FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Seminar Work

Fundamentals of Computer Networks. Multiplayer game "Reversi"

Volodymyr Pavlov

PILSEN, CZECH REPUBLIC

2024

FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Seminar Work

Fundamentals of Computer Networks. Multiplayer game "Reversi"

Volodymyr Pavlov

PILSEN, CZECH REPUBLIC 2024

Citation in the bibliography/reference list:
PAVLOV, Volodymyr. *Fundamentals of Computer Networks. Multiplayer game "Reversi"*. Pilsen, Czech Republic, 2024. Seminar Work. University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering.

Contents

Task 1

Create a multiplayer game 'Reversi'. Solution should contain a server (written in C) with TCP scoket, which can process many players and at the same time can save game state. Client application (written in Kotlin), should support client login. After a login, client should continue with a last known state.

Implemented solution should be stable and be able to handle unexpected behavior (wrong input, client/server lost connection).

Game rules 2

Player color (white and black) is distributed randomly between two players. White player begins. Each turn, the player places one piece on the board with their color facing up.

Game starts with placing 4 pieces (2 of each color) in the middle of the board, so called *initial board state*, which can be seen in the Figure 2.1.

Each piece played must be laid adjacent to an opponent's piece so that the opponent's piece or a row of opponent's pieces is flanked by the new piece and another piece of the player's color. All of the opponent's pieces between these two pieces are 'captured' and turned over to match the player's color.

It can happen that a piece is played so that pieces or rows of pieces in more than one direction are trapped between the new piece played and other pieces of the same color. In this case, all the pieces in all viable directions are turned over.

The game is over when neither player has a legal move (i.e. a move that captures at least one opposing piece) or when the board is full.

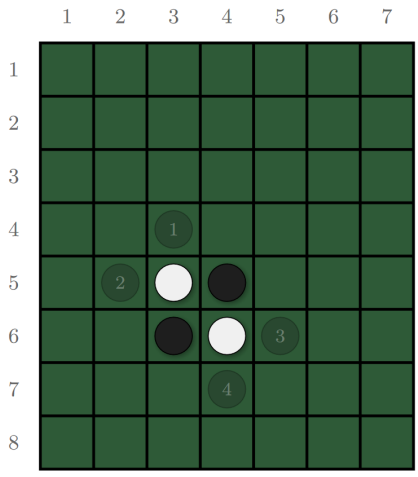

Figure 2.1: Possible game initial state.

Task Analysis 3

Due to a task restriction of using any existing data-interchange format (like *JavaScript Object Notation (JSON)* or *Extensible Markup Language (XML)*), a proprietary one should be created to be used in communication between server and client. Flexible data-interchange format should improve extensibility of requests and responses. The most challenging part will be to ensure multi-thread clients handing on server side, and proper error/unexpected behavior handing on both sides.

3.1 Key Components for Implementation

1. **Game engine** - this component will be response for all game-related processes.
    - **Game board** - create a data structure to store current game state.
    - **Move validation** - check if player move is correct and can be processed.
    - **Move processing** - process a correct move within all possible directions.
2. **Communication protocol**
    - **Representational State Transfer (REST)** - REST-like connection protocol will be used for communication between client[1] and server. It means, that whole communication will work in the way one client request will have one server response. Server will not send any non-client requested data.
    - **Message header** - Each message should have a header of a predefined length. The header should contain an identifier (to be sure that the socket is used by correct client), and a payload length (can be zero in case of empty payload).

[1] From now the *client* will be used in the meaning of a client application, connected to the server.

- **Message payload** - The payload is not mandatory. Payload should have a predefined structure to ensure correct composition, parsing and to make it easily extendable with additional values.
- **HTTP-like codes** - Each header should contain HTTP-like status code for easy flow and error handling.
- **Operation codes** - Each header should contain so-called *operation code* to differentiate message types handling.

3. **Server components**

   - **Clients handling** - each client will have its own thread on server side, which will process every client request.
   - **Operations synchronization** - every data-related action should be synchronized to avoid concurrent resource access.
   - **Client timeouts** - if a client has not performed a request for the last **N** minutes (N is configurable), then the client should be disconnected and related socket should be closed.
   - **Invalid client operations** - server should correctly react on any client's invalid operation (malformed request, action which is not possible in the current client state).
   - **Client flow state** - each client can be in a one of the flow state: [*LOGIN, MENU, LOBBY, GAME*] [2]. The client flow state should be synchronized between server and client, and the server value will be the only source of truth [3].

4. **Client components** - Entire client application will be implemented using Kotlin multi-platform (KMP) for backend and frontend.

   - **User Interface (UI)** - Easy and user-friendly UI will be implemented using Compose multi-platform (CMP).
   - **State synchronization** - Client state should be server-driven and synchronized to avoid inconsistent client state.
   - **Continue on client login** - After login, client should synchronize state with server and continue from a state, where client was.

[2] Described more detailed in the [section ref]

[3] It means, that a client-side flow state for some reason does not match server-side value, then it should be adjusted on the client side.

Custom data-interchange format

4

For a data-interchange format for payload a string with semicolon delimiters was chosen. Supported data types can be found in the Table 4.1. A payload then can be composed into a string, which then can be transmitted via TCP socket. Also, the payload can be easily parsed from a string into objects format using regex expressions.

Data Type	**Syntax**	**Example**
Integer	key={int_value}	key=123
String	key="{string_value}"	key="123"
Boolean	key={bool_value}	key=true
Integer Array	key= [{int_value0},{int_value1}]	key=[1,2,3,4]
String Array	key= ["{string_value0}","{string_value1}"]	key=[“str1”, “str2”]
No Value	key=null	key=null

Table 4.1: Payload supported data types.

Communication Protocol

5

Communication protocol is inspired by REST style and Hypertext Transfer Protocol (HTTP). Any information transmitting is driven by a client (client sends a request to a server and waits for a response).

Each message (request or response) contains a header, which has predefined constant length, and a payload, which contains useful information (may be empty for some cases).

5.1 Header

Message *header* has length of 17 bytes and contains 5 sections described in Table 5.1. Header has always constant structure: `[Identificator,Length,Type,Subtype,Status]`.

Section	Length (bytes)	Data Type	Description
Identificator	4	String	Unique identificator for client-server connection
Length	7	Integer	Length of the message payload (max is 9999999)
Type	1	Integer	Message Type
Subtype	2	Integer	Message Subtype
Status	3	Integer	Response status

Table 5.1: Header sections

5.2 Payload

Message *payload* is optional. It uses Custom data-interchange format for data transmitting. *Payload* can have variable length up to 9999999 bytes. The message *payload* length is defined in the message *header*, in case of empty payload zero value in the *header* is set.

Client Flow 6

Each client has its own flow state, which is synchronized between server and client. It's used to show correct UI screen to a player, to perform correct ping/get-state requests from a client, restrict or permit some user actions and to make a client able to continue from a state it was after reconnection (in case if the client is stored in server's cache). A client can be in four states: **No State**, **Menu**, **Login** and **Game**. Detailed description of client's states can be found in Table 6.1

No State	If a client has no state, it means that login was not performed and a client state could not be fetched from a server. In this case a user is asked to login.
Menu	This state means, that the client is logged in, but has no lobby or game assigned. In this case open lobbies list is displayed to a user. Player can create a new lobby, or connect to am existing one.
Lobby	In this state client is assigned to a lobby, but game was not started. Lobby screen is displayed to a user. Player can leave the lobby, or start the game in case if he/she is a lobby host and there are enough players.
Game	In this state game is already started and client is participating it. User is allowed to make game moves or leave the game. If one of the game players leaves, the game is terminated for all players.

Table 6.1: Client flow states.

Some player actions (e.g. creating a new lobby or leaving current lobby) can affect current client flow state, which will lead to UI changes. If the client state is changed on server side (e.g. game is terminated), then client-side flow state value is adjusted to ensure consistency with a server. Entire client flow overview can be seen in Figure 6.1.

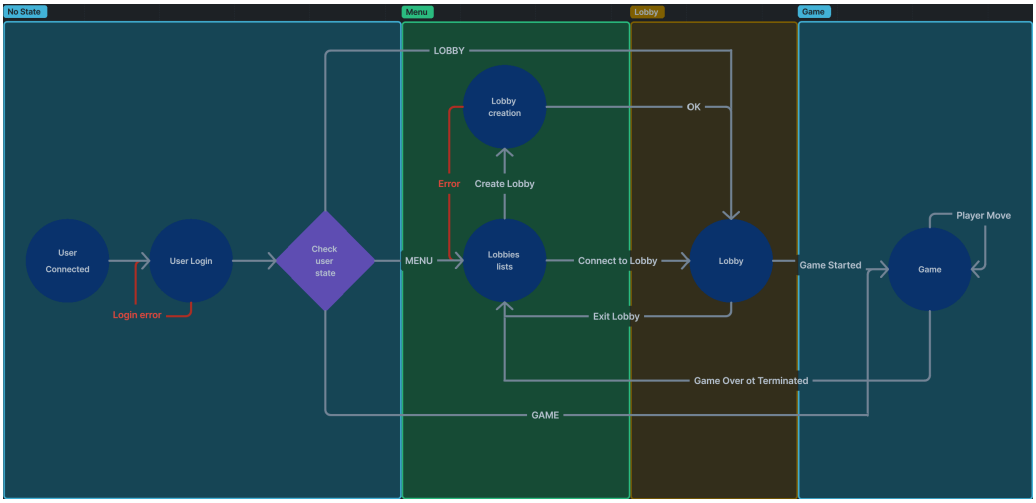

Figure 6.1: Client flow overview.

Requests and Responses

7

In this chapter used client requests and server responses will be described. On some request types there can be more than one response type, HTTP-like response codes are used to differentiate response processing.

Each request has its own type (see Table 7.1) and subtype (see Table 7.2), which indicates how it should be processed on server. They are a mandatory part os a message header.

Type	Code	Description
GET	1	Is used to retrieve information from a server.
POST	2	Is used to change/post some information on a server.

Table 7.1: Request types.

Subtype	Code	Description
PING	01	Is used to ping a server.
LOGIN	02	Is used to perform client login.
CREATE_GAME	03	Is used to perform creating a new game.
LOBBY_EXIT	04	Is used to exit a lobby or game.
LOGOUT	05	Is used to perform client logout.
START_GAME	06	Is used to start a game.
LOBBIES_LIST	07	Is used to get open lobbies list.
LOBBY_CNCT	08	Is used to perform client connection to a lobby.
GAME_STATE	09	Is used to retrieve current game state.
GAME_MOVE	10	Is used to perform a player game move.
LOBBY_STATE	11	Is used to retrieve current lobby state.
HANDSHAKE	12	Is used to perform a client-server handshake.

Table 7.2: Request subtypes.

7.1 Common

These requests and responses are used when a client is not logged in or for performing actions with its login.

7.1.1 Ping

A simple ping request-response which is used to check if the server is online. Client request and server response structures can be seen in Table 7.3.

Section	Value	Description
Type	1	GET
Subtype	01	PING

Request header.

Request payload is empty.

Section	Value	Description
Type	1	GET
Subtype	01	PING
Status	200	OK

Response header.

Response payload is empty.

Table 7.3: Ping request/response.

7.1.2 Handshake

It is used to verify new client connection. If handshake was not performed, any other requests from client are aborted. Client request and server response structures can be seen in Table 7.4.

Section	Value	Description
Type	2	POST
Subtype	12	HANDSHAKE

Request header.

Request payload is empty.

Section	Value	Description
Type	2	POST
Subtype	12	HANDSHAKE
Status	200	OK

Response header.

Response payload is empty.

Table 7.4: Handshake request/response.

7.1.3 Login

This request i subtype is used to perform client login (see Table 7.5). If login is accepted by server, OK response is returned (see Table 7.6). Otherwise, if username is already in use, CONFLICT response is sent (see Table 7.7).

Section	Value	Description
Type	2	POST
Subtype	02	LOGIN

Request header.

Field name	Type	Description
username	string	Username for login.

Request payload.

Table 7.5: Login client request.

Section	Value	Description
Type	2	POST
Subtype	02	LOGIN
Status	200	OK

Response header.

Field name	Type	Description
state	string	Client flow state.

Response payload.

Table 7.6: Login server response - OK.

Section	Value	Description
Type	2	POST
Subtype	02	LOGIN
Status	409	CONFLICT

Response header.

Field name	Type	Description
msg	string	Error message.
state	string	Client flow state.

Response payload.

Table 7.7: Login server response - NOK - username declined.

7.1.4 Logout

This request i subtype is used to perform client logout. If client is not logged in, no actions are performed and OK response is returned. Client request and server response can be seen in Table 7.8

Section	Value	Description
Type	2	POST
Subtype	05	LOGOUT

Request header.

Request payload is empty.

Section	Value	Description
Type	2	POST
Subtype	05	LOGOUT
Status	200	OK

Response header.

Response payload is empty.

Table 7.8: Logout request/response.

7.2 Menu and Lobby

These subtypes of requests are used to retrieve information about current lobbies state and to interact with a lobby.

7.2.1 Retrieve Open Lobbies

This request is used to get open lobbies, to which a client can connect (see Table 7.9 and Table 7.10).

Section	Value	Description
Type	1	GET
Subtype	07	LOBBIES_LIST

Request header.

Request payload is empty.

Table 7.9: Get lobbies client request.

Section	Value	Description
Type	1	GET
Subtype	07	LOBBIES_LIST
Status	200	OK

Response header.

Field name	Type	Description
state	string	Client flow state for synchronization.
lobbies	string array	A list of open lobbies names.
lobby_hosts	string array	A list of open lobbies host usernames.

Response payload.

Table 7.10: Get lobbies server response.

7.2.2 Create Lobby

This request is used to process new lobby creation, triggered by a user. Client sends a request with a lobby name (see Table 7.11). Server respond with OK if a lobby is created successfully (see Table 7.12), and with status CONFLICT if a lobby with requested name is already exists (see Table 7.13).

Section	Value	Description
Type	2	POST
Subtype	03	CREATE_GAME

Request header.

Field name	Type	Description
name	string	Lobby name.

Request payload.

Table 7.11: Create lobby client request.

Section	Value	Description
Type	2	POST
Subtype	03	CREATE_GAME
Status	200	OK

Response header.

Field name	Type	Description
state	string	Client flow state for synchronization.
name	string	Lobby name.
user	string	Client name.

Response payload.

Table 7.12: Create lobby server response - OK.

Section	Value	Description
Type	2	POST
Subtype	03	CREATE_GAME
Status	409	CONFLICT

Response header.

Field name	Type	Description
msg	string	Error message.
state	string	Client flow state for synchronization.

Response payload.

Table 7.13: Create lobby server response - CONFLICT.

7.2.3 Connect to Lobby

This request subtype is used to perform client connection to an existing open lobby (see Table 7.14). If connection was successful, OK response is returned (see Table 7.15), otherwise, if connection was not performed, NOT FOUND response is returned (see Table 7.16).

Section	Value	Description
Type	2	POST
Subtype	08	LOBBY_CNCT

Request header.

Field name	Type	Description
lobby	string	Lobby name.

Request payload.

Table 7.14: Connect to lobby client request.

Section	Value	Description
Type	2	POST
Subtype	08	LOBBY_CNCT
Status	200	OK

Response header.

Field name	Type	Description
state	string	Client flow state for synchronization.
lobby	string	Lobby name.
host	string	Host name.
players	string array	Players names list.

Response payload.

Table 7.15: Connect to lobby server response - OK.

Section	Value	Description
Type	2	POST
Subtype	08	LOBBY_CNCT
Status	404	NOT FOUND

Response header.

Field name	Type	Description
msg	string	Error message.
state	string	Client flow state for synchronization.
lobby	string	Lobby name.

Response payload.

Table 7.16: Connect to lobby server response - NOT FOUND.

7.2.4 Get Lobby State

This request is used to get current state of the lobby, in which client is (see Table 7.17). If state is retrieved successfully, then OK response is returned (see Table 7.18). In case when the game is started, REDIRECT response is used (see Table 7.19).

Section	Value	Description
Type	1	GET
Subtype	11	LOBBY_STATE

Request header.

Request payload is empty.

Table 7.17: Get lobby state client request.

Section	Value	Description
Type	1	GET
Subtype	11	LOBBY_STATE
Status	200	OK

Response header.

Field name	Type	Description
state	string	Client flow state for synchronization.
lobby	string	Lobby name.
host	string	Host name.
players	string array	Players names list.

Response payload.

Table 7.18: Get lobby state server response - OK.

Section	Value	Description
Type	1	GET
Subtype	11	LOBBY_STATE
Status	301	MOVED

Response header.

Field name	Type	Description
state	string	Client flow state for synchronization.
lobby	string	Lobby name.
host	string	Host name.
players	string array	Players names list.

Response payload.

Table 7.19: Get lobby state server response - REDIRECT.

List of Abbreviations

UI User Interface

REST Representational State Transfer

KMP Kotlin multi-platform

CMP Compose multi-platform

JSON JavaScript Object Notation

XML Extensible Markup Language

HTTP Hypertext Transfer Protocol

List of Figures

List of Tables

List of Listings