Design Document

Submitted On: March 17, 2015

Design Team

Programming Team 1:

Naba Sadia Siddiqui (998072945)

Sameer Patel (998352526)

Programming Team 2:

Hossein Haider (998423353)

Taras Vertypolokh (998470747)

Programming Team 3:

Ahsan Sardar (998422158)

Fateh Singh (998088199)

1. Design Overview

The implementation of a decentralized version of Mazewar will be realized through the application of Token Ring algorithm and a naming service. The naming service will be used to locate players who have joined the game. The Token Ring algorithm will be used to achieve mutual exclusion on each action that can be performed in the maze.

2. Components

The main components of the game are:

- Token-Ring algorithm
- Naming service
- Communication protocol

2.1 Token-Ring algorithm

This algorithm is responsible for synchronizing player actions on the maze. One token will be shared among all nodes (players) in the ring. A node (player) can only perform an action if he/she has the shared token. If the node has the token, and it wants to perform an action, it will lock the token. The node will then perform that single action (e.g. MOVE_FORWARD or FIRE) and multicast it to other nodes in the ring. It will keep the lock on the token till it has received N-1 ACKs from the nodes that they have performed the action. This will ensure consistency across all nodes. When the node holding the token gets N-1 ACKs, it will release the token and pass it to the next node in the ring. If a node receives a token but does not wish to perform any action, it will simply pass it on to the next node in the ring. In this fashion, if no one needs the token, it will circulate in high-speed across the ring.

2.2 Naming service

The naming service is a server to which all clients will register against by sending their identity (name) and location (port number and hostname). The naming service will store this information in a concurrent hash map. Each key-value pair in the hash map will be of form <client name,

client location>. This information will then be multicasted to all N players in the game. In this way, all clients will be able to locate other players in the game. The game is not implementing dynamic joins. Every player will wait for others to join the game before starting.

The naming service will make the following assumptions:

1. All clients will connect from a different location.

The clients will making the following assumption about the naming service:

1. It is started before any client has joined the game.
2. It is reliable (i.e. it is always running).

The naming service however will **not have fault tolerance**, despite these assumptions the client will make.

When a client quits the game, it will send its state to the naming service which will remove the client from its data structure that tracks all nodes in the game. The client will also multicast this action to other peers in the network, who will remove this node from their local circle.

2.3 Communication protocol

Each packet that a client sends to other players in the game will consist of the following information:

1. Client identity (name)
2. Client action (e.g. forward, fire, etc.). This is the critical section it is trying to enter

3. Design Evaluation

3.1 Starting, maintaining and exiting a game

The game does not support dynamic joins. This is a weakness because every player has to wait for every other player to join the game. Furthermore, the server needs to know exactly how many players will join the game. If the server is started with the knowledge that only 2 players will join the game, then only 2 players can join the game after the server has started. If after starting the server, players decide they want to change the number of players, they will have to stop the server, set the new player mode and then start the server again.

The server is only needed for lookup service, but it is still recommended to keep the server running. This will allow the server to track which players have quit the game. Regardless, once the game has started, the server can quit or crash without effecting any other aspect of the game.

Another strength is that exiting the game is dynamic. That is, a user can quit the game and let other players continue playing.

3.2 Performance

The performance of the game on the ug machines is actually very fast, and moves are very smooth. The only downside is the number of packets which are exchanged. Each trip of the token around the circle involves sending N packets. Additionally, if any player wants to perform an action and has the token, he/she will send N-1 packets and wait for N-1 ACKS. But the rest of the players will send the ACK to every other player (not just one who multicasted the action). So whenever an action is performed, a total of (N-1)*(N-1) ACKS will be passed around. This is $O(N^2)$. So, one can argue that the performance in terms of number of packets exchanged is really poor when an action is performed. Furthermore, all these packets are sent over TCP. TCP headers have a high performance overhead. Therefore, as the number of packets increase, the overhead from TCP headers also increases. Such high bandwidth is of little concern when the number of players is small (i.e. 2-4). However, as the number of players will increase, the amount of bandwidth consumption will become a performance bottleneck.

3.3 Scalability

Given that the communication is based on TCP/IP and the large number of packets exchanged per action executed by a player, scaling the game (which is easy and very possible with current implementation) will make it considerably slow. At the same time, TCP is a reliable networking protocol. Therefore, there are no concerns related to packet losses, etc.

3.1 Consistency

The algorithm ensures consistency by implementing mutual exclusion on moves, and only releasing locks on the critical section once every other player has sent back an ACK. Since TCP/IP is a reliable networking protocol, we can be assured that no packets will be dropped and all communication will go through. As a result, all moves will be synchronized and consistent across different screens.