Chess in LAN

15.01.2025

Îndrumător:

dr. ing. Daniel Morariu

Student:

Diță, Vlad-Ilie (23/2)

Istoric Versiuni

Data	Versiune	Descriere	Autor
03/12/2024	0.1	Crearea proiectului si planuirea claselor de baza	Diță Vlad-Ilie
04/12/2024	0.2	Un backend initial ce contine clasele de baza Board si Piece impreuna cu fiecare piesa si validare a mutarilor. Din clasa Game se poate „incepe” un joc si se initializeaza tabla	Diță Vlad-Ilie
05/12/2024	0.3	Un frontend initial ce contine o interfata de inceput (tabla si piesele) si o legatura cu backend-ul.	Diță Vlad-Ilie
06/12/2024	0.4	Dezvoltarea interfetei. Au fost adaugate Events ale caror actiuni ajung intr-un final la clasa Board si permite actiuni propriu-zise in joc. Mutarea pieselor este posibila.	Diță Vlad-Ilie
07/12/2024	0.5	Adaugarea unor features de tip „Quality of Life” pentru a imbunatati experienta utilizatorului (functia de hover peste patratele ce pot fi selectate, dupa selectia piesei sunt dezactivate toate patratele mai putin cele in care piesa poate fi mutata)	Diță Vlad-Ilie
10/01/2025	1.0	Jocul in faza initiala completa. Micile erori gasite au fost rezolvate, functii imbunatatie (din punct de	Diță Vlad-Ilie

		vedere al codului sau a eficientei) si jocul se poate juca pe rand	
11/01/2025	1.1	Reteaua a fost adaugata si este functionala. Jocul se poate juca de pe doua dispozitive diferite conectate la aceeasi retea	Diță Vlad-Ilie
12/01/2025	1.2	Rezolvarea micilor erori ale features de QoL aparute in urma adaugarii retelei, detectarea de Sah-Mat si retusuri de final	Diță Vlad-Ilie
14/01/2025	1.3	Mici imbunatatiri aditionale	Diță Vlad-Ilie, Bărglăzan Adrian

Cuprins

1 Specificarea cerințelor software

1.1 Introducere

Acest proiect ii permite utilizatorului sa joace un joc de sah in retea care include atat functionalitati de baza ale jocului cat si functionalitati aditionale pentru o experienta mai buna.

1.1.1 Obiective

Aplicatia propriu-zisa:

- Logica de baza a jocului de sah
- Regula piesei atinse
- Interfata simpla cu aspect placut
- Functionalitati de confort
- Retea
- Pawn promotion
- Reguli aditionale ale jocului (rocada, en passant)

Codul aplicatiei:

- Organizat
- Curat
- Usor de citit
- Eficient

1.1.2 Definiții, Acronime și Abrevieri

Pawn promotion: Regula a jocului de sah care presupune ca un pion ce a ajuns la capatul tablei va lua rol de regina, tura, cal sau nebun la alegerea jucatorului

Quality of Life (QoL): Functionalitati ale aplicatiei adaugate cu scopul de a face interfata mai usor de folosit, de a ajuta utilizatorul sau de a spori confortul la navigarea prin aplicatie

Camel Case: Convetie de denumire folosita in programare (ex: „camelCase”). A fost folosita pentru variabile si obiecte

Pascal Case: Convetie de denumire folosita in programare (ex: „PascalCase”). A fost folosita pentru clase, metode si proprietati

Prescurtari WinForms: Pentru elementele din ToolBox am folosit varianta prescurtata ca prefix (ex: button -> btn, textbox -> txt)

1.1.3 Tehnologiile utilizate

- Visual Studio Community 2022
- C# WinForms
- .NET Framework 4.7.2

1.2 Cerințe specifice

Backend:

- Logica jocului care cuprinde clasa Board si clasa abstracta Piece
- Fiecare tip de piesa are o clasa individuala care mosteneste clasa Piece. Metoda abstracta „GetValidMoves” este surpascrisa pentru fiecare piesa si returneaza o lista de pozitii pe care piesa are voie sa se mute, conform regulilor. Aceasta lista va fi folosita ulterior intr-o functionalitate prezenta in interfata
- Clasa Board sta la baza backend-ului. Aceasta contine o matrice de piesa care reprezinta tabla. La declararea unui obiect de tip Board se apeleaza constructorul care initializeaza tabla, adica populeaza matricea cu piese pe pozitiile corespunzatoare
- Clasa Board contine si metode folosite pentru validari precum IsSquareEmpty, IsOpponentSquare si IsInBounds
- In Board gasim si functia MoveTo care se ocupa de mutarea in matrice a piesei de pe o pozitie pe alta

Frontend:

- Interfata jocului si folosirea datelor primite din backend. Cuprinde StartForm si ChessForm
- StartForm este fereastra ce se afiseaza la pornirea aplicatiei. Contine un TextBox si doua Button prin care poti crea un joc sau te poti conecta la unul
- ChessForm apare dupa StartForm si este locul efectiv unde se afiseaza tabla de sah si piesele. Aici sunt facute retusurile de final ce tin de regulile jocului, cum ar fi regula piesei atinse, si

sunt adaugate functionalitatile de Quality of Life, cum ar fi Event-uile MouseEnter si MouseLeave pentru piesele pe care le poti muta, EnableSquares si DisableSquares in functie starea jocului si HighlightValidMoves

- In ChessForms tabla de sah este reprezentata ca un Panel al carei imagine de fundal este tabla. Peste acesta se afla o matrice de PictureBox care reprezinta fiecare patrat de pe tabla si i-a fost sau nu atribuita o imagine cu piesa corespunzatoare in functie de datele primite din backend

Legatura:

- Legatura dintre frontend si backend este clasa Game. Toate apelurile efectuate intre cele doua bucati de aplicatie si datele trec prin aceasta clasa ce functioneaza, la nivel superficial, similar unui Controller dintr-un API
- In momentul pornirii unui joc dupa ce trecem de fereastra StartForm, in ChessForm se initializeaza un obiect de tip Game care la randul lui initializeaza un obiect de tip Board si efectueaza procesul de intializare a inceputului de joc precum a fost explicat pe pagina anterioara
- In plus, clasa Game este cea in care se tine randul intr-o proprietate „CurrentTurn”, iar metoda „SwitchTurn” este apelata la finalul fiecare mutari

Retea:

- Codul de retea a fost scris in ChessForm. Exista o metoda „StartServer” care esta apelata daca ai apasat pe butonul „Host” din StartForm si o metoda „ConnectToServer” daca ai apasat pe butonul „Connect”. Reteaua a fost facuta cu TCP
- Metoda „ListenForMoves” are la baza „ThreadPool” care asteapta un semnal pe stream prin care se transmite piesa mutata de catre celalalt jucator. In urma interceptarii semnalului, se decodifica mesajul intr-un string care contine coordonatele X si Y separate printr-o virgula ale piesei mutate si cele ale locului in care a fost mutata, separate prin doua puncte (ex: 0,1:0,3). Aceasta actiune va fi executata folosind metoda „Invoke” urmata de „new Action”

2 Crearea Tablei

2.1 Descriere

Crearea tablei sta la baza jocului de sah si este prezenta atat in backend cat si in frontend. Fara aceasta functionalitate nu ai avea cum sa joci jocul

2.2 Fluxul de evenimente

2.2.1 Crearea Tablei

Tabla de joc este creata in urma initializarii obiectului Board in clasa Game. Constructorul explicit si fara parametri al clasei Board apeleaza metoda „InitializeBoard" care la randul ei contine cateva alte metode ce initializeaza fiecare tip de piesa.

```
13 references
public class Board
{
    private Piece[,] Grid = new Piece[8, 8];

    1 reference
    public Board()
    {
        IntializeBoard();
    }

    1 reference
    public void IntializeBoard()
    {
        InitializePawns();
        InitializeRooks();
        InitializeKnights();
        InitializeBishops();
        InitializeQueens();
        InitializeKings();
    }
```

Aceasta initializare inseamna ca proprietatea Grid, matricea de tip Piece, al clasei Board este populata pe pozitiile corespunzatoare cu obiecte derivate din clasa Piece ce reprezinta fiecare piesa in parte pentru ambele culori. La crearea unui obiect derivat din Piece se precizeaza pozitia in care se afla piesa ca Point si culoarea acesteia.

```
1 reference
private void InitializeKings()
{
    Grid[4, 0] = new King(new Point(4, 0), PlayerColor.Black);
    Grid[4, 7] = new King(new Point(4, 7), PlayerColor.White);
}
```

Initializarea tuturor pieselor este facuta in acelasi mod ca initializarea regilor din imaginea de mai sus cu exceptia pionilor unde a fost folosit un for-loop unde pozitia a fost de tip [i,1].

Ulterior in clasa ChessForm incepe initializarea tablei. Un Panel este creat care primeste ca imagine de fundal tabla, iar peste se creeaza o matrice de PictureBox-uri lipsita de imagini. Acestea primesc doar cateva date legate de proprietatile lor ce includ dimensiunea si pozitia si sunt dezactivate. Apoi sunt atribuite Panel-ului.

```
PictureBox piecePictureBox = new PictureBox
{
    Size = new Size(squareSize, squareSize),
    Location = new Point(i * squareSize, j * squareSize),
    BackColor = Color.Transparent,
    SizeMode = PictureBoxSizeMode.StretchImage,
    Tag = new Point(i, j),
    Enabled = false
};

piecePictureBox.MouseClick += Piece_MouseClick;

PanelBoard.Controls.Add(piecePictureBox);
PiecePictureBoxes[i, j] = piecePictureBox;
```

In final matricea din clasa Board este apelata in ChessForms care intr-un for-loop compara fiecare pozitie din matricea tablei cu fiecare pozitie din matricea imaginilor, iar cele in care sunt gasite piese, adica nu sunt null, inca faptul ca acel PictureBox trebuie sa contina o imagine. In acest caz se apeleaza o metoda care primeste ca parametru un obiect de tip Piece ce reprezinta piesa din matricea tablei si verifica tipul clasei derivate pe baza caruia va returna imaginea din folder-ul Resources ce va fi atribuita PictureBox-ului.

```
1 reference
private Image GetPieceImage(Piece piece)
{
    PlayerColor color = piece.GetColor();

    if (piece is Pawn)
        return color == PlayerColor.White ? Properties.Resources.Pawn_White : Properties.Resources.Pawn_Black;
    if (piece is Rook)
        return color == PlayerColor.White ? Properties.Resources.Rook_White : Properties.Resources.Rook_Black;
    if (piece is Knight)
        return color == PlayerColor.White ? Properties.Resources.Knight_White : Properties.Resources.Knight_Black;
    if (piece is Bishop)
        return color == PlayerColor.White ? Properties.Resources.Bishop_White : Properties.Resources.Bishop_Black;
    if (piece is Queen)
        return color == PlayerColor.White ? Properties.Resources.Queen_White : Properties.Resources.Queen_Black;
    if (piece is King)
        return color == PlayerColor.White ? Properties.Resources.King_White : Properties.Resources.King_Black;

    return null;
}

1 reference
private void InitializePieceImages()
{
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            Piece piece = Game.GetPieceAtSquare(new Point(i, j));
            PictureBox piecePictureBox = PiecePictureBoxes[i, j];

            if (piece != null)
            {
                piecePictureBox.Image = GetPieceImage(piece);
            }
            else
            {
                piecePictureBox.Image = null;
            }
        }
    }
}
```

In contextul retelei, cel ce s-a conectat nu va putea face nimic deoarece nu este randul lui, iar toate PictureBox-urile sunt dezactivate implicit la creare, iar host-ul are activate doar piesele pe care are voie sa le mute. Din acest moment poate incepe jocul.

2.2.2 Pre-condiții

Pentru ca utilizatorul sa ajunga in aceasta faza trebuie sa porneasca programul si sa apese pe butonul „Host Game” sau „Connect”. Dupa ce ambii jucatori s-au conectat la retea, initializarea jocului va avea loc si in interfata, iar alb (host-ul) va incepe.

2.2.3 Post-condiții

In acest moment tabla este deja creata. Tot ce mai au de facut jucatorii este sa se bucure de un joc amical de sah.

3 Validarea Mutarilor

3.1 Descriere

Validarea mutarilor asigura ca piesa este mutata corect. De asemenea, interfata ajuta jucatorul sa efectueze o mutare corecta a piesei prin indicii vizuale prezente pe tabla dupa selectarea piesei.

3.2 Fluxul de evenimente

3.2.1 Validarea Mutarilor

In aceasta aplicatie validarea este facuta dupa selectarea piesei, dar inainte mutarii ei. Motivul acestui lucru va fi inteles pe parcursul explicatiei.

Dupa alegerea piesei, metoda „GetValidMoves” a piesei este apelata. In aceasta metoda se verifica daca pozitiile pe care s-ar putea muta o piesa sunt permise, iar cele care trec verificarile sunt adaugate intr-o lista numita „validMoves”. La finalul verificarilor se returneaza aceasta lista. Validarea de mutari este specifica fiecarei piese, insa tipul lor poate fi impartit in doua categorii ce vor fi explicate in sectiunea 3.2.2.

Este de mentionat ca in acest moment, dupa selectarea piesei, toate patratele de pe tabla sunt dezactivate si matricea de PictureBox-uri asteapta un nou Click Event care reprezinta pozitia pe care va fi mutata piesa.

ChessForm primeste lista cu mutari valide si o parcurge in metoda „HighlightValidMoves” care va activa si colora distinctiv doar PictureBox-urile de pe pozitiile ce apartin listei.

```
1 reference
private void HighlightValidMoves(Piece piece)
{
    List<Point> validMoves = piece.GetValidMoves(Game.GetBoard());

    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            PictureBox square = PiecePictureBoxes[i, j];
            Point squareLocation = new Point(i, j);

            if (validMoves.Contains(squareLocation))
            {
                square.Enabled = true;
                square.BackColor = Color.FromArgb(128, Color.LightBlue);
            }
        }
    }
}
```

In acest mod jucatorul va putea vedea pe tabla toate patratele pe care piesa selectata poate fi mutata, iar acele patrate sunt singurele pe care are voie sa dea click. In imaginea de mai jos putem observa mutarile posibile pentru calul alb.

Validarea mutarilor in acest fel nu doar ca imbunatateste interfata, experienta utilizatorului si in esenta este o functionalitate de tipul „Quality of Life”, dar imi asigura in acelasi timp ca jucatorul va face mereu o mutare corecta. Prin implementarea acestei functionalitati am eliminat cu succes un factor ce ar fi putut afecta rularea jocului in parametri optimi.

3.2.2 Fluxuri alternative

3.2.2.1 Validarea Mutarilor pe Pozitie

Agloritmul de validare a mutarilor este putin diferit in functie de modul in care se misca piesa. Piesele ce le-am numit eu ca „piese ce se muta pe pozitie" sunt cele care se misca un singur patrat sau nu tin cont de prezenta unui obstacol pana la acel patrat. Un bun exemplu este calul care spre deosebire de majoritatea pieselor nu baga de seama daca exista vreo piesa in calea lui. Singurele criterii care trebuie sa fie indeplinite pentru ca el sa se mute pe un patrat sunt sa fie pe tabla si sa nu se afle o piesa de aceeasi culoare pe acel patrat.

```
Point jump = new Point(Position.X + direction.X, Position.Y + direction.Y);
if (board.IsInBounds(jump) && (board.IsSquareEmpty(jump) || board.IsOpponentSquare(jump, Color)))
{
    validMoves.Add(jump);
}
```

Desi se muta diferit, mutarile pionului si al regelui sunt validate indentic datorita faptului ca se pot muta un singur patrat.

3.2.2.2 Validarea Mutarilor pe Directie

Celalalt algoritm este pentru piesele ce le-am numit ca „piese ce se muta pe directie". Acestea sunt cele care se pot muta pe o succesiune de patrate pana intalnesc marginea tablei, o piesa a adversarului sau o piesa de aceeasi culoare. Pentru aceste piese nu mai este suficient sa verific daca pozitia pe care mut este valida, ci trebuie sa verific si daca toate pozitile in drumul sau sunt valide la randul lor.

```
while (true)
{
    currentPosition = new Point(currentPosition.X + direction.X, currentPosition.Y + direction.Y);
    if (!board.IsInBounds(currentPosition))
    {
        break;
    }

    if (!board.IsSquareEmpty(currentPosition))
    {
        if (board.IsOpponentSquare(currentPosition, Color))
        {
            validMoves.Add(currentPosition);
        }

        break;
    }

    validMoves.Add(currentPosition);
}
```

Acest mod de validere este folosit de catre regina, tura si nebun.

3.2.3 Pre-condiţii

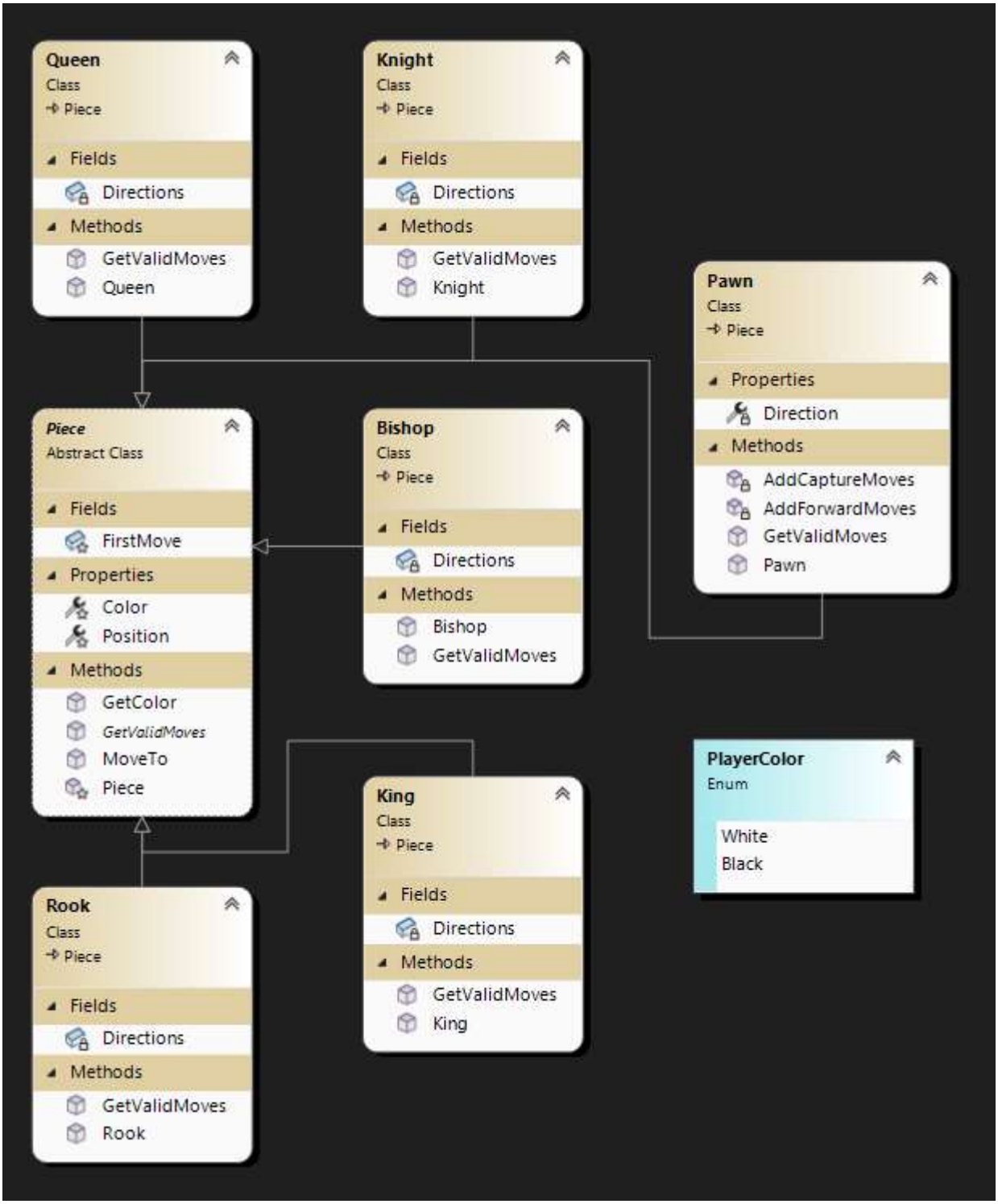
Aceasta functionalitate este accesibila dupa initializarea tablei, explicata mai sus, si cand este randul jucatorului. In prima faza acesta va trebui sa selecteze o piesa, iar apoi sa selecteze unul dintre patratele colorate diferit.

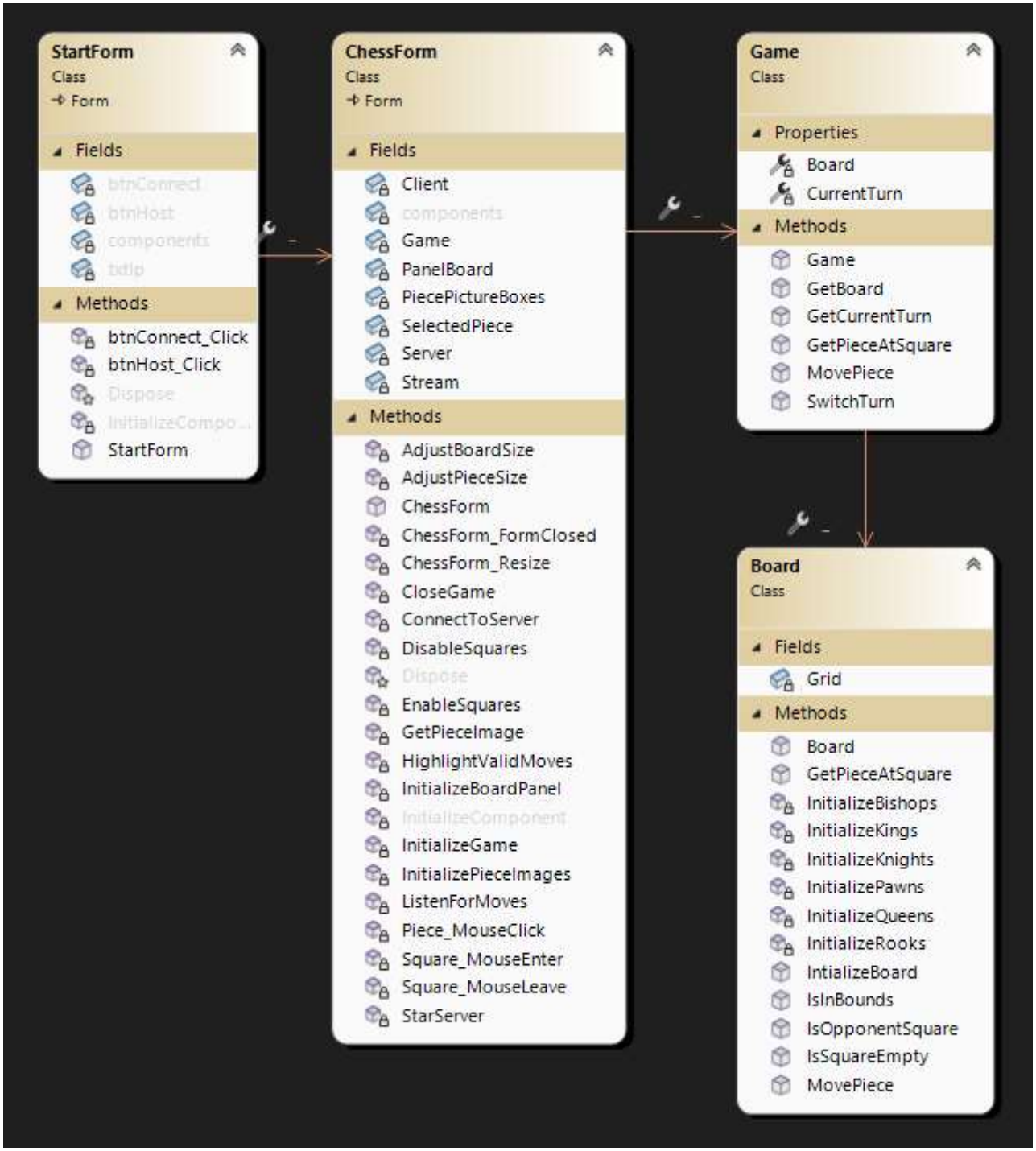
3.2.4 Post-condiţii

Dupa selectarea piesei, toate patratele vor fi dezactivate mai putin cele pe care piesa poate fi mutata. Patratele active sunt colorate cu o nuanta de albastru diferita de restul tablei. Utilizatorul selecteaza unul dintre aceste patrate, iar apoi va observa ca piesa a fost mutata pe acel patrat.

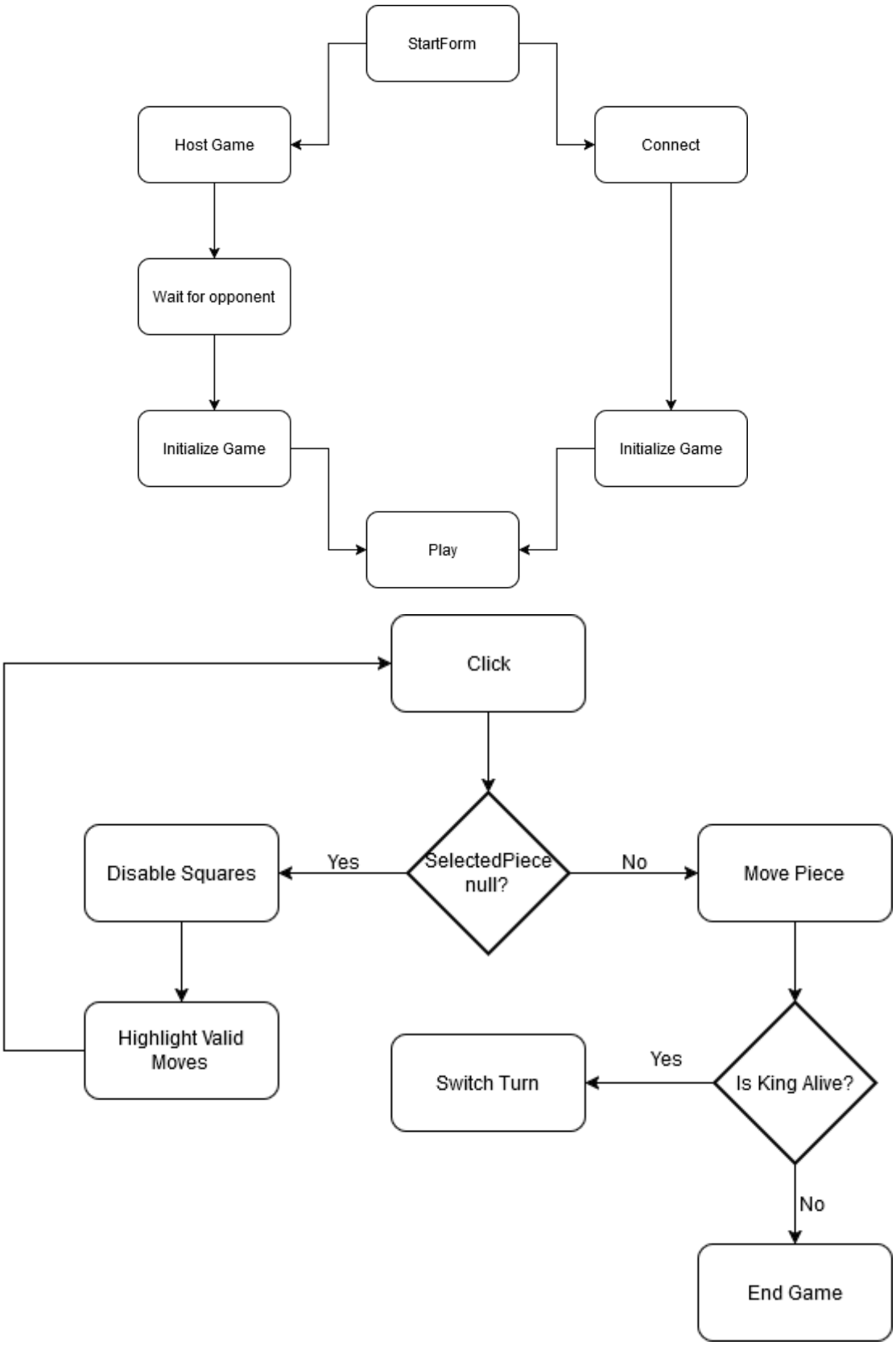
4 Implementare

4.1 Diagrama de clase




StartForm
Class
Form
Fields
btnConnect
btnHost
components
tdip
Methods
btnConnect_Click
btnHost_Click
Dispose
InitializeCompo...
StartForm
ChessForm
Class
Form
Fields
Client
components
Game
PanelBoard
PiecePictureBoxes
SelectedPiece
Server
Stream
Methods
AdjustBoardSize
AdjustPieceSize
ChessForm
ChessForm_FormClosed
ChessForm_Resize
CloseGame
ConnectToServer
DisableSquares
Dispose
EnableSquares
GetPieceImage
HighlightValidMoves
InitializeBoardPanel
InitializeComponent
InitializeGame
InitializePieceImages
ListenForMoves
Piece_MouseClick
Square_MouseEnter
Square_MouseLeave
StarServer
Game
Class
Properties
Board
CurrentTurn
Methods
Game
GetBoard
GetCurrentTurn
GetPieceAtSquare
MovePiece
SwitchTurn
Board
Class
Fields
Grid
Methods
Board
GetPieceAtSquare
InitializeBishops
InitializeKings
InitializeKnights
InitializePawns
InitializeQueens
InitializeRooks
IntializeBoard
IsInBounds
IsOpponentSquare
IsSquareEmpty
MovePiece

4.2 Descriere detaliată



5 Bibliografie

https://learn.microsoft.com/en-us/dotnet/desktop/winforms/windows-forms-overview?view=netframeworkdesktop-4.8

https://stackoverflow.com/

https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/sockets/socket-services

https://www.youtube.com/results?search_query=c%23+sockets

https://chatgpt.com/

https://app.diagrams.net/#

https://www.chess.com/learn-how-to-play-chess