Proiect - Chess - Raport Tehnic

Iov Alexandru-Constantin

Universitatea Alexandru Ioan Cuza

1 Introducere

1.1 Viziunea Generala

In cadrul acestui raport tehnic, voi prezenta proiectul ales la matera "Retele de Calculatoare", "Chess", grad de dificultate B. Viziunea generala a proiectului implica realizarea unei interfate de tip CLI (in terminal) ce permite jucarea unui joc de sah intre doi utilizatori, dar care ofera si o interfata grafica cu ajutorul careia jucatorii pot face miscarile. In acelasi timp, aplicatia permite jucarea mai multor jocuri simultane (e.g. daca sunt 2 jucatori deja implicati intr-un joc si la server se mai conecteaza 2 clienti, acestia vor juca un joc separat).

1.2 Obiectivele proiectului

Asa cum am mentionat mai sus, proiectul va permite jucarea unui joc de sah intre 2 persoane, fiind posibila rularea mai multor jocuri concurent. Proiectul va avea o interfata grafica, ce afiseaza tabla de joc actuala, permite miscarea pieselor si afiseaza statutul jucatorului (castigator/pierzator).

2 Tehnologii aplicate

La baza, proiectul va fi bazat pe o comunicare de tip TCP-IP. Motivul pentru care am ales TCP-IP in defavoarea UDP-ului este deoarece jocul de sah nu necesita o latenta mica a miscarilor, asa ca principalul avantaj al UDP-ului este irelevant. In schimb, siguranta TCP-IP-ului este mult apreciata.

Pentru a crea interfata grafica, proiectul utilizeaza libraria "raylib". Motivul pentru care am ales aceasta librarie in defavoarea altora este familiaritatea mea cu aceasta. In trecut, am mai facut proiecte in C cu ea, si sunt obisnuit cu sintaxa. Mai mult, este usoara de folosita si ofera un aspect placut interfatei grafice.

O alta tehnologie importanta folosita in proiect este functa fcntl(), utilizata pentru a face apelurile de read() si write() din client neblocante. Fara a fi neblocante, interfata grafica s-ar bloca.

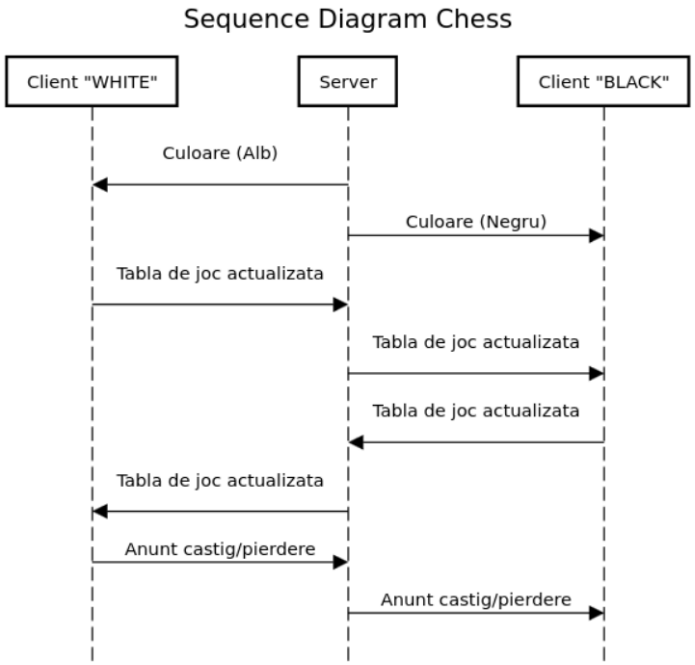

Fig. 1. Structura

3 Structura Aplicatiei

Structura de baza a aplicatiei este formata dintr-un server central, care are rol de a permite comunicarea intre clienti si doi client care participa la un joc de sah. Cei doi clienti au identitati diferite, si anume clientul de tip "WHITE" si clientul de tip "BLACK". Diferenta evidenta intre cei doi clienti sta in ordinea miscarilor de sah. Clientul "WHITE" va trimite si, implicit, va primi primul raspuns de la server, urmand ca, ulterior, clientul "BLACK" sa trimita si el informatiile, pastrand ordinea pana la finalul jocului.

De remarcat faptul ca sectiunea din mijloc ("Tabla de joc actualizata") se va repeta pana cand unul din clienti se afla in ipostaza castigatorului. De asemenea, anuntul de castig/pierdere poate veni, evident, din partea oricarui client. Nu neaparat din partea Clientului "WHITE".

La inceput, server-ul transmite printr-un apel write() culoarea fiecarui Client, in functia de ordinea de conectare la server. Apoi, cei doi trimit, pe rand, tabla actualizata cu miscarea lor (miscarile sunt validate in cadrul clientului, pentru a mentine marimea obiectului trimis permanent constant (o tabla de sah 8x8)).

4 Aspecte de Implementare

Bucati importante de cod:

```
int color = 0;
if (write(tdL.cl, &color, sizeof(color)) < 0)
{
  printf("[Thread %d]\n", tdL.idThread);
  perror("Eroare la write() la client.\n");
}
color++;

if (write(tdL.cl2, &color, sizeof(color)) < 0)
{
  printf("[Thread %d]\n", tdL.idThread);
  perror("Eroare la write() la client.\n");
}
```

Asociarea unei culori fiecarui client, facut la inceputul thread-ului.

```
  if (chessboard[0][0] == '0')
  {
    printf("Player has conceded.\n");
    //char buff[10] = "You win!";
    write(tdL.cl2, &chessboard, sizeof(chessboard));
    return;
  }
  else if (chessboard[0][0] == '1')
  {
    printf("Player has won.\n");
    //char buff[10] = "You lose!";
    write(tdL.cl2, &chessboard, sizeof(chessboard));
    return;
  }
```

Testul de castig: Tabla de 0 inseamna ca cel care citeste tabla a castigat, si este transmisa in momentul in care celalalt jucator a folosit comanda "concede". In schimb, tabla de 1 inseamna ca cel care citeste a pierdut, fiind trimisa in momentul in care celalalt jucator a capturat regele culorii opuse.

```
void drawMatrix(Texture2D *assets)
{
  for (int row = 0; row < BOARD_SIZE; row++)
  {
    for (int col = 0; col < BOARD_SIZE; col++)
    {
      // Alternate colors
      Texture2D piece;
      Color color = ((row + col) % 2 == 0) ? DARKGRAY : LIGHTGRAY;
      DrawRectangle(col * SQUARE_SIZE, row * SQUARE_SIZE, SQUARE_SIZE, SQUARE_SIZE, color);
```

```
      if (chessboard[row + 1][col + 1] != ' ')
      {
        switch (chessboard[row + 1][col + 1])
        {
        ... // alegerea de textura pentru piese
        }
        DrawTextureEx(piece, (Vector2){col * SQUARE_SIZE, row * SQUARE_SIZE},
        0.0f, 1.25f, WHITE);
        }
      }
  }
}
```

Functia care deseneaza piesele, asa cum se gasesc in matrice. Fiecarei piese ii corespunde o textura specifica, iar daca spatiul este gol, nu va fi selectata nici o textura.

```
    Vector2 getTile(Vector2 click)
        {
            Vector2 new = {floor(click.x / SQUARE_SIZE) + 1,
                           floor(click.y / SQUARE_SIZE) + 1};
            return new;
        }
```

Functie pentru a obtine patratul apasat de catre mouse.

Doua variabile importante pentru proiect sunt winFlag si loseFlag. Aceste doua variabile schimba modul de desenare a interfatei si afiseaza fie mesajul "You win", fie mesajul "You lose". In momentul in care clientul citeste o matrice plina de 1, atunci loseFlag este initializat. Daca clientul citeste o matrice de 0 sau se determina ca a facut o miscare castigatoare, atunci winFlag este initializat cu 1.

```
    if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON) && start.x != -9 && turn == 1)
        {
          flipBoard();
          int ok = 1;
          end = getTile(GetMousePosition());

          printf("Mouse button moved to tile %f,%f\n", end.x, end.y);
          fflush(stdout);

          int nr1 = (int)start.y;
          int nr2 = (int)start.x;

          int nr3 = (int)end.y;
          int nr4 = (int)end.x;
```

```
if (!isStartingPosValid(nr1, nr2, color))
{
  printf("Invalid move! Try again! (You're WHITES,
  you can only move uppercase pieces.) \n");
  ok = 0;
}
if (!isEndingPosValid(nr3, nr4, color))
{
  printf("Invalid move! You cannot capture your own pieces!\n");
  ok = 0;
}
if (!isValidMove(nr1, nr2, nr3, nr4))
{
  ok = 0;
}
if (ok)
{

  if (updateMatrix(nr1, nr2, nr3, nr4, sd) == -1)
  {
    winFlag = 1;
  }
  flipBoard();
  if (write(sd, &chessboard, sizeof(chessboard)) > 0)
  {
    printf("Wrote chessboard to server!\n");
  }
  printMatrix(color);
  turn = 0;

  start.x = -9;
  start.y = -9;
  end.x = -9;
  end.y = -9;
  count = 0;
}
else
{
  flipBoard();
  start.x = -9;
  start.y = -9;
  end.x = -9;
  end.y = -9;
}
```

```
}
if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON) && start.x == -9 &&
turn == 1)
{
  start = getTile(GetMousePosition());
}
```

Logica de input a miscarii cu ajutorul interfatei grafice. Mai intai, o casuta este apasata. Apoi, este apasata si a 2 a. In cazul in care miscarea este valida, aceasta este efectuata si devine randul celuilalt jucator. Daca miscarea este invalida, vectorii sunt restartati si jucatorul trebuie sa apese casute de miscare noi. Pentru clientul negru, apar si apeluri ale functiei flipBoard(), pentru a putea desena tabla invers.

Pentru a da un exemplu real, daca jucatorul alb incearca sa mute pionul alb la inceputul jocului cu 3 casute in fata, va apasa mai intai pe pion si vectorul start va fi initializat. Apoi va apasa cu 3 casute in fata, dar va esua deoarece regulile nu permit aceasta mutare. Apoi, utilizatorul va trebui sa aleaga in continuare perechi de cate 2 casute pana cand eventual va face o miscare valida.

5 Concluzii

O potentiala imbunatatire a sistemului propus este o metoda de detectare a deconectarii premature a unui client. In momentul de fata, daca primul client se deconecteaza inainte ca cel de-al doilea client sa intre in joc, al doilea client se va afla intr-o situatie nedefinita.

O alta imbunatatire ar fi permiterea clientilor sa isi aleaga oponentul. In momentul de fata, nu este posibila alegerea inamicului, pentru ca utilizatorii sunt grupati in ordinea in care intra in aplicatie. Prin implementarea unui sistem de "lobby-uri", fiecare client ar putea sa isi creeze propria camera de sah, unde ar putea intra, cu ajutorul unei parole, inamicul cu care acestia doresc sa se infrunte.

In ceea ce priveste domeniul retelelor, o potentiala imbunatatire ar putea fi folosirea select() - urilor in loc de read() si write() neblocant. Motivul este ca natura neblocanta a acestor apeluri este imprevizibila si poate duce usor la SegFault-uri.

O ultima imbunatatire este legata de faptul ca nu toate miscarile posibile in sah sunt implementate. Cele 3 miscari care lipsesc sunt rocada mica, rocada mare si miscarea "en passant" a pionului.

References

1. Cursul Retele de Calculatoare, model de server concurent cu thread-uri, https://edu.info.uaic.ro/computer-networks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c
2. Cursul Retele de Calculatoare, model de client, https://edu.info.uaic.ro/computer-networks/files/NetEx/S12/ServerConcThread/cliTcpNr.c