Vlad Baderca, 998894488
Bohan Zhang, 998310162

Introduction

The following design document explains the methods used by our group to change the centralized version of Mazewar to a fully working, decentralized version of the game.

Naming Service

The only centralized component of the game will be the naming service. When a new client joins the game, the client will connect to the naming service through a known port number and hostname. The naming service will then transmit the new client a list of currently existing clients, followed by a SERVER_DONE packet so that the new client is aware it has finished receiving all existing players. After a new client gets the list of already existing players, the new client will draw the existing players first, then draw itself, and then notify the naming service of its hostname and the port it is listening on. Clients stay connected to the naming service for the duration of the game, so that they can be notified when a new user joins the game.

When a new client joins the game, the naming service will notify all currently existing clients of the hostname and port number of the new client. They will then draw the new client on their own end. Our game assumes that all clients have joined before any player makes a move.

Handling Player Actions

The game must not rely on a dedicated server when handling player actions. To avoid information collisions, our design uses a token ring. The naming server creates and passes the token to the second client that joins the game. Our implementation has three running threads: one thread to listen to the naming service, one thread to listen to keyboard presses and queue up actions, and one thread to listen to incoming information from other clients/ naming server, as well as broadcast actions to other players when the token is acquired. When a user presses a button, the respective action is enqueued in the “actions” queue. When a client receives the token, they immediately check the actions queue. If it is not empty, they dequeue **one** action from its queue to broadcast to all other clients. We only dequeue one action per token acquire to avoid possible hogging by a single client.

Broadcasting is done manually in our implementation; we do not use the multicast library available in java. To broadcast an action, a client connects to the port of another client using TCP, and sends a packet describing the action that is taking place. The client who receives the action responds with an ACK, and then draws the action on their end. After the ACK is received by the token holder, the token holder connects to another client, and the process repeats until every client has been notified of the action. Once the broadcast is finished, the token holder passes the token to the next client in the token ring. The player making the broadcast draws the position of itself on the map once it receives an ACK from the remaining players.

Token Ring

To ensure every client has the same ring order, a protocol is followed. The naming service transmits a new client the list of already existing players in the same order as the ring order. Then, the new client is added as the last element in the ring order. After receiving the list of existing players, the client creates its own local ring order, and adds itself as the last element in the ring order. This way, we ensure that no token loops occur, which would not allow certain clients to make any moves.