Documentatia proiectului ReversiS

Vivdici Ina[1[2B4]]

[1] Universitatea “Alexandru Ioan Cuza”, Iasi, Romania
sendmailtoina@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 rularea 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 viteza mai mica decat UDP, dar un nivel de securitate a transmiterii de date mai mare. Natura jocului Othello nu este una ce necesita reactie rapida, dar analiza strategica, deci nu necesita neaparat cea mai buna viteza de transmitere a datelor (mai concret a unei miscari). De asemenea, este crucial sa obtinem datele unei miscari nedistorsionate, altfel natura jocul va fi alterata. Folosind UDP as fi avut nevoie de implementat manual aceasta verificare de date primite, dar din considerentul ca viteza nu este un factor de decizie in acest joc este mai convenabil TCP. Pe langa aceasta, nu am servicii de tip broadcast. Un joc include doar 2 clienti, iar singurul care interactioneaza cu ei este serverul. Informatia trimisa de server este mereu una individuala, deci nu vom avea nevoie de un serviciu multicast sau broadcast.

2.2 SQLite - baza de date

Chiar daca proiectul ales nu impune utilizarea unei baze de date am decis in loc de a folosi un simplu fisier sa folosesc o baza de date. Motivatia a fost bazata pe gestionarea facila a datelor, usurand cautarea unui nume de utilizator, parola sau afisarea clasamentului. De asemenea, actualizarea informatiilor despre utilizatori (punctajul sau adaugarea unui nou utilizator) este mai usoara, existand interogari anumite pentru a executa aceste operatii. Am ales SQLite in loc de XML pentru ca structura bazei mele de date este mai usor de vizualizat in forma de tabel, decat arbore. De asemenea, nu aveam nevoie sa transfer date dintre diferite entitati pentru a avea probleme cu formatarea, adica sa am nevoie de XML sau altceva de genul. In cazul meu, doar serverul va fi cel ce acceseaza, deci nu voi avea probleme legate de formatul datelor. Faptul ca SQLite este usor de utilizat si este rapid a fost un factor decisiv. Libraria folosita pentru interactiunea dintre SQLite si C++ este sqlite3 din considerentul popularitatii ei si vechimii.

2.3 graphics.h – libraria pentru interfata grafica

Am decis sa creez tabla de joc folosind o librarie grafica pentru a fi mai usor de vizualizat miscarile si tabla de joc in sine. Am decis sa folosesc libraria graphics pentru a atinge acest scop. Libraria in cauza are functii prestabilite pentru actiunile necesare mie. De exemplu, desenare cerc, dreptunghi, linii si colorare de forme. Comparativ cu nana, o librarie pentru GUI, graphics este mai simplist in abordarea pe care o ofera. Nana, pe de alta parte are o multitudine de diverse functii specializate pe crearea unei interfete de utilizator. In ciuda acestui fapt, graphics a fost mai convenabila, caci scopul meu a fost crearea unei simple table de joc, nu a unui GUI complet. Un fapt important a fost si ca nana nu contine o functie prestabilita pentru crearea unui cerc, fiind necesara o implementare creata de mine. Daca va fi dorita o implementare mai complexa a interfetei grafice, graphics.h ofera cateva functii, iar la

necesitate se pot implementa functionalitatile dorite. Chiar daca folosind o librarie mai complexa ar fi fost mai usor de imbunatatit libraria grafica, scopul proiectului nu era o interfata grafica atractiva sau complexa.

2.4 fork – entitatea care dirijeaza o partida de joc

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 clientiilor 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-ului 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.

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 „username” si clientul va scrie numele. Dupa aceea va fi ceruta si parola, iar serverul va scrie clientului: „password”. Daca exista asa jucator, jucatorul va intra in joc. Daca nu exista se creeaza un cont nou, iar daca parola nu este corecta se va trimite clientului: „wrong password or username”. 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. Clientul va cere aceste date scriind comanda: „n”. 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

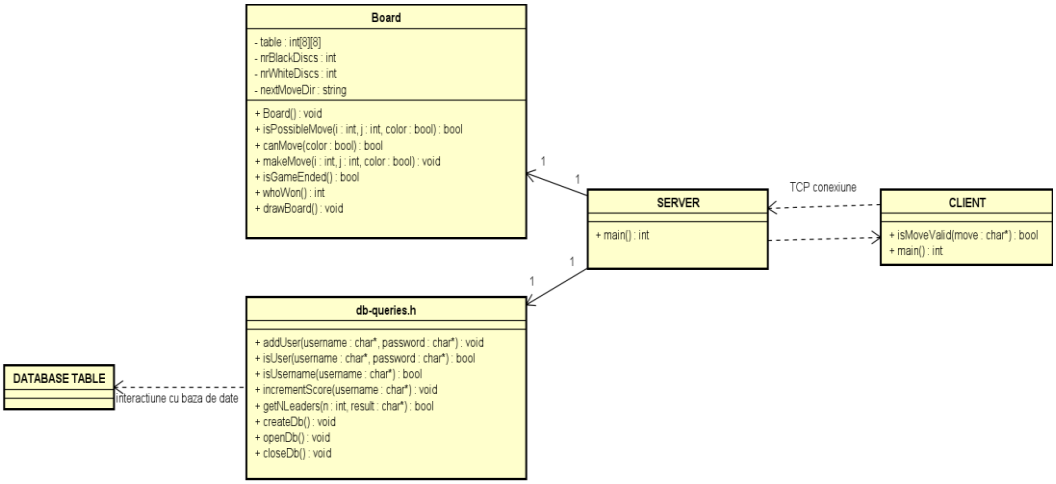

Fig. 1. Diagrama UML de clase a proiectului

4 Detalii de implementare

4.1 Logica proiectului reprezentata prin pseudocod

board.h

```
class Board
{
    int table[8][8]; // table e tabla de joc; table[i][j] = 0 avem un disc negru pe
    //pozitia i, j; table[i][j] = 1, avem un disc alb; table[i][j] = 2 este liber
    int nrBlackDiscs; //numarul discurilor negre plasate pe tabla
    int nrWhiteDiscs; //numarul discurilor albe plasate pe tabla
    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
    //valori posibile a variabilei si semnificatia lor:
```

```
        //'d' | 'st' | 's' | 'j' | 'sd' |'sst' |'jd' |'jst' | '0'  -> dreapta | stanga | sus | jos | sus
        dreapta // | sus stanga | jos dreapta | jos stanga | nu avem vreo directie la moment

        public:
        Board(); //va reproduce in table starea initiala a tablei din jocul Othello;
        //nrBlackDisc = nrWhiteDiscs = 2
        bool isPossibleMove (int i, int j, bool color); // i - nr liniei, j - nr coloanei, pe
        //care dorim sa punem un disc, iar color culoarea jucatorului care doreste sa
        //efectueze miscarea; color = 0 e pentru jucatorul negru, color = 1 pentru
        //jucatorul alb; isPossible move va memora „directia de capturare" a miscarii
        //daca este una corecta
        bool canMove(bool color); //verificam daca jucatorul care are culoarea color
        //poate plasa un disc pe tabla in aceasta runda
        void makeMove(int i, int j, bool color); //pozitionam discul pe pozitia dorita,
        //schimbam culoarea discurilor capturate de color cu miscarea i, j si actualizam
        //nrBlackDiscs si nrWhiteDiscs
        bool isGameEnded(bool nextPlayer); //verificam daca conditiile de terminare a
        //unui joc sunt intrunite: nu avem nici un loc liber pe tabla de joc; unul dintre
        //jucatori nu are nici un disc pe tabla;  nici jucatorul urmator nici celalalt jucator
        //nu mai au miscari posibile;
        int whoWon(); //returnam cine are mai multe discuri de culoarea lui pe tabla, iar
        //daca numarul lor e egal returnam 2, ce semnifica ca ambii au castigat
        void drawBoard(); //sterge ecranul de tot ce a fost desenat inainte si deseneaza
        //starea curenta a tablei de joc
}
```

db-queries.h

```
void addUser(char * username, char * password); //adauga utilizatorul cu numele
//username cu parola password in baza de date
bool isUser(char * username, char * password); //verifica daca exista un utilizator cu
//numele username cu parola password in baza de date, returneaza true daca este si
//false altfel
bool isUsername(char * username); //avem un utilizator cu numele username in baza
//de date -> returneaa true, altfel false
void incrementScore(char * username); //mareste scorul utilizatorului cu numele
//username cu 1
bool getNLearders(int n, char * result); //returneaza numele primilor n jucatori din
//clasament in variabila result daca n <= multimea numelor de jucatori, valoarea
//returnata este true, daca n > valoarea returnata este false
void createDb(); //creaza tabelul users continand coloanele: username, password,
//score
void openDb(); //crearea conexiunii cu baza de date
void closeDb(); //inchiderea conexiunii cu baza de date
```

client.cpp

```
bool isMoveValid(char * move); //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
asteptam sa aflam daca avem cu cine juca
daca mesajul primit == „sorry” facem shutdown si notificam clientul ca nu are cu cine
se juca
daca mesajul primit == „succes” inseamna ca avem cu cine juca si continuam
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(move) 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 „wrong” 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 numele castigatorului sau afisam „It’s a draw. Your opponent was...”
cu numele oponentului
afisam clasamentul daca utilizatorul doreste
inchidem conexiunea cu serverului
```

server.cpp

```
pregatim socketul de listen si facem listen
while (1)
{
   acceptam un client
   isLogged = 0
   while (!isLogged)
      scriem clientului „user data”
      citim numele si parola clientului
   if (isUsername(username1))
      if(isUser(username1, password))
         trimite clientul „user logged”
```

```
        isLogged = 1
    else
        trimite clientului „username or password wrong”
else
    addUser(username1, password)
    trimite clientului „user created”
    isLogged = 1
```

mai acceptam un client si il logam (folosim username2)
daca nu gasim un client (timp de 1 min) cu care sa se joace primul ii trimitem clientului ”sorry” pe primul despre asta si facem exit
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
    Board board;
    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 (isPossibleMove(move, color))
            scriem clientului „success”
            makeMove(move, color)
        else
            cerem o noua miscare valida scriind clientului „wrong”
        board.drawBoard();
        if (board.isGameEnded())
            end = 1
            spunem ambilor jucatori „end”
        else
            schimbam urmatorul jucator
    }
    afisam fiecarui utilizator whoWon(), iar daca e egalitate scriem „draw+numele
    oponentului”
```

```
        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 = "0"; //reinitilizam directia

    //i si j nu se incadreaza in tabel
    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; //ii va fi folosit pentru linie pentru parsarea table
    int jj; //jj va fi folosit pentru coloana pentru parsarea table

    //testam daca putem captura discuri daca ne-am pozitiona pe i, j
    //sus
    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 discul dorit
        if (ii >= 0 && table[ii][j] == int(color))
        {
            nextMoveDir = "s";
            return true;
        }
    }

    //jos
    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 = "j";
            return true;
        }
    }

    //dreapta
    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 = "d";
            return true;
        }
    }

    //stanga
    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 = "st";
            return true;
        }
    }

    //pe diagonala, sus dreapta
    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 = "sd";
            return true;
        }
    }

    //pe diagonala, sus stanga
    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 = "sst";
            return true;
        }
    }
    //pe diagonala, jos dreapta
    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 = "jd";
            return true;
        }
    }
    //pe diagonala, jos stanga
    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 = "jst";
            return true;
        }
    }
    //nu putem captura nici un disc in vreo directie
    return false;
}
```

Implementarea functiei board.canMove(bool color)

```
{
    //cautam discurile albe si incercam sa le capturam in diferite directii
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            if ( table[i][j] == int(!color)
                //verificam daca nu suntem intr-un caz in care nu avem
                //cum captura discul
                && (i != 0 && j != 0) && (i != 7 && j != 0)
                && (i != 0 && j != 7) && (i != 7 && j != 7)
                && /*cautam daca putem captura discul*/
                (   //vertical
```

```
                isPossibleMove(i-1, j, color) || isPossibleMove(i+1, j, color)
                //orizontal
                || isPossibleMove(i, j-1, color) || isPossibleMove(i, j+1, color)
                //diagonal spre dreapta
                || isPossibleMove(i-1, j-1, color) || isPossibleMove(i+1, j+1, color)
                //diagonal spre stanga
                || isPossibleMove(i-1, j+1, color) || isPossibleMove(i+1, j-1, color)
            )
           )
            return true; //putem efectua o miscare
        }
      }
   //nu am putut captura nici un disc, deci nu avem ce miscari sa facem
   return false;
}
```

4.3 Scenarii de utilizare

Identificarea unui utilizator

Flow-ul programului

1. Utilizatorul i se scrie „Type your username...”
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 „Type a username that’s smaller 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 have spaces! Type a username without spaces.”. Utilizatorul este intors la pasul 1.

3. Utilizatorul i se scrie „Type your password...”
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 „Type a password that’s smaller 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 have spaces! Type a password without 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. Utilizatorul necesar este gasit timp de 1 min
      (1) Utilizatorului i se scrie „We found a player to play with!”
      (2) Utilizatorul este transferat in „Vizualizarea clasamentului”
      (3) Fiecarui utilizator ii este asignata o culoare si o ordine. Ambii sunt informati despre culoarea si ordinea jocului.
      (4) Ambii jucatori sunt transferati in „Efectuarea unei miscari”
   b. Utilizatorul necesar nu este gasit timp de 1 min
      (1) Utilizatorul este notificat despre lipsa unui partener pentru un joc, iar conexiunea cu el este intrerupta. Utilizatorul primeste mesajul „We didn’t find any player you could play with. Sorry!”

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 „Type a number greater than 0...”
         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 spune „Please type a number less than (marime+1)”, marime fiind marimea clasamentului. Utilizatorul este intors la pasul 1b.

      (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. Your turn is skipped". Jucatorul curent este modificat.
3. Jucatorul curent primeste mesajul: „Please type a 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, utilizatorul primeste mesajul „The move was correct."

    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 „Impossible move in the current state of the game! Please type a possible move." 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 primeste numele castigatorului.
            ii. In caz ca este egalitate li se spune utilizatorilor ca este egalitate si numele oponentului.
            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 unora 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. Procese 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 fiecarei 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. Programming digest page code examples for graphics, https://programmingdigest.com/c-graphics-with-example-codes-for-drawing-lines-rectangle-and-circle-using-graphics-function/, ultima accesare: 2021/12/02
3. Programming simplified page graphics h, https://www.programmingsimplified.com/c/graphics.h, ultima accesare: 2021/12/02
4. Reference Manual Nana C++ Library, http://nanapro.org/en-us/documentation/, ultima accesare: 2021/12/02
5. Introduction to Nana C++ Library, http://nanapro.org/en-us/blog/2016/05/an-introduction-to-nana-c-library/, ultima accesare: 2021/12/03
6. Reference for Users Nana C++ Library examples, http://qpcr4vir.github.io/nana-doxy/html/examples.html, ultima accesare: 2021/12/02

7. An Introduction to the SQLite C/C++ Interface, https://www.sqlite.org/cintro.html, ultima accesare: 2021/12/02
8. SQLite – C/C++ tutorialspoint page, https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm, ultima accesare: 2021/12/02
9. Reversi Rules - Reversi Documentation, https://documentation.help/Reversi-Rules/rules.htm, ultima accesare: 2021/12/02
10. 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
11. 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
12. Laborator 4 Retele de Calculatoare - Georgiana Calancea page, https://profs.info.uaic.ro/~georgiana.calancea/Laboratorul_4.pdf, ultima accesare: 2021/12/03
13. Laborator 7 Retele de Calculatoare - Georgiana Calancea page, https://profs.info.uaic.ro/~georgiana.calancea/Laboratorul_7.pdf, ultima accesare: 2021/12/02