Proiect - Reversi (B)

Bocicov Vitalie

Facultatea de Informatică
Universtatea "Alexandru Ioan Cuza" Iași
vitalie.bocicov@info.uaic.ro

1 Introducere

Proiectul *ReversiS* propune creearea unui server care să ofere unor perechi de clienți posibilitatea să joace *Reversi*, un joc de masă de strategie pentru doi jucători, jucat pe o tablă 8x8. Jucătorii vor fi identificați prin nume unice. Primii N cei mai buni jucători vor fi salvați într-un clasament.

2 Tehnologii utilizate:

2.1 TCP

Protocolul de comunicație a fost implementat folosind **TCP**. Am considerat că, **TCP** este mai potrivit acestui proiect decât **UDP** pentru că avem nevoie de:

- o conexiune între clienți și server;
- siguranța transmiterii (în ordine) a datelor;
- mecanisme de control al fluxului și control al congestiei;

2.2 SQLite

Am folosit biblioteca **SQLite** care implementează un motor de baze de date **SQL** încapsulat, pentru a salva un clasament al primilor N cei mai buni jucători. Am ales această bibliotecă pentru că necesită zero-configurare, și pentru că ușurează procesul de creare si manipulare a bazei de date.

2.3 SFML

Interfața grafică a fost implementată cu ajutorul bibliotecii **SFML** (Simple and Fast Multimedia Library), o bibliotecă multimedia portabilă și ușor de folosit, scrisă în **C++**. **SFML** pune la dispoziție grafică 2D accelerată prin hardware, folosind **OpenGL**, și câteva module pentru a ușura programarea jocurilor și a aplicațiilor multimedia.

3 Arhitectura aplicatiei

3.1 Conceptele implicate

Serverul creează un socket **TCP**, inițializează baza de date și așteaptă conectarea jucătorilor. Vor fi acceptați, în mod concurent, perechi de jucători. Serverul va crea pentru fiecare pereche un **thread**, astfel niciun jucător nu va fi nevoit să aștepte terminarea unei altei partide ca să poată juca. Jucătorul va aștepta doar conectarea unui adversar, ca să poată începe jocul.

În continuare, **thread-ul** creat de server se va ocupa de comunicarea cu cei doi jucători. Fiecare jucător va primi tabla, care conține piesele celor doi jucători, scorul și mutările posibile. Ordinea mutării va fi în ordinea conectării acestora la server(va începe primul jucător conectat). Procesul va primi pe rând coordonatele mutării de la fiecare jucător și va trimite înapoi tabla, scorul și mutările posibile actualizate.

Când jocul se va termina, procesul îi va cere învingătorului să-și trimită numele, ca să poată fi salvat, alături de scor, în clasament.

3.2 Diagrama aplicației

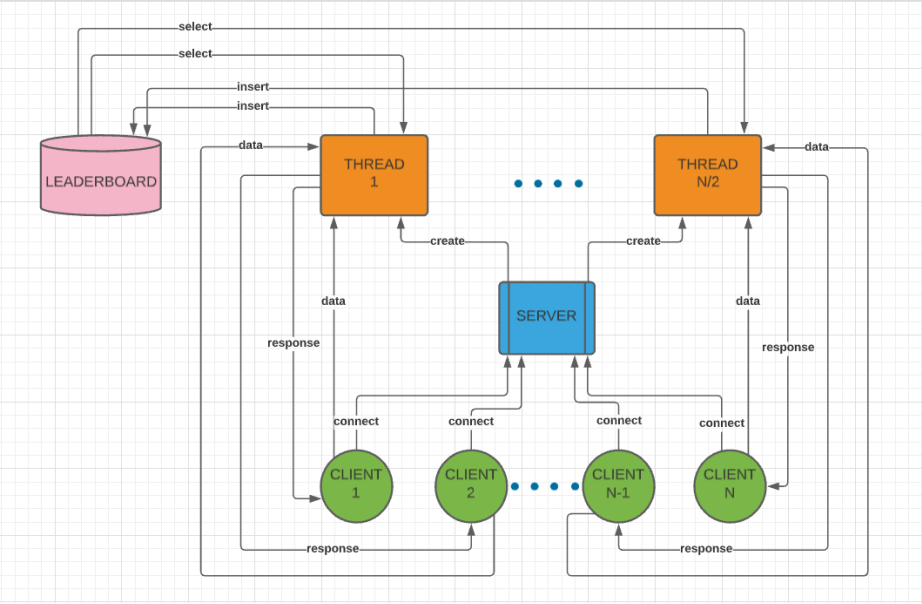


Figura 1. Diagrama aplicatiei

4 Detalii de implementare

4.1 Server

Serverul este compilat și executat folosind comanda *make*. Fiecare jucator conectat va avea asociată o paritate (**0** daca este jucătorul 1, respectiv **1** daca este jucătorul 2), astfel serverul va ști dacă sunt doi jucători conectați. Dacă s-a conectat jucătorul 1, serverul salvează descriptorul jucătorului într-o variabilă și așteaptă conectarea altui jucător. Dacă s-a conectat jucătorul 2, descriptorii jucătorilor vor fi salvați într-o structură predefinită, ce va fi folosită drept parametru pentru apelarea funcției de tratare a fiecătui **thread**.

```
while (1)
   {
        int player;
        int length = sizeof(from);

        if ((player = accept(sd, (struct sockaddr *)&from, &
   length)) < 0)
        {
            perror("error: accept");
            continue;
        }
        // player connected

        if (players == 0) // player 1 connected
        {
            printf("PLAYER 1 CONNECTED!\n");
            fflush(stdout);
            players++;
            p1_desc = player;         // saving player 1
   descriptor
            msg_player_1(player, 0); // connected, waiting
   for player 2
        }
        else // player 2 connected
        {
            printf("PLAYER 2 CONNECTED!\n");
            fflush(stdout);
            players--;
            msg_player_2(player, 0); // connected, starting
   game

            th_desc *td;
            td = (struct thData *)malloc(sizeof(struct thData
   ));
            td->desc1 = p1_desc;
            td->desc2 = player;
```

Dacă avem o pereche de jucători conectați, serverul creează un **thread**, care se va ocupa de desfășurarea jocului. Va fi apelată funcția *treat*, funcția de tratare

a fiecărui *treat*, cu structura ce conține descriptorii celor doi jucători, drept parametru.

```
printf("CREATING THREAD...\n");
fflush(stdout);
pthread_t thread;
pthread_create(&thread, NULL, &treat, td);
```

4.2 Thread

thread-ul se va ocupa în întregime de desfășurarea jocului și de comunicația cu cei doi jucători.

Funcțiile cele mai importante sunt:

- `static void *treat(void *);`

  - se ocupă de fiecare **thread**, apelează funcția de start a jocului, cu cei doi descriptori ca paramatri.

- `void start_game(int desc1, int desc2);`

  - inițializează scorul, tabla și clasamentul.

- ```
  void play(int **board, char *ldboard, int *score, int
      desc1, int desc2);
  ```

  - se ocupă de toate stările jocului și apelează toate funcțiile necesare.

- `void mark_possible_moves(int **board, int player);`

  - actualizează tabla, inserând mutările posibile a jucătorului *player*.

- `int rcv_move(int **board, int desc1, int desc2)`

  - citește mutarea jucătorului cu descriptorul *desc*1.

- `int is_playable(int **board, int x, int y);`

  - verifică dacă mutarea jucătorului este validă.

- ```
  void capture_pieces(int **board, int *score, int x, int y,
       int player);
  ```

  - capturează piesele adversarului conform regulilor jocului.

- ```
  int move_player_1(int **board, int *score, int desc1, int
      desc2);
  int move_player_2(int **board, int *score, int desc1, int
      desc2);
  ```

  - funcțiile ce se ocupă de citirea mutărilor, validarea acestora, actualizarea scorului și a tablei.

- ```
  void pc_vs_player(int **board, char *ldboard, int *score,
      int desc, int player);
  ```

- funcția ce se va ocupa de meciul jucătorului împotriva calculatorului.

```
void msg_player_1(int desc, int status);
void msg_player_2(int desc, int status);
void msg_board(int **board, int desc);
void msg_score(int desc, int *score);
void msg_winner(int desc);
void msg_loser(int desc);
void msg_tie(int desc);
void msg_invalid_username(int desc);
void msg_endgame(char *ldboard, int desc1, int desc2, int
    *score);
char *msg_rcv_username(int desc);
void msg_leaderboard(int desc, char *ldboard);
void msg_player_disconnected(int desc);
void msg_invalid_answer(int desc);
```

- toate funcțiile de mai sus trimit mesaje jucătorilor. Fiecare mesaj este prefixat de lungimea acestuia.

```
void endgame(int **board, char *ldboard, int *score, int
    desc1, int desc2);
```

- funcția finală ce se va ocupa de alegerea câștigătorului, trimiterea mesajelor, citirea numelui, actualizarea clasamentului.

4.3 Baza de date

Baza de date va conține un singur tabel ce va stoca scorurile și numele câștigătorilor. Aceasta va fi inițializată de către server, dar va fi manipulată de către procesele copil.

```
void init_db()
{
    sqlite3 *db;
    char *msg_err = 0;

    int ok = sqlite3_open("leaderboard.db", &db);

    if (ok != SQLITE_OK)
    {
        printf("Cannot open database: %s\n",
    sqlite3_errmsg(db));
    }

    char *sql = "CREATE TABLE Leaderboard(Score INT, Name
    TEXT);";
    ok = sqlite3_exec(db, sql, 0, 0, &msg_err);
```

La final, numele câștigătorului va fi adaugăt în clasament, de către **thread**, alături de scorul obținut. Dacă numele există deja în clasament, jucătorul va trebui să reintroducă alt nume.

```
do
    {
        char *query = sqlite3_mprintf("INSERT into
    Leaderboard VALUES (%d,'%q');", score, winner);
        ok = sqlite3_exec(db, query, 0, 0, &msg_err);
        sqlite3_free(query);
        if (ok != SQLITE_OK)
        {
            printf("SQL INSERT error: %s\n", msg_err);
            sqlite3_free(msg_err);
            msg_invalid_username(desc);
            winner = msg_rcv_username(desc);
            query = sqlite3_mprintf("INSERT into
    Leaderboard VALUES (%d,'%q');", score, winner);
            ok = sqlite3_exec(db, query, 0, 0, &msg_err);
            sqlite3_free(query);
        }
    } while (ok != SQLITE_OK);
```

Clasamentul va fi trimis jucătorului învingător, folosind funcțiile de mai sus. Baza de date va fi interogată folosind comanda select *SELECT*.

```
sqlite3_stmt *stmt;

char *sql = "SELECT * FROM Leaderboard ORDER BY 1 DESC";

int rc = sqlite3_prepare_v2(db, sql, -1, &stmt, NULL);
```

În urma interogării, vom primi câte un rând din clasament, ordonat după scor, descrescător. Fiecare rând va fi salvat în vectorul de caractere *ldboard*, ce va fi trimis jucătorilor.

```
int rows = 1;
while (rows++ <= N && (rc = sqlite3_step(stmt)) ==
    SQLITE_ROW)
{
    char *line = malloc(sizeof(char) * 100);
    line[0] = '\0';
    strcat(line, "|");
    const char *score = sqlite3_column_text(stmt, 0);
    const char *name = sqlite3_column_text(stmt, 1);
    strcat(line, score);
    strcat(line, "   |");
    strcat(line, name);
    strcat(line, "\n");
    strcat(ldboard, line);
    free(line);
}
```

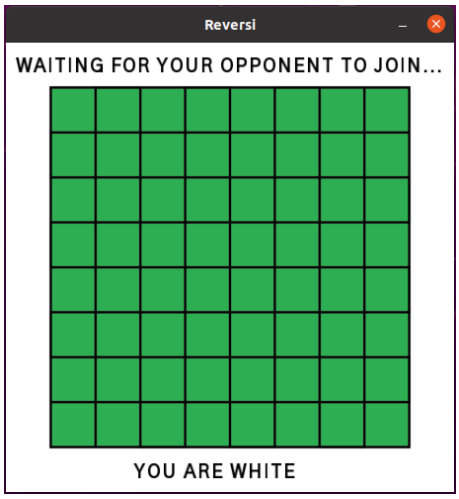
4.4 Client

Clientul se va conecta la server folosind comanda *make run* și va primi mesaje de la server (mai întâi mesajul de conectare), apoi doar de la **thread**. Va citi mai întâi lungimea mesajului, un spațiu(delimitator între lungime și mesaj) apoi mesajul propriu-zis. În funcție de conținutul acestura, funcția de citire va returna un cod, ca să ușureze interpretarea mesajelor. Citirea mesajelor se va fi realizată în mod neblocant, pentru că interfața grafică se blochează o dată cu un apel de funcție blocant, astfel jucătorul nu mai poate interacționa cu fereastra până mută oponentul său, iar sistemul de operare îi propune jucătorului să închidă ferestra pentru că nu mai răspunde. Culoarea jucătorului, scorul și tabla vor fi afișate în mod grafic. Mutările vor vi primite de la jucător prin intermediul click-ului.

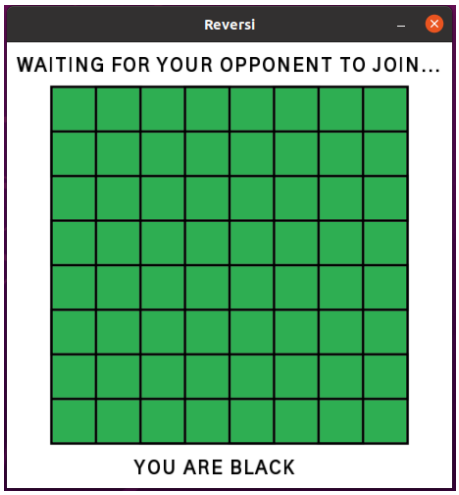
```
int readS(int sd)
{
    int m_len, ok;
    m_len = msg_length(sd);
    char buff[m_len];
    ok = read(sd, buff, m_len);
    if (ok <= 0)
        return ok;
    buff[ok] = '\0';
    if (check_conn(sd, buff))
        return 1;
    if (check_score(sd, buff))
        return 2;
    if (check_board(buff, m_len))
        return 3;
    if (check_insert(sd, buff))
        return 4;
    if (check_player_disconnected(sd, buff))
        return 6;
    if (check_gameover(sd, buff))
    {
        printf("%s\n", buff); fflush(stdout);
        if (check_win(sd, buff))
            return 7;
        if (check_lost(sd, buff))
            return 8;
        if (check_draw(sd, buff))
            return 9;
        if (check_username(sd, buff))
            return 10;
        if (check_leaderboard(sd, buff))
            return 11;
        if (check_invalid_username(sd, buff))
            return 12;
    }
    return 0;}
```

4.5 Scenarii de utilizare

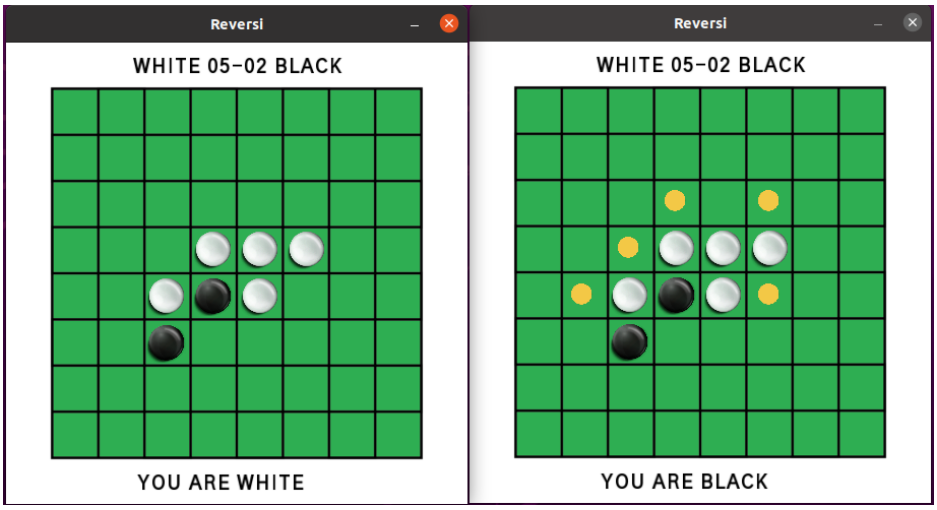
– Când se conectează jucătorul 1, va primi culoare pentru piese, va fi înștiințat că s-a conectat și că așteaptă un adversar să se conecteze, prin intermediul interfeței grafice.



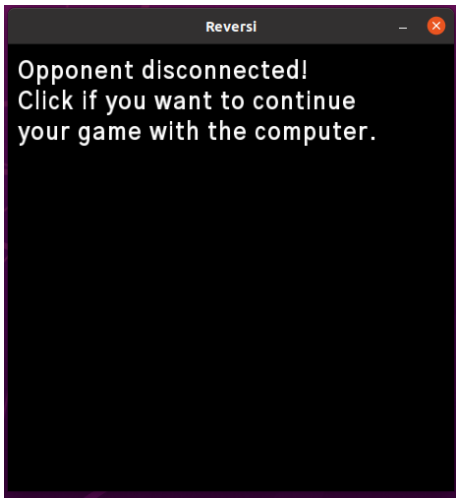
– Când se conectează jucătorul 2, va primi de asemenea o culoare pentru piese și jocul începe. Fiecare jucător va aștepta mutarea celuilalt jucător



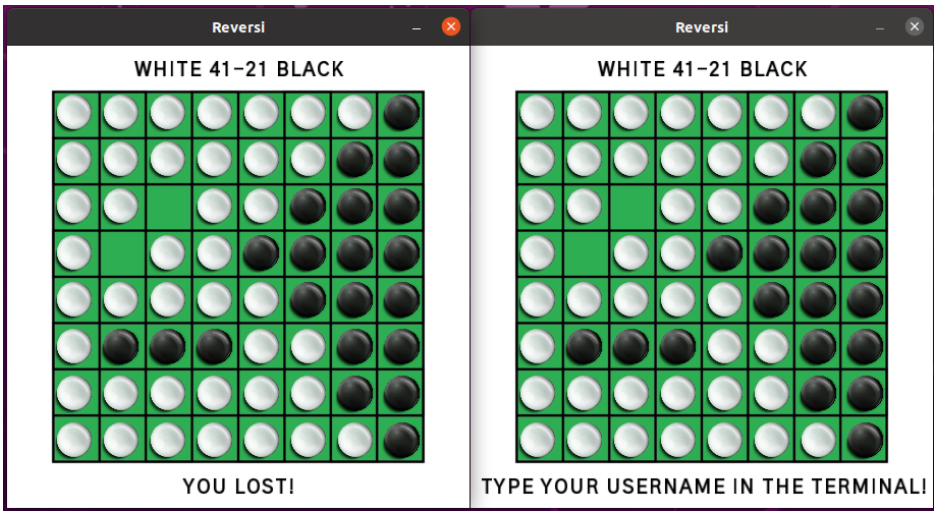
– Fiecare jucător va primi scorul și tabla și piesele în funcție de culoarea primită la început. Cu puncte galbene vor fi marcate mutările posibile, calculate individual pentru fiecare jucător.

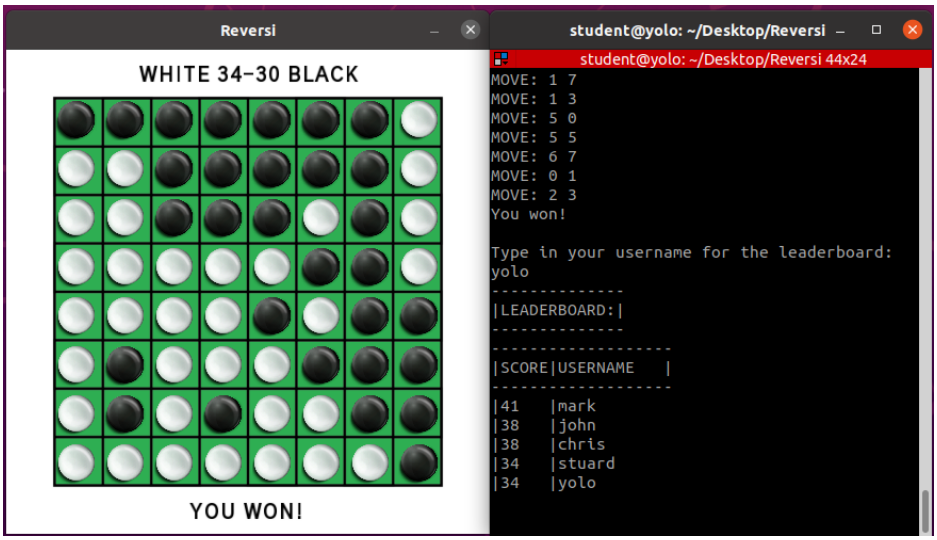


– Jucătorii nu pot alege mutări imposibile.
– În caz că un jucător va părăsi partida, jucătorul rămas va avea opțiunea de a juca în continuare împotriva calculatorului.



– La finalul jocului fiecare jucător va fi înștiințat dacă a câștigat sau a pierdut. Jucătorul învingător va introduce numele său pentru a fi salvat în clasament, nume ce trebuie să nu existe în clasament.





5 Concluzii

Proiectul poate fi îmbunătățit prin crearea unei interfețe grafice pentru clasament.

Bibliografie

1. Pentru instalarea bibliotecii SQLite:
https://www.tutorialspoint.com/sqlite/sqlite_installation.htm

2. Pentru instalarea bibliotecii SFML :
https://www.sfml-dev.org/tutorials/2.5/start-linux.php
3. https://profs.info.uaic.ro/~computernetworks/files/4rc_NivelulTransport_En.pdf
4. https://profs.info.uaic.ro/~computernetworks/files/5rc_ProgramareaInReteaI_en.pdf
5. https://docs.google.com/document/d/1rEZO1PwFMpYbu5WviAyPwOIjvsQTwETpGgMsxBL8Rb4/edit#heading=h.jq4sq8sou4pr
6. https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm
7. https://www.sfml-dev.org/learn.php
8. https://ro.wikipedia.org/wiki/SQLite
9. https://ro.wikipedia.org/wiki/SFML