Introduction to Network Programming HW2 Report

112652033 賴柏瑋

1. System structure:
    a. MessageFormat: a message format class used to turn arguments into json format and parse it back to list of arguments.
    b. MessageFormatPasser: a class used to send json format messages by length-prefixed framing protocol.
    c. Protocols: a class defining all message formats.
    d. Words: a class defining strings used in keys in json.
    e. LobbyServer: the lobby server class. It accepts connections from users and database server. When user sends requests about data, the thread of server processing the connection with that client generates a corresponding uuid4 request id and send this id together with the request to database server and append this request into a dictionary. After receiver thread gets the response for that id, receiver then adds response into dictionary, then the sender thread gets the response. If player wants to start game, lobby server creates a GameServer object and sends port to players.
    f. DatabaseServer: the database server class. Connects to lobby server and modifies user_db.json, the json file saving status of each player, and room_db.json, the json file saving all room status.
    g. Client: the client class. It connects to lobby server and interacts with it. After starting game, client connects to the game server with the host same as the lobby server and the port number sent by lobby server. After that, client start running game window and starts a thread listening the messages from game server.
    h. GameServer: the game server class listening actions from players and sending snapshots of the game. For avoiding blocks caused by sending messages due to internet lags, one thread of game server puts the

snapshot message into each queue of players and spectators having maximum size 100 (if full, pop then put) and other threads send messages to players and spectators independently (one player one thread, one spectator one thread).

2. Protocol formats:
    a. clients connecting to server will first send { “connection_type”: “client”}. Database server will send {“connection_type”: “database_server”}, then Lobby will send back {“result”: “confirmed” or “error”, “message”: <additional information>} To send command, client sends {"command": str, "params": dict}. For response and events, server sends {"message_type": str, "responding_command": str, "event_type": str, "result": str, "data": dict}. If it is a response, message_type will be “response”. If it is an event, message_type will be “event”. “event_type” can be user joined room or game started, etc. Result is used in responses. Data for additional informations.
    b. When lobby server wants to send requests to database server, it sends {"request_id": str, "collection": str, "action": str, "data": dict} where request_id: unique identifier for the request, collection: e.g., 'user', 'room', action: e.g., 'create', 'read', 'update', 'delete', 'query', data: additional data as a dictionary. For response from database server, it sends {"responding_request_id": str, "result": str, "data": dict} where responding_request_id: the request_id this response is for, result: 'success' or 'failure', data: additional data as a dictionary.
    c. When client connects to game server, it sends {"username": str, "room_id": str, "role": str} where username: player or spectator's username, room_id: ID of the game room, role: "player" or "spectator", then gameserver will response {"result": str, "role": str, "seed": int, "bagRule": str, "gravityPlan": dict} where result: 'success' or 'failure', role: 'player1' or 'player2' or ‘spectator’, seed: random seed for the game session, bagRule: rule for piece bag generation, gravityPlan: plan for gravity
    d. When game starts, game server sends {"result": str, "message": str, "player1_username": str, "player2_username": str, "player_health": int,

"now_piece": str, "next_pieces": list, "goal_score": int} where result: 'success' or 'failure', message: additional information, player1_username: username of player 1, player2_username: username of player 2, player_health: initial health of each player, now_piece: initial piece on the game field, such as "I", next_pieces: initial next pieces, such as ["J", "L", "O"] \ngoal_score: score needed to win.

e. For sending screenshots, game server sends {"player1": dict, "player2": dict, "data": dict} where

player1: game state update for player 1 \n
player2: game state update for player 2
the dictionary mainly contains:
'board': string representing the game board, contains width * height chars
'now_piece': current piece shape (list)
'color': current piece color
'position': current piece position
'next_pieces': list of next piece types (list[str])
'score': current score
'health': current health
'revive_time': current revive time remaining

data: additional data as a dictionary, such as game over info

f. For sending actions to game server, player sends {"action": str, "data": dict} where action: e.g., 'move_left', 'rotate', 'drop', 'ready', etc. data: additional data as a dictionary

3. Synchronization Strategies:
   a. Input actions: For example, when player press the left arrow, player sends {"action": “move_left”, "data": {}}; when player press c, player sends {"action": “change_color”, "data": {“color”: 3}}.
   b. Snapshot frequency: 0.1s per snapshot.
   c. Synchronization: each thread send messages of current game status to the client it is processing to. The client will try receive the messages as fast as possible and put the informations to game window to render.

This ensures that each client can receive the latest game status as long as the internet is not laggy.

4. Game Rules:
    a. Players initially have 40 maximum health.
    b. If health becomes 0 or tetris board dies, player dies and have initially 15 seconds of reviving.
    c. Each death adds 5 seconds of reviving time of next death.
    d. Players can change the color of the piece of tetris they are controlling:
        i. Press z for score cell (orange), if this cell is cleared by full line, the player earns 1 point.
        ii. Press x for heal cell (green), if this cell is cleared by full line, the player heals 1 health.
        iii. Press c for attack cell (red), if this cell is cleared by full line, opponent lose 1 health.
    e. First player reaching 50 points wins.