concurrent players, and very fault-tolerant.

In terms of technology, it'd be easiest to use Python, since it has great libraries and I'm very familiar with it, but I might actually try to do it in something like Erlang / Elixir instead, since they're particularly adept at handling concurrent, message-passing programs (plus I always like to take opportunities to learn new languages). In addition, I believe I'll use some kind of concurrent message-passing queue, like zeroMQ, and (based on what I've seen in class) probably some kind of dynamic leader-election system on the server side to ensure that there's always someone managing the incoming messages in that queue.

Proving correctness should be pretty straightforward – as long as messages from each player are received quickly, have very little chance of failing (even when a "server" goes down), and are always reflected at the same time to all other players, the system should be acting correctly.