CPSC 416 Project 2 Proposal

Jerome Rasky, Madeleine Chercover, Raunak Kumar, Vaastav Anand

Introduction

Video games have always been an important aspect in the lives of people as they provide an escape from reality and life problems. in the last year or so, a new genre of video games, Battle Royale, has taken over the world by storm. Video games of this genre, like Fortnite and PUBG, are essentially last-man standing games where the last player wins the game. These games typically involve altercations between players and heavy interaction with the game map. These features place a lot of restirctions on maintaining the game state like validity and consistency of the world that has been modified and making sure the players that have already been eliminated are unable to modify, but maybe view, the game state. Additionally, like any network game these games also require that each palyer has very low latency.

We're interested in building a distributed game of the Battle Royale genre for our term project. We feel that this area is interesting because it introduces real-time constraints into our distributed system. As mentioned before, such a game requires low latency whereas the blockchain can afford to take ten minutes to confirm transactions, players might be upset if it takes ten minutes to move at all in a game. In addition, we also intend to have a lot of distributed state in our game.

Background

To build an online real-time multiplayer game, there were initially two popular architectures, shown in Figure 1, which were used in industry. The first one is the peer-to-peer architecture in which all clients start in the same initial state and each client broadcasts every move to the all other nodes. With clients communicating directly, the gamestate can not advance until each client's move is received by every other client. The overall latency of the system is then dependent on the slowest client in the system. This system is also not tolerant to faulty clients, as each client will wait and decide themselves if a client has failed. The second architecture is the client-server architecture. In this architecture the gamestate is stored on a server and clients send updates to the server. This architecture reduces latency, for each client the latency is determined by the connection between that client and the server. However, this architecture is still too slow for real-time online multiplayer games, which lead to the introduction of client-side prediction. The client-side prediction allows the client to simulate its own version of the game whilst sending the results of the moves to the server. This essentially means that each client maintains its own gamestate but this can be overriden by the server. This architecture allows for the server to detect malicious clients as well because each action by the client is being validated on the server.

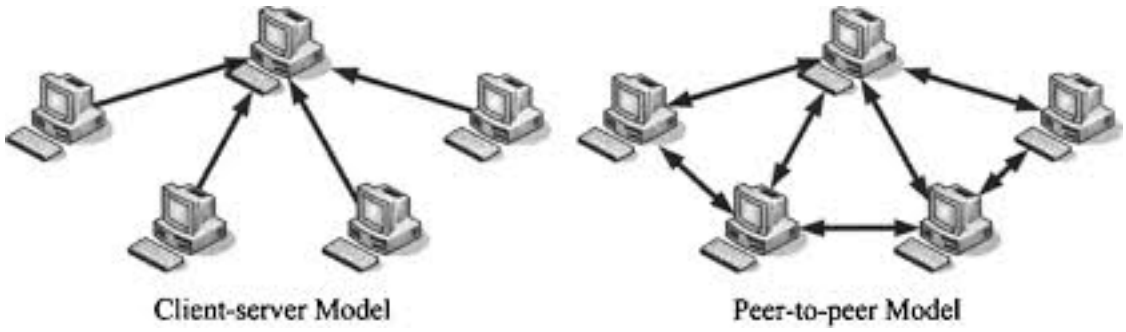

Figure 1: Network Architecture Models

Game Mechanics

Basic Idea

We'll build a game not unlike Tanks, where players can move around and shoot at each other. The last player standing wins. Initially, our game will be very simple, without any health or stage hazards or anything. We'll grow our game mechanics as time goes on, adding things like stage hazards, status effects, and alternative goals. With this approach, we can grow the complexity to an almost arbitrary degree by adding features, but we can also scale back our ambitions if we run out of time.

Peer Discovery

Clock Synchronization

Player Disconnections

Stat Collection

We'll keep track of player stats, such as a k/d ratio, using conflict-free replicated data types. This will provide a dimension of distributed systems design that is less latency-bound than the regular game mechanics.

How we'll use the cloud

We'll have a centralized server for peer discovery, which will be hosted on Azure. We also plan to have a "backup" client on Azure, so that the game state persists even if no clients are online.

Technology stack

We'll use go for nearly everything. We'll use a simple 2D game engine, so that we don't have to work too hard on graphics. We'll use a games library to facilitate that development, but otherwise we'll stick to relatively standard go.

Development Plan

SWOT Analysis

Strengths

- Team members have worked with each other on the previous assignments of the course.
- Team members are diligent and punctual.
- All members are good at researching as well as solving potential issues.

Weaknesses

- There are very limited resources available for building a low-latency distributed game.
- None of the members have any prior experience with making multiplayer online games.

Opportunities

- We can make the game as complex as we would like given the time limitations that would allow us to expand on the distriuted system we are aiming to make.
- Conflict-Free Replicated Data Types is a very new concept for all of the members and would require a lot of work for a proper implementation.

Threats

- Commitment to exams or assignments from other courses may interfere with progress.
- None of us have any experience with GoVector, ShiViz or Dinv, all of which we are aiming to incorporate in our project.

Resources

https://www.cs.ubc.ca/~gberseth/projects/ArmGame/ARM%20Game%20With%20Distributed%20States%20-%20Glen%20Berseth,%20Ravjot%20%20%20%20%20%20Singh.pdf

http://www.it.uom.gr/teaching/distrubutedSite/dsIdaLiu/lecture/lect11-12.frm.pdf

https://www.microsoft.com/en-us/research/uploads/prod/2016/12/Time-Clocks-and-the-Ordering-of-Events-in-a-Distributed pdf

https://en.wikipedia.org/wiki/Berkeley_algorithm

http://pmg.csail.mit.edu/papers/osdi99.pdf

https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type