Universitatea Tehnică “Gheorghe Asachi” din Iași
Facultatea de Automatică și Calculatoare
Domeniul Calculatoare și Tehnologia Informației
Specializarea Tehnologia Informației

INTELIGENȚĂ ARTIFICIALĂ

Implementarea jocului "Connect 4" cu ajutorul algoritmului minimax cu retezarea alfa-beta

Coordonator,
Prof. dr. Florin Leon

Studenți,

Bîzgan Alina, 1410B
Modiga Camelia-Maria, 1410B
Ștefan Alexandra, 1410B

An universitar 2021-2022

Cuprins

1. Descrierea problemei considerate

Connect Four (cunoscut și sub numele de Four Up , Plot Four sau Find Four) este un joc cu doi jucători în care jucătorii aleg o culoare și apoi aruncă pe rând discuri colorate într-o grilă suspendată vertical cu șapte coloane și șase rânduri. Fiecare jucător are inițial un număr egal de piese (21) pe care să le arunce pe rând din partea de sus a tablei. Apoi, ei vor juca pe rând și cel care face o linie dreaptă fie vertical, orizontal sau diagonal câștigă.

2. Aspecte teoretice privind algoritmul

2.1. Algoritmul minimax

Algoritmului minimax este un algoritm recursiv pentru a găsi cea mai bună mișcare într-o situație dată. Algoritmul minimax constă dintr-o funcție de evaluare pozițională care măsoară bunătatea unei poziții (sau a stării de joc) și indică cât de dorit este ca jucătorul dat să atingă acea poziție; jucătorul face apoi mișcarea care minimizează valoarea celei mai bune poziții atinse de celălalt jucător.

Cei doi jucători sunt numiți maximizant și minimizant. Maximizantul încearcă să obțină cel mai mare scor posibil, în timp ce minimizantul încearcă să obțină cel mai mic scor posibil. Dacă asociem fiecărei table de joc un scor de evaluare, atunci unul din jucători încearcă să aleagă o mutare care să îi maximizeze scorul, iar celălalt alege o mutare care are un scor minim, încercând să contra-atace.

Se presupune că jocul este de sumă nulă, deci se poate folosi o singură funcție de evaluare pentru ambii jucători. Dacă $f(n) > 0$, poziția n este bună pentru calculator și rea pentru om, iar dacă $f(n) < 0$, poziția n este rea pentru calculator și bună pentru om. Se construiește arborele până la limita de adâncime.

Se calculează funcția de evaluare pentru frunze și se propagă evaluarea în sus, selectând minimele pe nivelul minimizant (decizia omului) și maximele pe nivelul maximizant (decizia calculatorului).

2.2. Retezarea alfa-beta

O îmbunătățire substanțială a Minimax este Alpha-beta pruning (tăiere alfa-beta). Acest algoritm încearcă să optimizeze Minimax profitând de o observație importantă: pe parcursul examinării arborelui de soluții se pot elimina întregi subarbori, corespunzători unei mișcări m, dacă pe parcursul analizei găsim că mișcarea m este mai slabă calitativ decât cea mai bună mișcare curentă.

Parametrul alfa reprezintă cea mai bună valoare pe care maximizantul o poate garanta la nivelul curent sau la nivelul următor, iar parametrul beta reprezintă cea mai bună valoare pe care minimizantul o poate garanta la nivelul curent sau la nivelul următor.

Căutarea alfa-beta înlocuiește valorile lui alfa și beta pe măsură ce traversează algoritmul și elimină ramurile rămase ale unui nod atunci când valoarea nodului curent este sigur mai nefavorabilă decât valorile curente ale lui alfa și beta.

Astfel, considerăm că pornim cu o primă mișcare M1. După ce analizăm această mișcare în totalitate și îi atribuim un scor, continuăm să analizăm mișcarea M2. Dacă în analiza ulterioară găsim că adversarul are cel puțin o mișcare care transformă M2 într-o mișcare mai slabă decât M1 atunci orice alte variante ce corespund mișcării M2 (subarbori) nu mai trebuie analizate.

3. Modalitatea de rezolvare

În prezentul proiect algoritmul minimax este setat pentru un joc cu doi jucători: omul și computerul, jucătorul uman fiind cel care maximizează și, respectiv, computerul fiind minimizatorul. Această terminologie se datorează faptului că jucătorul uman își maximizează șansa de a câștiga, în timp ce jucătorul pe computer minimizează această șansă. Pentru ca minimax să se desfășoare eficient, valorile scorurilor trebuie să fie adaptate fiecărei poziții specifice a tablei și trebuie să se adapteze la orice stare/turnă dată de joc.

Definim următoarele variabile:

- alfa: este cea mai bună valoare pe care maximizantul o poate garanta la nivelul curent sau la nivelul următor;
- beta: este cea mai bună valoare pe care minimizantul o poate garanta la nivelul curent sau la nivelul următor.

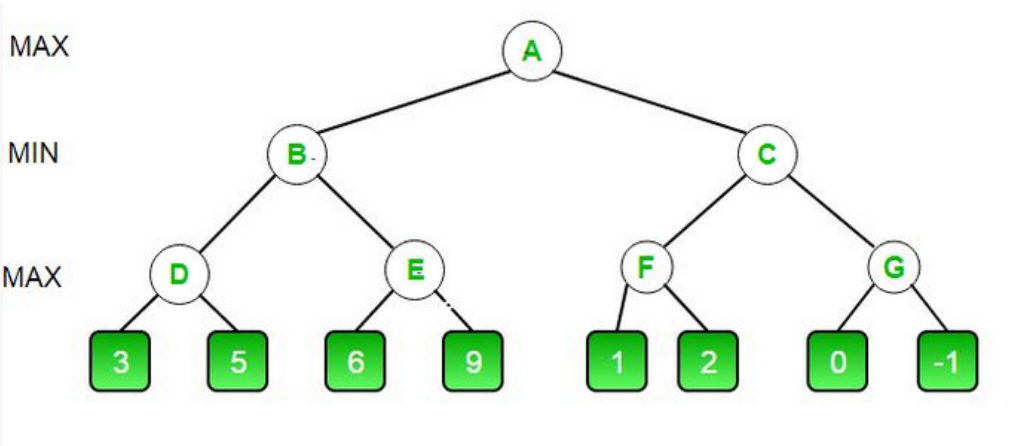

Figura 1. Arborele inițial

Apelul inițial începe de la A. Valoarea lui alfa aici este minus infinit și valoarea beta este plus infinit. În nodul A, maximizantul trebuie să aleagă maximul dintre B și C, așa că îl alege pe B, iar B alege nodul D. Nodul D se uită la copilul său drept, care are valoarea 3. Acum valoarea lui alfa la D este max(-INF, 3) care este 3.

Pentru a decide dacă merită să se uite la nodul său drept sau nu, verifică condiția beta<=alfa. Acest lucru este fals deoarece beta = +INF și alfa = 3. Deci continuă căutarea.

D acum se uită la copilul său drept, care returnează o valoare de 5. La D , alfa = max(3, 5) care este 5. Acum valoarea nodului D este 5. D returnează o valoare de 5 la B . La B, beta = min(+INF, 5) care este 5. Minimizantul este acum garantat de o valoare de 5 sau mai mică. B verifică nodul E pentru a vedea dacă poate obține o valoare mai mică decât 5. La E valorile alfa și beta nu sunt -INF și +INF, ci în schimb -INF și respectiv 5, deoarece valoarea beta a fost schimbată la B și asta este ceea ce B a transmis la E. Acum E se uită la copilul său stâng care este 6. La E, alfa = max(-INF, 6) care este 6. Aici condiția devine adevărată. beta este 5 și alfa este 6. Deci beta<=alpha este adevărat. Prin urmare, E întoarce 6 la B. La B, beta = min(5, 6) care este 5. Valoarea nodului B este de asemenea 5.

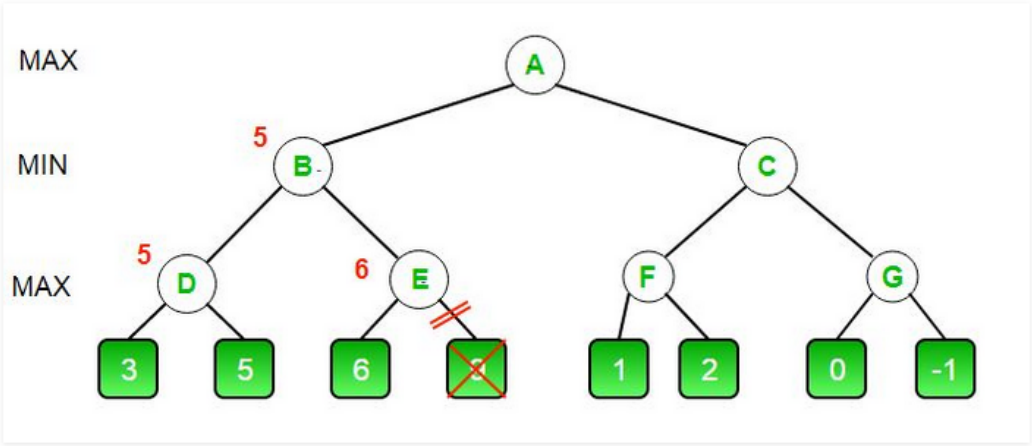

Figura2. Arborele după calcularea valorilor pentru nodurile E, D și B

În continuare se respectă aceeași pași pentru calcularea valorilor pentru fiecare nod până se ajunge la următoarea structură arborescentă.

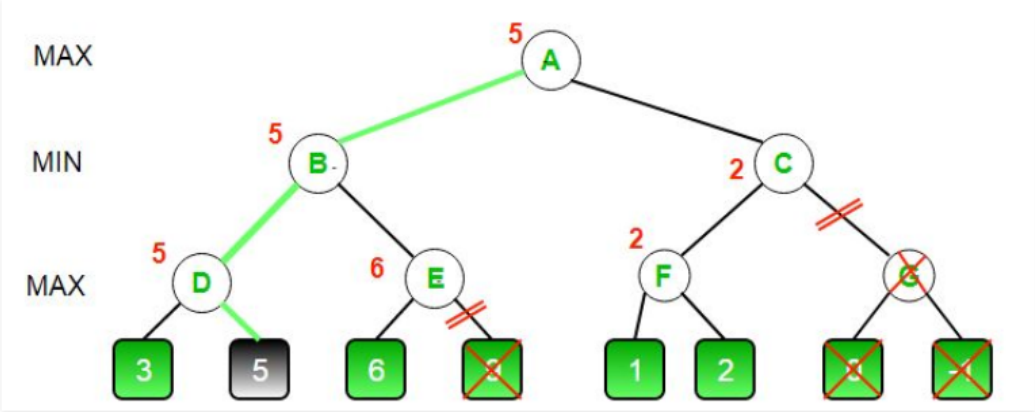

Figura 3. Arborele final

4. Părți semnificative din codul sursă

4.1. Funcția de evaluare

Funcția de evaluare implementată pentru minimax, limitată de cele 3 nivele, calculează valoare de utilitate care va fi folosită de către algoritm pentru a determina următoarea mișcare, care va fi cea mai bună posibilă.

Calculatorul alege cea mai mare valoare utilă pe care o găsește în timpul calculului și alege coloana corespunzătoare pentru următoarea mișcare.

Verifică, în primul rând dacă poate obține patru bile consecutive, apoi trei și apoi două. Se verifică numărul de bile pe care le poate face jucătorul și după ce termină calculele, alege coloana care are cea mai mare valoare corespunzătoare și face mutarea.

```
private int functieEvaluare(int[,] tabla)
{
    var castigator = verificaCastigator(tabla);
    string s;
    // dacă avem un castigator
    if (castigator != -1)
    {
        // și castigatorul este jucătorul
        if (castigator == 1)
            return int.MinValue;
        else
            return int.MaxValue;
    }

    var nr = 0;

    // calculeaza ponderea pe orizontală
    for (var i = 0; i < L; i++)
    {
        for (var j = 0; j <= C - 4; j++)
        {
            s = $"{tabla[i, j]}{tabla[i, j + 1]}{tabla[i, j + 2]}{tabla[i, j +
3]}";
            nr = calculPondere(s, nr);
        }
    }

    // calculeaza ponderea pe verticală
    for (var i = 0; i <= L - 4; i++)
    {
        for (var j = 0; j < C; j++)
        {
            s = $"{tabla[i, j]}{tabla[i + 1, j]}{tabla[i + 2, j]}{tabla[i + 3,
j]}";
            nr = calculPondere(s, nr);
        }
    }
```

```
    // calculează ponderea pe diagonala spre dreapta (\)
    for (var i = 0; i <= L - 4; i++)
    {
        for (var j = 0; j <= C - 4; j++)
        {
            s = $"{tabla[i, j]}{tabla[i + 1, j + 1]}{tabla[i + 2, j + 2]}{tabla[i +
3, j + 3]}";
            nr = calculPondere(s, nr);
        }
    }

    // calculeaza ponderea pe diagonala spre stanga (/)
    for (var i = 0; i <= L - 4; i++)
    {
        for (var j = 3; j < C; j++)
        {
            s = $"{tabla[i, j]}{tabla[i + 1, j - 1]}{tabla[i + 2, j - 2]}{tabla[i +
3, j - 3]}";
            nr = calculPondere(s, nr);
        }
    }
    return nr;
}
```

4.2. Calcularea următoarei poziții a calculatorului

Metoda calculeaza următoarea poziție a calculatorului în funcție de pozițiile libere găsite pe tabla de joc. La fiecare mutare a calculatorului se apelează algoritmul minimax cu retezare alfa-beta pentru a se asigura că se alege poziția optimă.

```
public void mutareCalculator()
{
    // algoritmul Minimax se apelează după fiecare mutare a jucătorului
    var mutare = algoritmMinimax(tabla, 0, 2, int.MinValue,
int.MaxValue);

    if (mutare.Item1 != -1 && mutare.Item1 != -2)
    {
        for (var i = L - 1; i >= 0; --i)
        {
            // dacă există spațiu pe tablă, atunci se pune piesa (pe
primul loc liber)
            if (tabla[i, mutare.Item1] == 0)
            {
                tabla[i, mutare.Item1] = 2;
                break;
            }
```

```
        }
    }
}
```

4.3. Algoritmul minimax cu retezare alfa-beta

```
private (int, int) algoritmMinimax(int[,] tabla, int adancime, int
utilizator, int alfa, int beta)
{
   var castigator = verificaCastigator(tabla);

   // adâncimea arborelui este 3
   // condiția de terminare
   // am ajuns la nodul frunza
   if (adancime == 3)
       return (-1, functieEvaluare(tabla));
   // dacă există un câștigător
   else if (castigator != -1)
       // și câștigătorul este jucătorul
       if (castigator == 1)
           // jucătorul devine minimizant
           return (-1, int.MinValue);
       else
           // altfel devine maximinzant
           return (-1, int.MaxValue);
   else
   {
       var mutari = mutareValida(tabla);
       // dacă nu mai avem poziții libere pe tabla de joc se returnează
-2
       if (mutari.Count == 0)
           return (-2, functieEvaluare(tabla));
       else
       {
           // dacă nodul este maxim
           // adică utilizatorul este reprezentat de calculator
           if (utilizator !=1)
           {
               var scorOptim = int.MinValue;
               var mutareOptima = -1000;

               for (var i = 0; i < mutari.Count; i++)
               {
                   var aux = mutareNoua(tabla, mutari[i], utilizator);
                   var mutare = algoritmMinimax(aux, adancime + 1, 1,
alfa, beta);
                   if (mutare.Item2 >= scorOptim)
                   {
```

```
                        scorOptim = mutare.Item2;
                        mutareOptima = mutari[i];
                    }

                    alfa = Math.Max(alfa, scorOptim);
                    // Condiția necesară pentru retezarea alfa-beta

                    if (alfa > beta)
                        break;
                }

                return (mutareOptima, scorOptim);

            }
            // dacă nodul este minim
            // adică utilizatorul este reprezentat de jucător
            else
            {
                var scorOptim = int.MaxValue;
                var mutareOptima = -1000;

                for (var i = 0; i < mutari.Count; i++)
                {
                    var aux = mutareNoua(tabla, mutari[i], utilizator);
                    var mutare = algoritmMinimax(aux, adancime + 1, 2,
alfa, beta);
                    if (mutare.Item2 <= scorOptim)
                    {
                        scorOptim = mutare.Item2;
                        mutareOptima = mutari[i];
                    }

                    beta = Math.Min(beta, scorOptim);

                    // Condiția necesară pentru retezarea alfa-beta
                    if (alfa > beta)
                        break;
                }

                return (mutareOptima, scorOptim);
            }
        }
    }
}
```

5. Rezultatele obținute prin rularea programului

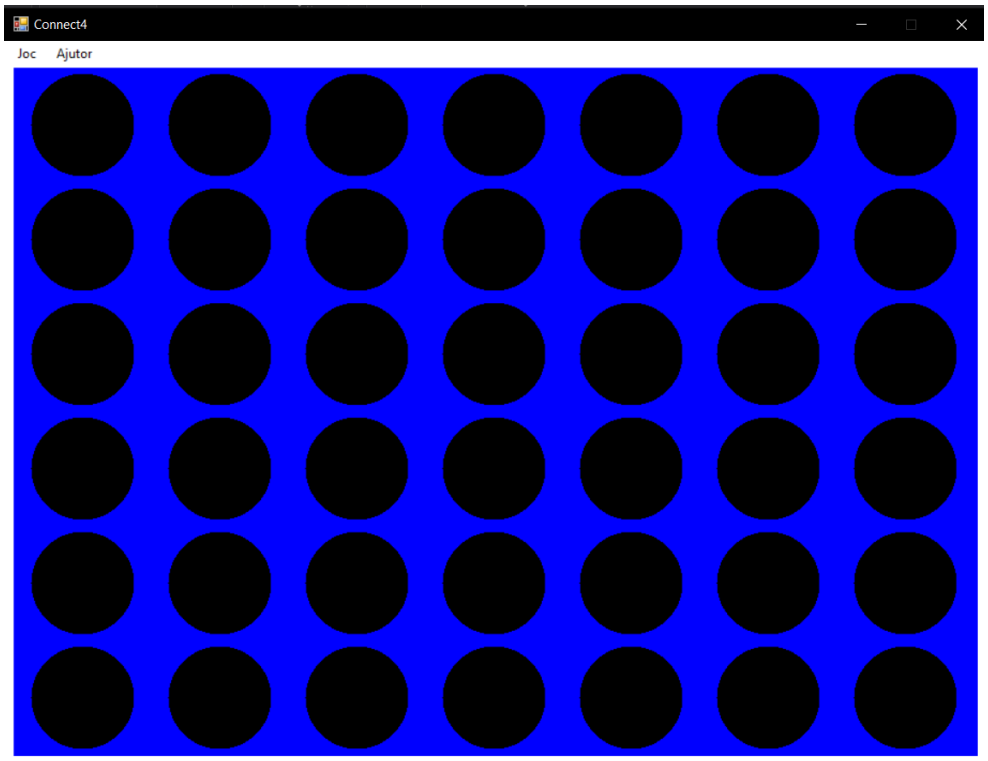

Figura 4. Tabla de joc

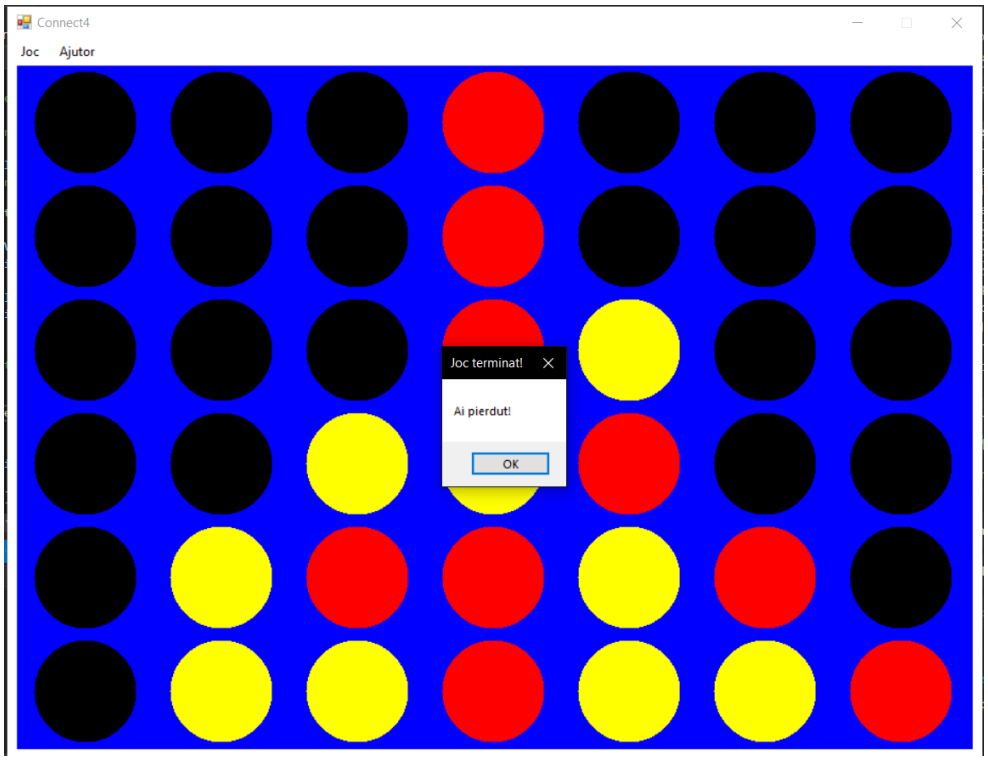

Figura 5. Exemplu de rulare a aplicației

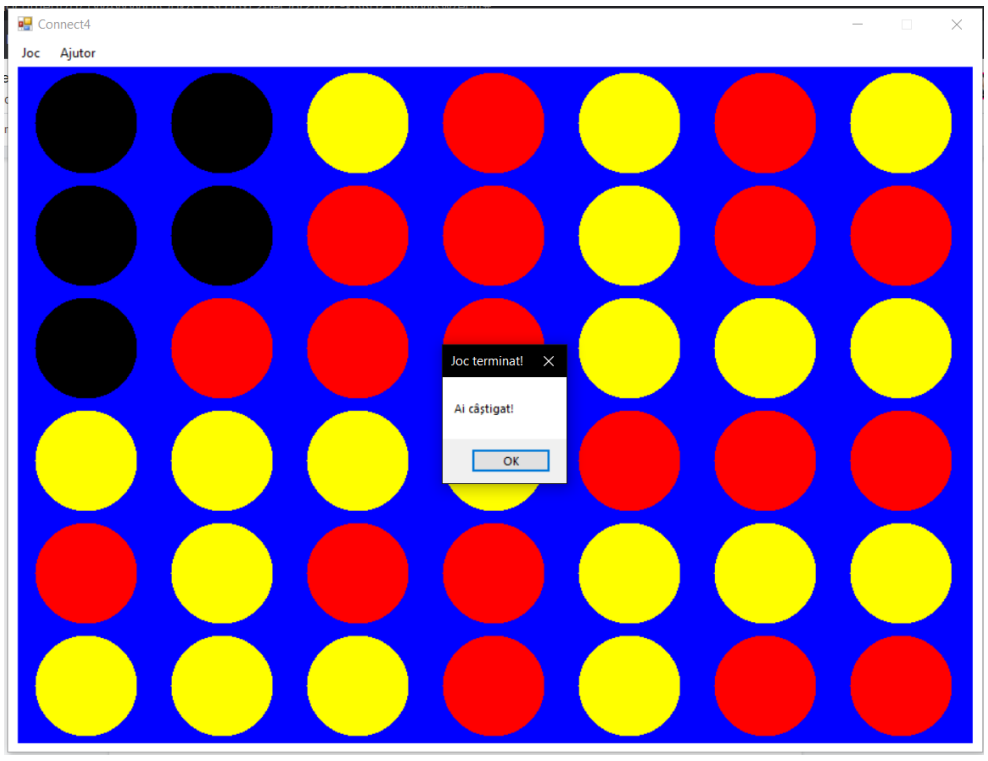

Figura 6. Exemplu de rulare a aplicației

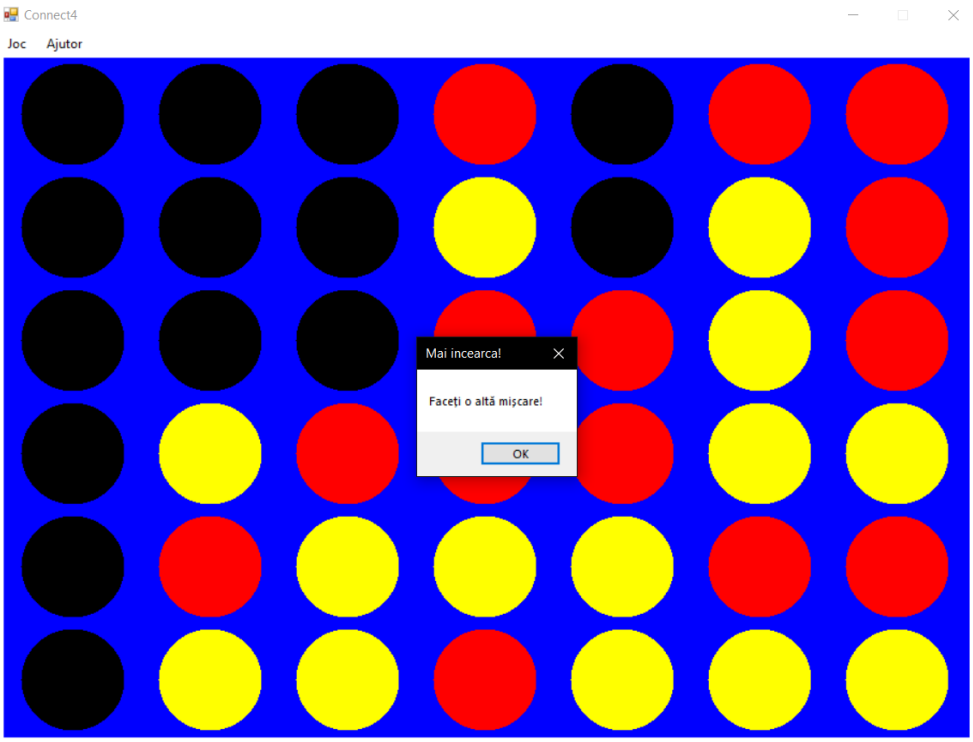

Figura 7. Exemplu de rulare a aplicației atunci când avem o mutare invalidă

6. Concluzii

Algoritmul minimax are o aplicabilitate mare pentru jocurile de tip sumă zero. Acesta găsește soluția optimă examinând întreg spațiul de soluții al problemei. Această abordare nu este eficientă din punct de vedere al vitezei de calcul și a memoriei, deoarece multe stări de joc inutile sunt explorate.

Retezarea alfa-beta reprezintă o îmbunătățire substanțială a algoritmului minimax. În cazul ideal, atunci când cea mai bună mutare este analizată prima, celelalte vor fi eliminate din căutare. Însă în cel mai nefavorabil caz, atunci când cea mai bună mutare este analizată ultima, retezarea alfa-beta are aceeași complexitate cu algoritmul minimax.

7. Bibliografie

[1] https://koaha.org/wiki/Minimax
[2] https://medium.com/analytics-vidhya/artificial-intelligence-at-play-connect-four-minimax-algorithm-explained-3b5fc32e4a4f
[3] https://en.wikipedia.org/wiki/Connect_Four
[4] https://ocw.cs.pub.ro/courses/pa/laboratoare/laborator-06
[5] Inteligență artificială - LaboratorIA07.pdf, Curs03.pdf
[6] https://connect4me102b.weebly.com/about.html
[7] https://www.youtube.com/watch?v=MMLtza3CZFM

[8] https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

8. Rolul fiecărui membru al echipei

Bîzgan Alina

- metoda functieEvaluare
- metoda deseneazaTabla
- metoda calculPondere
- constructorul Implementare
- interfața grafică și funcțiile asociate acesteia

Modiga Camelia

- metoda algoritmMinimax
- metoda stareCurenta
- metoda mutareNoua
- interfața grafică și funcțiile asociate acesteia
- documentație

Ștefan Alexandra

- metoda verificaCastigator
- metoda mutareJucator
- metoda mutareCalculator
- metoda mutareValida
- interfața grafică și funcțiile asociate acesteia