Introduction to Distributed Systems

Project: Game

Abbas Khreiss

Farah Maria Majdalani

Software Architecture

The project is a distributed game. It is made up of a playground separated into zones, where players can connect to and move freely inside it. When two players get close to each other, they say “Hello” to one another.

There are three main entities in our design:

- Entry Node:
    - The entry node is responsible for setting up the playground during the initialization phase, where it assigns a specific section of the playground to each zone that will be registered, and it links the neighboring zones together.
        - Once all the zones are linked together, it marks each zone as ready for them to start accepting new players.
    - It allows players to register to the game by registering them to a specific zone corresponding to the input coordinates.
- Zone Nodes:
    - Upon registration through the entry node, each zone is given its own zone base and bounds with respect to the full playground.
    - Once the zone is ready, it will start accepting new players to be registered to it at a specific location in the zone’s grid.
    - It allows the player to move around the zone, checking if the player’s position is valid within the zone, in addition to registering the player to a neighboring zone if the player wished to move to that location in the playground.
    - Players can unregister from the zone, removing them from the playground.
    - The zone keeps the zone players updated about the postions of the other players in the same zone.
- Players:
    - Upon registration through the entry node into a specific zone, the player can communicate with its zone.
    - The player can move in four directions in the zone playground: up, down, left, and right.
    - It receives updates from the zone about other players in the zone, as well as, messages from the zone that may include its neighbor saying “hello” to it.

In the game, the player is allowed to move freely in the playground as long as there is no other player in the position they want to move to. They can only see the players that are in the zone they are registered to, and the rest of the zones are blackened out. When the player moves right next to another player, they both send “Hello” to each other.

Players crashing is handled by the zone, where the zone broadcasts to all the players inside it that that player disconnected. However, fault handling of the entry and zone nodes was not implemented. If the entry node gets disconnected, the game can still run normally but cannot register new players.

The project is implemented in Java RMI. All the entities mentioned are Remote with their respective interfaces.

IEntryNodePlayer:

- (IZoneNodePlayer zoneNode, String Message, boolean success) registerPlayer(IPlayer player, PlayerCoordinates coords);

The entry node exposes this interface to the player. It allows the player to register anywhere in the playground, and checks the validity of the player registering to the input coordinates. It returns the status of the operation. If the operation is successful, it returns the zone node with an optional message. However, if it fails, it returns an error message without a zone node.

IEntryNode extends IEntryNodePlayer:

- (int XBase, int YBase, int XBound, int YBound, int matrixSize) registerZone(IZoneNode zoneNode);

The entry node exposes this interface to the zone. It allows the zone to register to the game, giving it its own base and bound with respect to the playground matrix size.

IZoneNodePlayer:

- (IZoneNodePlayer zoneNode, String Message, boolean success) registerPlayer(IPlayer player, Coordinates playerCoordinates);
- void unRegisterPlayer(String playerId);
- (IZoneNodePlayer zoneNode, String Message, boolean success) movePlayer(IPlayer player, Direction direction);

The zone node exposes this interface to the player. It allows the player to register anywhere in the zone, and checks the validity of the player registering to the input coordinates. It also allows the player to unregister from the zone. Moreover, it allows the player to move throughout the zone grid. The player can move one cell in four directions: up, down, left, and right.

IZoneNode extends IZoneNodePlayer:

- void linkNeighbors(ZoneNeighbors neighbors);
- void MarkAsReady();

The zone node exposes this interface to the entry node. It includes all the actions the player can do when it extends IZoneNodePlayer. The entry node can provide the neighbors that the zone would be linked to – i.e. the neighbors from the top, bottom, left, and right of the zone. The zone is marked as ready when all the zones are linked to their respective neighbors.

IPlayer:

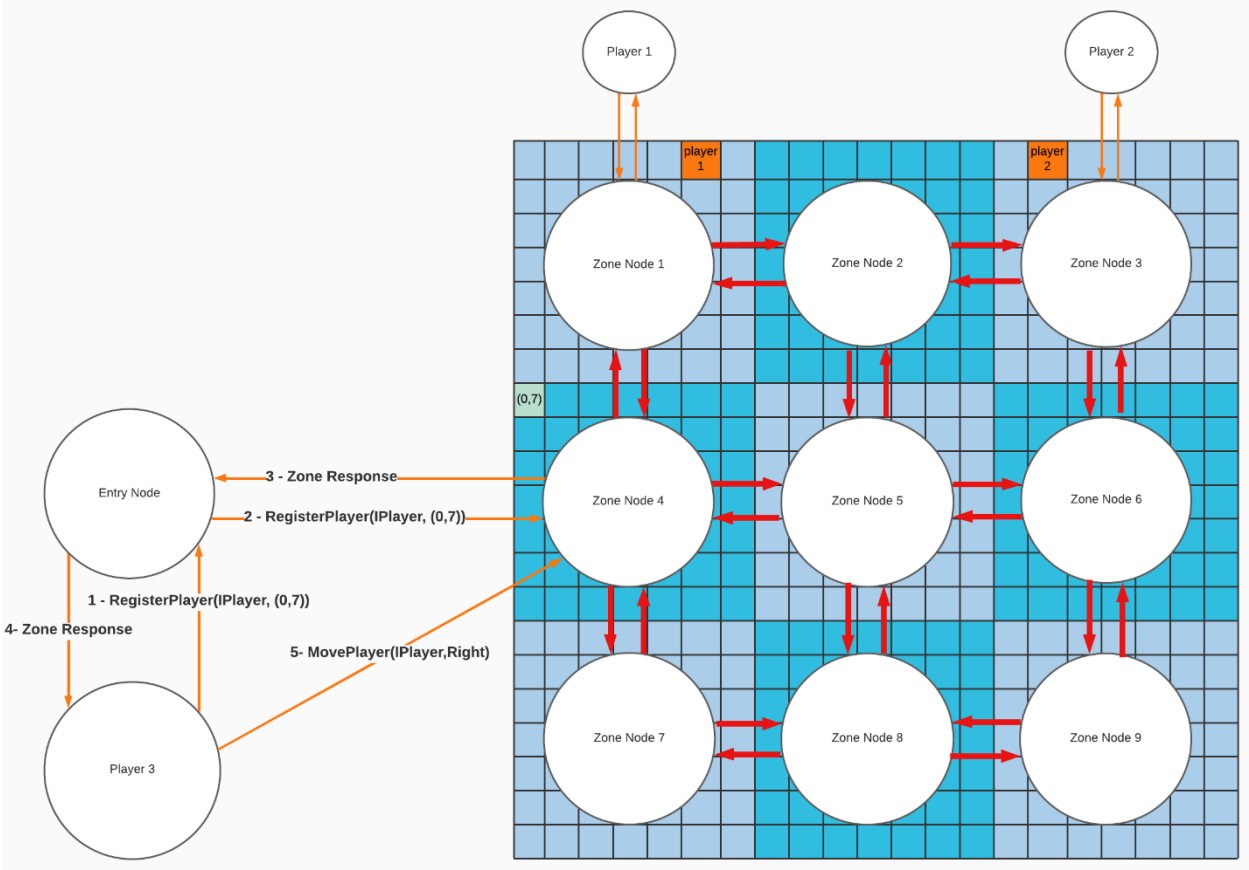
- string getId();

- void setId(string id);
- void receiveUpdate(HashMap<(string playerId, int x, int y)> players, boolean zoneChanged, ZoneDescription<IZoneNodePlayer> zoneDescription);
- void receiveMessage(String message);

The player exposes this interface to the entry and zone nodes. The nodes can get and set the player's id to assign a global unique identifier to it. The entry node is responsible to set that unique id. The zone node sends updates to the player, and sends the zone description if the player registers to a new zone. The zone description includes the zone's base and bound and the full playground size. This description is used to draw the playground in the player's client. The player can receive any message from the nodes, for example, receiving a "Hello" message from other players.

Java RMI was chosen for two main reasons:

- Synchronous Messges:
    - Many messages in our design require synchronous communication. For example, when a player wants to move between zones, the zone needs to communicate with its neighbors, but it should know that the user is registered to the new zone before unregistering it from the previous one, or else problems would be faced like disconnecting the player from the game if another player was already there at that location. This can be easily managed using RMI.
- Clear Interfaces:
    - The design and interfaces are clearly organized, and the roles and functionalities of each entity are easily understandable.



Implementation

EntryNode.java

- The entry node class implements the IEntryNode interface.
- It takes the playground size and the split size. The split size specifies the number of zones that would be split vertically and horizontally. For example, if we have a split size equal to three then the playground will be split into 9 zones with 3 columns and 3 rows.
- It registers the zone nodes until the required number of zones are registered. It specifies the bases and bounds corresponding to the full matrix size and sets them in the zone description. When all the zones are registered, it links them together and marks itself and all the zones as ready.
- When the player wants to register, it finds the zone corresponding to the coordinates that were picked. Then it tries registering the player in that position in the zone, returning the zone response back to the player.

ZoneNode.java

- The zone node class implements the IZoneNode interface.
- It contains the neighboring zone remote objects to that zone – the top, bottom, right, and left zones.

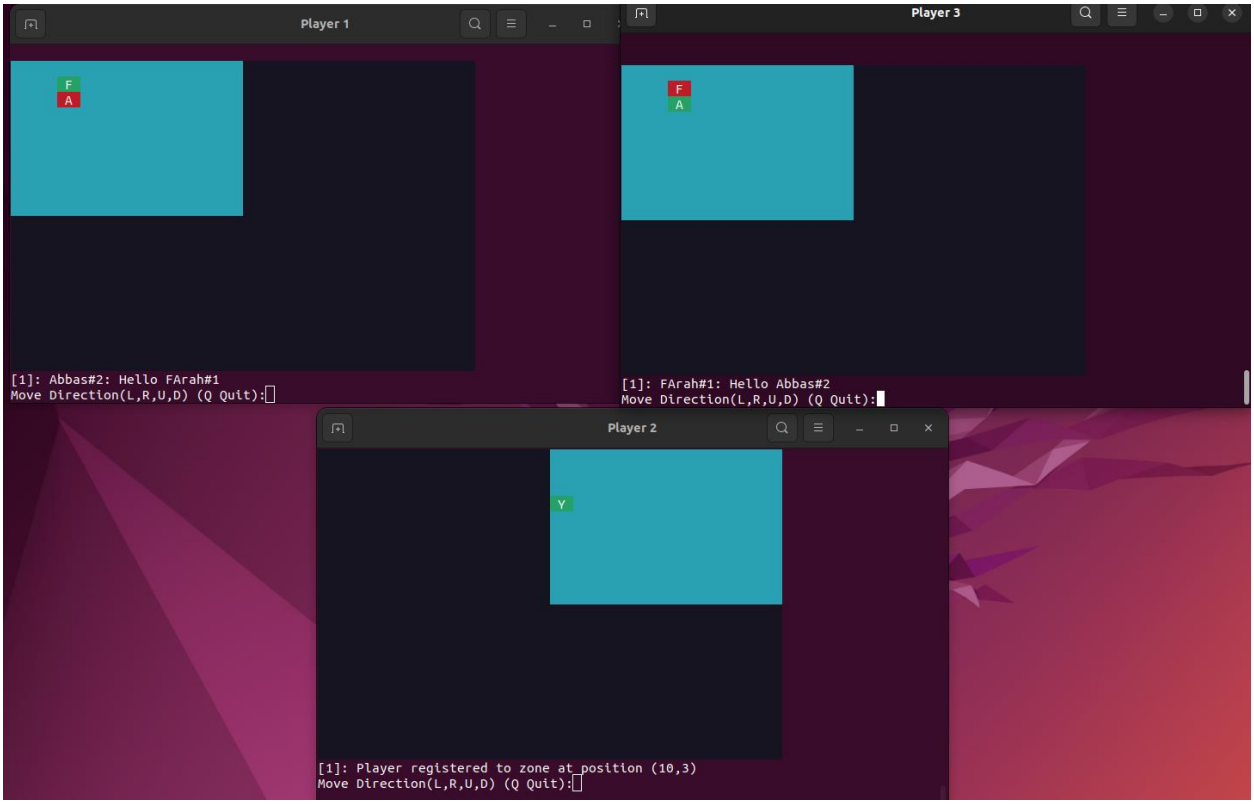
- It also contains two concurrent hash maps, one mapping the player's id to its coordinates, and the other mapping the occupied coordinates to the IPlayer remote object and its id to always know the id of the user in case of disconnection of the IPlayer.
- The register players function fails if the zone is not ready, or if the player is already registered, or if there is another player in the target coordinates. If the destination coordinates are occupied by a player, it tries getting its id from the remote object to check if that player is still connected. If it returns an error that means that the player has crashed and is removed automatically from the registered users and the map, and the current player is allowed to register. If the destination coordinates are free, then the player is allowed to register. If the player is registered near other players, it sends and receives "hello" from them.
- In the unregister player function, it removes the player from both hash maps.
- The move player function fails if the player is not registered, or if the coordinates are out of bound, or if the cell is already occupied by another player. If the player is moving in the same zone, we update its coordinates in the internal hash maps and sends "hello" if moving next to other players. If the player is moving to another zone, then the current zone tries to register to its neighboring zone, returning the result. On success, the player is unregistered from the current zone.

Player.java

- The player class implements the IPlayer interface.
- The player has an id.
- It contains a hash map that maps other players in the zone to their coordinates, in addition to, a linked list of the last three sent messages.
- Upon receiving an update from the zone node, the hash map containing the other players' coordinates is updated and the UI is redrawn based on the new information received.
- When a new message is received, the UI is redrawn to display the new message.

GUI

The GUI is drawn in a function called DrawUI in the Player.java. Based on the zone description and the playground size, it draws the current zone in Cyan, and the rest of the zones would be blackened. The current player is colored in Green with the first character of his name inside. The other players in the same zone are colored in Red with the first character of his name inside. The player would be prompted to set which direction they want to move. Moreover, the last three received messages will be displayed.



GitHub Repository Link

https://github.com/AbbasLB/IDS_Game_Project

Deployment

Requirements:

- JavaJDK
- Java Runtime
- Ant (for building the project)
- Terminal supporting ANSI escape codes

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
```

If “gnome-terminal” not is installed, 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
```

Conclusion

Possible Improvements

- Create a better GUI for the client, as there are a lot of limitations using the terminal.
- Add a timestamp to the messages.
- Handle failures when entry and zone node disconnect.
- Add more functionalities, like allowing the player to send custom messages to other players.

What we learned

- We learned about how to design and implement a distributed solution.
- We learned how to deal with concurrency issues.
- We learned to separate concerns into interfaces.