SENG 485 Assignment

(Quake 3 Arena)

Luo Dai(V00838474)

Table of Contents

Introduction

Quake 3 Arena is a first-person shooting game released in 1999 by id Software, it is the third game in the Quake franchise. It was a game that focused heavily on online multiplayer matches. At the time, online games were just lifting off the ground. The game "Doom" released in 1993 had popularized the concept of a deathmatch mode for FPS games, where players go head to head in an online environment, and Quake 3 Arena was built based on this mode. Quake 3 Arena was made popular mostly due to its great technology at the time, it had gathered a large and loyal fanbase, and many people still play it even up until this day.

Requirements

Being a competitive online multiplayer shooting game, there are many critical requirements to provide a good gaming experience. This includes requirements on security, graphical performance, and network performance. The game will need to have mechanisms in place to combat intrusions and protect players' information, it will also need to be able to smoothly display graphical components, and finally it will have to be able to support multiple players in a single game session while keeping everyone in sync and account for lost datagrams. In this assignment, the attribute requirement we will be focusing on is network performance.

For an online shooting game, the packets received are very time critical, thus using a TCP/IP protocol would make the connection slower due to its robustness. For this reason, Quake 3 uses the UDP protocol instead. But due to the unreliability of UDP, the game will have to implement some measure to deal with things like packet dropping, and that is what Quake 3 have done with its "snapshot" mechanism.

Use Case Scenario

Aspect	Details
Business Goals	To provide an online game that ensure correct synchronization
Quality Attributes	Network Performance
Stimulus	Playing in an online session
Stimulus Source	Any player
Response	Client stay in sync with master game state
Response Measure	Everything displayed on screen is not-bogus (actually registered in the game)

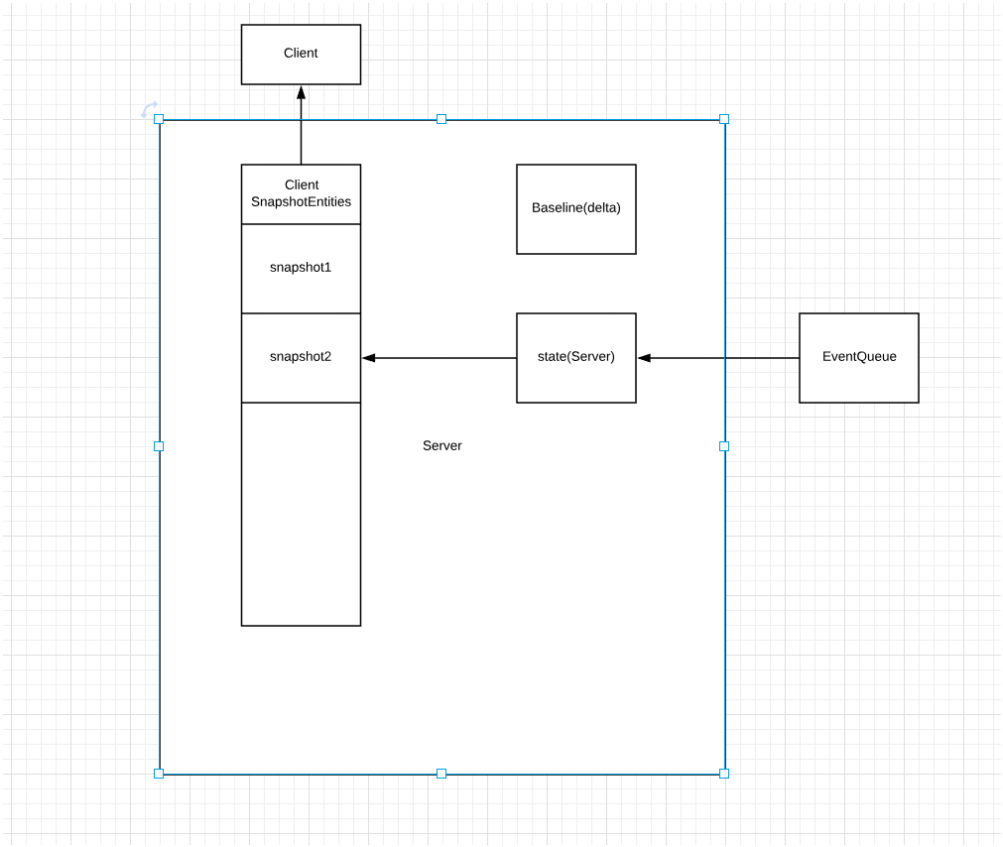
Growth Scenario

Aspect	Details
Business Goals	To support large number of players in a game session
Quality Attributes	Network Performance
Stimulus	Multiple players playing in an online session
Stimulus Source	Player
Response	All client stays in sync with master game state and each other
Response Measure	Everything displayed on screen is not-bogus (A dead person can't shoot and hurt anyone)

Investigation

The investigation of the code included first cloning the source code on to my pc, and then open it inside an IDE, the one I choose was Microsoft Visual Studio. Once the solution file is loaded, we can observe that it is made up of 8 different projects. I had issues when trying to run some of the analyze tools in VS, but was able to see it in object view and get a sense after reading all the comments. Upon further research and investigation of the code in the server folder, I found something interesting that is the "snapshot" mechanism that Quake 3 Arena uses to compensate for the unreliability of UDP and keep the client and server in sync.

Use Case Diagram

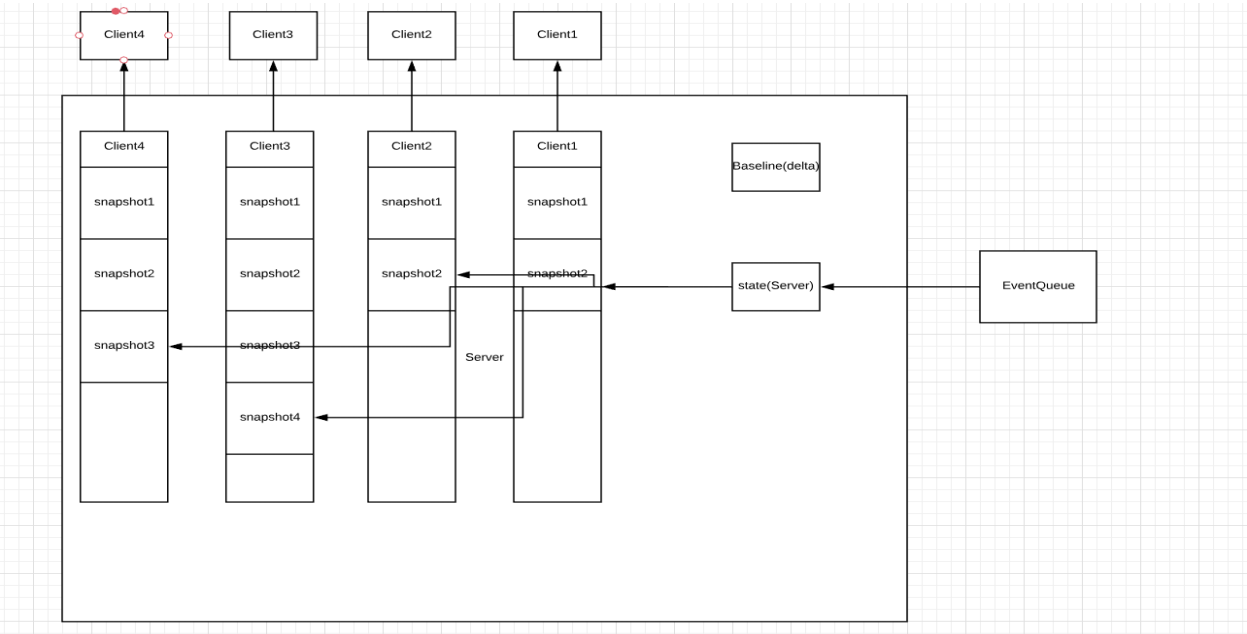


The game server will have a state saved that is used to update the state of all clients. When user commands are passed through as event from the queue to the server, it updates the server state. The new server state is passed on to the snapshot list of a player, and then compared with the most recent snapshot to perform an update to the client, finally the snapshot state is marked ACK. This way, if a datagram gets dropped and does not update the client, the snapshot will not be marked ACK, thus the update won't be performed based on it. This will ensure that the client will receive all the updates, and that everything the client sees is updated from the server state.

```
/*
==================
SV_WriteSnapshotToClient
==================
*/
static void SV_WriteSnapshotToClient( client_t *client, msg_t *msg ) {
    clientSnapshot_t    *frame, *oldframe;
    int                 lastframe;
    int                 i;
    int                 snapFlags;

    // this is the snapshot we are creating
    frame = &client->frames[ client->netchan.outgoingSequence & PACKET_MASK ];

    // try to use a previous frame as the source for delta compressing the snapshot
    if ( client->deltaMessage <= 0 || client->state != CS_ACTIVE ) {
        // client is asking for a retransmit
        oldframe = NULL;
        lastframe = 0;
    } else if ( client->netchan.outgoingSequence - client->deltaMessage
        >= (PACKET_BACKUP - 3) ) {
        // client hasn't gotten a good message through in a long time
        Com_DPrintf ("%s: Delta request from out of date packet.\n", client->name);
        oldframe = NULL;
        lastframe = 0;
    } else {
        // we have a valid snapshot to delta from
        oldframe = &client->frames[ client->deltaMessage & PACKET_MASK ];
        lastframe = client->netchan.outgoingSequence - client->deltaMessage;

        // the snapshot's entities may still have rolled off the buffer, though
        if ( oldframe->first_entity <= svs.nextSnapshotEntities - svs.numSnapshotEntities ) {
            Com_DPrintf ("%s: Delta request from out of date entities.\n", client->name);
            oldframe = NULL;
            lastframe = 0;
        }
    }
```

```
/*
===============
SV_AddEntToSnapshot
===============
*/
static void SV_AddEntToSnapshot( svEntity_t *svEnt, sharedEntity_t *gEnt, snapshotEntityNumbers_t *eNums ) {
    // if we have already added this entity to this snapshot, don't add again
    if ( svEnt->snapshotCounter == sv.snapshotCounter ) {
        return;
    }
    svEnt->snapshotCounter = sv.snapshotCounter;

    // if we are full, silently discard entities
    if ( eNums->numSnapshotEntities == MAX_SNAPSHOT_ENTITIES ) {
        return;
    }

    eNums->snapshotEntities[ eNums->numSnapshotEntities ] = gEnt->s.number;
    eNums->numSnapshotEntities++;
}
```

Growth Diagram



This “Snapshot” system can handle multiple players in an online session while making sure every client will get all the updates from server state. The process is the same but extended to multiple clients, thus there would be more event updates and more client state updates and will require more bandwidth. So, the number of clients that can be connected and still provide a smoothing experience dependent on the bandwidth of the server. With current technology, it will be able to easily support large numbers of players.