COMPSCI 3N03 Assignment 3

Aditya Sahni, Karan Sandhu, Ankur Pandey, Manavi Manavi

March 2025

Detailed Explanation of the Protocol

Overview

This report details the communication protocol for a networked ASCII "Battle Game" implemented in C, consisting of a server (server.c) and a client (client.c). The game supports up to 4 players in a turn-based system, where players can move, attack with shurikens, or quit the game. The server manages the game state, processes client commands, and broadcasts updates to all connected clients. The client sends commands to the server and displays the game state received from the server.

The protocol is built on TCP for reliable communication, using a simple text-based message format. Below, we describe the protocol, the recognized commands, the message flow, and provide a sequence diagram to illustrate the client-server interaction.

Connection Establishment

1. Connection Setup
    - **Transport Layer**: The protocol uses TCP for reliable, ordered message delivery.
    - **Server Initialization**:
        - The server starts by binding to a specified port (e.g., `./server 3000`) and listens for incoming connections.
        - It can accept up to 4 clients (`MAX_CLIENTS = 4`).
        - If the server is full, it sends the message **"Server full. Please try again later."** to the new client and closes the connection.
    - **Client Connection**:
        - The client connects to the server using the server's IP address and port (e.g., `./client 127.0.0.1 3000`).

  - Upon successful connection, the server assigns the client a player ID (0 to 3, corresponding to players A to D) and initializes their position on the grid (e.g., Player A at (0, 0), Player B at (1, 0), etc.).
  - The server increments its `clientCount` and logs: "**New client connected! Connected to (<hostname>, <port>). Active clients: <count>/4**".

2. Game State

   - **Grid**: A 5x5 ASCII grid (`GRID_ROWS = 5, GRID_COLS = 5`)
     - `.` represents an empty cell.
     - `#` represents an obstacle (in our case, at positions (2, 2) and (1, 3)).
     - `A, B, C, D` represent players.
     - `*` represents a shuriken.
   - **Players**:
     - Each player has a position (x, y), health points (HP, initially 100), and one shuriken.
     - Players are identified as A (index 0), B (index 1), C (index 2), and D (index 3).
   - **Turn-Based System**:
     - The game operates on a turn-based system, starting with Player A (index 0).
     - The server maintains a `currentTurn` variable to track whose turn it is.
     - After each turn, the server rotates the turn to the next active player with HP > 0 using `rotateTurn()`.

3. **Recognized Commands**:

   The client sends commands as plain text strings. The server recognizes the following commands:

   - `MOVE <DIRECTION>`:
     - Format: `"MOVE UP", "MOVE DOWN", "MOVE LEFT", "MOVE RIGHT"`.
     - Action: Moves the player one cell in the specified direction if the move is within bounds and not blocked by an obstacle (`#`).
     - Example: If Player A is at (0, 0) and sends "MOVE DOWN", they move to (1, 0) if the cell is not an obstacle.
   - `ATTACK <DIRECTION>`:
     - Format: `"ATTACK UP", "ATTACK DOWN", "ATTACK LEFT", "ATTACK RIGHT"`.

    - Action: Launches a shuriken in the specified direction from the player's position.
        * The shuriken spawns one cell away in the direction of the attack (e.g., if Player A at (0, 0) sends `"ATTACK DOWN"`, the shuriken spawns at (1, 0)).
        * If the spawn position is valid (within bounds and not an obstacle), the shuriken becomes active and moves one cell per turn in the same direction.
        * If the shuriken hits a player (i.e., occupies the same cell), it deals 50 damage and deactivates.
        * If the shuriken hits a wall (`#`) or goes out of bounds, it deactivates.
    - Example: If Player A at (0, 0) sends "ATTACK DOWN" and Player B is at (1, 0), Player B takes 50 damage (HP: 100 $\rightarrow$ 50).
- `QUIT`:
    - Format: `"QUIT"`.
    - Action: The player quits the game, their state is reset, and their socket is closed.
    - Example: If Player A sends `"QUIT"`, they are removed from the game, and other players are notified.

4. Server Responses and Messages:

   The server sends the following messages to the clients:

   - **Game State Update**:
       - Format:

```
STATE:

<grid row 1>
<grid row 2>
<grid row 3>
<grid row 4>
<grid row 5>

ACTIVE PLAYER INFO (IF EXISTS)
Player <ID>
Player position: (<x>, <y>)
Player health points <HP>
...
```

       - Sent: After every command (via `broadcastState()`), when a player connects, disconnects, or quits.
       - Example:

```
STATE:

A....
B...#
..#..
.....
.....

ACTIVE PLAYER INFO (IF EXISTS)
Player 0
Player position: (0, 0)
Player health points 100
Player 1
Player position: (1, 0)
Player health points 100
```

- **Turn Notifications**:
  - To the current player: "**It's your turn, Player X**" (e.g., "**It's your turn, Player A**").
  - To other players: "**It's Player X's turn**" (e.g. "**It's Player A's turn**").
  - Sent: After every turn rotation (via `rotateTurn()`).
- **Not Your Turn**:
  - Message: "**Sorry, it's not your turn**".
  - Sent: If a player sends a command when it's not their turn.
- **Shuriken Hit**:
  - The server logs: "**Player X hit by shuriken! HP reduced to Y**", but this is not sent to clients.
  - The updated HP is reflected in the game state broadcast.
- **Player Defeated**:
  - To the defeated player: "**You have died!**".
  - Sent: When a player's HP drops to 0 or below (via `checkShurikenCollision()`).
  - The server logs: "**Player X has been defeated!**".
- **Player Quit**:
  - To the quitting player: "**You have quit the game.**".
  - To other players: "**Player X has quit the game.**".
  - Sent: When a player sends "`QUIT`".
- **Player Disconnected**:
  - To other players: "**Player X has disconnected.**".
  - Sent: When a player disconnects unexpectedly (e.g., closes their client).

5. Message Flow

   The message flow follows a turn-based pattern:

   (a) Client Connects:
      - Client connects to the server.
      - Server assigns a player ID, initializes the player's state, and broadcasts the updated game state to all clients.
      - If the client is the first player (Player A), the server sends: "**It's your turn, Player A**".

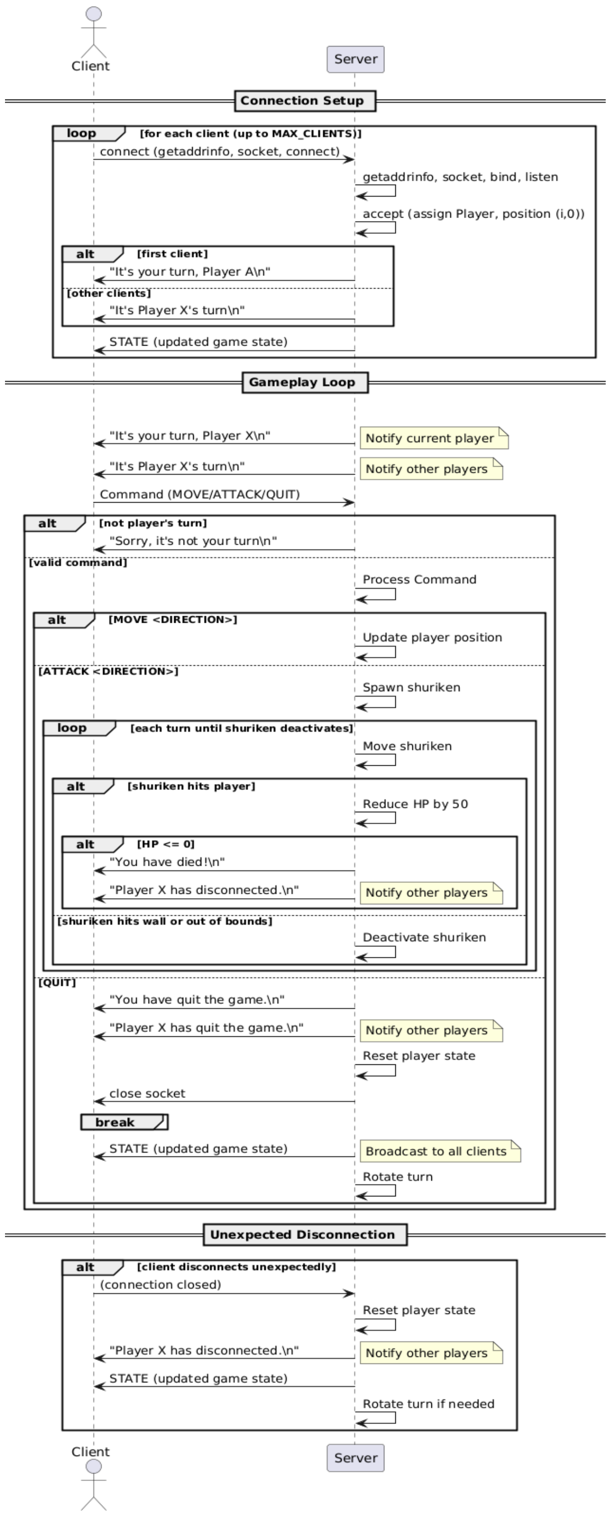
   (b) Turn-Based Command Processing:
      - The client whose turn it is sends a command (e.g., `"MOVE DOWN"`, `"ATTACK RIGHT"`, `"QUIT"`).
      - If it's not the client's turn, the server responds with "**Sorry, it's not your turn**".
      - Otherwise, the server processes the command:
        – Updates the game state (e.g., moves the player, launches a shuriken, or removes the player if they quit).
        – Broadcasts the updated game state to all clients.
        – Rotates the turn and sends turn notifications.

   (c) Shuriken Movement:
      - At the start of each turn, the server moves all active shurikens and checks for collisions.
      - If a shuriken hits a player, their HP is reduced, and the updated state is broadcast.

   (d) Disconnection:
      - If a client sends "`QUIT`" or disconnects unexpectedly, the server resets their state, notifies other players, and broadcasts the updated state.

Sequence Diagram

Below is a sequence diagram illustrating the communication between two clients (Client A and Client B) and the server during a game session. The diagram includes connection, command processing, shuriken hits, and disconnection.



Client
Server
Connection Setup
loop [for each client (up to MAX_CLIENTS)]
connect (getaddrinfo, socket, connect)
getaddrinfo, socket, bind, listen
accept (assign Player, position (i,0))
alt [first client]
"It's your turn, Player A\n"
[other clients]
"It's Player X's turn\n"
STATE (updated game state)
Gameplay Loop
"It's your turn, Player X\n"
Notify current player
"It's Player X's turn\n"
Notify other players
Command (MOVE/ATTACK/QUIT)
alt [not player's turn]
"Sorry, it's not your turn\n"
[valid command]
Process Command
alt [MOVE <DIRECTION>]
Update player position
[ATTACK <DIRECTION>]
Spawn shuriken
loop [each turn until shuriken deactivates]
Move shuriken
alt [shuriken hits player]
Reduce HP by 50
alt [HP <= 0]
"You have died!\n"
"Player X has disconnected.\n"
Notify other players
[shuriken hits wall or out of bounds]
Deactivate shuriken
[QUIT]
"You have quit the game.\n"
"Player X has quit the game.\n"
Notify other players
Reset player state
close socket
break
STATE (updated game state)
Broadcast to all clients
Rotate turn
Unexpected Disconnection
alt [client disconnects unexpectedly]
(connection closed)
Reset player state
"Player X has disconnected.\n"
Notify other players
STATE (updated game state)
Rotate turn if needed
Client
Server

Summary

The networked ASCII Battle Game protocol defines a set of commands and server messages to facilitate turn-based gameplay. Players can issue commands such as `MOVE <DIRECTION>` to move their character in the specified direction (`UP, DOWN, LEFT, RIGHT`), `ATTACK <DIRECTION>` to launch a shuriken in the chosen direction (`UP, DOWN, LEFT, RIGHT`), and `QUIT` to exit the game, which removes the player from the session.

The server communicates with clients through various messages, including the game state, formatted as "STATE: \n <grid> \n ACTIVE PLAYER INFO (IF EXIST) \n <player info>", which provides the current grid and player details.

Turn notifications are sent to inform the current player with "**It's your turn, Player**" and other players with "**It's Player X's turn**", while an error message, "**Sorry, it's not your turn**", is sent if a player attempts an action out of turn.

When a player's HP drops to zero, the server sends "**You have die**" to the affected player. For the `QUIT` command, the quitting player receives "**You have quit the game**", and other players are notified with "**Player X has quit the game**". If a player disconnects unexpectedly, the server broadcasts "**Player X has disconnected**" to the remaining players.

The message flow of the protocol follows a structured sequence to ensure smooth gameplay. When a client connects, the server assigns a player ID and broadcasts the updated game state to all clients. The game then enters a turn-based loop where the server notifies the current player of their turn, prompting them to send a command, which the server processes before broadcasting the updated state and rotating the turn to the next player. At the start of each turn, the server handles shuriken movement and collision checks, updating the game state accordingly. If a client disconnects, either by sending `QUIT` or unexpectedly, the server resets the player's state, broadcasts the updated state to all remaining clients, and continues the game. This flow ensures a fair and consistent multiplayer experience.

Additional Features

Here are some additional features we've introduced. We won't go into detail here, as we've already explained them previously.

- **Shuriken**: Added a shuriken mechanic, allowing players to attack in a specified direction, dealing 50 damage on hit.
- **Turn-by-Turn Gameplay**: Implemented a turn-based system, ensuring fair play by rotating turns among active players.

- **Proper QUIT Mechanics**: Enhanced the QUIT command to notify all players, reset the quitter's state, and close their socket cleanly.
- **Border Adherence**: Fixed movement to prevent players from going out of the 5x5 grid, ensuring valid moves within boundaries.