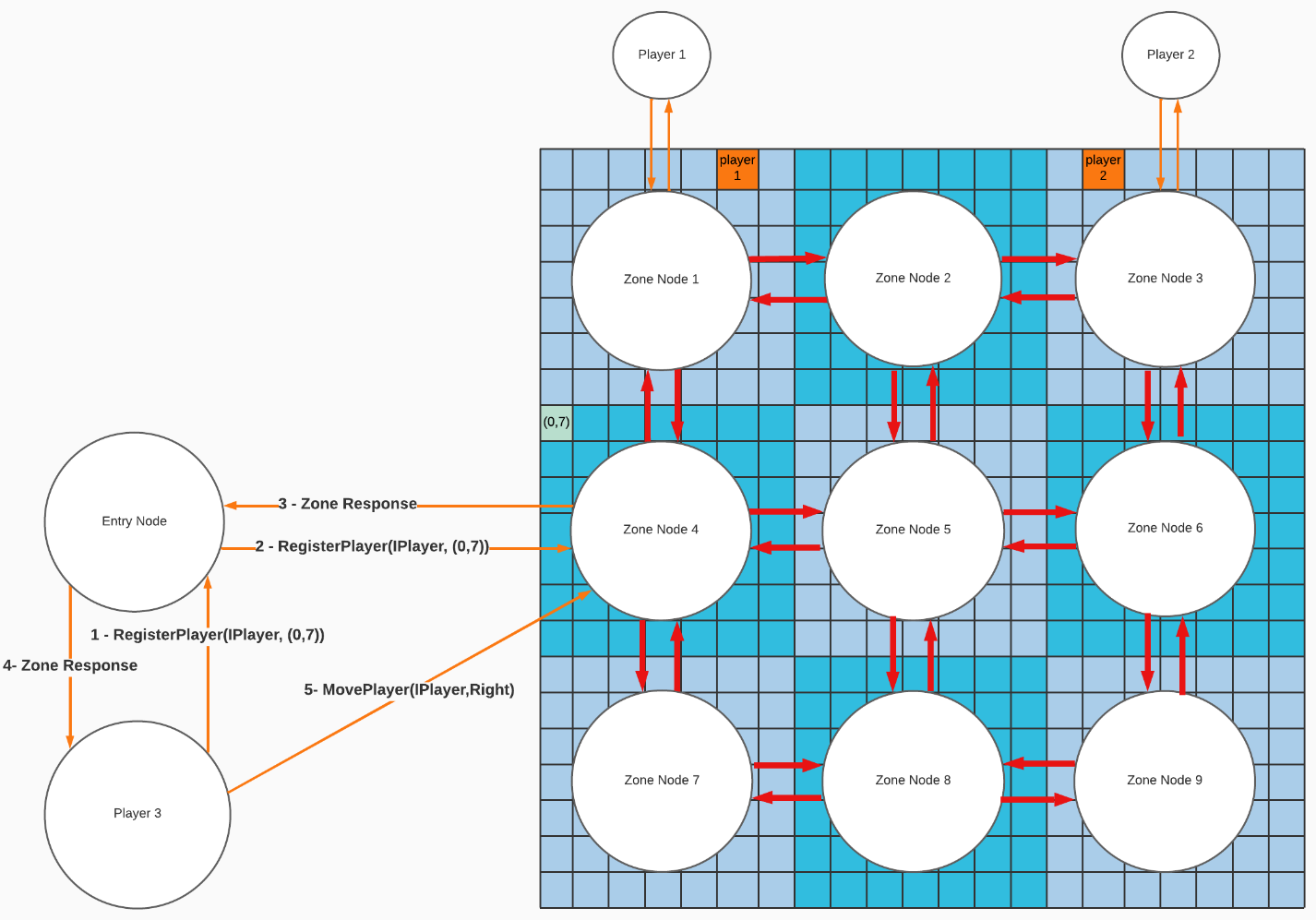
Introduction to Distributed Systems

**Lab 2: Chat Application**

Abbas Khreiss

Farah Maria Majdalani

# Software Architecture



The clients start by registering to the server. They have the option to send a message, view history or logout (unregister). Upon sending a message to the server, all the other registered clients will receive that sent message.

# Global Functionality

## Server Functionality

Server starts, creates an instance of the chat service and registers it into the RMI registry.

## Chat Service Functionality

* It contains a list of all messages sent to the sever.
* It contains a Concurrent Hashmap that contains usernames associated the users’ remote object.
* It saves and restores the messages from a file called ‘Messages.bin’.
* Once the service is created, it restores all messages found in ‘Messages.bin’.
* It contains four functionalities: register, unregister, send message, get history.
  + The register function adds the user to the list of registered users if he was not registered beforehand.
  + The unregister function removes that user from the list of registered users.
  + The send message function checks if the sender is registered. If so, the message is sent to all the registered users.
  + The get history function returns all the messages from the list of messages previously sent to the server.
* If the server fails to call a user’s remote object, the user is unregistered from the service.  
  ex: if the client abruptly closes the terminal or performs Ctrl-C.

## Client Functionality

Gets the Chat Service from the RMI registry.

The Client will try to register to the server by sending a remote object that allows the server to get his username or send him messages. If the username exists, then he will be asked to enter another unique one. Then, the history will be displayed with the ability to send messages to the chat room. The messages received from the server are colored in cyan. In order to unregister, they could use the ‘\exit’ in the chat.

# GitHub Repository Link

<https://github.com/AbbasLB/IDS_Game_Project>

# How to compile and run

If “gnome-terminal” is installed, run the following script, it will compile the code, run all nodes, and start 3 player clients:

./run\_game.sh matrixSize splitSize

For Example, the following command will create a 20x20 map managed by 4 nodes (2x2 nodes):

ex: ./run\_game.sh 20 2

Compile the code and create the jar files by running the ant command:

ant

Run the rmiregistry using the provided script:

./start-rmi.sh

Run the Entry Node with the preferred parameters:

java -jar dist/EntryNodeServer.jar matrixSize splitSize

For Example, the following command will create an entry node that handles 20x20 map managed by 4 nodes (2x2 nodes):

java -jar dist/EntryNodeServer.jar 20 2

Start Zone nodes depending on the splitSize set, you need splitSize\*splitSize nodes:

java -jar dist/ZoneNodeServer.jar localhost

Run a player client:

java -jar dist/PlayerClient.jar localhost

# Possible Improvements

1. Create a GUI for the client, as there are a lot of limitations using the terminal.
2. Notify all the clients when someone joins or leaves the chat room.
3. Add a timestamp to the messages.

## To Include:

* Unique Identifier
* How to build
* Terminal that supports ansi escape codes to run the game
* Why rmi
* Run on both win and linux
* Handled player crashing ()