ECE419S - Lab 3

Design Document

Design Team:

Jonghyo David Kim 998960824
Huanzhong Wei 998830993

Soon Kyu Lee 999021892
Guanzhou Ye 999842602

Aldrich Wingsiong 998735775
Howard Zhang 998851871

The Design

Gameplay Use-cases

1. Beginning Game (Initialization)
    a. How do players start the game?
        i. Players start the game by entering their username. (The first client will become the first naming service that knows the location of everyone, and every additional client will be a backup naming service in case the previous naming service fails)
        ii. For clients that are not a naming service or backup, they only need to know their neighbours (due to the simple token ring algorithm)
    b. How do players register with their name?
        i. A naming service is initialized and listens on a predetermined port. This is the server which will register all players and broadcast all player profiles to every single client. This way, they will have info such as player name, direction, and what port to listen on.
    c. Where do players get placed?
        i. Players gets placed pseudorandomly on the map.
    d. How do players synchronize their initial location placement?
        i. Players have the same game initialization seed. All players will have the same placement at the start.
2. In-Game
    a. How do players move around and know where other players are?
        i. Consult algorithm below
    b. When a player fires at someone, how do missile ticks get handled?
        i. Handled the same way as a movement event. The first client to join is elected as the person who adds the missile tick event to the token ring queue every 200ms, which will move all current projectiles on the map.
    c. How do players get revived?
        i. Players have the same game initialization seed. All players will see the same revival coordinates locally.
    d. How does the scoreboard get updated?
        i. As long as the game is consistent, score will also be updated consistently.
3. End-Game
    a. How do players leave the game?
        i. No dynamic joining or leaving. Player leaves the game when they close down their client
    b. How does the game end?
        i. It doesn't. It's too much fun!!!
        ii. The game ends when all players quit and exit the game window

Synchronization Component

1. A single token is passed around using a round robin system between all clients.
2. When a client does not have the token, he stores all his movements and fire events into the local queue. The first client who registered also stores missile tick movements in the queue every 200 ms.
3. When a client receives the token, client will perform the following steps:
    a. Dequeue, and remove the events that belong to the current client until an event belonging to a different client is at the front of the token queue
    b. Dequeue each event, process the event (update UI), and enqueue back into the token queue
    c. Continue step b) until the first event seen in b reappears (circular queue)
    d. Process the local queue events
    e. Push the local queue events onto the token queue
    f. Send token queue to the next client on the token ring
4. When the client receives the token, there will be a variable number of events in the queue
5. When a client receives the token, the current client sends an acknowledgement back to the client who sent the token

Network Reliability Component (addresses issue of dropped packets)

1. Send token to next client on the ring.
2. Every time token is received, an internal token sequence number is incremented by 1.
3. When token received by a client, an acknowledgement with the same sequence number is sent
4. If no acknowledgement received within a 500ms timeout period, the client will resend the token.
5. If acknowledgement is dropped and a duplicate token is received (distinct via sequence number), acknowledgement is resent

Evaluate the portion of your design that deals with starting, maintaining, and exiting a game – what are its strengths and weaknesses?

	Strengths	**Weaknesses**
Starting	- Easy for users to join - just enter username and naming service address - the rest is handled by our code - Naming service ensures efficient startup process without the need for clients to elect coordinator - Pseudo-random player location reduces network latency, as	- Username uniqueness must be enforced - Must wait for a preset number of users to join before game starts - Players cannot join dynamically mid-game

	players don’t need to broadcast their locations since locations are pre-set across all clients	
Maintaining	- Guaranteed consistency using token ring algorithm - Token ring algorithm guarantees O(n) efficiency when waiting for token, which is fairly efficient - No reliance on centralized server, fully peer to peer - Fair - each client has to wait equal period of time before they can make their moves - Takes advantage of low latency characteristic of our environment, while not being affected by low parallelism characteristic (ring algorithm requires less parallelism since token passed one at a time) - Low bandwidth - moving token around doesn’t require a lot of network traffic. Movement via token ring happens in O(N) runtime	- Token ring algorithm is O(n) even when other clients don’t make any moves (still needs to pass token) - Not scalable - the design works within our constraints, but adding more clients linearly reduces the speed of the algorithm. - Client will see delay between the time they press their key and time the move is received
Exiting	- Easy to exit - players just exit their client - Continuous gameplay - Unless all players quit, the game continues, the fun goes on :)	- No message for server or other clients to shut down, so they can hang

Evaluate your design with respect to its performance on the current platform (i.e. ug machines in a small LAN). If applicable, you can use the robot clients in Mazewar to measure the number of packets and packet sizes sent for various time intervals and number of players. Analyze your results.

We don’t have full performance metrics because we haven’t written the code yet. However, we can perform analysis with estimates from ping experiments we’ve performed in the lab.

- UG machines have approximately 1-2ms of round trip time when pinging other UG machines
- We have a ring token algorithm which means at worst case, the token needs to make a full circle before events are processed by all clients in the ring.

- Assuming 1000 machines, this means events will require (worst case) 1 to 2 seconds to be processed. On average, it is 500ms to 1 second - approximately the delays in lab 2

How does your current design scale for an increased number of players for the type of environment you chose? You can give examples for environments that might fit the hypothetical scenario you chose e.g., game is played on desktops or mobile devices, wired/wireless. Use Big O notation in your performance analysis.

We chose **environment 2**, which has high network bandwidth, very low parallelism in network, very low network latency, and low end nodes.

Our design can scale in this environment because of low network latency. As the token is passed to each client once before being passed to a client for the second time, the latency is O(N). For example, if we have 1000 clients and a latency of 2ms, it will take at least 2000ms between two consecutive token possessions by one client. This consequently makes our design extremely latency sensitive, and our design will not work without low latency.

Our design also does not require high-end nodes since each node only has to process and enqueue its local moves once it retrieves the token. So at any time, it only needs to send one token and one acknowledgement packet. Furthermore, since each node only queues its own local movements, the memory required to run the game is reduced.

High network bandwidth ensures that our variable length token packets can be sent successfully.

Our design does not require much parallelism in the network since only one client is sending the token at any given point in time. This means that every time we add a client, we only add 1 more client to the token ring which means the time it takes to circle the token around the ring is linearly increased. Compared to multicast where each client broadcasts and acks everybody else, our design is much more suited to the environment specified.

Evaluate your design for consistency. What inconsistencies can occur? How are they dealt with?

In terms of movements and missile ticks, our design does not have any inconsistencies. We have achieved total order through the token ring structure.

Our design solves the inconsistency problem by providing a global ordering point through the token system. The token system ensures that at any point in time, only 1 client adds all its queued up moves (which is ordered locally) to the token queue and this set of moves will not be interleaved with any other client's moves. Thus, inconsistencies cannot occur. Also, since missile fires and ticks are now part of movement (sent as an event message), they will be consistent between clients as well.