University of Toronto
Faculty of Applied Science and Engineering
ECE419 - Design Document

Date	March 14, 2016
Prepared By (Names and Student #s of Team Members)	Shafaaf Khaled Hossain(998891515) Sai Kiran Varikooty (998440307) Abdelrahman El Shimy (998695540)

Distributed Algorithm

The distributed algorithm will be implemented using total order multicast based on sorted queues for each client based on their lamport clocks. When a client makes a move, its lamport clock is incremented by 1, and the event generated from the move is added to its own queue for the client along with the updated lamport clock value. The client then broadcasts the event that is at the head of the queue to all clients, as there could be many events at the queue.
For each other client in the game, when the client receives an event it will place the event in it's queue in the correct position based on the logical clock value and then broadcast the acknowledgement for the head of the queue to all other clients in the game. If it has been sent already, it will not send again.
The client will also wait for acknowledgements from all other clients for this event. When every client has received an acknowledgment from every other client in the game for the head element of the queue, it will dequeue the event and execute the event.
Here each client will only execute when it has n acknowledgments and so has to ack itself.

Timing of Events

To maintain the correct order of packets, clients will use a logical clock value to assign to every event. A client will examine the logical clock value of an event to determine where to place the event in it's queue. If a client receives an event with a logical clock value that is smaller than the event at the head of its queue, it will place this newly received event at the head of the queue. If the logical clock value of the event it receives is greater than the event at the head of its queue, it will place the event in a later position in the queue. The use of logical clocks allows events to be totally ordered and the use of sorted queues for clients will allow events to be executed in the proper order.
All this works because eventually all clients will have the same event as its head in its queue.

The client broadcasting the event will first put it in its queue and then broadcast and therefore make sure it would block a later event from getting all acknowledgements. This is the key in making sure events are consistent.

Ties of events with same Lamport Clock value are broken with their unique pid. The client with the lower pid will have the lower lamport clock valued event.

Communication Protocol

A player locates the Mazewar game by querying a well known address of a naming server. The naming server will contain information about clients currently playing in the game. Particularly, the naming server will have all the clients' ip addresses and ports to be connected to. When a new client tries to join the game, it will send a packet to the naming server which will contain the client's ip address and port. The naming server stores the client details and replies back with the list of stored clients and their ip addresses and port numbers. The new client will then proceed to connect with every client from that list and joins the game.
All clients are connected to each other via a TCP connection which ensure reliable stream service using a packet drop rate o.

Joining and Leaving

To join the game, players must run the client and specify a port which they will be listening to. The client then registers itself with a nameserver and obtains a list of other players who are connected, it then connects to all of them. We do not support dynamic joins or quits in our design.

Our current implementation does not handle failures or degrade gracefully.

Strengths and Weaknesses:

Joining and Leaving

- Our design assumes a fixed number of players will join the game and wait until all players have joined before beginning the game.
- This does not allow players to dynamically join the game and if players leave the game, the design does not handle this gracefully. Thus a weakness of this approach is that the game is not dynamic.

Performance

- On the ug machines on a small LAN, although there are observable delays in playing the game, it can be played reasonably well without delays significantly affecting the ability to play the game.
- For the environment of high network bandwidth, high parallelism in network (many routers), high network latency, high-end nodes:

    - This environment is preferred since our algorithm uses total ordered multicasting. Acknowledgements can be sent in parallel due to high network parallelism and bandwidth.
    - High end nodes allow the maintenance of queues and multiple threads executing on each end node efficiently.
- Increased number of players would make the game slower. This is because the global ordering method we choose is Total Order Multicast and its complexity is O(N^2).
- This would make the game extremely slow as every client would be waiting for n-1 acknowledgements from other clients. Even if a client has all acknowledgements for an event it would only execute that if it is the head of the queue anyways.
- Since specific packets are expected, the whole system may wait based on just 1 client not receiving a specific packet. Therefore it could be said that the game can be played on any enviroment like desktops, mobile devices etc and would be consistent but would be slower with more number of players as more packets could be dropped and this blocking events.
- The game would not scale as much, but would be consistent.

Consistency

- The game uses Total Ordered Multicasting to show consistent moves of players on different clients. Therefore all clients would see the same screen at any time.
- The game is also consistent regardless of a FIFO channel since the design uses Lamport clocks and total ordered multicasting to maintain consistency of events among different clients
- However, our design did not implement a reliable channel and does not account for packet dropping and thus these inconsistencies appear if packets are dropped.