Reversi Game

Lungu Fabian, 2B3

Facultatea de Informatică, Universitatea Alexandru Ioan Cuza Iași

1 Introducere

Viziunea acestui proiect este să creeze o aplicație care să le permită mai multor jucători să joace Reversi, utilizând o arhitectură client-server. Fiecare jucător va fi identificat printr-un nume unic, iar aplicația va permite conectarea și jocul în perechi, astfel încât doi jucători să se conecteze în timpul unui meci.

Obiective

- Crearea partidelor de joc între doi jucători diferiți.
- Autentificarea și înregistrarea jucătorilor.
- Implementarea clasamentului.
- Asigurarea mai multor partide de joc simultan.

2 Tehnologii Aplicate

Limbajul C++: Este un limbaj rapid și eficient, perfect pentru a crea jocuri și aplicații care necesită performanță bună. Este folosit pentru a implementa logica jocului, gestionarea utilizatorilor și comunicarea între server și client.

Protocolul TCP: Este un protocol de comunicare care se asigură că mesajele trimise între server și client ajung corect, fără a pierde informație. Este important pentru jocuri, deoarece ajută la transmiterea comenzilor și mișcărilor fără erori.

Multi-Threading: Permite rularea mai multor procese în paralel, astfel încât serverul să poată gestiona mai mulți clienți în același timp. Fiecare client va fi tratat de un thread separat, ceea ce îmbunătățește performanța și permite serverului să răspundă rapid la cerințele mai multor jucători simultan.

SQLite: Este o bază de date ușor de folosit, care stochează informațiile. În acest proiect, va fi folosită pentru a salva datele despre jucători, scoruri și un flag special pentru a cunoaște dacă un jucător este deja conectat.

3 Structura Aplicației

3.1 Diagrama

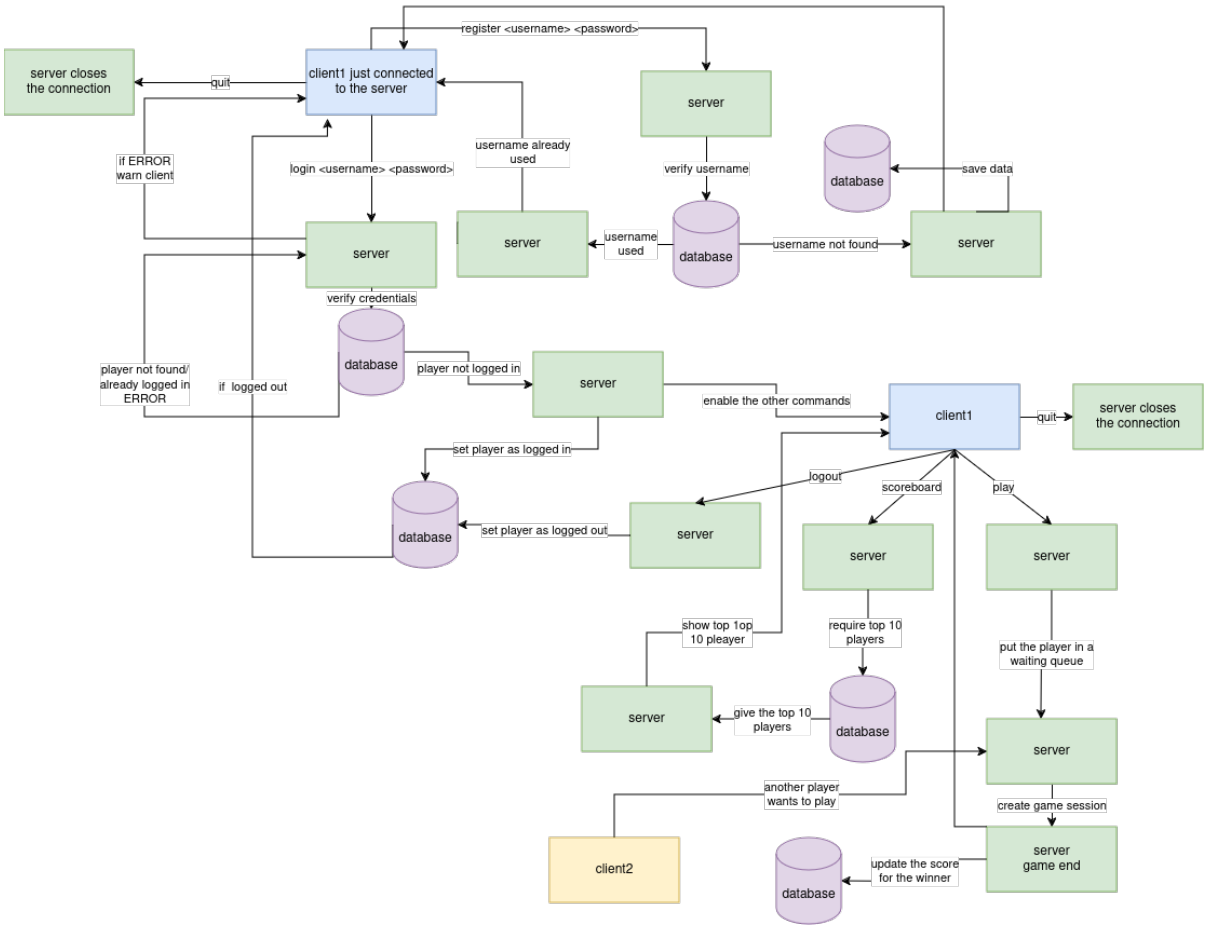

Fig. 1. Diagrama aplicației

3.2 Funcționalități

Aplicația funcționează pe baza unui model client-server, unde fiecare client poate interacționa cu serverul pentru diverse comenzi.

- Conectare: Clienții se conectează la server folosind socket-uri TCP. Fiecare client este gestionat de un thread dedicat pentru a permite mai multe conexiuni simultane.
- Autentificare și înregistrare: Clienții trimit comenzi precum login sau register. Serverul verifică credențialele sau creează conturi noi în baza de date SQLite.
- Logout: Clienții pot folosi comanda logout pentru a încheia sesiunea curentă, fără a închide complet conexiunea la server. Serverul marchează utilizatorul ca deconectat, iar acesta poate relua autentificarea dacă dorește.
- Inițierea unui joc: Când doi jucători sunt disponibili, serverul îi conectează pentru o partidă de Reversi. Fiecare mutare este validată de server, care gestionează starea jocului și o trimite ambilor jucători.
- Mutările în timpul unui joc sunt gestionate prin comanda move <linie> <coloană>, iar dacă nu este rândul unui jucător sau acesta nu se află într-un joc activ va fi anunțat de acest lucru.
- În timpul jocului un jucător poate alege să renunțe la acel joc prin comanda surrender , astfel adversarul său devenind câștigător , iar partida încheindu-se acolo.În cazul în care conexiunea se întrerupe cu unul dintre jucători același mecanism este aplicat, adversarul devenind câștigător.
- Clasament: După fiecare joc, serverul actualizează scorurile, salvând aceste date în baza de date SQLite. Jucătorii pot consulta clasamentul printr-o comandă dedicată.
- Deconectare: Clienții pot folosi comanda quit pentru a închide complet conexiunea cu serverul, moment în care resursele asociate clientului sunt eliberate.

4 Aspecte de implementare

4.1 Comunicarea client-server

În această aplicație, comunicarea între clienți și server este realizată prin intermediul socket-urilor TCP, care asigură o conexiune stabilă și sigură. Serverul ascultă pe un port predefinit (în cazul de față, portul 8080) și acceptă conexiuni noi de la clienți folosind funcția accept.

Fiecare client trimite comenzi către server sub formă de mesaje text (ex. "login", "register", "play"), iar serverul procesează aceste comenzi printr-o funcție dedicată, handle_command. Serverul returnează răspunsuri către client prin funcția send, asigurând comunicarea bidirecțională.

```
server_socket = socket(AF_INET, SOCK_STREAM, 0);
if (server_socket == 0)
{
    perror("Eroare la crearea socket-ului");
    exit(EXIT_FAILURE);
}

setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR,
&optval, sizeof(optval));

server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = INADDR_ANY;
server_address.sin_port = htons(PORT);

...

*client_socket = accept(server_socket, (struct sockaddr *)
                &client_address,&client_addr_len);
if (*client_socket < 0)
{
    perror("Eroare la accept");
    free(client_socket);
    continue;
}

.....
```

```
void handle_command(int client_socket, char *command)
{
    char response[BUFFER_SIZE];
    if (strcmp(command, "login") == 0)
    {
        snprintf(response, BUFFER_SIZE,
                "Serverul a primit comanda [%s]",command);
    }else if ...
    send(client_socket, response, strlen(response), 0);
}
```

4.2 Multi-Threading

Serverul folosește multi-threading pentru a gestiona mai mulți clienți simultan. Fiecare conexiune este tratată într-un thread separat, creat prin funcția pthread_create. Aceasta permite serverului să continue să asculte și să accepte noi conexiuni în timp ce procesează cererile clienților existenți.

```
pthread_t thread_id;
if (pthread_create(&thread_id, NULL, handle_client,
                    client_socket) != 0)
{
    perror("Eroare la crearea thread-ului");
    close(*client_socket);
    free(client_socket);
}else
{
    pthread_detach(thread_id);
}
```

Fiecare thread rulează funcția handle_client, care gestionează interacțiunea cu un singur client.

1. Jucătorii care trimit comanda "play" vor fi plasați într-o coadă de așteptare.
2. Serverul va asocia primii doi jucători din coadă și va crea o sesiune de joc.
3. Sesiunea de joc va fi salvata intr-o structura de tipul Game_Info unde vor fi salvate informatii despre jocul curent.

Fiecare thread rulează funcția handle_client, care gestionează interacțiunea cu un singur client.

Jucătorii care trimit comanda "play" vor fi plasați într-o coadă de așteptare. Serverul va asocia primii doi jucători din coadă și va crea o sesiune de joc.

```
    void handle_play(Client_Info *client_info)
{
    char response[BUFFER_SIZE];
    bzero(response, BUFFER_SIZE);
```

```
    std::lock_guard<std::mutex> lock(waiting_mutex);
    if (!waiting_queue.empty())
    {
        Client_Info *player1 = waiting_queue.front();
        waiting_queue.pop();
        client_info->status = IN_GAME;
        player1->status = IN_GAME;
        create_new_game(player1, client_info);
    }
    else
    {
        waiting_queue.push(client_info);
        snprintf(response, BUFFER_SIZE, "Asteptati un adversar!\n");
        client_info->status = WAITING_FOR_PLAYER;
        send_message_to_client(client_info->socket, response);
    }
}
```

Cum funcționează jocul între două perechi de jucători:

1. Conectarea: Fiecare jucător se conectează la server și trimite o comandă "play". Serverul îi pune în coada de așteptare.
2. Crearea sesiunii: Când sunt disponibili doi jucători, serverul îi asociază și inițiază un joc nou.
3. Jocul: Serverul trimite starea tablei de joc ambilor jucători și primește mutările acestora, actualizând tabla și trimițând răspunsurile.
4. Finalizarea: La sfârșitul jocului, serverul actualizează scorurile în baza de date și informează jucătorii despre rezultat.

4.3 Comenzi

Dupa ce conexiunea intre client si server este stabilita , clientul poate trimite urmatoarele comenzi:

- register <username> <password> - Creaza un nou cont
- login <username> <password> - Autentificare
- logout - Log-out din contul curent
- play - Pregateste un joc Reversi
- stop - Opreste cautarea unui meci
- move <linie> <coloana> - Executa o mutare in joc
- surrender - Abandoneaza jocul curent
- scoreboard - Top 10 jucatori
- help - Afiseaza pe ecran toate comenzile posibile si ce fac acestea
- quit - Deconeteaza clientul de la server

Fiecare comanda este prelucrata in functia sa proprie:

```
int login_user(const char *username, const char *password)
void logout_user(const char *username)
void scoreboard(Client_Info *client_info)
void handle_move(Client_Info *client_info, char *move_str)
void handle_play(Client_Info *client_info)
```

Sau este prelucrata in cadrul functiei handle_command:

```
void handle_command(Client_Info *client_info, char *command)
```

4.4 Sesiunea de joc

Dupa ce un jucator s-a autentificat acesta poate executa comanda play pentru a putea juca o partida de Reversi.In cazul in care un alt jucator a executat aceasta comanda cei doi vor fi pusi intr-un meci, altfel el va fi pus intr-o coada de asteptare pana cand alt jucator doreste sa joace:

```
void handle_play(Client_Info *client_info)
{
    char response[BUFFER_SIZE];
    bzero(response, BUFFER_SIZE);
    std::lock_guard<std::mutex> lock(waiting_mutex);
    if (!waiting_queue.empty())
    {
        Client_Info *player1 = waiting_queue.front();
        waiting_queue.pop();
        client_info->status = IN_GAME;
        player1->status = IN_GAME;
        create_new_game(player1, client_info);
    }
    else
    {
        waiting_queue.push(client_info);
        snprintf(response, BUFFER_SIZE, "Asteptati un adversar!\n");
        client_info->status = WAITING_FOR_PLAYER;
        send_message_to_client(client_info->socket, response);
    }
}
```

In timpul partidei cei doi fac miscari pe rand pana cand nu mai sunt miscari valabile sau tabla de joc este plina.Atunci se va calcula scorul si in functie de rezultat se vor actualiza scorurile in baza de date:

```
if (black_count > white_count)
    {
        update_score(game.player1->username, 3);
        update_score(game.player2->username, 1);
    }
```

```
else if (white_count > black_count)
{
    update_score(game.player1->username, 1);
    update_score(game.player2->username, 3);
}
else
{
    update_score(game.player1->username, 2);
    update_score(game.player2->username, 2);
}
```

Ca si functionalitati speciale, daca in timpul cautarii unui adversar clientul doreste sa opreasca acest lucru el poate oferi comanda stop si va fi scos din coada de asteptare, iar daca in timpul unui meci un jucator doreste sa renunte acesta poate sa abandoneze utilizand coamnda surrender.

5 Concluzii

Aplicația permite utilizatorilor să joace Reversi în timp real, cu un sistem funcțional de autentificare, clasament și suport pentru jocuri multiple simultan.

Pentru realizarea acestei aplicații, s-au folosit următoarele tehnologii și concepte:

- C++ și threading: Pentru a implementa logica aplicației și gestionarea simultană a mai multor conexiuni client-server.
- Protocolul TCP: Pentru asigurarea unei comunicări sigure și fără pierderi de date între clienți și server.
- SQLite: Pentru stocarea informațiilor despre utilizatori, scoruri și starea de conectare.
- Arhitectură client-server: Modelul client-server a fost implementat astfel încât serverul să gestioneze partidele și să conecteze jucătorii.

5.1 Îmbunătățiri viitoare

- Adăugarea unei interfețe grafice pentru clienți în locul unei interfețe text-based.
- Implementarea unei funcționalități de chat între jucători.
- Opțiunea de antrenament împotriva unui robot cu dificultăți variate.

References

1. https://www.sqlite.org/cintro.html
2. https://en.wikipedia.org/wiki/Reversi
3. https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php
4. https://www.geeksforgeeks.org/multithreading-in-cpp/
5. https://www.techtarget.com/whatis/definition/multithreading
6. https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/
7. https://en.cppreference.com/w/cpp/standard_library