ECE419 Lab 3 Design Document

Due to the fact that my lab partner dropped this semester as a result of her severe injuries during a skiing trip after the initial design document, this lab is carried out by myself, and major design changes have been made to accommodate the schedule and workload. A simpler design is employed and a few optional features are skipped due to time constraints.

---

1. Design Description:

1.1 Overall distributed algorithm functionality, components, interactions

The overall design uses a centralized sequencer to issue a sequence number of all events happening among peers. Peers inter-connect to each other and also connect to the sequencer service. When an event is taking place, it will be first sent to the sequencer service first to obtain an incrementing sequence number, and then broadcasted to all peers. Each peer keeps a queue sorted by the sequence number and implements the queue head according to the sequence number. Therefore all events will happen in the same order across all nodes.

1.2 Communication protocol – packet types and formats

Packets contain information about

- the sender's name, used to identify client object on the receiver's side to apply game events
- sequence number, used to order events on the receiver's side
- event, used to indicate which event is needed to be applied on the receiver.

The packet may also contain information about the client's location and score, which is used for new player joining the game

1.3 How a player locates, joins and leaves a game of Mazewar

The game is designed to support dynamic joining and leaving the game. The game is located by using lists of host addresses and port number. Each peer knows whom the expect when starting the game, and it will connect to the peers that's previously in the game as "client", and operate as "server" for peers who will join the game in the future.

When joining the game, the new peer will send out an initialization packet to all peers currently in the game. The peers already in the game receive the initialization packet from the new player, and they will reply with initialization packets, indicating their current location and score. The init packets are then received by the new player to create clients and eventually get a synchronized copy of the game locally. The initialization packets are numbered by the sequence number, however they will not cause the sequencer service's internal counter to increment. This is because order isn't necessary among the initialization packets, and they will be assigned the current sequence number to order them against other game events.

When a player leaves the game, it will broadcast a QUIT event to all peers. Peers receive the event and they will take the leaving client off the broadcasting list and other peer listening threads, and finally remove the client from the game.

1.4 What happens when a process loses contact with another process?
It will detect the peer is disconnected and it will remove the peer from the game and its broadcasting list.

1.5 Timings of protocol events, and how they provide sufficient consistency for the game state
Game events are ordered by its sequence number. Packets of game events may arrive at a peer out of order, and the game consistency is ensured by using a priority queue ranked by the event's sequence number. As the sequence number increments 1 by 1, the peer process will wait until any game resulted from early arrival of later packets are filled. The peer process will then dequeue the head of the queue and apply it to the game application.

1.6 Pseudo-code or an abstract description of process behaviour
Each peer process executes the following:
- Create a GUI client
- Contact the sequencer service to establish connection
- Connect to existing peers in the game. for each peer, creates a handler thread which reads the input stream and enqueue events onto the event queue sorted by sequence number
- Start a thread to act as server to later peers who wants to connect to this peer
- The peer process enters the main loop to dequeue events from the event queue and apply events to the game application in order
- The key listener will create the appropriate packet based on the keypress, obtain a sequence number from the sequencer service, put the event on its own event queue and broadcast the packet to other peers.

The sequencer service executes the following:
- Start a thread as the server to handling each client connected
- The sequencer keeps an internal counter of event's sequence number. When contacted by a client, it will reply with the current sequence number, and increment the internal counter by 1.
- When contacted by client with initialization packet, it replies with the current sequence number but does not increment the counter.

---

2. Additional Questions:

2.1 Evaluate the portion of your design that deals with starting, maintaining, and exiting a game – what are its strengths and weaknesses?
The game was designed with the idea of dynamic joining and leaving in mind, and it handles player's dynamic behaviour really well. The weakness is the centralized sequencer service: although it does not broadcast to clients, it is identified as a single point of failure.

2.2 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.
Robot player is not implemented due to time constraints. However, missile tick can be used to for testing performance. For a 2-player local game, the game runs smoothly with missile ticks broadcasted from one player up to every 5 millisecond, and it starts to fail when the broadcast

interval is reduced to 1 millisecond. For a 4 player multiplayer game, the game starts to fail with missile tick period down to 10 milliseconds, which would translate to 25 moves per second per player. The result indicates the game is fairly robust as a multiplayer game for human players, and it should be capable of handling more peers.

2.3 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?

Because the game supports dynamic join and leave, it's very simple to increase the number of players. With more players, the sequencer would start to become a bottleneck, as it has to handle the increased number of players.

When the game is played over poor network connections, delays will be introduced as the peer processes must wait for the sequencer service to respond before an event can be applied. If the connection is lossy, the game will fail as there is no implementation of ACK and timeout, with the assumptions of reliable FIFO network (TCP/IP).

If it's played on the mix of devices, it depends on the nature of the networks. It is more likely to fail with inconsistencies due to lost packets as the current design is not robust enough to handle lossy networks. Delay would also become an issue as the bandwidth may vary. But the dynamic join and leave feature can allow players to join and leave freely without interrupting other's gameplay, no matter by choice or lost connections.

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

Packets from different peers arriving out of order:
Sequencer service and synchronized priority queue sorted by sequence number ensures all packets are applied in the same order at any client

Synchronizing to the current sequence number when joining a game:
Each peer must know what the current sequence number is, and initialization packets ensures new player is "on the same page" as the existing players regarding the sequence number. The initialization request also does not increment the sequence number in the sequencer service, and hence all peers will have the same sequence number after the new player has joined.

Score and client location when dynamically joining a game:
The score and client location is sent in the initialization packets when new player joins the game. The new player will receive initialization packets from each existing peer, and it will use the information to create the game locally and stay consistent.

Random number:
The random number generator is set with the same seed initially. However it would cause inconsistency as the new player joins the game, who has no knowledge of the current status of the random number generator. This issue is handled by using the player's unique name string to initialize the random number generator when a new player joins the game. Therefore the player location is determined in a consistent manner as the name is unique.