Universitatea din Craiova
Facultatea de Automatică,Calculatoare şi Electronică

5.iunie.2018

Proiect : Inteligenta Artificiala
Titlu : Tic Tac Toe

Student : Voica Catalin Gabriel
Sectie : Calculatoare Romanaă
Anul 2
Grupa : CR 2.1 C

Contents

1 Declararea Problemei

1.1 Titlu

Tic Tac Toe

1.2 Descriptie

Pentru proiectul meu a trebuit sa implementez jocul Tic Tac Toe (X si 0),iar apoi sa testez si sa evaluez performanta a doua algoritme de cautare euristice.Cele doua functii euristice pe care trebuia sa le fac sunt A* Search si Recursive Best First Search.

La algoritmul pe care trebuie sa il implementez se vor verifica urmatoarele lucruri: -Verificarea completitudinii : daca algoritmul respectiv determina o solutie,cate solutii poate gasi algoritmul. -Verificarea optimitatii:se va verifica daca algoritmul determina o solutie optima cand exista cel putin o solutie.Se va verifica deasemenea si daca algoritmul respectiv determina solutia optima din prima incercare. -Complexitatea Timpului:cat timp dureaza sa gaseasca o solutie si/sau solutia optima.

Se cere sa se implementeze seturi de date nontriviale care sa contina cel putin 10 cazuri de teste de marimi variabile :mici,medii,mari si foarte mari astfel incat fisierele de date sa incapa intr-o arhiva de 8 MB.Fiecare caz de test trebuie sa defineasca starea spatiului,dar si valoarea functiei euristice pentru fiecare stare.

La aplicatia pe care trebuia sa o implementez,Tic Tac Toe,nu am reusit sa aplic niciuna dintre cele doua functii euristice,A* Search si Recursive Best First Search.In shimb am reusit sa termin jocul,construind o clasa "checkRules" care contine cate o functie pentru fiecare caz in care jocul Tic Tac Toe se poate termina,o clasa "board" prin care se introduc mutarile jucatorului,mutarile acestuia fiind simulate de catre program.Tot in clasa "board" se actualizeaza jocul la fiecare mutare a jucatorului,urmand apoi sa se faca afisarea.La final am mai construit o clasa "gameRules" care sa ruleze functiile care verifica daca jocul s-a terminat sau nu.La sfarsitul jocului simulat,aceasta clasa va afisa rezultatul final.

Proiectul s-a ales in functie de formula 1 + (suma codului ASCII al numelui de familie + primul prenume modulo 5)

voica catalin (v - 118,o-111,i-105,c-99,a-97,space-32,c-99,a-97,t-116,a-97,l-108,i-105,n-110)

Formula : ((118+111+105+99+97+32+99+97+116+97+108+105+110)mod5) + 1 = 5 (a rezultat problema 5)

2 Pseudocod

2.1 board checkWhoStarts()

```
1: randomize v1
2: if v1 == 1 then
3:    return 1;
4: end if
5: return 0;
```

2.2 board playGame (n,matrix,currentPlayer,playerChoice)

```
1: randomize line
2: randomize column
3: while matrix[line][column] ≠0 or line ≥n orcolumn ≥n or line ≥n and
   column ≥n do
4:    if line ≥n orcolumn ≥n or line ≥n and column ≥n then
5:       randomize line
6:       randomize column
7:    end if
8:    if matrix[line][column] ≠0 then
9:       randomize line
10:      randomize column
11:   end if
12: end while
13: if currentPlayer == 1 and playerChoice == 1 then
14:    matrix[line][column] ←playerChoice
15:    updateMatrix(n,matrix,playerChoice)
16: end if
17: if currentPlayer == 2 and playerChoice == 2 then
18:    matrix[line][column] ←playerChoice
19:    updateMatrix(n,matrix,playerChoice)
20: end if
```

2.3 board updateMatrix (n,matrix,playerChoice)

```
1: for i =0 to n do
2:    for j =0 to n do
3:       if matrix[i][j] ≠ 1 and matrix[i][j] ≠2 then
4:          matrix[i][j] ←0
5:       end if
6:       matrix[line][column] ←playerChoice
7:    end for
8: end for
```

2.4 board showMatrix (n,matrix)

```
1: for i =0 to n do
2:    for j =0 to n do
3:       print matrix[i][j]
4:    end for
5: end for
```

2.5 checkRules checkIfBoardFull(n,matrix)

```
1: k ←0
2: for i =0 to n do
3:    for j =0 to n do
4:       if matrix[i][j] == 0 then
5:          k ←k + 1
6:       end if
7:    end for
8:    if k==0 then
9:       return 0
10:   end if
11:   return 1
12: end for
```

2.6 checkRules checkTheLines(n,matrix,playerchoice)

```
1: if playerChoice == 1 then
2:    k ←0
3:    for i =0 to n do
4:       for j =0 to n do
5:          if matrix[i][j] ≠playerChoice and k <n and j == n-1 then
6:             k ←0
7:          end if
8:          if matrix[i][j] == playerChoice and k <n then
9:             k ←0
10:            if k==n then
11:               save ←i
12:            end if
13:            if j == n-1 and k <n then
14:               k ←0
15:            end if
16:         end if
17:      end for
18:   end for
19:   if k==n then
20:      return 1
21:   end if
```

```
    return 0
end if
if playerChoice == 2 then
    k ←0
    for i =0 to n do
        for j =0 to n do
            if matrix[i][j] ≠playerChoice and k <n and j == n-1 then
                k ←0
            end if
            if matrix[i][j] == playerChoice and k <n then
                k ←0
                if k==n then
                    save ←i
                end if
                if j == n-1 and k <n then
                    k ←0
                end if
            end if
        end for
    end for
    if k==n then
        return 1
    end if
    return 0
end if
```

2.7 checkRules checkTheColumns(n,matrix,playerChoice)

```
if playerChoice==1 then
    k ←0
    for i =0 to n do
        for j =0 to n do
            if matrix[i][j] == playerChoice and k <n and i == 0 or (matrix[i][j] ≠playerChoice
            and k <n and i==0) then
                k ←0
            end if
            if matrix[i][j] == playerChoice and k <n then
                k ←k+1
                if k==n then
                    save ←j
                end if
            end if
        end for
    end for
    if k==n then
        return 1
```

```
    end if
    return 0
end if
if playerChoice==2 then
    k ←0
    for i =0 to n do
        for j =0 to n do
            if matrix[i][j] == playerChoice and k <n and i == 0 or (matrix[i][j] ≠playerChoice
            and k <n and i==0) then
                k ←0
            end if
            if matrix[i][j] == playerChoice and k <n then
                k ←k+1
                if k==n then
                    save ←j
                end if
            end if
        end for
    end for
    if k==n then
        return 1
    end if
    return 0
end if
```

2.8 checkRules checkTheDiagonalOne(n,matrix,playerChoice)

```
if playerChoice == 1 then
    k ←0
    for i =0 to n do
        for j =0  TO m do
            if i==j and matrix[i][j] == playerChoice then
                k ←0
            end if
        end for
    end for
end if
if k==n then
    return 1
end if
return 0
if playerChoice == 2 then
    k ←0
    for i =0 to n do
        for j =0  TO m do
            if i==j and matrix[i][j] == playerChoice then
```

```
            k ←0
          end if
        end for
      end for
    end if
    if k==n then
      return 1
    end if
    return 0
```

2.9 checkRules checkTheDiagonalTwo(n,matrix,playerChoice)

```
1:  if playerChoice == 1 then
2:    k ←0
3:    i ←0
4:    j ←n-1
5:    while i <n and j ≥0 do
6:      if matrix[i][j] == playerChoice then
7:        k ←k+1
8:      end if
9:      i ←i+1
10:     i ←j-1
11:   end while
12:   if k==n then
13:     return 1
14:   end if
15:   return 0
16: end if
17: if playerChoice == 2 then
18:   k ←0
19:   i ←0
20:   j ←n-1
21:   while i <n and j ≥0 do
22:     if matrix[i][j] == playerChoice then
23:       k ←k+1
24:     end if
25:     i ←i+1
26:     i ←j-1
27:   end while
28:   if k==n then
29:     return 1
30:   end if
31:   return 0
32: end if
```

2.10 gameRules checkTheGame(n,matrix,playerChoice,currentPlayer)

```
1: if checkTheLines(n,matrix,playerChoice) == 1 then
2:    showMatrix(n,matrix)
3:    return 0
4: end if
5: if checkTheColumns(n,matrix,playerChoice)==1 then
6:    showMatrix(n,matrix)
7:    return 0
8: end if
9: if checkTheDiagonalOne(n,matrix,playerChoice)==1 then
10:   showMatrix(n,matrix)
11:   return 0
12: end if
13: if checkTheDiagonalTwo(n,matrix,playerChoice)==1 then
14:   showMatrix(n,matrix)
15:   return 0
16: end if
17: if checkIfBoardFull(n,matrix)==0 then
18:   showMatrix(n,matrix)
19:   return 0
20: end if
21: return 1
```

3 Designul Aplicatiei

3.1 Main

In functia main primul lucru pe care l-am facut a fost sa initializez toate casutele din jocul Tic Tac Toe cu cifra '0',presupunand ca acele casute sunt casutele care nu sunt ocupate,jucatorii urmand sa le completeze ulterior(Jucatorul 1 joaca cu cifra '1' iar Jucatorul 2 joaca cu cifra '2').Tot in functia main se va stabili prin functia "checkWhoStarts" din clasa "board" care jucator va incepe primul,apoi va urma un while care va rula functia "checkTheGame" din interiorul clasei "gameRules" pana cand aceasta va returna 0,adica atunci cand una din conditiile ca jocul Tic Tac Toe sa se termine este indeplinita.

3.2 Input Data

Ca input am ales functia "playGame" din clasa "board" sa dea valori aleatorii pentru numarul casutei care va fi selectata de unul din cei doi jucatori,aceasta generand numere aleatorii intre 0 si numarul de casute dorit pentru jocul de X si 0.Jocul de Tic Tac Toe pe care l-am dezvoltat poate rula pe n casute,numarul acestora stabilindu-se la inceputul functiei "main".

3.3 Output Data

Atunci cand programul a terminat de rulat se va afisa pe ecran un joc de X si 0 pe n x n casute,casutele fiind completate de catre playeri,programul mentionand ce jucator a castigat,dar si pe ce linie,coloana sau diagonala a reusit acest lucru.Daca nici un jucator nu a castigat programul va mentiona acest lucru.

3.4 Clasa board

3.4.1 checkWhoStarts

Functia "checkWhoStarts" din clasa "board" are rolul de a decide ce jucator va incepe primul jocul de X si 0,programul generand un numar la intamplare,acest numar poate sa fie '1' (caz in care incepe primul jucator) sau numarul '2' (caz in care va incepe al doilea jucator).

3.4.2 playGame(n,matrix,currentPlayer,playerChoice

Functia "playGame" din clasa "board" are rolul de a completa casutele din interioriul matricei,in interioriul functiei generandu-se numere aleatorii.Functia

deasemenea verifica daca vreo casuta din interioriul matricei a fost completata,sau daca numerele generate sunt prea mari decat capacitatea matricei respective,cazuri in care va genera alte numere.Tot din interioriul acestei functii se va actualiza si matricea dupa fiecare mutare a jucatorilor.

3.4.3 updateMatrix(n,matrix,playerChoice)

Functia "updateMatrix" din clasa "board" are rolul de a actualiza matricea respectiva,dupa ce unul din jucatori a ales casuta din interioriul matricei.Aceasta functie este apelata din interiorul functiei "playGame" din aceeasi clasa.

3.4.4 showMatrix(n,matrix)

Functia "showMatrix" din clasa "board" este apelata din clasa "gameRules".Functia este apelata in momentul in care una din conditiile pentru a se termina jocul a fost indeplinita,functia urmand sa afiseze rezultatul final al jocului de X si 0.

3.5 Clasa checkRules

3.5.1 checkIfBoardFull(n,matrix)

Aceasta functie are rolul de a verifica daca mai sunt mutari posibile,si niciun jucator nu a reusit sa invinga.In acest caz se,nu va mai exista nici o casuta goala (nu va mai exista cifra '0' in matrice la rezultatul final).Aceasta functie este apelata din interiorul functiei "checkTheGame" din clasa "gameRules.Aceasta functie este una din conditiile pe care eu le-am pus pentru a incheia jocul.(1/5)

3.5.2 checkTheLines(n,matrix,playerChoice)

Aceasta functie este una din cele 5 functii pe care le-am folosit pentru a verifica daca jocul s-a terminat,functia avand rolul de a verifica pe linie daca mutarile jucatorului (cifra '1' sau '2') sunt egale cu n.Aceasta functie este si ea apelata tot din interiorul functiei "checkTheGame".(2/5)

3.5.3 checkTheColumns(n,matrix,playerChoice)

Functia are acelasi rol ca functia precedenta,"checkTheLines",diferenta consta in faptul ca aceasta verifica daca unul din jucatori are numarul de mutari (cifra '1' sau '2') egal cu n.(3/5)

3.5.4 checkTheDiagonalOne(n,matrix,playerChoice)

Cu aceasta functie se verifica pe prima diagonala (din coltul din stangas sus pana in coltul din dreapta jos) daca numarul de mutari al jucatorului (cifra '1' sau '2') este egala cu n.(4/5)

3.5.5 checkTheDiagonalTwo(n,matrix,playerChoice)

Aceasta functie este ultima din cele 5 conditii ca jocul sa se incheie,si are rolul de a verifica pe a doua diagoanala (din coltul din dreapta de sus pana in coltul din stanga jos) daca numarul de mutari al jucatorului (cifra '1' sau '2') este egal cu n.(5/5)

3.6 clasa checkRules

3.6.1 checkTheGame(n,matrix,playerChoice,currentPlayer)

Este functia care detine rolul cel mai important din program,aflandu-se in clasa "gameRules".Aceasta functie este apelata din interiorul functiei Main,este apelata in interiorul unui while,aceasta functie rulandu-se pana cand va returna 0.Se va returna valoarea 0 atunci cand una din cele 5 conditii pe care le-am prezentat mai sus va fi indeplinita.Tot din interiorul acestei functii se va apela si functia "showMatrix" in momentul in care jocul se va incheia.

3.7 Concluzii

In concluzie mi s-a parut foarte interesant proiectul de facut la Inteligenta Artificiala,reusind sa imi fac o idee despre cum gandeste un calculator in momentul in care o persoana interactioneaza cu acesta,spre exemplu daca se joaca chiar X si 0.Prin cele doua euristici pe care ar fi trebuit sa le impllementez,programul este mai "destept",generand o serie de cautari,in care el incearca sa gaseasca solutia optima,devenind tot mai dificl pentru acea persoana sa ii faca fata.

3.8 Referinte

Book:
1.$StuartRussel - ArtificialIntelligence : AModernApproach$

4 Source Code

Proiectul meu s-a numit "n size Tic-Tac-Toe Game" .Am ales sa folosesc limbajul de programare C++,si am folos Visual Studio 2015.

5 Experimente si rezultate

In prima poza,jucatorul 1(cel care joaca cu cifra '1') invinge,avand pe coloana a treia 3 de 1. In cea de a doua poza,jucatorul 1 invinge,avand 1 pe diagonala. In cea de a treia poza,cei doi jucatori termina la egalitate,nu mai exista casute goale (nu mai e cifra '0').