

CS1234 - L12

The Blind Maze

Background:

A maze lives entirely on our server. You cannot see it. Somewhere inside it is a secret room. Your job is to write a client program that navigates through the maze to find that room, one move at a time.

How it works:

When your client connects, the server initializes a maze and places a player at a random starting room and sends you the doors available from that room (some combination of North, South, East, West).

Each turn follows this pattern:

- Server sends client: which doors are open, and if the room was already visited or not.
- You send back: the direction you want to move
- Server moves the player and sends the updated door list and visited boolean for the new room
- Repeat until you find the secret room

When you reach the secret room, the server sends a **FOUND** message, which you have to **ACKNOWLEDGE** and the session ends. That is your passing condition.

What you must implement:

All socket and connection code is provided. You only need to modify **client.c**:

1. Deserialize the server's message.
2. Implement a **recursive** traversal strategy to decide which direction to move next.
3. Serialize your move and send it back.
4. Loop until **FOUND** is received.

Note: You only need to implement a recursive function in order to solve this problem.

For your reference, 5/10 maze structures are given as public test cases. **Client code should not open the test case files.**

In testcases/input0X.txt:

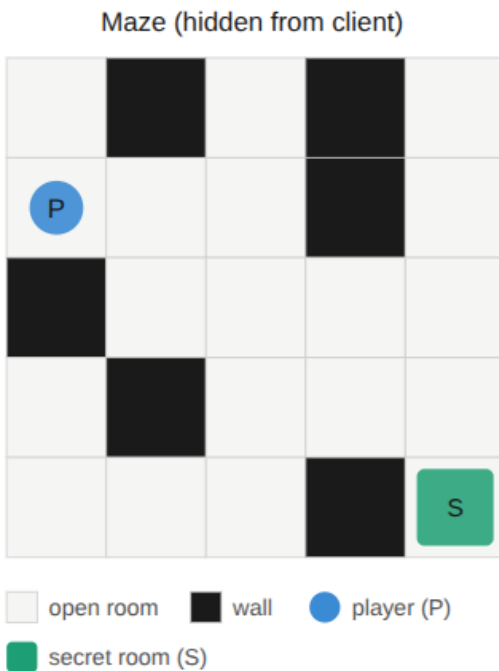
<M N>: Number of rows and columns, followed by the maze

. -> Valid room

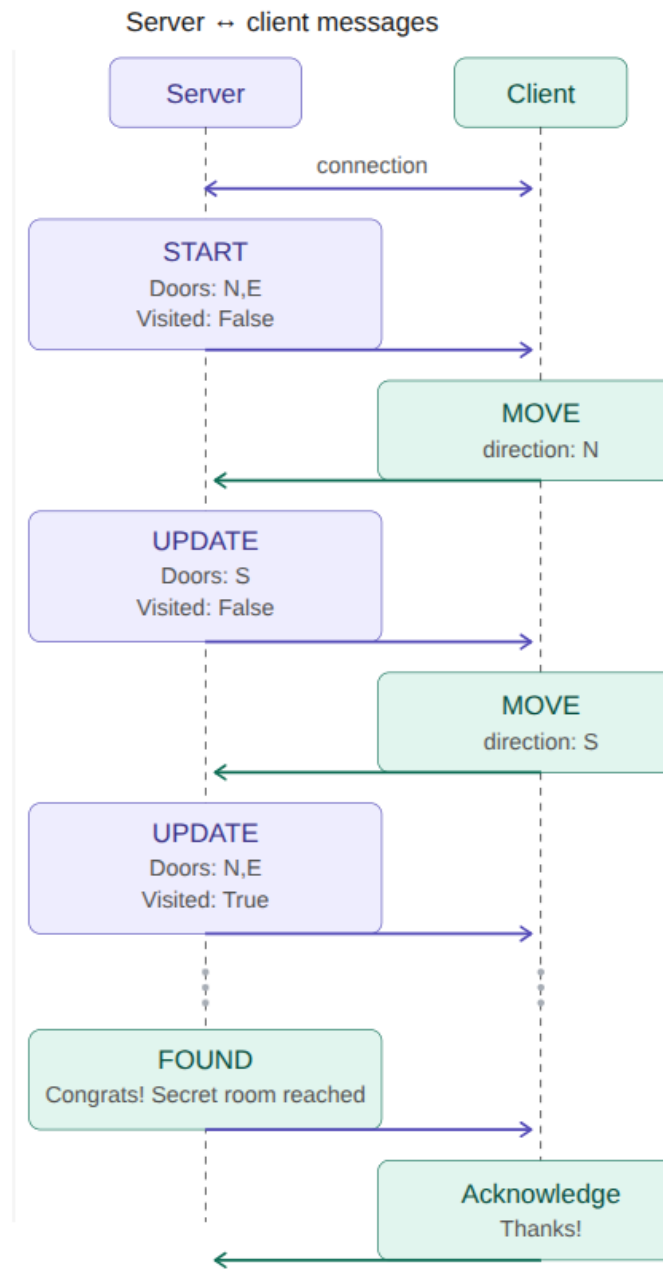
X -> Wall

P -> Player Start

S -> Secret Room



Visited: True means you've been there
Client sees only what server sends.



Communication Format:

Each message begins with a **message type**, followed by fields separated by spaces.

- **Server → Client**

<doors>: a subset of {N, S, E, W}, comma-separated, listing directions(fixed order) with open doors. Example: N,E or S or N,S,E,W

<visited>: **True** if this room was previously entered, **False** otherwise

At start:

START <doors> <visited>

Each move after that:

UPDATE <doors> <visited>

On reaching the secret room:

FOUND

On receiving an invalid move:

INVALID_MOVE

<then close the connection>

- **Client → Server**

<direction> — exactly one of N, S, E, W. Must be a door listed in the last server message.

Each turn:

MOVE <direction>

On receiving FOUND:

ACKNOWLEDGE

Constraints:

- The maze is a grid. Every room has at most 4 neighbours. A door exists only if the neighbour is an open room (not a wall).
- The maze is guaranteed to be **solvable**, secret room is reachable from the start.
- Sending a direction not listed in `<doors>` or sending MOVE when the secret room is found etc. are **invalid moves**.
- On receiving an **invalid move**, the server will close the connection and the test case will fail.
- At start, the player is guaranteed to not be in the secret room.
- $2 \leq \text{Number of rooms in maze (rows*columns)} \leq 1e5$
 $1 \leq \text{rows} \leq 1e5$
 $1 \leq \text{columns} \leq 1e5$.

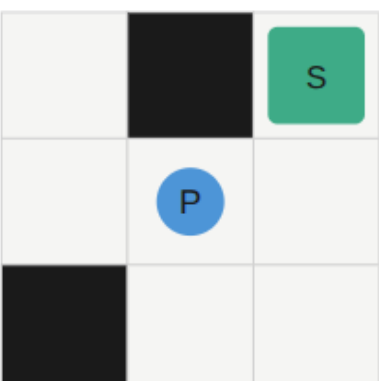
Example Flow:

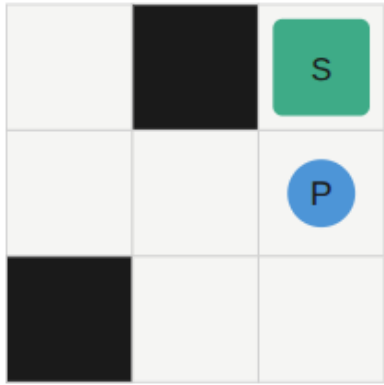
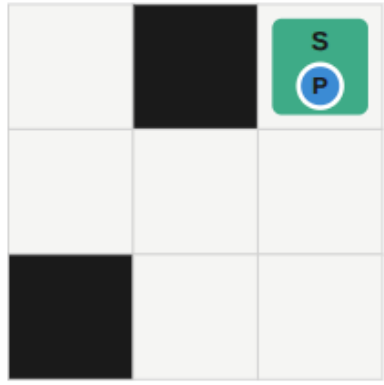
Suppose that server holds the following maze (hidden from client):



This is how a communication between the server and client might look like:

Server-side Maze	Server to Client	Client to Server
	START N,E False	

		MOVE N
	UPDATE S False	
		MOVE S
	UPDATE N,E True	
		MOVE E
	UPDATE E,S False	

		MOVE E
	UPDATE N,W,S False	
		MOVE N
	FOUND	
		ACKNOWLEDGE

Extension (ungraded)

Passcode collection

At the start, the server also sends **k**, the length of a passcode. Scattered across **k** specific rooms in the maze are passcode fragments (a number or short string), which the server delivers automatically when you step into one. Your client must collect all **k** fragments and print the full passcode in order once the secret room is found.

This means your traversal can no longer stop at the secret room, you must complete the passcode and then reach the secret room.