ECE 419 2015 Lab 3
Design Documntation

David Yin(998270767) , Yong Ro Park (997011816)

1. Introduction

In ECE 419 Lab3, we are asked to decentralize Mazewar game clients from the server, allowing players to communicate in a peer to peer manner and synchronize amongst themselves. Hence, the game must be consistent and share same view of the global states among all clients in the game.

2. Design / Algorithms

2.1 Connection Setup

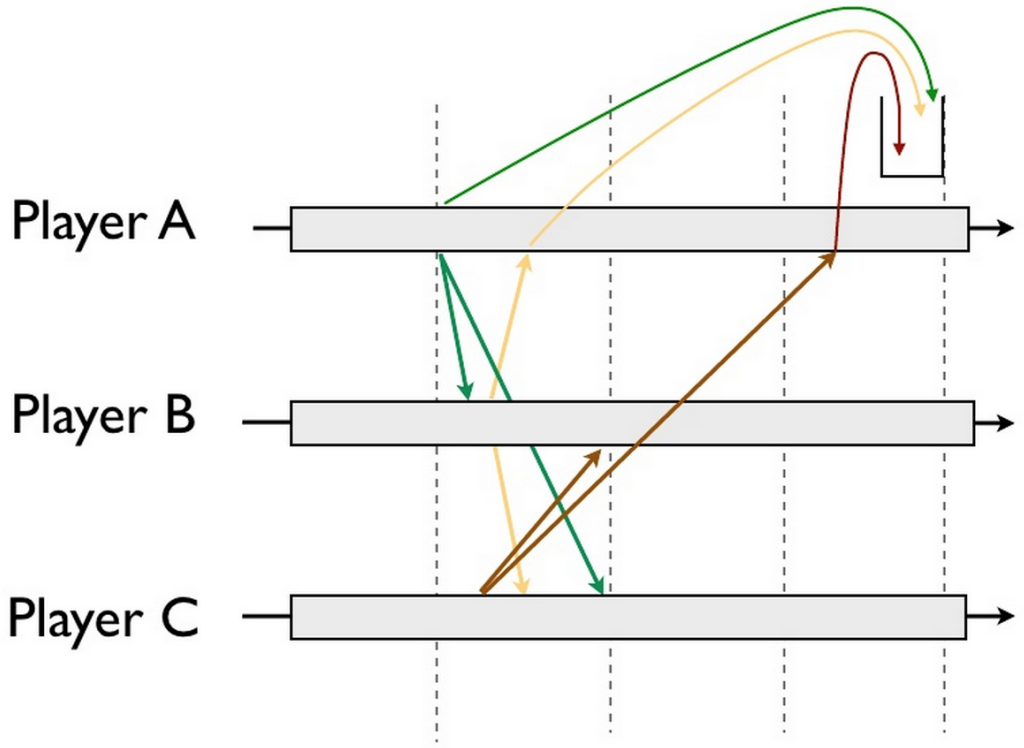
We'll use a naming service(centralized server) for initial connection setup. All clients who wants to join or initiate a game must go through this naming service. The naming service is responsible for:

- Making sure a client who wants to join a game has a unique username.
- Collecting clients' location information such as hostname and port number that the client is listening on.
- When a client joins, broadcasting collected clients' location information to all other clients.
- Sending start of game message when required number of players are met (as we will not handle dynamic join/leave)

Guideline

- Step 1: Client makes a join request with selected username to server (server will deny if same username already exists)
- Step 2: Server will include clients' socket information with the response. Upon receiving this message, client will broadcast to each of it's peers in the response notifying that client is joining the game.

2.2 Synchronization mechanism : Bucket Synchronization



Bucket Synchronization, Wei Tsang Ooi
(http://www.slideshare.net/weitsang/lec6-p2p-sync)

Before we start talking about the synchronization mechanism, we must make sure that each peer is responsible for broadcasting only the event consistent with its own avatar such as shoot, turn, forward, backward, death, and its spawn location.

Upon game start, we will use bucket synchronization to maintain the consistency of the game. In a bucket synchronization, the game is divided into rounds of *n* milliseconds. We chose *n* as 200 milliseconds hence there will be at the most 5 updates per-second. However, if we notice lag during the game purely due to the number of updates per second, we can always adjust *n* to smaller interval.

Each round is allocated with a bucket, which buffers both local and other peer's event. This bucket buffers both local and remote events according to message's sequence number and those events will be executed upon receiving all events from other peers. This means that when there is no event on a peer, it will have to broadcast an empty event message to

other peers. Since each message from each peer at round *i* is tagged with the same sequence number, this will guarantee the consistency of the game.

2.3 Communication protocol

We will use TCP as our transport layer protocol since we must guarantee the delivery of the message. We have two distinct types of packet - *ControlMessage* and *GameMessage*

Control Message is used for administrative purpose for game setup operations such as requesting peer informations on naming service and setup connection between peers.

messageType	username	inetAddr	portNo

Figure 2.3.1 ControlMessage Packet

(int) messageType: JOIN_GAME_REQUEST | JOIN_GAME_REQUEST_SUCCESS |
JOIN_GAME_REQUEST_FAILURE_USERNAME_EXISTS |
JOIN_GAME_REQUEST_FAILURE_MAX_CLIENT_REACHED |
CONN_INIT_REQUEST | CONN_INIT_SUCCESS
(string) username: peer's username
(InetAddr) inetAddr: peer's hostname
(int) portNo: peer's port number

Game Message is used for in-game actions such as movement, fire, respawn and etc.

messageType	username	clockValue

Figure 2.3.2 GameMessage Packet

(int) messageType: GAME_MESSAGE_TYPE_SPAWN_PLAYER |
GAME_MESSAGE_TYPE_MOVE_PLAYER_FORWARD |
GAME_MESSAGE_TYPE_MOVE_PLAYER_BACKWARD |
…
GAME_MESSAGE_TYPE_PROJECTILE_TICK |
GAME_MESSAGE_NULL

(string) username: sender's username
(int) clockValue: logical clock value of this mesage

3. Discussions (Q/A)

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

By utilizing a naming service, we can easily locate client who wants to join the game and broadcast it's socket information to every client. Once a peer has set up sockets to each of the other peers, it will send a READY message to server. Hence, the centralized server will know exactly when to start a game. Once the server knows that every peer is ready, the server will send a START message to each client. Therefore it will be a synchronized start. We guarantee synchronization during the game by using bucket synchronization as explained in *section 2.2*. Note that our protocol works like Stop-and-Wait protocol. This guarantees that every client will share the same states during the game. However, if one client suffers due to bad latency, every other clients will be affected which will result in lag in the game. Another weakness is that each peer has to broadcast a message in every round, in our case it is 5 messages per second.

Q: 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 sent for various time intervals and number of players. Analyze your results.

There is no way our message packet is greater than Maximum Transmission Unit(MTU) which is 1500 bytes. Therefore, our packet fits in a single Ethernet Packet. Therefore, transmission delay which is calculated as a function of packet size is not an issue. In a small LAN environment like the ug machines, propagation delay is also not an issue. By setting our bucket synchronization's round interval to 200ms, we will broadcast 5 packets to each clients per second, and receive 5 messages from each client per second.
Number of messages exchanged in every second can be generalized as:

*2 * (NumPlayers - 1) * 1 / roundIntervalInSecond*

Q: How does your current design scale for an increased number of players? What if it is played across a higher latency, lower-bandwidth wireless network – high packet loss rates? What if played on a mix of mobile devices, laptops, computers, wired/wireless?

The number of messages that need to be exchanged is a function of the number of players which scales linearly according to the generalized equation in the answer above. Lower

bandwidth would not cause lag in our case since the size of each packet is small. However, if latency (transmission delay) is bad for one player, everyone else in the game will suffer as well since we're using Stop-And-Wait like protocol.

Q: Evaluate your design for consistency. What inconsistencies can occur? How are they dealt with?
In our design, global game states and event ordering is guaranteed to be the same. However, each players might notice frame rate difference because their frame rate is dependent on the received packet.

References

http://www.slideshare.net/weitsang/lec6-p2p-sync
http://web.eecs.umich.edu/~sugih/courses/eecs494/fall06/lectures/lecture9-networking.pdf