Proiect- Draughts

Luculescu Emanuel

Facultatea de Informatică, Universitatea “Alexandru Ioan Cuza” din Iași

1 Introducere

Acest proiect implementează o aplicație client-server pentru un joc de dame. Serverul gestionează mai mulți clienți folosind tehnici concurente și permite operarea a două variante de joc: dame pe o tablă de 8x8 și dame pe o tablă de 10x10. De asemenea, aplicația gestionează un clasament pentru fiecare variantă de joc. Clientul este dezvoltat cu o interfață grafică Qt pentru interacțiune intuitivă cu utilizatorii.

Obiective:

- Gestionarea mai multor clienți conectați simultan.
- Gestionarea jocului între 2 clienți conectați.
- Implementarea unui protocol de comunicare între client și server.
- Crearea unei interfețe grafice care să permită selectarea jocului, afișarea tablei și a clasamentului.

2 Tehnologii Aplicate

În acest proiect, comunicarea între client și server se realizează prin socket-uri TCP, oferind o conexiune bidirecțională fiabilă. De asemenea thread-urile nu lipsesc datorită faptului că trebuie să gestionăm multiple conexiuni, concurente. Mesajele sunt trimise sub formă de comenzi text, ușor de interpretat, în formatul (comandă?nume_parametru=valoare.)

Am ales să folosesc aceste tehnologii deoarece sunt cele mai potrivite pentru acest tip de aplicație de joc, deoarece oferă o fiabilitate, ordonare și eficientă în gestionarea datelor și a sesiunilor de joc. De asemenea am ales să folosesc QT pentru a prezența și o interfață grafică ce face jocul mult mai interactiv.

3 Structura Aplicației

Aplicația este construită pe baza modelului **client-server,** cele două mari componente.

Modelul client-server:

- Serverul: Gestionează lista clienților conectați, starea tablei de joc pentru fiecare variantă și clasamentele pentru fiecare variantă.
- Clientul: Care oferă utilizatorului opțiuni pentru: înregistrare cu un nume, selectarea unei variante de joc și vizualizarea clasamentului si a tablei. Clientul permite jucatorului selectat de server să joace varianta selectată.

Diagrama- Înregistrare client pentru joc:

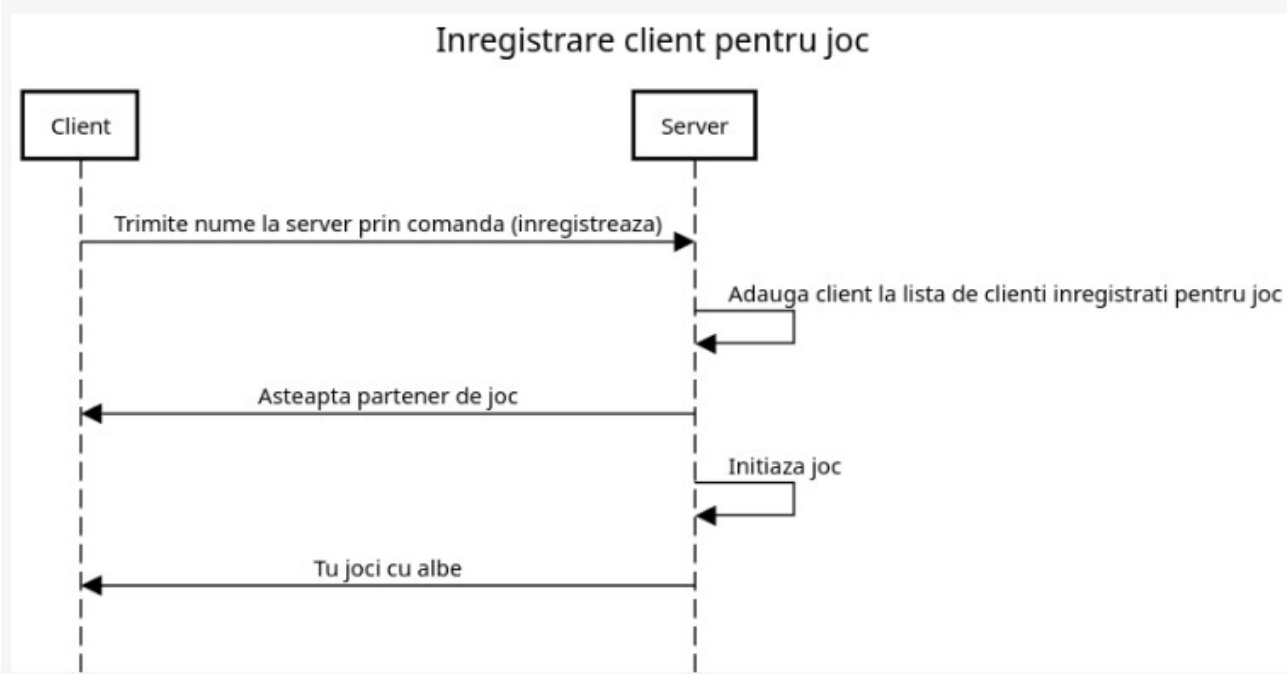


Diagrama- Structură joc

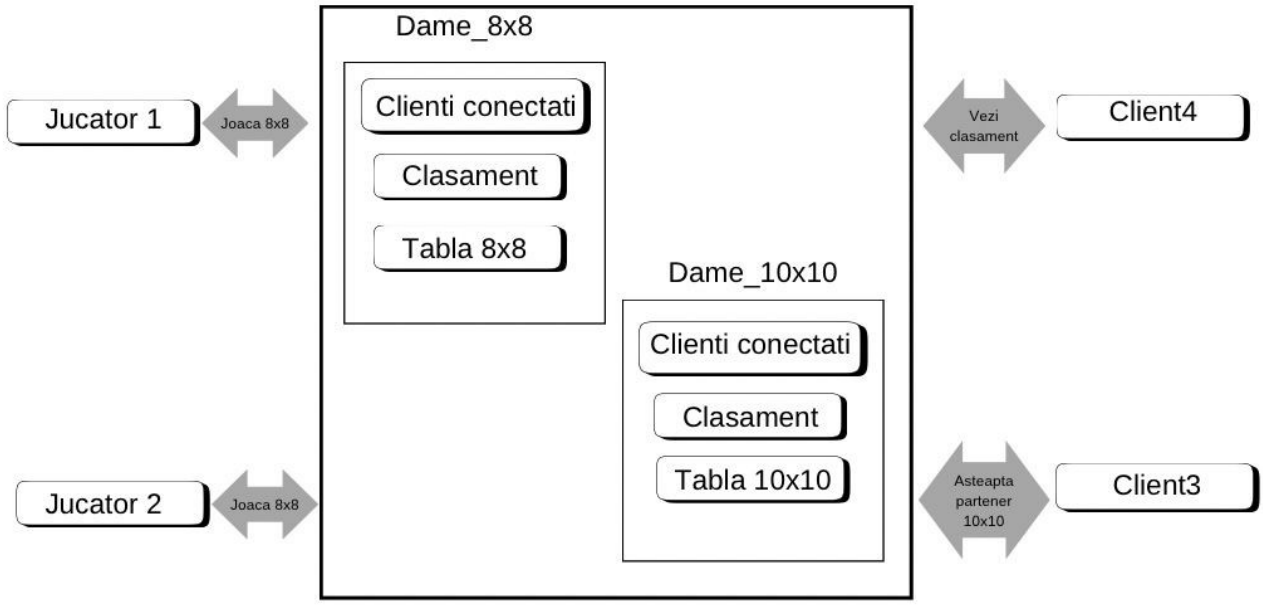


Diagrama → Client conectat neînregistrat sa joace:

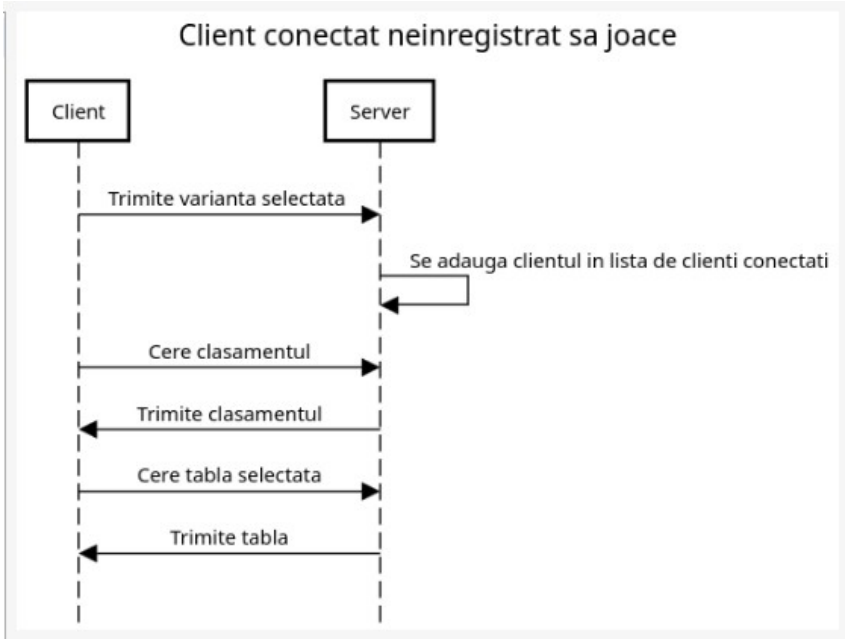
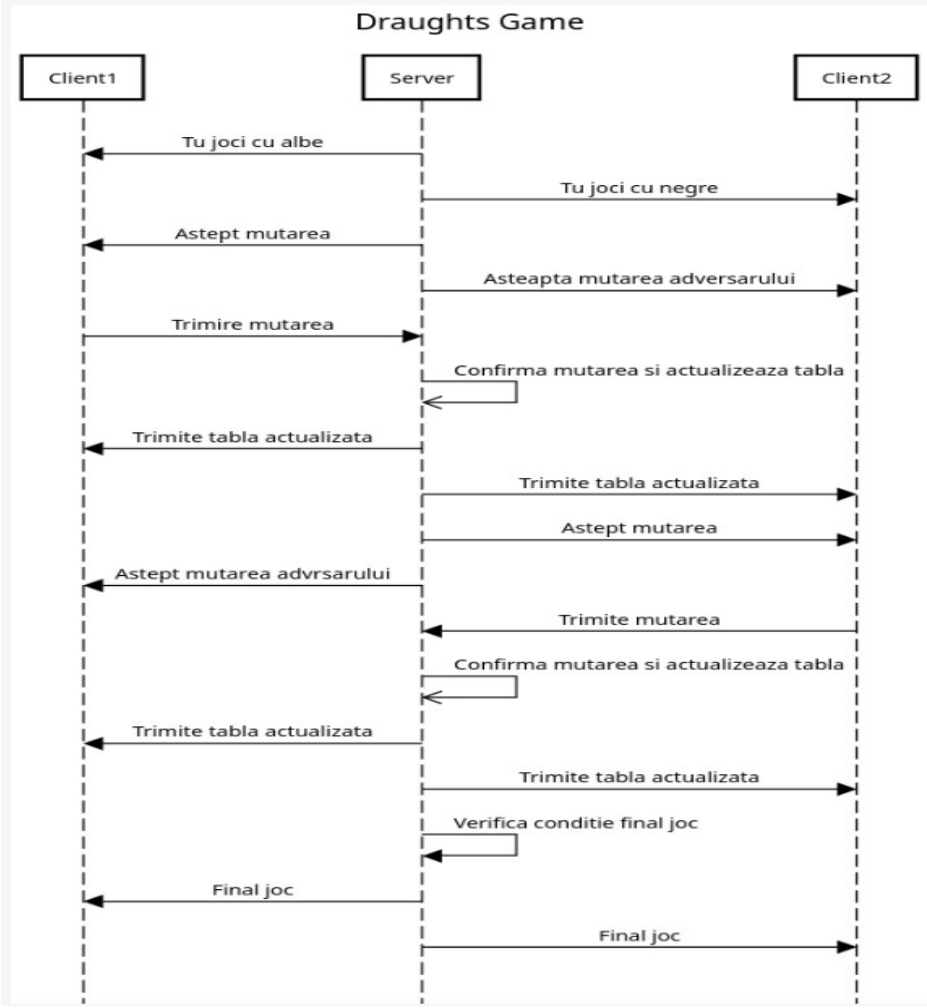


Diagrama → Joc



4. Aspecte de Implementare

Pe diverse actiuni din interfata client, ex: apasare buton, asamblam comenzi text cu parametri din context si le trimitem ca mesaje text la server.

La server, in funcția main() → initiaizeaza listening pe socket si creeaza threads pentru fiecare client.

```
if (listen(socket_fd, MAX_CONECTATI) < 0) {
    fprintf(log_file,"Eroare Listen pentru server socket a esuat\n");
    exit(EXIT_FAILURE);
}

fprintf(log_file,"Server pornit pe portul %d\n", PORT);

while (1) {
    int client_socket;
    if ((client_socket = accept(socket_fd, (struct sockaddr *)&addr, (socklen_t*)&addr_len)) < 0) {
        fprintf(log_file,"Eroare Accept conexiune a esuat\n");
        continue;
    }
    fprintf(log_file,"Conexiune acceptata %d\n", client_socket);
    pthread_t tid;
    if (pthread_create(&tid, NULL, gestiune_client, &client_socket) < 0) {
        fprintf(log_file,"Nu s-a putut crea thread-ul pentru %d\n", client_socket);
    } else {
        fprintf(log_file,"Handlerul client %d a fost asignat unui thread\n", client_socket);
        pthread_detach(tid);
    }
}
```

Pe evenimentul de show ne conectam la server si stocam file descriptor pt socket pentru comunicarea ulterioara cu serverul prin comenzi text.

```
void MainWindow::showEvent(QShowEvent *event) {
    event->accept();
    conectare_server();
}

void MainWindow::closeEvent(QCloseEvent *event) {
```

```
int conectare_server(){
    log_file = fopen("client_log.txt","w");
    setbuf(log_file,NULL);

    struct sockaddr_in servaddr;
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        fprintf(log_file,"Eroare socket respins\n");
        exit(EXIT_FAILURE);
    }
    fprintf(log_file, "Conectare client cu socket fd=%d\n", sockfd);
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    if (inet_pton(AF_INET, "127.0.0.1", &servaddr.sin_addr) <= 0) {
        fprintf(log_file,"Eroare Adresa invalida!\n");
        exit(EXIT_FAILURE);
    }
    if (connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        fprintf(log_file,"Eroare la conexiune\n");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

Iar pe evenimentul de close ne deconectam si eliberam resursele, inchidem socket si fisier de log.

```
void MainWindow::closeEvent(QCloseEvent *event) {
    event->accept();
    deconectare_server();
}
```

```
int deconectare_server() {
    fprintf(log_file,"Deconectare client cu socket fd=%d\n", sockfd);
    close(sockfd);
    fclose(log_file);
    return 0;
}
```

In client asamblam comenzi text si le trimitem ca mesaje la server.

```
void MainWindow::handle10x10Click() {
    strcpy(variantaJoc,"10x10");
    // Trimite varianta aleasa la server
    char sendBuffer[256];
    strcpy(sendBuffer, SET_VARIANTA_10x10_CMD);
    send(sockfd, sendBuffer, strlen(sendBuffer), 0);
    fprintf(logFile,"Am trimis la server comanda %s\n", sendBuffer);

    // Primeste tabla de la server
    char recvBuffer[1024];
    int recvSize = recv(sockfd, recvBuffer , sizeof(recvBuffer) , 0);
    if (recvSize > 0) {
        recvBuffer[recvSize] = '\0';
        Raspuns raspuns = parseaza_raspuns(recvBuffer);
        if (strcmp(TABLA_RSP, raspuns.tip_raspuns) == 0) {
            populeazaTabla(raspuns.date);
        } else {
            fprintf(logFile, "Nu am primit tabla de la server. Raspuns primit: %s\n", recvBuffer);
        }
    }
    // Primeste clasament de la server
    recvSize = recv(sockfd, recvBuffer , sizeof(recvBuffer) , 0);
    if (recvSize > 0) {
        recvBuffer[recvSize] = '\0';
        Raspuns raspuns = parseaza_raspuns(recvBuffer);
        if (strcmp(CLASAMENT_RSP, raspuns.tip_raspuns) == 0) {
            populeazaClasament(raspuns.date);
        } else {
            fprintf(logFile, "Nu am primit clasament de la server. Raspuns primit: %s\n", recvBuffer);
        }
    }
    numeField->setEnabled(true);
    inregistreazaButton->setEnabled(true);
}
```

In functia handleInregistreazaClick() se inregistreaza clientii si primesc mesaje in functie de caz.

```
            if (strcmp(IN_ASTEPTARE_RSP, raspuns.tip_raspuns) == 0) {
                mutareMesajLabel->setText("Așteaptă începutul unui nou joc.");
                serverPush->setEnabled(true);
            } else if (strcmp(INFO_JOC_RSP, raspuns.tip_raspuns) == 0) {
                populeazaStareJoc(raspuns.date);
                serverPush->setEnabled(true);
            } else if (strcmp("EroareNumeDuplicat", raspuns.tip_raspuns) == 0) {
                mutareMesajLabel->setText("Numele este deja utilizat. Alege alt nume.");
                numeField->setEnabled(true);
                inregistreazaButton->setEnabled(true);
            } else if (strcmp("EroareMaximClienti", raspuns.tip_raspuns) == 0) {
                mutareMesajLabel->setText("Numărul maxim de clienți a fost atins.");
                numeField->setEnabled(true);
                inregistreazaButton->setEnabled(true);
            } else {
                fprintf(logFile, "Răspuns neașteptat: %s\n", recvBuffer);
            }
        } else if (recvSize == 0) {
            fprintf(logFile, "Serverul a închis conexiunea\n");
        } else {
            fprintf(logFile, "Eroare la recepționarea datelor de la server\n");
        }
    } else {
        numeField->setEnabled(true);
        inregistreazaButton->setEnabled(true);
    }
}
```

In gestiune_client la server asteptam comenzi, le parsăm ca să extragem nume_comanda si parametri, apoi asamblam raspuns si il trimitem inapoi la client.

```c
void *gestiune_client(void *socket) {
    int socket_desc = *(int *)socket;
    char buffer[1024];
    char nume_client[50];

    while(1) {
        fprintf(log_file, "Asteptam de la client %d comanda\n", socket_desc);
        int recv_size = recv(socket_desc, buffer, sizeof(buffer), 0);
        if (recv_size > 0) {
            buffer[recv_size] = '\0';
            Comanda comanda = parseaza_comanda(buffer);
            fprintf(log_file, "Comanda parsata %s %s %s", comanda.nume,
            comanda.nume_parametru, comanda.valoare_parametru);
            sprintf(buffer, "Am parsat de la client comanda %s cu nume parametru %s si valoare parametru %s\n",
                    comanda.nume, comanda.nume_parametru, comanda.valoare_parametru);
            send(socket_desc, buffer, strlen(buffer), 0);
        } else {
            break;
        }

    }
    close(socket_desc);
    pthread_exit(NULL);
}
```

Serverul gestioneaza clientii conectati si le trimite mesaje precum: “Sa astepte conectarea altor jucatori” in cazul in care se doreste sa se joace. Informatii despre joc, culorile pieselor, tabla, etc.

```c
void gestiuneClientConectat(Dame* varianta_dame, int socket_desc) {
    if (varianta_dame->clienti_conectati_count == 1) {
        // Pentru primul client conectat trimit doar mesaj ca e pus in asteptare
        char raspuns[50];
        strcpy(raspuns, IN_ASTEPTARE_RSP);
        send(socket_desc, raspuns, sizeof(raspuns), 0);
        fprintf(log_file, "Am trimis la client %d raspuns %s\n", socket_desc, raspuns);
    } else if (varianta_dame->clienti_conectati_count >= 2) {
        // De al doilea client conectat, verificam daca avem jucatori
        if (varianta_dame->jucator1.nume_client[0] == '\0') {
            // Daca nu avem jucatori, alegem doi clienti conectati ca jucatori
            alege_jucatori(varianta_dame);
            // Toti clientii conectati primesc info cu starea noului joc
            char raspuns[1024];
            strcpy(raspuns, INFO_JOC_RSP);
            tipareste_stare_joc(raspuns, varianta_dame);
            for (int i = 0; i < MAX_CONECTATI; i++) {
                if (varianta_dame->clienti_conectati[i].socket_client > 0) {
                    send(varianta_dame->clienti_conectati[i].socket_client, raspuns, sizeof(raspuns), 0);
                    fprintf(log_file, "Am trimis la client %d raspuns %s\n", varianta_dame->clienti_conectati[i].socket_client, raspuns);
                }
            }
        } else {
            // Daca avem jucatori trimitem la noul client info joc cu jucatorii, piesele si cel care muta
            char raspuns_non_jucator[1024];
            strcpy(raspuns_non_jucator, INFO_JOC_RSP);
            tipareste_stare_joc(raspuns_non_jucator, varianta_dame);
            send(socket_desc, raspuns_non_jucator, sizeof(raspuns_non_jucator), 0);
            fprintf(log_file, "Am trimis la client %d raspuns %s\n", socket_desc, raspuns_non_jucator);
        }
    }
}
```

De asemenea, serverul preia comennzile trimise de clienti sub forma de string-uri si trimite inapoi clientului valoare comenzii trimise. De ex. Pentru “VARIANTA_CMD” cu parametrul “VARIANTA_PARAM_8X8” va trimite tabla de dame 8x8.

```
buffer[recv_size] = '\0';
Comanda comanda = parseaza_comanda(buffer);
if (strcmp(SET_VARIANTA_CMD, comanda.nume) == 0) {
    if (strcmp(VARIANTA_PARAM_8x8, comanda.parametri[0].valoare) == 0) {
        varianta_dame = &dame_8x8;
    } else if (strcmp(VARIANTA_PARAM_10x10, comanda.parametri[0].valoare) == 0) {
        varianta_dame = &dame_10x10;
    }
    // Pentru comanda Varianta se trimit la client tabla si clasamentul pentru varianta
    char raspuns[256];
    strcpy(raspuns, TABLA_RSP);
    tipareste_tabla(raspuns, varianta_dame);
    send(socket_desc, raspuns, sizeof(raspuns), 0);
    fprintf(log_file, "Am trimis la client %d raspuns %s\n", socket_desc, raspuns);
    memset(raspuns, 0, strlen(raspuns));
    strcpy(raspuns, CLASAMENT_RSP);
    tipareste_clasament(raspuns, varianta_dame->clasament);
    send(socket_desc, raspuns, sizeof(raspuns), 0);
    fprintf(log_file, "Am trimis la client %d raspuns %s\n", socket_desc, raspuns);
} else if (strcmp(INREGISTREAZA_CMD, comanda.nume) == 0) {
    // Protejam cu mutex pentru a nu alege jucatori simultan pe 2 threaduri diferite
    pthread_mutex_lock(&lock);
    // Pentru comanda inregistreaza se inregistreaza socket descriptor si nume client
    // la varianta aleasa si apoi trimit mesaje clientilor conectati
    int conectat = adauga_client_conectat(varianta_dame, comanda.parametri[0].valoare, socket_desc);
    if (conectat == 0) {
        gestiuneClientConectat(varianta_dame, socket_desc);
    }else if( conectat==-1){
        char raspuns[50];
        strcpy(raspuns,"EroareNumeDuplicat");
        send(socket_desc,raspuns,strlen(raspuns),0);
    }
    pthread_mutex_unlock(&lock);
} else if (strcmp(MUTARE_CMD, comanda.nume) == 0) {
    pthread_mutex_lock(&lock);//blocheaza mutex ul. Previne conflicte
```

Orice modificare a acestei structuri este protejata cu mutex, pentru a asigura consistenta datelor in contextul unor modificari concurente.

```
void adauga_client_conectat(ClientConectat *clienti_conectati,
int *clienti_conectati_count, int socket_client) {
    pthread_mutex_lock(&lock);
//  for (int i = 0; i < *clienti_conectati_count; i++) {
//      if (strcmp(clienti_conectati[i].nume_client, nume_client) == 0) {
//          pthread_mutex_unlock(&lock);
//          return 1; // Numele este deja utilizat
//      }
//  }
//  strcpy(clienti_conectati[*clienti_conectati_count].nume_client, nume_client);
    clienti_conectati[*clienti_conectati_count].socket_client = socket_client;
    (*clienti_conectati_count)++;
    pthread_mutex_unlock(&lock);
}
```

5 Concluzie

Acest proiect este unul complex, ce bineinteles mai poate fi imbunatatit. Au fost implementate concepte invatate in timpul orelor de curs/seminar. Aceasta aplicatie este de tip client-server si dezvolta jocul de dame intre clienti. Pentru a ne fi mult mai usor, au fost implementate si aspecte UI cu ajutoul framework-ului QT.

6 Bibliografie

1. Diagram: https://sequencediagram.org
2. Aspecte de implementare: https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php
3. QT: https://doc.qt.io/qt-6/qtwidgets-index.html
4. Phthread:https://man7.org/linux/man-pages/man3/pthread_create.3.html
5.Functions C/C++:https://www.geeksforgeeks.org/thread-functions-in-c-c/