

Documentatia proiectului ReversiS

Vivdici Ina^{1[2B4]}

¹ Universitatea "Alexandru Ioan Cuza", Iasi, Romania
sendmailto:ina@gmail.com

Abstract. Scopul acestui document este de a prezenta motivatia, functionalitatile, tehnologiile utilizate si modul de interactiune cu proiectul ReversiS.

1 Introducere

1.1 Motivatie

Am ales proiectul ReversiS din motive educative si personale. Am fost curioasa cum as putea crea interactiunea dintre o baza de date si un server in C++. Este interesant sa combin cunostintele invatate la Baze de Date cu cele de la Retele de Calculatoare. Pe langa aceasta, mi-a provocat curiozitate modul in care serverul va comunica cu multipli clienti si cum va gestiona partidele de joc multiple. Complexitatea proiectului este una medie, ceea ce imi va permite sa il pot face calitativ chiar si intr-un timp relativ scurt. De asemenea, a fost important sa descopar cum as putea imbunatati comunicarea dintre un client si server (mai concret crearea unui protocol). Motivul personal a fost invatarea unui nou tip de joc care pare relativ simplu. Potential, sa ma joc Reversi folosind acest proiect.

1.2 Prezentarea succinta a functionalitatilor

Proiectul ReversiS ofera functionalitati pentru un joc de Othello, care reprezinta o varianta moderna a jocului Reversi. Un client va putea fi identificat printr-un nume si o parola. Un joc va necesita prezenta a 2 jucatori care vor efectua diverse miscari. Serverul permite rulara a mai multor partide de joc in acelasi timp, efectuand rutina explicata pentru fiecare pereche de utilizatori. Odata cu autentificarea, clientii sunt notificati de ordinea in care vor juca si de culoarea lor. Fiecare client va trimite miscarea dorita cand ii este randul. Fiecare miscare va fi verificata daca este corecta si vor fi efectuate schimbarile necesare pe tabla (inversarea unor discuri si plasarea unui disc pe tabla). Fiecare jucator este notificat daca randul lui a fost trecut (daca nu are miscari posibile). Jucatorii vor putea vizualiza starea in care se afla tabla de joc in orice moment. La finalul jocului jucatorii vor fi notificati si li se va spune care este castigatorul. Clientii pot cere clasamentul primilor n jucatori inainte si dupa joc.

2 Tehnologiile utilizate

2.1 TCP - protocolul de comunicare

Este un fapt bine cunoscut ca TCP are o viteză mai mică decât UDP, dar un nivel de securitate a transmiterii de date mai mare. Natura jocului Othello nu este una ce necesită reacție rapidă, dar analiză strategică, deci nu necesită neapărat cea mai bună viteză de transmitere a datelor (mai concret a unei mișcări). De asemenea, este crucial să obținem datele unei mișcări nedistorsionate, altfel natura jocului va fi alterată. Folosind UDP s-ar fi avut nevoie de implementat manual această verificare de date primite, dar din considerentul că viteză nu este un factor de decizie în acest joc este mai convenabil TCP. Pe lângă aceasta, nu am servicii de tip broadcast. Un joc include doar 2 clienți, iar singurul care interacționează cu ei este serverul. Informația trimisă de server este mereu una individuală, deci nu vom avea nevoie de un serviciu multicast sau broadcast.

2.2 SQLite - baza de date

Chiar dacă proiectul ales nu impune utilizarea unei baze de date am decis în loc de a folosi un simplu fișier să folosesc o bază de date. Motivatia a fost bazată pe gestionarea facilă a datelor, ușurând căutarea unui nume de utilizator, parola sau afișarea clasamentului. De asemenea, actualizarea informațiilor despre utilizatori (punctajul sau adăugarea unui nou utilizator) este mai ușoară, existând interogări anumite pentru a executa aceste operații. Am ales SQLite în loc de XML pentru că structura bazei mele de date este mai ușor de vizualizat în forma de tabel, decât arbore. De asemenea, nu aveam nevoie să transfer date dintre diferite entități pentru a avea probleme cu formatarea, adică să am nevoie de XML sau altceva de genul. În cazul meu, doar serverul va fi cel ce accesează, deci nu voi avea probleme legate de formatul datelor. Faptul că SQLite este ușor de utilizat și este rapid a fost un factor decisiv. Biblioteca folosită pentru interacțiunea dintre SQLite și C++ este sqlite3 din considerentul popularității ei și vechimii.

2.3 SDL – biblioteca pentru interfata grafica

Am decis să creez tabla de joc folosind o bibliotecă grafică pentru a fi mai ușor de vizualizat mișcările și tabla de joc în sine. Am decis să folosesc biblioteca SDL pentru a atinge acest scop. Comparativ cu nana, o bibliotecă pentru GUI, SDL este mai simplă în abordarea pe care o oferă. Ambele biblioteci au o multitudine de funcții pentru interfata grafică, dar am preferat SDL căci era mai bine documentat și avea mai multe tutoriale.

2.4 fork – entitatea care dirijeaza o partida de joc si threads – unul care deseneaza si doua care transmit clasamentul

Implementarea unui server care poate dirija mai multe jocuri simultan este o conditie necesara a proiectului. Aceasta functionalitate poate fi implementata folosind fork sau threads. Fork ar fi mai efectiv decat un thread pentru ca chiar daca unul din procesele copii ar avea vreo eroare si ar iesi, celelalte partide de joc vor continua sa decurga normal. Astfel, chiar daca thread-urile sunt mai rapide, utilizandu-le daca am avea o problema la o pereche de clienti ea i-ar afecta pe restul. In asa mod, daca se produce o eroare intr-o partida de joc toti utilizatorii vor intampina probleme, ceea ce nu ar trebui sa se intample caci partidele de joc sunt independente. Pe langa aceasta, unicul schimb de date necesar dintre server si copil sunt descriptorii deschisi si numele clientilor din pereche. Aceste date nu vor fi actualizate pe parcursul unui joc, deci nu avem nevoie de memorie comuna dintre server si entitatile care vor dirija o partida de joc. Am creat un fork la aparitia unei noi perechi de clienti si am evitat preforking-ul pentru a minimiza timpul de asteptare a unui client pana la inceperea jocului dupa autentificare. De asemenea, nu voi avea diferente considerabile a timpului de asteptare a unui client dupa un anumit numar de partide de joc care ruleaza in acelasi timp.

Am folosit threads pentru desenarea tablei de joc caci aveam nevoie ca functia care deseneaza sa aiba mereu versiunea actualizata a tablei, astfel este mai facil sa aiba acces prin intermediul unei variabile globale decat prin trimiterea constanta prin socket. Pe langa aceasta, daca cade cel ce mentine jocul (fork-ul) e logic sa pice si desenarea tablei de joc, caci nu mai avem vreun joc.

Am decis sa folosesc threads pentru afisarea clasamentului pentru ca sa nu conteze ordinea logarii clientului, ci sa poata fi serviti ambii simultan. De asemenea, threads au permis impartirea memoriei. Am mai folosit threads pentru a mentine conexiunea cu clientii intr-un singur proces.

3 Arhitectura aplicatiei

3.1 Concepte utilizate

ReversiS contine un server, un client minimal, o baza de date in SQLite si o interfata grafica ce reprezinta tabla de joc. Serverul va reprezenta punctul central al functionarii aplicatiei, de asemenea un intermediar intre client si baza de date. Serverul va fi responsabil de conectare la o pereche de utilizatori si logarea lor. Dirijarea partidei de joc dintre acesti 2 utilizatori va fi responsabilitatea copilului nou creat. El va afisa tabla de joc, dirija un joc si interactiona cu baza de date la necesitate (pentru a incrementa scorul castigatorilor si pentru a afisa clasamentul). Clasa Board va avea rolul de a gestiona operatiile legate de tabla de joc: deciderea daca o miscare este valida, daca jocul s-a sfarsit, inversarea discurilor la o miscare anumita, verificarea daca exista miscari posibile pentru un jucator si va determina care este castigatorul jocului. Clientul se va conecta la server, alege numele si parola care il vor

identifica. Serverul va trimite cererea si clientul va scrie numele. Dupa aceea va fi ceruta si parola. Daca exista asa jucator, jucatorul va intra in joc. Daca nu exista se creeaza un cont nou, iar daca parola nu este corecta clientul va afla despre asta. Dupa aceea serverul va citi numele si parola din nou ca mai sus. Clientul, de asemenea va trimite serverului fiecare miscare dorita la un moment dat in joc. Formatul corect pentru o miscare este: „A-H1-8”, clientul verificand acest input. De asemenea, clientul poate cere clasamentul primilor n jucatori de la server, care va prelua informatia din baza de date. Baza de date va avea rolul de a memora numele, parola si numarul de partide castigate de fiecare utilizator. Clasamentul jucatorilor va fi creat folosind aceste date si afisat de server. Operatiile facute asupra bazei de date aflandu-se in fisierul db-queries.h.

3.2 Schema arhitecturii aplicatiei

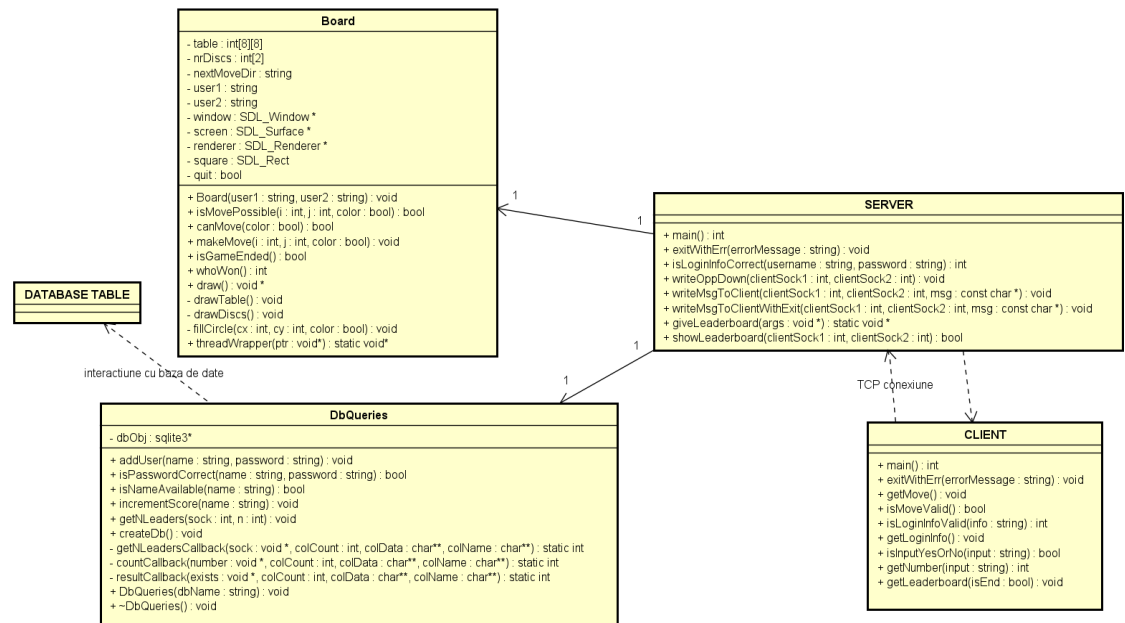


Fig. 1. Diagrama UML de clase a proiectului

4 Detalii de implementare

4.1 Logica proiectului reprezentata prin pseudocod

board.h

```

class Board
{

```

```

int table[8][8]; // 0 - black; 1 - white; 2 - free
int nrDiscs[2]; // nrDiscs[0] -> black; nrDiscs[1] -> white
string nextMoveDir; //va fi folosita de functia isPossibleMove pentru a memora
//care va fi directia de capturare a discurilor albe a miscarii identificate ca corecta.
//nextMoveDir va fi utilizat de makeMove pentru a simplifica efectuarea unei
//miscari

string user1 = ""; // the name of the first logged user
string user2 = ""; // the name of the second logged user
SDL_Window * window;
SDL_Surface * screen;
SDL_Renderer * renderer;
SDL_Rect square = { BOARD_XY, BOARD_XY, BOARD_SIZE,
BOARD_SIZE };
bool quit = false;

void drawTable();
void drawDiscs();
void fillCircle(int cx, int cy, bool color);

public:

Board(string user1, string user2);
bool isMovePossible(int i, int j, bool color);
bool canMove(bool color);
void makeMove(int i, int j, bool color);
bool isGameEnded();
int whoWon();

void * draw();
static void * threadWrapper(void * ptr);
};

```

db-queries.h

```

class DbQueries
{
    sqlite3 * dbObj;

    void createDb();
    // -- callbacks --
    // process the output of getNLeaders
    static int getNLeadersCallback(void * sock, int colCount, char** colData,
char** colName);
    // return the number of users in the database

```

```

static int countCallback(void * number, int colCount, char** colData, char**
colName);
// used for verifying if the select statement returned true or false
static int resultCallback(void * exists, int colCount, char** colData, char**
colName);

```

```

public:
DbQueries(string dbName);
~DbQueries();

void addUser(string name, string password);
bool isPasswordCorrect(string name, string password);
bool isNameAvailable(string name);
void incrementScore(string name);
void getNLeaders(int sock, int n);
};

```

client.cpp

```

bool isMoveValid(); //verificam daca primul index e din {A, ..., H}, iar al //doilea din
{1, ..., 8}, daca da, returnam true, daca nu, false
cream socketul necesar comunicarii cu serverul
conectarea la server
efectuam logarea utilizatorului
daca logarea esueaza, mai incercam sa ne logam
trimitem cerere de obtinere a clasamentului daca utilizatorul doreste
asteptam ca serverul sa ne asigneze culoarea si ordinea in joc

while(!end)
{

    if primim „skip” inseamna ca nu am avut miscari posibile, asteptam sa primim
    mesajul „move”
    else primim de la server mesajul „move” vom cere de la utilizator o miscare
    if isMoveValid() o trimitem serverului, daca nu, mai cerem de la client input inca
    odata
        if primim „succes” miscarea este posibila si acum jucatorul urmator va fi
        oponentul
        else primim „move” miscarea introdusa nu este una posibila in cadrul partidei
        curente, deci vom cere de la utilizator input inca odata
        if primim „end” stim ca jocul s-a terminat si asteptam numele castigatorului;
        end devine 1
    }
afisam clientului daca a castigat sau pierdut sau daca e egalitate
afisam clasamentul daca utilizatorul doreste

```

inchidem conexiunea cu serverului

server.cpp

pregatim socketul de listen si facem listen

```
while (1)
```

```
{
```

```
    acceptam un client
```

```
    isLoggedIn = 0
```

```
    while (!isLoggedIn)
```

```
        citim numele si parola clientului
```

```
    if (!isNameValid(username1))
```

```
        if(isPasswordCorrect (username1, password))
```

```
            trimite clientul „login”
```

```
            isLoggedIn = 1
```

```
        else
```

```
            trimite clientului „username or password wrong”
```

```
    else
```

```
        addUser(username1, password)
```

```
        trimite clientului „register”
```

```
        isLoggedIn = 1
```

mai acceptam un client si il logam (folosim username2)

daca am gasit un client nou trimitem primului si la al doilea „succes”

pastram in username1 si username2 numele clientilor logati

```
if ((pid = fork()) == 0)
```

```
{
```

```
    inchidem socketul de listen
```

daca vreun utilizator are nevoie de clasament i-l transmitem (2 threads create)

```
Board board(username1, username2);
```

```
Thread for board.drawBoard();
```

alegem o culoare pentru fiecare utilizator si notificam utilizatorii despre asta

alegem ordinea in care vor juca si notificam utilizatorii despre asta

```
end=0
```

```
while(!end)
```

```
{
```

```
    if( !board.canMove(jucatorul curent) )
```

```
        schimbam jucatorul curent si scriem clientului „skip”
```

```
    obtinem miscarea de la jucatorul curent
```

```
    if (isMovePossible (move, color))
```

```
        scriem clientului „success”
```

```
        makeMove(move, color)
```

```

    else
        cerem o noua miscare valida scriind clientului „move”
    if (board.isGameEnded())
        end = 1
        spunem ambilor jucatori „end”
    else
        schimbam urmatorul jucator
}
Trimitem fiecarui utilizator : „won” , „lost” sau „draw” in functie de caz
facem incrementScore(user) pentru castigator/castigatori
daca vreun utilizator doreste afisarea clasamentului o facem
terminam conexiunea cu ambii clienti notificandu-i
exit()
}
inchidem descriptorii pentru conexiunea cu perechea de clienti „preluata” de copil
}

```

4.2 Implementarea unor functii necesare

Implementarea functiei board.isPossibleMove(int i, int j, bool color)

```

{
    nextMoveDir = "";

    // i and j out of bound
    if (i < 0 || i >= 8 || j < 0 || j >= 8)
        return false;

    //deja avem un disc acolo
    if (table[i][j] != 2)
        return false;

    int ii; //linie pentru parsarea table
    int jj; //coloana pentru parsarea table

    // test if we can capture disc if on i, j
    // right
    if (j + 2 < 8 && table[i][j + 1] == int(!color))
    {
        for (jj = j + 2; jj < 8 && table[i][jj] == int(!color); jj++);
        if (jj < 8 && table[i][jj] == int(color))
            nextMoveDir += ":r";
    }

    // left
    if (j - 2 >= 0 && table[i][j - 1] == int(!color))

```



```

{
    for (jj = j - 2; jj >= 0 && table[i][jj] == int(!color); jj--);
    if (jj >= 0 && table[i][jj] == int(color))
        nextMoveDir += ":l";
}

// above
if (i - 2 >= 0 && table[i - 1][j] == int(!color))
{
    //verificam unde se termina sirul de discuri !color
    for (ii = i - 2; ii >= 0 && table[ii][j] == int(!color); ii--);
    //daca sirul se termina cu un disc de color, putem captura discuri
    if (ii >= 0 && table[ii][j] == int(color))
        nextMoveDir += ":ab";
}

// below
if (i + 2 < 8 && table[i + 1][j] == int(!color))
{
    for (ii = i + 2; ii < 8 && table[ii][j] == int(!color); ii++);
    if (ii < 8 && table[ii][j] == int(color))
        nextMoveDir += ":be";
}

// diagonally, above right
if (i - 2 >= 0 && j + 2 >= 0 && table[i - 1][j + 1] == int(!color))
{
    for (ii = i - 2, jj = j + 2; ii >= 0 && jj < 8 && table[ii][jj] == int(!color); ii--,
jj++);
    if (ii >= 0 && jj < 8 && table[ii][jj] == int(color))
        nextMoveDir += ":ar";
}

// diagonally, above left
if (i - 2 >= 0 && j - 2 >= 0 && table[i - 1][j - 1] == int(!color))
{
    for (ii = i - 2, jj = j - 2; ii >= 0 && jj >= 0 && table[ii][jj] == int(!color); ii--,
jj--);
    if (ii >= 0 && jj >= 0 && table[ii][jj] == int(color))
        nextMoveDir += ":al";
}

// diagonally, below right
if (i + 2 < 8 && j + 2 < 8 && table[i + 1][j + 1] == int(!color))
{

```

```

        for (ii = i + 2, jj = j + 2; ii < 8 && jj < 8 && table[ii][jj] == int(!color); ii++,
jj++);
        if (ii < 8 && jj < 8 && table[ii][jj] == int(color))
            nextMoveDir += "br";
    }
    // diagonally, below left
    if (i + 2 < 8 && j - 2 >= 0 && table[i + 1][j - 1] == int(!color))
    {
        for (ii = i + 2, jj = j - 2; ii < 8 && jj >= 0 && table[ii][jj] == int(!color); ii++,
jj--);
        if (ii < 8 && jj >= 0 && table[ii][jj] == int(color))
            nextMoveDir += "bl";
    }
    // check if we found directions from where we can capture discs
    if (nextMoveDir != "")
        return true;
    return false;
}

```

Implementarea functiei board.canMove(bool color)

```

{
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            if (table[i][j] == int(!color))
                // verify if this is a case where capturing a disc is impossible
                &&
                /*search if we can capture the disc*/
                ( // vertically
                    isMovePossible(i - 1, j, color) || isMovePossible(i + 1, j, color)
                    // horizontally
                    ||
                    isMovePossible(i, j - 1, color) || isMovePossible(i, j + 1, color)
                    // diagonally to the right
                    ||
                    isMovePossible(i - 1, j - 1, color) || isMovePossible(i + 1, j + 1, color)
                    //diagonally to the left
                    ||
                    isMovePossible(i - 1, j + 1, color) || isMovePossible(i + 1, j - 1, color)
                )
            )
            return true; // can make a move
        }
    }
}

```

```

    }
    // can't capture any disc, so no move possible
    return false;
}

```

4.3 Scenarii de utilizare

Identificarea unui utilizator

Flow-ul programului

1. Utilizatorul i se scrie sa se logheze si sa introduca mai intai numele
2. Utilizatorul scrie numele
 - a. Utilizatorul a scris un nume de max 50 caractere, care nu contine spatii

Cazuri de eroare:

- b. Utilizatorul a scris un nume de ≥ 50 caractere. Utilizatorului i se scrie „Your username can't have more than 50 characters!”. Utilizatorul este intors la pasul 1.
 - c. Utilizatorul a scris un nume care contine spatii. Utilizatorului i se scrie „Your " username can't contain spaces!”. Utilizatorul este intors la pasul 1.
 - d. Utilizatorul a scris un nume de ≥ 50 caractere care contine spatii. Utilizatorului i se scrie: „Your username can't have more than 50 characters or spaces!”. Utilizatorul este intors la pasul 1.
3. Utilizatorul i se scrie sa introduca parola
4. Utilizatorul scrie parola
 - a. Utilizatorul a scris o parola de max 50 caractere, care nu contine spatii.

Cazuri de eroare:

- b. Utilizatorul a scris o parola de ≥ 50 caractere. Utilizatorului i se scrie „Your password can't have more than 50 characters!”. Utilizatorul este intors la pasul 3.
 - c. Utilizatorul a scris o parola care contine spatii. Utilizatorului i se scrie „Your " password can't contain spaces!”. Utilizatorul este intors la pasul 3.
 - d. Utilizatorul a scris o parola de ≥ 50 caractere care contine spatii. Utilizatorului i se scrie: „Your password can't have more than 50 characters or spaces!”. Utilizatorul este intors la pasul 3.
5. Utilizatorul este identificat
 - a. Utilizatorul a introdus un nume si o parola care se afla in baza de date. Utilizatorul afla ca a fost logat.
 - b. Utilizatorul a introdus un nume care nu se afla in baza de date, deci a fost creat un user nou cu parola data. Utilizatorul afla ca a fost creat un cont nou.

Cazuri de eroare:

- c. Utilizatorul a introdus un nume existent in baza de date, dar parola data a fost gresita. Utilizatorul i se scrie „Username or password incorrect! Try again.” Utilizatorul este intors la pasul 1.
- 6. Utilizatorul asteapta aparitia unui alt utilizator pentru a se juca
 - a. Utilizatorului i se scrie ca a fost gasit un oponent si numele
 - b. Utilizatorul este transferat in „Vizualizarea clasamentului”
 - c. Fiecarui utilizator ii este asignata o culoare si o ordine. Ambii sunt informati despre culoarea si ordinea jocului.
 - d. Ambii jucatori sunt transferati in „Efectuarea unei miscari”

Vizualizarea clasamentului

Flow-ul programului

- 1. Utilizatorul este intrebat daca doreste vizualizarea clasamentului
 - a. Utilizatorul nu doreste vizualizarea clasamentului
 - b. Utilizatorul doreste vizualizarea clasamentului
 - (1) Utilizatorul vede mesajul „How many users do you want to see from the leaderboard?”
 - i. Inputul reprezinta un numar > 0 si nu depaseste marimea clasamentului
- Cazuri de eroare:
- ii. Inputul nu este un numar
 - (a) Utilizatorului i se spune „Please type a number!”. Utilizatorul este returnat la pasul 1b.
 - iii. Inputul este un numar ≤ 0
 - (a) Utilizatorului i se spune „Please type a number greater than 0!”. Utilizatorul este returnat la pasul 1b.
 - iv. Inputul este un numar, dar depaseste marimea clasamentului curent
 - (a) Utilizatorului i se returneaza tot clasamentul.
- (2) Sunt returnati primii n jucatori din clasament

Efectuarea unei miscari

Flow-ul programului

- 1. Este verificat daca utilizatorul curent are miscari posibile
- 2. Daca nu are, jucatorul curent primeste mesajul „You don't have any available moves at the moment.”. Jucatorul curent este modificat.
- 3. Jucatorul curent primeste mesajul: „Please type a valid move.”
- 4. Jucatorul curent introduce o miscare
- 5. Miscarea se valideaza

- a. Miscarea este de formatul {A...H}{1...8} si miscarea este una posibila in cadrul partidei.

Cazuri de eroare:

- b. Miscarea nu este de formatul {A...H}{1...8}
 - (1) Utilizatorul este notificat sa introduca o miscare valida cu formatul necesar. Utilizatorul vede textul „Please type a move with the correct format: first character must be a letter from A to H, and the second a number from 1 to 8!”. Utilizatorul este reintors la pasul 3
- c. Miscarea este de formatul {A...H}{1...8}, dar nu este una posibila in cadrul partidei (nu captureaza nici un disc de culoarea opusa)
 - Utilizatorul i se spune ca miscarea aleasa nu este posibila in cadrul partidei curente. Utilizatorul vede mesajul „Move impossible! Try again.” Utilizatorul este intors la pasul 3.

6. Se efectueaza miscarea.

7. Este verificat daca miscarea a generat vreun caz in care jocul sfarseste

- a. Jocul s-a terminat
 - (1) Ambii utilizatori sunt notificati despre terminarea jocului si castigator.
 - i. In caz ca este doar un castigator fiecare utilizator afla ca a castigat sau pierdut in functie de caz.
 - ii. In caz ca este egalitate li se spune utilizatorilor ca este egalitate.
 - iii. Se incrementeaza scorul fiecarui castigator in baza de date
 - iv. Fiecare jucator este transferat in „Vizualizarea clasamentului”
 - v. Conexiunea cu fiecare jucator este intrerupta
- b. Jocul nu s-a terminat
 - (1) Este schimbat urmatorul jucator si flow-ul este intors la punctul 1

5 Concluzii

5.1 Rezumat

Proiectul va fi divizat in mai multe entitati.

Prima va fi clasa Board. Ea va fi responsabila de memorarea starii tablei de joc si numarului discurilor negre si albe de pe tabla. Board va presta toate serviciile necesare interactiunii cu tabla de joc. Board va desena tabla de joc la cerere conform datelor membre prezente in ea. Board va verifica daca o mutare captureaza un disc (in orice directie). De asemenea, va verifica daca un jucator poate efectua vreo mutare (adica captura vreun disc de culoarea opusa de pe tabla de joc). Board va avea un constructor care va crea starea initiala din jocul Othello. Board va efectua o mutare primita de la server. Mutarea este compusa din: plasarea unui disc pe tabla de joc si inversarea tuturor discurilor de culoare opusa capturate. Board va mai avea rolul de a stabili daca jocul s-a incheiat (daca nici un jucator nu mai poate face mutari sau un jucator nu are nici un disc pe tabla). Board va fi cel ce stabileste cine este castigatorul unui joc (care culoare) si daca este egalitate intre ei.

A doua entitate este db-queries.h. Db-queries se va ocupa de toata interactiunea cu baza de date, primind inputul de la server. Astfel, serverul nu va interactiona „direct” cu baza de date, dar printr-un „intermediar”. Db-queries va fi responsabil de crearea bazei de date (unui tabel ce va contine coloanele: username, password, score). Db-queries va deschide si inchide conexiunea cu baza de date la cererea serverului (respectiv, cand serverul are nevoie de baza de date si cand nu mai are). Db-queries va adauga un utilizator in baza de date pe baza unui username si a unei parole. De asemenea, va cauta daca exista in baza de date un user cu un username anume si daca exista un user cu un username si o parola anume. Pe langa aceasta, la necesitate db-queries va incrementa scorul unui username cu 1 (cand acel user a castigat). Db-queries se va ocupa de afisarea clasamentului ceruta de server (cererea fiind receptionata de la client).

A treia entitate este clientul. Clientul se va conecta la server. Clientul va fi responsabil de colectarea datelor de la utilizator: numele si parola dorita pentru autentificare, numarul n pentru afisarea primelor n persoane din clasament si miscarea dorita spre efectuare intr-un anumit moment al jocului. Clientul va mai fi responsabil de validarea unei miscari, adica verificarea daca miscarea este de tipul {A...H}{1..8}. De asemenea, clientul va fi cel ce va prelua datele de la server si transmite utilizatorului un mesaj concludent. Clientul va cere date valide in cazul prestarii unor invalide, de exemplu a unui username si parole gresite sau a scrierii unei mutari invalide sau imposibile. Clientul va notifica utilizatorul daca randul lui a fost schimbat pentru ca nu poate efectua o mutare. Per general, clientul va procesa orice cerere venita de la server si va trimite orice cerere de la utilizator la server.

Ultima entitate este serverul. Serverul este punctul central al programului, conectand clientul la tabla de joc, baza de date si la restul functionalitatilor jocului. Serverul va fi cel ce apeland la db-queries va garanta logarea sau crearea unui utilizator in cadrul bazei de date. Serverul va trimite mereu mesaj clientului despre starea logarii: cont creat, utilizator logat sau parola invalida pentru contul specificat (in client e trimis textul ca „username or password wrong” pentru a mentine un nivel de securitate mai inalt). Serverul va fi cel ce va accepta cereri de la clienti si crea perechi pentru a incepe un joc. In cazul imposibilitatii crearii unei perechi timp de 1 minut clientul este notificat, iar conexiunea intrerupta. Serverul va crea un proces copil pentru gestionarea unei partide de joc pentru fiecare pereche de clienti. Procesele copil poate cere informatii despre clasament la inceputul sau sfarsitul jocului de la db-queries, in caz ca vreun client cere aceasta. Procesele copil vor dirija desfasurarea corecta a jocului utilizand functiile din clasa Board. Vor stabili ordinea miscarilor, vor garanta efectuarea unei miscari in caz ca ea e corecta si terminarea unui joc. Procesele copil vor fi cele ce va trimit cereri pentru db-queries in caz ca e necesara incrementarea scorului unui utilizator.

5.2 Imbunatatiri posibile

O imbunatatire posibila ar putea fi legata de securitatea logarii. Parola ar putea fi pastrata codificat pentru a minimiza riscul de „spargere” a unui cont. Implementarea e posibila folosind diverse mecanisme de criptare disponibile. O alta imbunatatire ar fi

o blocare a utilizatorilor care gresesc de prea multe ori o miscare sau datele de logare, astfel incarcand serverul fara motiv. Numele si parola acestor utilizatori va fi pastrata intr-un nou tabel in baza de date, denumit „Banned users”. Acest tabel ar reprezenta o penalizare, ea putand fi eliminata dupa o anumita perioada de timp. Fiecare utilizator va fi notificat daca devine banned si daca incearca sa se logheze fiind banned. Crearea unei interfete grafice mai prietenoasa unui client ar usura comunicarea dintre client si server. Interfata noua creata ar scrie mereu ce date sunt necesare de la client si ar prezenta notificari despre joc. Interfata ar deveni astfel una individuala, la moment ea fiind pentru o pereche de utilizatori. Astfel, interfata ar reusi sa creeze o interactiune personalizata si focusata pe client, nu pe un cuplu de clienti. Pe langa aceasta, memorarea si afisarea fiecărei miscari efectuate de un utilizator pe parcursul unui joc ar ajuta utilizatorii sa invete din greseli sau sa observe mai usor parcursul jocului. Va fi afisat cine a efectuat o miscare si miscarea respectiva. Ar fi util, de asemenea, de acordat posibilitatea unui utilizator sa memoreze acest parcurs al jocului intr-un fisier.

References

1. SQLite page When To Use, <https://sqlite.org/whentouse.html>, ultima accesare: 2021/12/02
2. Reference Manual Nana C++ Library, <http://nanapro.org/en-us/documentation/>, ultima accesare: 2021/12/02
3. Introduction to Nana C++ Library, <http://nanapro.org/en-us/blog/2016/05/an-introduction-to-nana-c-library/>, ultima accesare: 2021/12/03
4. Reference for Users Nana C++ Library examples, <http://qpcr4vir.github.io/nanadoxy/html/examples.html>, ultima accesare: 2021/12/02
5. An Introduction to the SQLite C/C++ Interface, <https://www.sqlite.org/cintro.html>, ultima accesare: 2021/12/02
6. SQLite – C/C++ tutorialspoint page, https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm, ultima accesare: 2021/12/02
7. Reversi Rules - Reversi Documentation, <https://documentation.help/Reversi-Rules/rules.htm>, ultima accesare: 2021/12/02
8. Curs 5 Retele de Calculatoare - Programarea in Retea I, https://profs.info.uaic.ro/~computernetworks/files/5rc_ProgramareaInReteaI_ro.pdf, ultima accesare: 2021/12/02
9. Curs 7 Retele de Calculatoare - Programarea in Retea III, https://profs.info.uaic.ro/~computernetworks/files/7rc_ProgramareaInReteaIII_Ro.pdf, ultima accesare: 2021/12/02
10. Laborator 4 Retele de Calculatoare - Georgiana Calancea page, https://profs.info.uaic.ro/~georgiana.calancea/Laboratorul_4.pdf, ultima accesare: 2021/12/03
11. Laborator 7 Retele de Calculatoare - Georgiana Calancea page, https://profs.info.uaic.ro/~georgiana.calancea/Laboratorul_7.pdf, ultima accesare: 2021/12/02

12. Laborator 12 Retele de Calculatoare - Georgiana Calancea page,
https://profs.info.uaic.ro/~georgiana.calancea/Laboratorul_12.pdf, ultima
accesare: 2022/01/11
13. SDL wiki, <https://wiki.libsdl.org/>, ultima accesare: 2022/01/11
14. Reversi Rules, yourturnmyturn page,
<https://www.yourturnmyturn.com/rules/reversi.php?pdf=1>, ultima accesare:
2022/01/11