UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAŞI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DOMENIUL: Calculatoare şi Tehnologia Informaţiei
SPECIALIZAREA: Tehnologia Informaţiei

Inteligență artificială

~ proiect ~

Algoritmul Minimax cu retezarea alfa-beta

Student:Ichim Paula-Mădălina
Grupa: 1411A

An 2021-2022

Cuprins

Capitolul 1. Descrierea problemei considerate

Jocuri de sumă 0 sunt jocurile care de obicei se joacă în doi jucători şi în care o mutare bună a unui jucător este în dezavantajul celuilalt jucător în mod direct. Aplicațiile de acest fel sunt în orice joc de sumă 0: șah, table, X și 0, dame, go etc. Orice joc în care se poate acorda un scor / punctaj pentru anumite evenimente - de exemplu la șah pentru capturarea unor piese sau la X și 0 pentru plasarea unui X pe o anumite poziție).

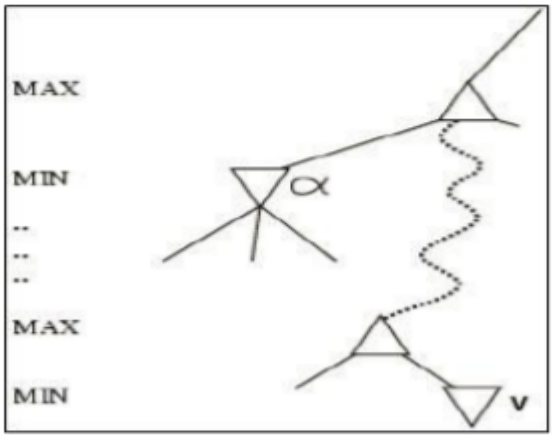
Algoritmul Minimax reprezintă una din cele mai cunoscute strategii pentru a juca jocurile de sumă 0. Minimax este un algoritm ce permite minimizarea pierderii de puncte într-o mutare următoare a jucătorului advers. Concret, pentru o alegere într-un joc oarecare se preferă cea care aduce un risc minim dintre viitoarele mutări bune ale adversarului.

O îmbunătățire substanțială a algoritmului 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.

Astfel ne-am propus să implementăm acest algoritm într-un joc de tip X şi 0 , un joc între 2 jucători, un player și un Computer, în care logica mișcărilor Computerului vor avea la bază acest algoritm .

Capitolul 2. Aspecte teoretice privind algoritmul

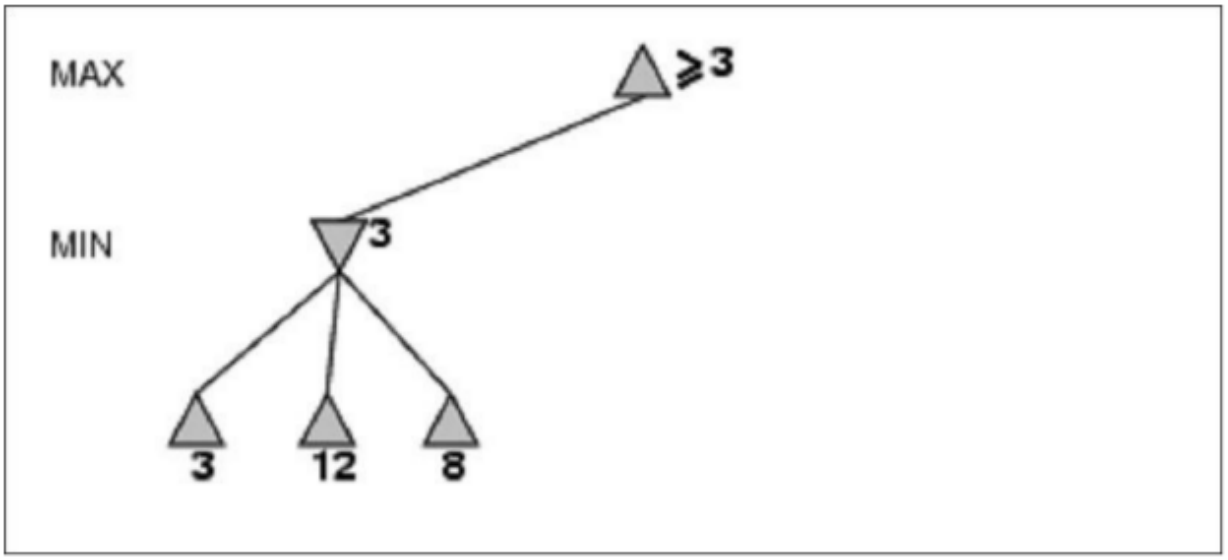
Algoritmul minimax poate determina cea mai buna mutare pentru un jucător, presupunand ca adversarul joacă perfect, prin enumerarea întregului arbore al jocului. Algoritmul alfa-beta face aceleași calcule ca și minimax, dar este mai eficient pentru ca elimina ramurile arborelui de căutare care nu au relevanță pentru rezultatul final. De obicei nu este convenabil sa se considere întregul arbore al jocului, chiar dacă se folosește și alfa beta, motiv pentru care se oprește căutarea la un anumit punct și se aplica o funcție de performanta care estimează utilitatea unei stări.
Astfel este posibil a se calcula decizia Minimax fără a vizita fiecare nod din arborele de căutare. Procesul constă în retezarea unor ramuri ale arborelui şi presupune neluarea în considerarea a acelor ramuri.

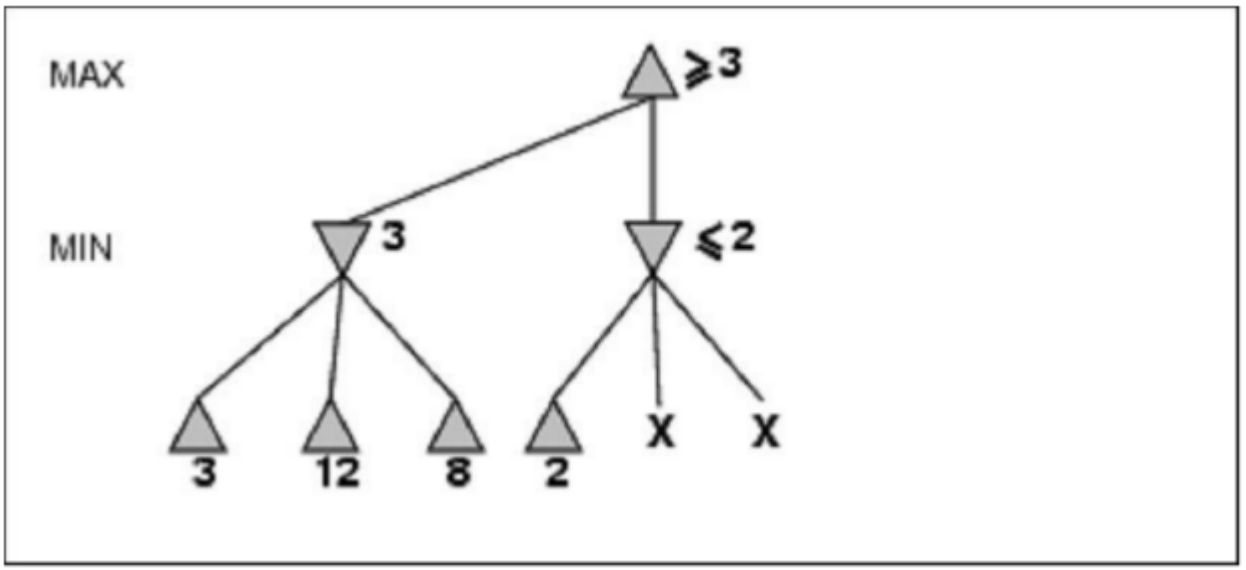


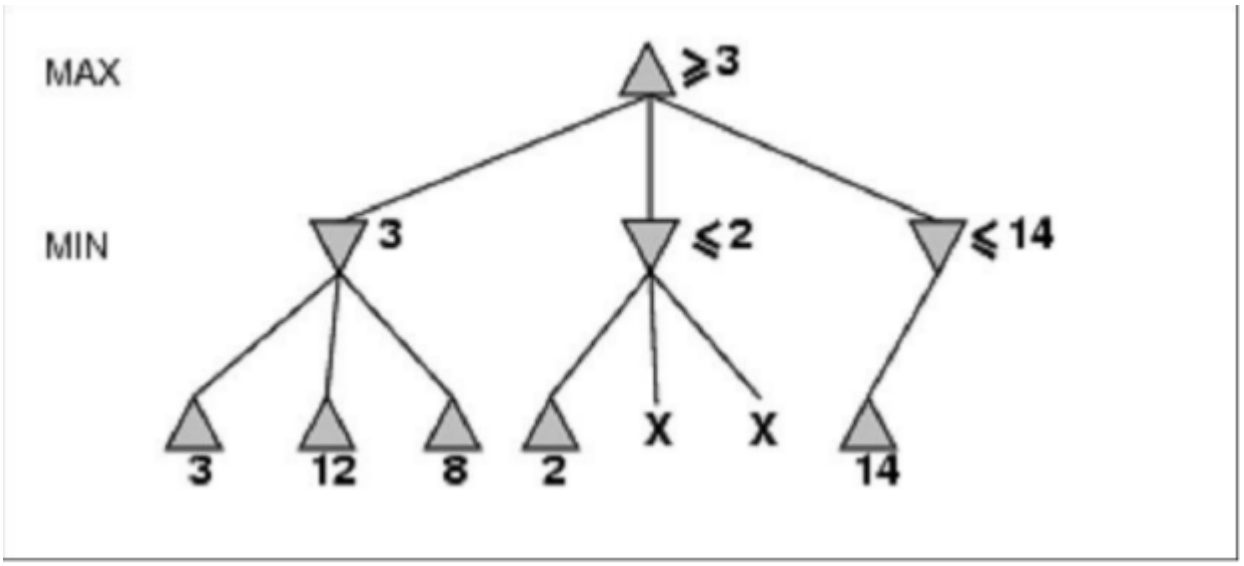
Parametrii alpha și beta :

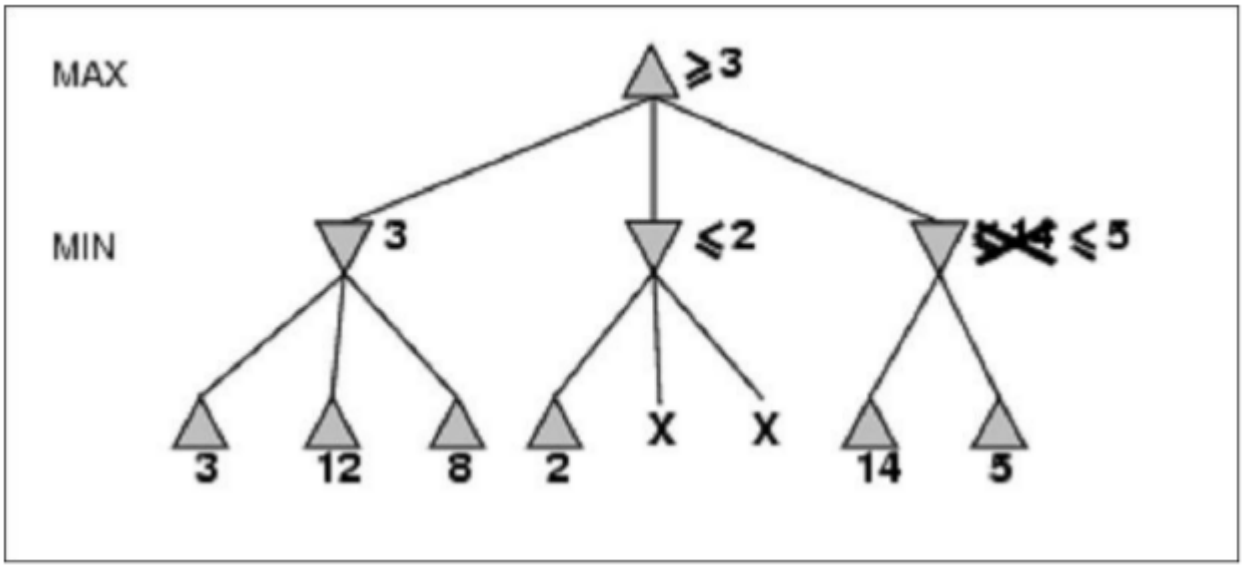
- **α** este valoarea celei mai bune alegeri găsite până la momentul curent, la orice punct de-a lungul unui drum, pentru MAX. Dacă v este mai prost decât α, MAX îl va evita, prin eliminarea acelei ramuri.
- **β** este definit în mod similar pentru MIN, adică cea mai mică valoare găsită de-a lungul unui drum, pentru MIN.

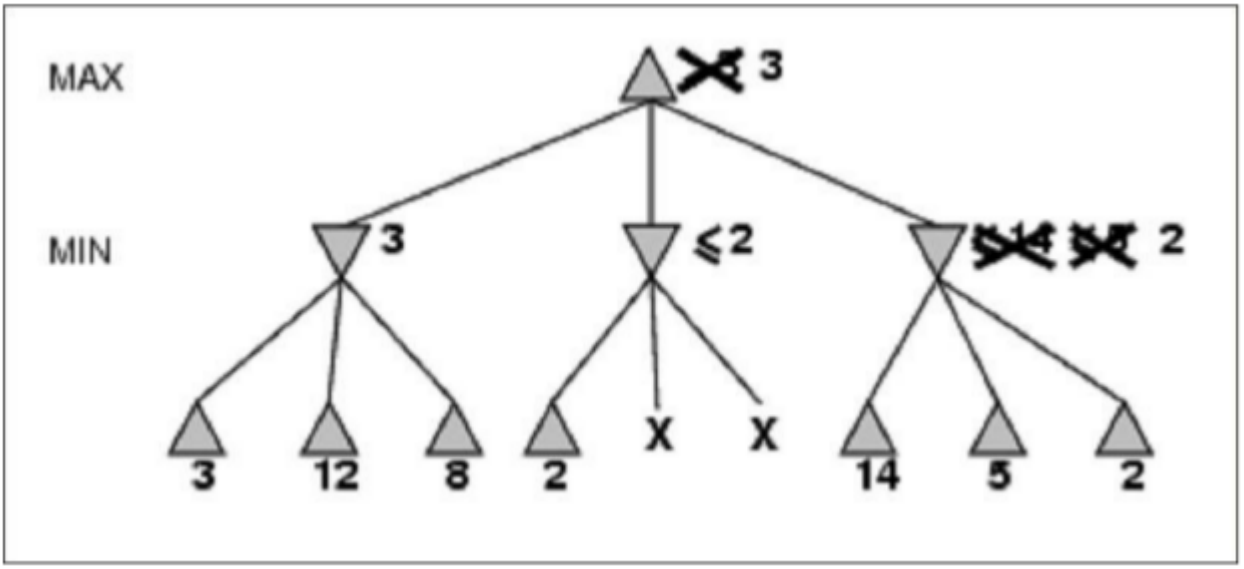
Figurile de mai jos reprezintă o exemplificarea a modului în care tăietura α – β se aplică unui arbore.





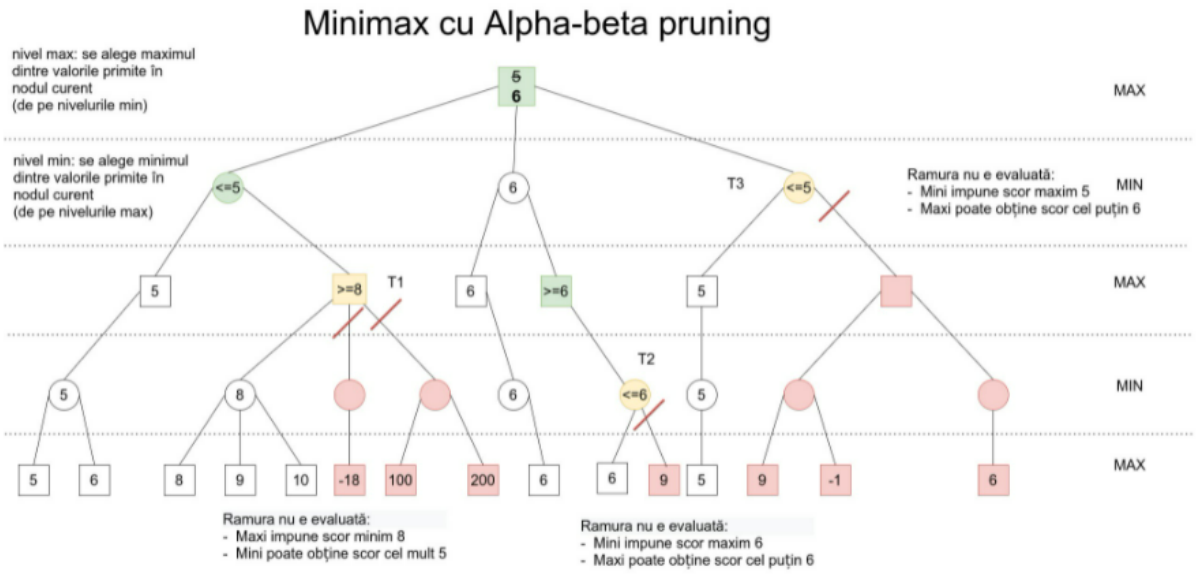

MAX
MIN
≥3
3
≤2
≤14
3
12
8
2
X
X
14


MAX
MIN
≥3
3
≤2
≤5
3
12
8
2
X
X
14
5

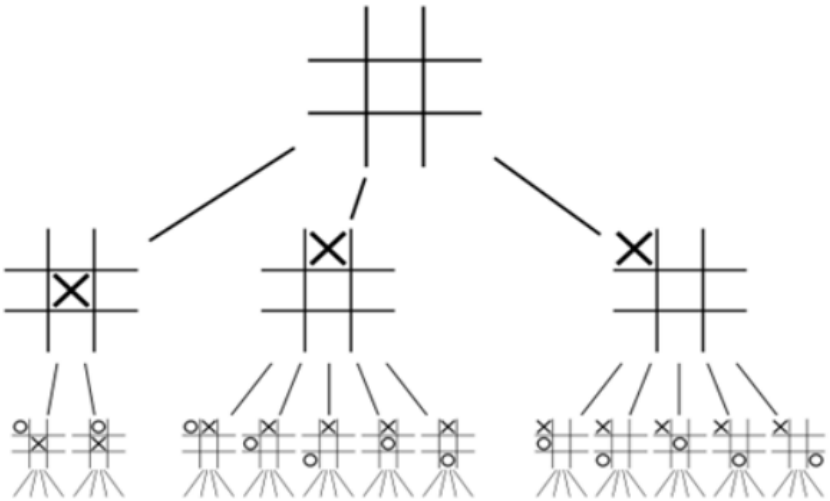

MAX
MIN
3
3
≤2
2
3
12
8
2
X
X
14
5
2

Pseudocode :

```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):

    if node is a leaf node :
        return value of the node

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each child node :
            value = minimax(node, depth+1, false, alpha, beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal

    else :
        bestVal = +INFINITY
        for each child node :
            value = minimax(node, depth+1, true, alpha, beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal
```



Exemplu pentru jocul de X și 0 :



Capitolul 3. Modalitatea de rezolvare

Descrierea formală a jocului:

- Un joc poate fi definit formal ca o problema de căutare cu următoarele componente:
- Starea inițială include pozițiile de pe tabla și cine este cel care este la mutare.
- O mulțime de acțiuni care definesc mutările admise pe care le poate face un jucător.
- O stare terminala care determina cand se sfarseste jocul. Stările în care jocul se încheie se numesc stări terminale.
- O funcție de utilitate care întoarce o valoare numerică pentru rezultatul jocului. În general, posibilitatile sunt victorie, egal sau infrangere care pot fi reprezentate ca 1, 0 sau -1.

Strategie de joc :

- MAX trebuie să găsească o strategie care să îl ducă la o stare terminala în care el este câștigătorul, indiferent de ce mutari face MIN. Strategia presupune ca MAX face mutările corecte, indiferent de mutările lui MIN.

Exemplificarea strategiei:

- De la starea inițială, MAX are posibilitatea de a alege din 9 stări posibile. Jucătorii alternează punand X și 0 pana cand se ajunge la o stare terminala – stare în care un jucător are trei elemente pe o linie, coloană sau diagonală ori toate căsuțele sunt completate. Numărul atașat la fiecare nod frunza se referă la utilitatea stării terminale pentru jucătorul MAX. Valorile mari sunt considerate bune pentru MAX și proaste pentru MIN (și invers), de aici și numele celor doi jucători. Sarcina lui MAX este sa folosească arborele de căutare pentru a determina cele mai bune mutări, tinand cont de utilitatile stărilor terminale.

Jucătorii:

- Cei doi jucători sunt numiți maximizant şi 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 contraatace.

Capitolul 4. Listarea părților semnificative din codul sursă însoțite de explicații și comentarii

Funcția algoritmului Minimax cu retezare alpha-beta:

Această funcție evaluează toate mutările posibile folosind algoritmul Minimax și returnează cea mai bună mutare posibilă.
Dacă valoarea alfa este mai mare sau egală decât valoarea beta a unui nod descendent, atunci se oprește generarea fiilor nodului descendent.
Dacă valoarea beta este mai mică sau egală decât valoarea alfa a unui nod descendent, atunci se oprește generarea fiilor nodului descendent.
Pseudocodul a fost prezentat în capitolul anterior.

```
public int GetBestMove_alphabeta(Board board, Board.State player, int depth, double alpha, double beta)
    {
      Board.State computer = (player == Board.State.X) ? Board.State.O : Board.State.X; // verificăm dacă Computerul este cu X sau cu 0

      if (board.GameOver() || depth++ == maxDepth)  //conditia de terminare
      {
        return EvaluationFunction(board, player);
      }

      if (board.getTurn() == player)
      {
        int index_bestmove = -1;//initializam cu o valoare imposibila
        List<int> valid_moves = board.getAvailableMoves(); //lista mutari posibile
        for (int i = 0; i < valid_moves.Count; i++)
        {
          int current_move = valid_moves[i]; // luam mutarile la rand
          Board modifiedBoard = board.GetCopy(); //facem copie la tabla de joc
          modifiedBoard.Move(current_move);  //facem mutarea curenta
          int best_move = GetBestMove_alphabeta(modifiedBoard, player, depth, alpha, beta);
```

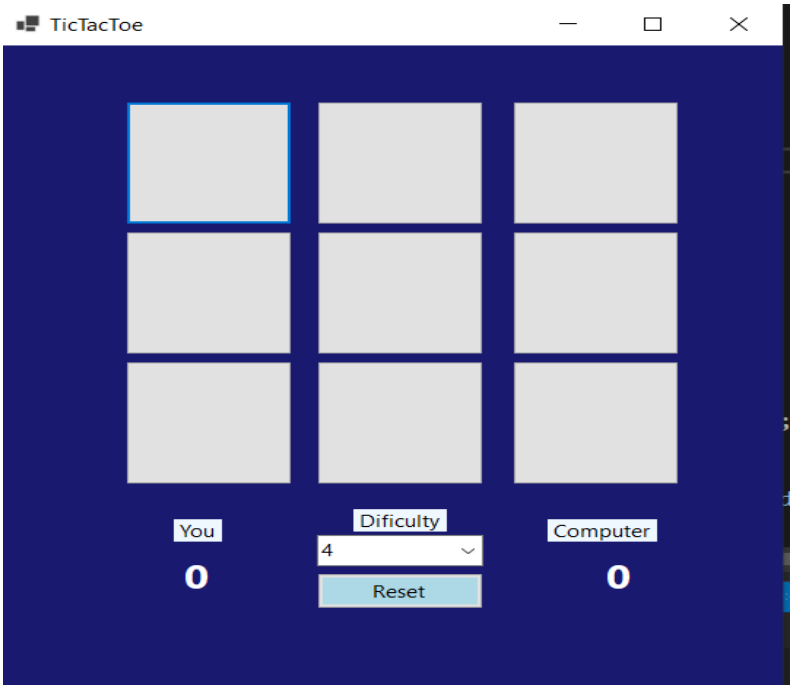
```
//apelam functia

                if (best_move > alpha) // daca miscarea are scorul mai mare decat alfa
                {
                    alpha = best_move;    //alfa=miscare
                    index_bestmove = current_move;// indexul este miscarea curenta
                }
                if (alpha >= beta)  //nu este o miscare buna
                {
                    break;
                }
            }
            if (index_bestmove != -1) //exista miscare mai buna
            {
                board.Move(index_bestmove); //facem miscarea
            }
            return (int)alpha;
        }
        else // este randul computerului , procedam asemanator dar cu beta
        {
            int index_bestmove = -1;//initializam cu o valoare imposibila
            List<int> valid_moves = board.getAvailableMoves();//lista mutari posibile
            for (int i = 0; i < valid_moves.Count; i++)
            {
                int current_move = valid_moves[i]; //luam fiecare mutare in parte
                Board modifiedBoard = board.GetCopy();//copiem tabla de joc
                modifiedBoard.Move(current_move); // facem mutarea curenta
                int score = GetBestMove_alphabeta(modifiedBoard, player, depth, alpha, beta);
//vedem score-ul ei
                if (score < beta)   // daca este mai mica decat valoarea beta o salvam ca miscare mai
buna
                {
                    beta = score;
                    index_bestmove = current_move;
                }
                if (alpha >= beta)
                {
                    break;
                }
            }
            if (index_bestmove != -1)//există mișcare mai bună
            {
                board.Move(index_bestmove);//efectuam miscarea
            }
            return (int)beta;
        }

    }
```

Funcția de evaluare:
Calculează funcția de evaluare statică pentru configurația (tabla) curentă. Pentru un joc oarecare, programatorul trebuie să imagineze funcția de evaluare. Pot exista mai multe funcții posibile pentru un joc. Cu cât funcția aceasta este mai „inteligentă”, cu atât calculatorul va juca mai bine. De exemplu, în cazul nostru, o funcție de evaluare returnează 1 în cazul în care playerul a câștigat, -1 în cazul în care Computerul a câștigat, altfel returnează 0.

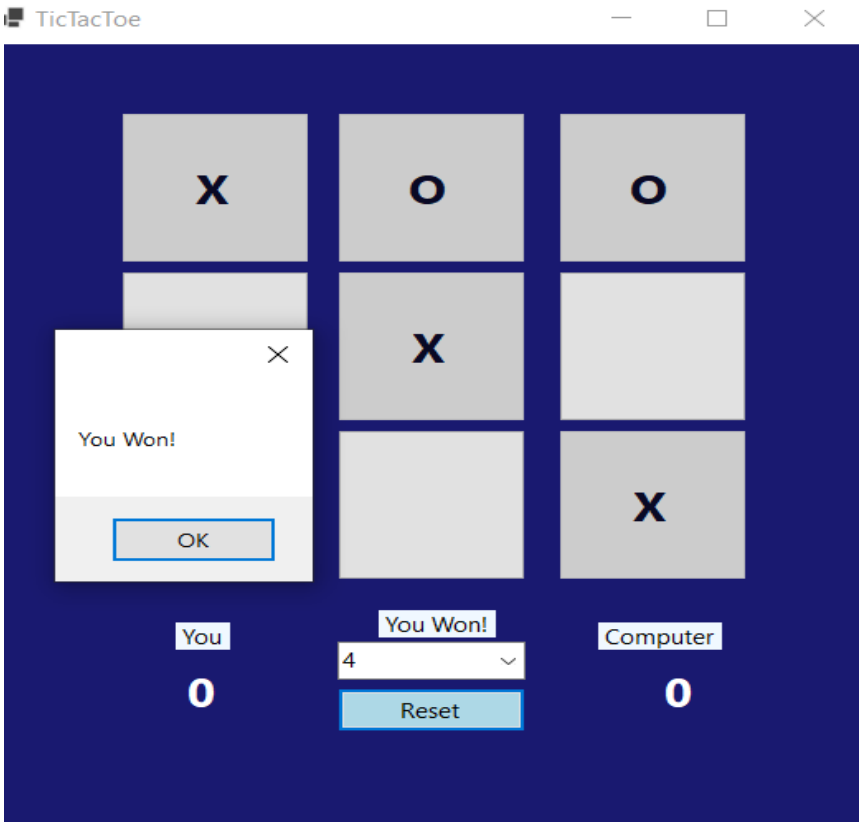
```
public int EvaluationFunction(Board board, Board.State player)
        {
            Board.State computer = (player == Board.State.X) ? Board.State.O : Board.State.X; //
verificam daca este cu 0 sau X computerul
            if (board.GameOver() && board.getWinner() == player)//a castigat playerul
            {
                return 1;
            }
            else if (board.GameOver() && board.getWinner() == computer) //a castigat calculatorul
            {
                return -1;
            }
            else
            {
                return 0;
            }
        }
```

Capitolul 5.Rezultatele obținute prin rularea programului în diverse situații

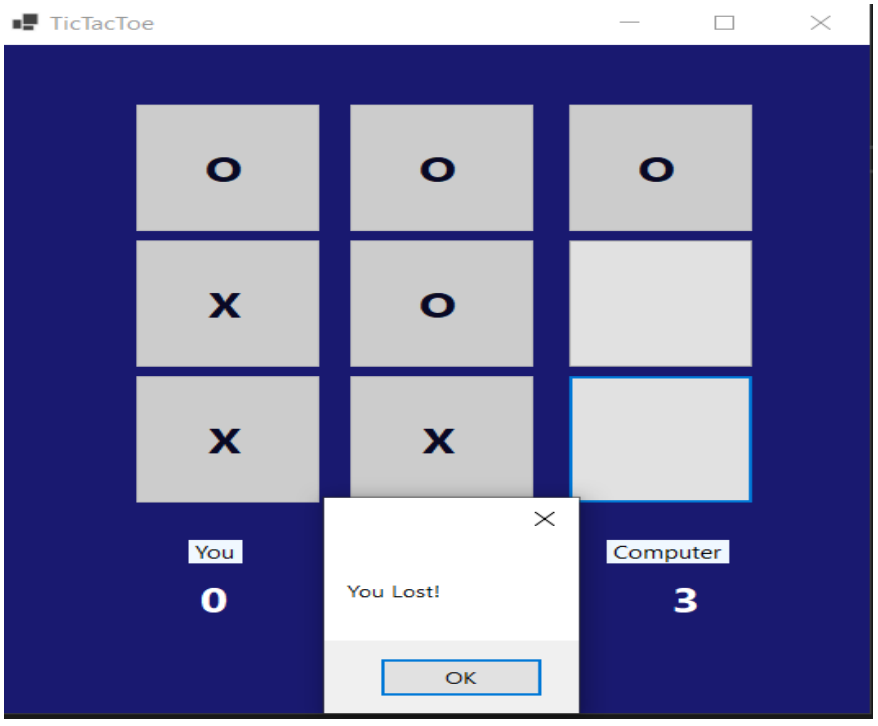
Interfața jocului :



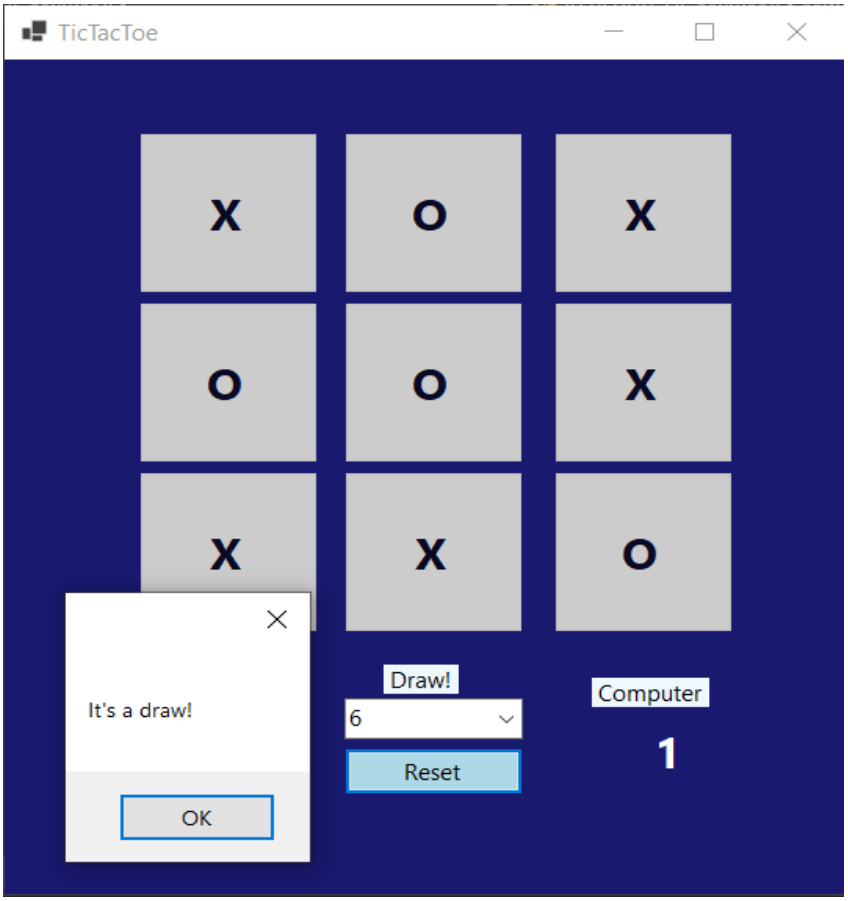
Captură atunci când playerul câștigă jocul: (nu am găsit astfel de cazuri utilizând algoritmul, dar am modificat programul ca să arăt acest caz că funcționalitate de câștig este implementată)



Captură atunci când Computerul câștigă jocul :



Captură atunci când este remiză:



Capitolul 6. Concluzii

Algoritmul minimax poate determina cea mai buna mutare pentru un jucător, presupunand ca adversarul joacă perfect, prin enumerarea întregului arbore al jocului. Algoritmul alfa-beta face aceleași calcule ca și minimax, dar este mai eficient pentru ca elimina ramurile arborelui de căutare care nu au relevanță pentru rezultatul final.
Alfa-beta garantează calcularea aceleiași valori pentru rădăcină ca și minimax, cu o complexitate mai redusă deoarece reducerea nu afectează rezultatul final.
O bună ordonare a mutărilor îmbunătățește algoritmul de reducere. Dacă succesori sunt puși perfect în ordine (cei mai buni se află primii), atunci complexitatea temporară ar fi O(b^d/2), în loc de O(b^d) cât are Minimax. Deci α – β, poate căuta de două ori mai mult decât Minimax și astfel tratând cu mare atenție calculele care afectează decizia, putem transforma un program de nivel începător în expert.

Capitolul 7.Bibliografie

- https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/
- http://inf.ucv.ro/documents/cstoean/c8IA_13.pdf
- https://ocw.cs.pub.ro/courses/pa/laboratoare/laborator-06
- https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning
- curs Inteligența Artificială, prof. dr. ing. Florin Leon
- http://aspc.cs.upt.ro/ia/bia3.htm