Mesterséges intelligencia

gyakorlat

5. feladat

(kétszemélyes játék)

Készítette:

Szathmáry László
II. PM.

Feladat: egy tetszőleges kétszemélyes játék elkészítése. Az általam választott játék a "HARAPÁS", melynek lényege: egy 9x9-es táblán korongok vannak elhelyezve; bármelyik korong levehető, s ekkor a tőle jobbra ill. alatta lévők eltűnnek. Cél, hogy a bal felső zsetont az ellenfélnek kelljen levennie.

Stratégiai játékok osztályozása:

1., résztvevők száma szerint (2,3,...,n személyes játékok)
(néha a véletlent is személynek veszik)

2., véges játékok: olyan stratégiai játékok, amelyekben minden állásban véges sok szabályos lépés közül lehet választani, s véges sok lépés után véget ér

3., zérusösszegű: ha a nyeremények és veszteségek összege zérus. Ellenkező esetben: nem zérusösszegű játékról beszélünk.

4., sztochasztikus, determinisztikus (az alapján, hogy a véletlennek mekkora szerepe van)

5., teljes információjú: a játékosok a játékkal kapcsolatos összes információt ismerik, s felváltva lépnek

Harapás

```
#include <stdio.h>
#include <dos.h>
#include <ctype.h>
#include <stdlib.h>
#include <iostream.h>
#include <conio.h>
#define  N 9                        //a tábla mérete 9x9-es lesz
#define  X 3
#define  Y 2
enum elem   { ures, korong };       //a tábla mezőinek lehetséges állapotai
enum player { human, gep   };       //a két játékos

class Harap {
  public:
    Harap();                        //konstruktor, szerepe: kezdőállapot meghatározása
    void vezerlo();
  private:
    elem tomb[N][N];                //maga az állapottér
    player jatekos;
    int sor, oszlop;

    void rajz();
    int  elofeltetel(int, int);
    int  celallapot();
    void kiharap(int, int);
    void lepes();
    void keres();
};
```

I. Állapottér megadása

Az állapottér egy 9x9-es tábla, melyet egy kétdimenziós tömbbel, azaz mátrixszal reprezentálunk.

A tábla minden egyes mezője két lehetséges állapot valamelyikét veheti fel: **ures**, ill. **korong**.

ures: azon a mezőn már nincs korong, levettük

korong: arról a mezőről még nem vettük le a korongot, még rajta van

II. Kezdőállapot megadása

A kezdőállapot meghatározásához nagyon jól használható a C++ konstruktora:

```
Harap::Harap()
{
  clrscr(); randomize();

  window(40,1,80,25); textcolor(LIGHTGRAY);
  cprintf("Adott egy %dx%d-es tabla, melyen korongok\r\n"
          "vannak. Egy korong felvetele maga utan\r\n"
          "vonja az alatta levok, ill. a tole jobbra"
          "levok felvetelet is.\n\r"
          "Cel: a bal felso zsetont az ellenfel\r\n"
          "vegye fel.", N,N);

  for (int i=0; i<N; ++i)
    for (int j=0; j<N; ++j)
      tomb[i][j]=korong;

  rajz();

  char kezd;
  do {
  gotoxy(3,22); clreol();
  cprintf("Akarsz kezdeni? "); cin>>kezd; kezd = toupper(kezd);
  } while (!(kezd=='I' || kezd=='N'));
  jatekos = ((kezd=='I') ? (human) : (gep));
  gotoxy(3,22); clreol();
}
```

Először a felhasználó tájékoztatása végett kiíratunk egy információs szöveget magáról a programról, annak használatáról. Ezután inicializáljuk az állapotteret, azaz a mátrix valamennyi mezőjére korongot helyezünk.

Ezután kirajzoljuk a táblát "grafikusan", majd meghatározzuk, hogy ki fog kezdeni.

```
void Harap::rajz()
{
  int i,j;

  window(1,1,80,25);
  gotoxy(X, Y); textcolor(YELLOW);
  for (i=0; i<N; ++i) cprintf("%d  ", i);
  gotoxy(1,Y+2);
  for (i=0; i<N; ++i) cprintf("%d\n\n\r" ,i);

  textcolor(LIGHTGRAY);
  for (i=0; i<N; ++i)
  {
    gotoxy(X,(Y+2)+i*2);
    for (j=0; j<N; ++j)
       cprintf("%c  ", ((tomb[i][j]==korong) ? 'O' : ' '));
  }
}
```

Ez az "eljárás" a tábla kirajzolásáért felelős. Amely pozíción még van korong ott egy **O** fog megjelenni.

III. Végállapot megadása

```
int Harap::celallapot()
{
  return (tomb[0][0]==ures);
}
```

Akkor vagyunk végállapotban, ha a bal felső korongot levette valaki. Az a játékos, aki levette, az elveszítette a játékot.

Mivel C-ben nincs logikai típus, ezért azt integrálissal helyettesítjük: 1-igaz, 0-hamis.

IV. Operátorok

Egyetlenegy operátort fogunk alkalmazni: azt, amelyik egy adott pozícióról levesz egy korongot, s ez maga után vonja a tőle jobbra ill. alatta lévő korongok törlését is.

Ennek az előfeltétele:

```
int Harap::elofeltetel(int x, int y)
{
  return (x>=0 && x<N && y>=0 && y<N && tomb[x][y]==korong);
}
```

Vagyis: amely mezőről le akarunk venni egy korongot annak a táblán kell lennie, ill. csak olyan mezőről vehetünk le bármit is, amelyen VAN korong, azaz üres mező esetén az előfeltétel nem fog teljesülni.

```
void Harap::kiharap(int x, int y)
{
  for (int i=x; i<N; ++i)
    for (int j=y; j<N; ++j)
      tomb[i][j]=ures;
}
```

Maga az operátor a megadott pozícióról leveszi a korongot, s a tőle jobbra ill. alatta lévő mezőkkel is ugyanezt teszi => ezen mezők állapota **ures** lesz.

```
void Harap::lepes()
{
  int sor, oszlop;

  window(3,22,80,25);
do {
  do {
  gotoxy(1,1); clreol();
  cprintf("Sor:    "); cin>>sor;
  } while (!(sor>=0 && sor<N));
  do {
  gotoxy(1,3); clreol();
  cprintf("Oszlop: "); cin>>oszlop;
  } while (!(oszlop>=0 && oszlop<N));
} while (!(elofeltetel(sor, oszlop)));
  Harap::sor=sor;
  Harap::oszlop=oszlop;
}
```

A **lepes** nevű eljárás feladata, hogy ha a humán játékos lép, akkor bekérje tőle, hogy mely pozícióról akarja a korongot levenni. A bekéréskor megtörténik annak vizsgálata, hogy a kiválasztott mező rajta van-e a táblán, ill. azon a mezőn van-e egyáltalán korong (előfeltétel leellenőrzése).

Maga a főprogram a következőképpen néz ki:

```
void main()
{
  Harap harap;
  harap.vezerlo();
}
```

A Harap osztályt példányosítjuk, majd meghívjuk ezen új objektum **vezerlo()** metódusát:

A vezérlő:

```
void Harap::vezerlo()
{
  while (!celallapot())
  {
    if (jatekos==human)
    {
        lepes();                    //mit lép a játékos?
        jatekos=gep;
    }
    else
    {
        delay(1000);
        keres();                    //mit lép a gép?
        jatekos=human;
    }
    if (elofeltetel(sor, oszlop)) kiharap(sor, oszlop);
    rajz();
  }
  window(1,1,80,25);
  gotoxy(40,10);
  if (jatekos==human) cprintf("G Y O Z T E L ! ! !");
  else                cprintf("E N  G Y O Z T E M ! ! !");
}
```

A vezérlő egész addig dolgozik, míg nem teljesül a célfeltétel. Addig az egyes játékosoktól felváltva bekéri, hogy hova lépnek, azaz mely pozícióról vesznek le korongot. Ha ez egy szabályos lépés, vagyis teljesül az előfeltétele, akkor a megfelelő részt "kiharapja" a táblából (**kiharap** operátor alkalmazása), majd a módosult táblát újrarajzolja.

Mihelyt valaki felvette a bal felső korongot a vezérlő még tájékoztatást küld arról, hogy ki nyert, majd kilép.

Most már csak az a kérdés, hogy a gép hogyan választja ki a számára legkedvezőbb lépést, hiszen az ő célja ugyanaz, mint a miénk: nyerni szeretne. Mi az ő nyerő stratégiája?

```
void Harap::keres()
{
  int  keresx=-1, keresy=-1;

  while (keresx+1<N && tomb[keresx+1][0]!=ures) ++keresx;
  while (keresy+1<N && tomb[0][keresy+1]!=ures) ++keresy;

  if (keresx==0 && keresy==0)          //már csak a bal felső korong van fent
  {
    sor=0; oszlop=0;
    return;
  }
  if (keresx==keresy)
  {
    if (tomb[1][1]==korong)
    {
        sor=1; oszlop=1;
        return;
    }
    /* else if tomb[1][1]==ures */
    int szam=random(2);
    if (szam==0) sor=0, oszlop=keresy;
    else         sor=keresx, oszlop=0;
    return;
  }
  /* else if keresx!=keresy */
  if (tomb[1][1]==ures)
  {
    if (keresx<keresy) sor=0, oszlop=keresx+1;
    else               sor=keresy+1, oszlop=0;
    return;
  }
  /* else if tomb[1][1]==korong */
  if (keresx>1 && keresy>1)            // 2xn -esre hozzuk
  {
    sor=2, oszlop=0;
    return;
  }
  if (keresx==1)
  {
    if (tomb[1][keresy]==korong)
    {
        sor=1, oszlop=keresy;
        return;
    }
    /* else if tomb[1][keresy]==ures */
    int temp=-1;
    while (temp+1<N && tomb[1][temp+1]!=ures) ++temp;
```

```
    sor=0;
    if (temp==keresy-1) oszlop=temp+1;
    else                oszlop=temp+2;
    return;
  }


if (keresy==1)
  {
    if (tomb[keresx][1]==korong)
    {
        sor=keresx, oszlop=1;
        return;
    }
    /* else if tomb[keresx][1]==ures */
    int temp=-1;
    while (temp+1<N && tomb[temp+1][1]!=ures) ++temp;

    oszlop=0;
    if (temp==keresx-1) sor=temp+1;
    else                sor=temp+2;
    return;
  }
}
```

Ebben az "eljárásban" fogja a gép meghatározni, hogy mely lépést tegye meg. Akkor van nyerő stratégia, ha a korongok elrendezése négyzet alakú, ill. ha sikerül 2xn alakú elrendezést elérni (az nx2-es is jó).

Ha a humán játékos átengedi a kezdő lépést a gépnek, akkor a gép nyerni fog, hiszen az (1,1) pozíción lévő [Vigyázat! C-ben az indexelés 0-tól kezdődik!] korong felvétele után már csak a legelső sor ill. oszlop fog a táblán maradni, s ezután a gép mindig azt fogja lépni az ellentétes oldalon, amit a játékos. Bármit is lép a játékos, nála mindig kevesebb korong lesz, s a gép csupán egyenlőre hozza az ellentétes oldalt.

A 2xn-es alakú elrendezés esetén szintén az nyerhet, aki először lép. Ekkor két sor vagy oszlop közül mindig úgy kell lépni, hogy 1-gyel kevesebb korong legyen az adott sorban vagy oszlopban, mint a másikban.

A gép tehát vagy megpróbálja levenni az (1,1) koordinátájú korongot, vagy megpróbál egy 2xn-es elrendezést kihozni.

Futási eredmény:

```
  0 1 2 3 4 5 6 7 8
0 O O O O O O O O O
1 O O O O O O O O O
2 O O O O O O O O O
3 O O O O O
4 O O O O O
5 O O O O O
6 O O O O O
7 O O O O O
8 O O O O O
```

Adott egy 9x9-es tabla, melyen korongok vannak.
Egy korong felvetele maga utan vonja az alatta
levok, ill. a tole jobbra levok felvetelet is.
Cel: a bal felso zsetont az ellenfel vegye fel.

Sor: 3
Oszlop: 5