Colegiul Național ”Roman-Vodă”
Roman

JOCUL MINESWEEPER

Lucrare pentru obținerea atestatului profesional la informatică

Candidat,
Munteanu Andreea-Elena

Coordonator,
prof. Florin Moldovanu

2018

Cuprins

I. Argument

Am ales ca temă de studiu pentru lucrarea de atestat jocul Minesweeper pentru că în codul necesar realizării jocului sunt înglobate noțiuni teoretice fundamentale de programare în limbajul C++, precum implementare de matrice, recursivitate sau funcții booleene, care îmi atestă cunoștințele acumulate pe parcursul anilor de liceu în ceea ce privește domeniul informatic.

Am considerat că simularea unui joc va prezenta mai mult interes decât implementarea unui program obișnuit, având avantajul interactivității. Jocul Minesweeper, care are o istorie veche de mai bine de 50 de ani, fiind creat în jurul anilor 1960, este cunoscut publicului larg, fiind unul dintre jocurile de strategie consacrate ale tuturor timpurilor. De aceea, acest program poate prezenta interes oricărui jucător amator. Programul de față prezintă o versiune adaptată a jocului, care reproduce întocmai regulile și parcursul jocului clasic de Minesweeper.

II. Limbajul C++

1. Structura unui program C++

Ex :

```
#include <iostream.h>          //directiva preprocesor
void main()                    //antetul functiei principale
{                              //inceputul blocului
                               //parte declarativa
cout<< "Un exemplu ";          //partea procedurala care contine instructiuni ce descriu
//pasii algoritmului
cout<<"foarte simplu" ;
}                              //sfarsitul blocului
```

Instrucțiunile unui program C++ sunt grupate în entități numite *funcții.*Orice program trebuie să conțină cel putin o funcție, cu numele **main**, numită ***funție principală***. Ea se execută automat atunci când lansăm în execuție programul. Pe lângă funcția principală, un program poate să conțină oricâte alte funcții, numite ***funcții definite de utilizator***.

Orice funcție C++ este alcătuită din două părți :

- ***antetul*** funcției
- ***corpul*** funcției

Antetul funcției trebuie să fie de forma:

<tipul_valorii_returnate> <numele_functie> (<lista_parametrilor_formali>)

Parametrii formali sunt identificatori de variabile pe care le folosește funcția la execuție. Dacă funcția are parametri formali, atunci la apel trebuie să dăm valori concrete ale acestora, valori numite **parametri actuali (efectivi)**. O funcție poate să returneze o anumită valoare, de un anumit tip. Dacă o funcție nu returnează nicio valoare, atunci spunem că tipul valorii returnate este **void**(nimic).

Dacă nu există parametri, se vor scrie numai parantezele, sau cuvântul "void" între paranteze. Dacă tipul valorii returnate este void, el poate să lipsească.

Putem folosi și alte forme ale antetului, ca de exemplu: main(), void main(void), main(void).

Totalitatea instrucțiunilor unei funcții alcătuiesc **corpul funcției**. Acesta trebuie cuprins între acoladă deschisă și acoladă închisă, chiar dacă nu conține nicio instrucțiune. Limbajul C++ face distincție între minuscule și majuscule.

2. Directive preprocesor

Orice mediu de programare în C++ dispune de un set important de funcții predefinite. Ele au fost scrise de autorii limbajului, fac parte intrisecă din limbaj și pot fi folosite oriunde în program.

Toate funcțiile predefinite își au prototipul (definiția) în fișiere cu extensia **.h**, numite **header-e.** Pentru a folosi o funcție predefinită, trebuie să scriem la începutul programului o așa-numită **directiva preprocesor**, adică biblioteca din care face parte funcția respectivă.

Sintaxa unei directive preprocesor: **#include <nume_header>**

Ex:

#include <math.h> - sunt incluse functiile matematice
#include <iostream.h> - permite efectuarea citirilor/afisarilor

3. Vocabularul limbajului

Este format din: setul de caractere, identificatori, separatori și comentarii. **Setul de caractere** este alcătuit din:

- litere mari și mici ale alfabetului englez: a, b, …, z, A, B,…, Z
- cifre: 0, 1,…,9
- caractere speciale: + - * / = < > ( ) [ ] { } . , : ; ” ` ~ ! @ # $ % ^ & _ etc.

Identificatori: prin identificator se înțelege un nume asociat unei constante, variabile, proceduri, funcții, etc. El poate conține numai litere, cifre și caracterul special „_” (liniuța de subliniere) și trebuie să înceapă obligatoriu cu o literă.

Ex: a2, b, med_aritm

O categorie specială de identificatori este reprezentată de cuvintele-cheie ale limbajului. Ele au un înteles bine definit și nu pot fi folosite ca identificatori ai unor constante sau variabile definite de utilizator.

Ex: *define, include, if, else, case, for, while, do, char, int, return, struct,* etc.

Separatorii pot fi dupa caz: blanc (caracterul spațiu), caracterul ”;” .
Comentariile pot fi scrise în doua moduri:

- între secvențele de caractere: /* și */
- punând // în fața comentariului

4. Instrucțiunea vidă

Ex:

```
#include <iostream.h>
void main()
{
;
}
```

Corpul programului conține un singur caracter, și anume caracterul „**;”.** El joacă rol de instrucțiune vidă, nu are niciun efect, dar uneori prezența lui într-un program este necesară.

5. Constante, tipuri, variabile, operatori, expresii

1. **Constantele** sunt valori care nu se modifică pe parcursul execuției unui program. Ele se împart in :
   - **constante numerice**
     - întregi: 32, -158
     - reale : -45.88, 1. , .2, -2.5E-12

   Sintaxa: **const [tip] nume=valoare;**

Ex: const int numar=10; const numar=10; const float pi=3.14;

Constantele întregi aflate între limitele 0 si 32767 pot căpăta nume folosind tipul **enum**.

Ex: **enum timp {ieri, azi, maine};** am declarat 3 constante, *ieri*, *azi* și *maine*, inițializate implicit cu 0, 1, 2

- **enum timp {ieri, azi=3, maine=7};** *ieri* este inițializat cu 0
- **enum timp {ieri, azi=3, maine=azi};**
- **enum timp {ieri=7; azi};** *azi* va fi inițializat cu valoarea 8.

- **constante nenumerice**
  - *caracter* : Orice caracter are un cod ASCII propriu, adică un număr întreg cuprins între 0 si 255.
    - Caracterele litere mari ’A’, ..., ’Z’ au codurile ASCII cuprinse între [65,90]
    - Caracterele litere mici ’a’, …, ’z’ au codurile ASCII cuprinse între [97,122]
    - Caracterele cifra ’0’, ... , ’9’ au codurile ASCII cuprinse între [48,57]
    - Caracterele speciale sunt intercalate în tot intervalul [0,255]
  - **șir de caractere:** sunt șiruri de caractere cuprinse între ghilimele ”Borland C++ 7.0”

- **simbolice**: unei constante de orice tip i se poate atașa un nume. Ele se declară la început cu ajutorul directivei preprocesor ”#define”, sintaxa fiind **#define <nume> <valoare>. *Ex:*** #define pi 3.14

Limbajul C++ are câteva constante simbolice predefinite:

- MAXINT=32,767 (definită în header-ul **values.h**)
- MAXLONG=2,147,483,647 (definită în header-ul **values.h**)
- M_PI=3.1415927… (definită în header-ul **math.h**)

Constantele întregi cu sens logic (boolean) : limbajul C++ nu dispune de un tip de date logic, ADEVĂRAT și FALS, el interpretând orice valoare nenulă drept ADEVĂRAT. DacĂ valoarea este nulă, atunci condiția nu este indeplinită (FALS).

2. **<u>Tipurile de date</u>** standard ale limbajului C++ sunt:

1) tipuri întregi
2) tipuri reale

Tipurile întregi sunt:

- **unsigned char**: caracter fără semn, ocupă 8 biți și ia valori între [0,255]
- **char**: caracter, ocupă 8 biți, ia valori între [-128,127]
- **unsigned int**: întreg fără semn, 16 biți, ia valori între [0,65535]
- **short int**: întreg scurt, 16 biți, ia valori între [-32768,32767]
- **int** : întreg, ocupă de regulă 16 biți (numărul lor diferă de la o implemantare la alta) și ia valori între [-32767, 32767]
- **unsigned long**: întreg lung fără semn, 32 biți, ia valori între[0, 4 294 967 295]
- **long** : întreg lung cu semn, 32 biți, [-2 147 483 648, 2 147 483 647]

Tipurile reale sunt:

- **float** : ocupă 32 biți, ia valori între [$3.4*10^{-38}$, $3.4*10^{38}$]
- **double**: ocupă 64 biți, ia valori între [$1.7*10^{-308}$, $1.7*10^{308}$]
- **long double**: ocupă 80 biți, ia valori între [$3.4*10^{-4932}$, $3.4*10^{4932}$]

3. **<u>Variabile</u>** : variabila este o entitate caracterizată prin trei atribute: *tip, valoare, adresă*, cu proprietatea că atributul valoare se poate modifica.

În limbajul C++, declarațiile de variabile pot fi plasate oriunde în program.

Sintaxa: **<tip> <v1>, <v2>, <v3>,…;**

Pentru ca variabilele să fie văzute în tot programul, le putem declara în două locuri:

- la începutul programului, imediat după directivele preprocesor și înaintea funcției **main**; acestea se numesc **variabile globale**;
- la începutul funcției **main**, imediat după acolada deschisă; acestea se numesc **variabile locale** funcției main.

La declararea unei variabile, putem să o inițializăm:

- cu o valoare constantă;

- cu valoarea altei variabile
- cu valoarea unei expresii

Sintaxa: **<nume_variabila>=<valoare_initiala>**

Ex:

```
#include <iostream.h>
int a2,x,m=2;
float med, x, y=3.5, z=y, t=2*y-1;
void main()
{
int p; float w;
…
}
```

4. Operatori

a) operatori aritmetici: +, - , * , / (când deîmpărțitul și împărțitorul sunt întregi, x/y va fi de tip întreg), % (restul împărțirii întregi)

b) operatori relaționali : = = (egalitate), != (diferit), <=, >=, <, >

3>5 – rezultatul este 0 (fals)

3+7>=11-1 rezultatul este 1 (adevarat)

3= =3, rez 1; 3!=3 rez 0; 3!=4, rez 1

c) operatori logici:

- && (ȘI logic): dacă ambii operanzi sunt diferiți de 0, rezultatul este 1, altfel este 0: 3&&5=1; 0&&4=0; 5&&0=0
- || (SAU logic): dacă cel puțin unul dintre operanzi este diferit de 0, rezultatul este 1, altfel este 0: 0||3=1; 3||0=1; 0||0=0
- ! (negația): dacă operandul este valoare diferită de 0, rezultatul este 0, altfel este 1: !7=0; !0=1

d) operatorul de atribuire: **<nume_variabila>=<valoare>**

Atribuire multiplă: **<var1>=<var2>=<var3>=…=<valoare>**

Ex :

```
int a=2, b=3, x, y, z;

m=n=p=q=2*a-b;
```

Atribuiri cu sens logic: unei variabile de tip intreg i se poate atribui valoarea de adăvar a unei expresii logice.

```
int a=2, b=5, c=6, x,y;
x=(a<b&&b<c); rezultatul este 1 – adevărat
y=(a+b+c<=10); rezultatul este 0 – fals
```

e) operatorii +=, -=, *=, /=
Sintaxa: <var>+=<expr> ; <var>-=<expr> ; <var>*=<expr> ; <var>/=<expr> ;

x=4, y=5, z=7;
x=x+2; ⇔ x+=2; => x=6
x=x+y ⇔ x+=y;
x=x*2 ⇔ x*=2;
x=x/2 ⇔ x/=2;

f) operatorii ++ și - -
Sintaxa: <var>++ / <var>- - formă postfixată

++<var> / - - <var> formă prefixată

int x=3, y=5;
++y; y=5+1=6
z=x-2*y++; z=4-2*6=-8

g) operatorul " , " (virgula) exp1, exp2, exp3,…,expn;

int a=1, b=5; float c;
Expresia c=a=b+1, a=c+2, b=b+1; b=5+1=6, a=6, c=6, a=6+2=8, b=5+1=6
Expresia (în ansamblu) este de tipul int și are valoarea 6

h) operatorul conditional exp1?exp2:exp3

Se evaluează exp1. Dacă rezultatul este diferit de 0, se evaluează exp2, altfel se evaluează exp3.

i) operatorul sizeof sizeof(expresie) sizeof(tip)

Returnează numărul de octeți (1 octet=8 biți) utilizați pentru memorarea unei valori. Expresia nu este evaluată.

int a; sizeof(a)=2;
float a; sizeof(a)=4;
sizeof(float)=4;
int a; double b; sizeof (a+b)=8;
float x; sizeof(++x)=4

j) operatorul de conversie explicită: conversia unui întreg la tipul float se face în două moduri:

1. cu ajutorul operatorului punct:

int x=4, y=13;
float z;
z=(x+y)/2;

Deoarece x si y sunt intregi, rezulta ca z=(4+13)/2=17/2=8 va fi de tip întreg, și nu 8.5. Pentru a-l converti la float vom scrie: **z=(x+y)/2.** și atunci z=8.5

2. folosind operatorul de conversie explicita: (float) (int)

z=((float)x+y)/2; - s-a convertit x la float
z=(x+(float)y)/2; - s-a convertit y la float
z=(float)(x+y)/2; - s-a convertit valoarea expresiei (x+y)
float x=-1.9;
(int)x=-1;
(int) (++x+3)=(int)(-1.9+1+3)=(int)(2.9)=2

5. Expresii

O expresie este alcatuită din operanzi și operatori și se caracterizează printr-un rezultat numit valoarea expresiei. Operanzii sunt valorile care intră în calculul expresiei, iar operatorii desemnează operațiile care se execută în cadrul expresiei asupra operanzilor.

Tipul unei expresii reprezintă tipul valorii expresiei.

Într-o expresie, ordinea în care se execută operațiile este dată de **prioritatea operatorilor**:

- Operații aritmetice: * / + -
- Operații logice: ! && ||
- Operații relaționale

Expresiile aritmetice sunt cele care au ca rezultat un număr.

Expresiile logice sunt cele care au valoarea 1 în cazul în care condiția este adevarată, și 0 când este falsă. Folosesc operatorii relaționali și pe cei logici.

!1=0	!0=1
0&&0=0	0||0=0
0&&1=0	0||1=1
1&&0=0	1||0=1
1&&1=1	1||1=1

III. JOCUL MINESWEEPER

1. Introducere. Reguli de bază

Minesweeper este un joc de strategie single-player care are ca obiectiv deschiderea celulelor ”sigure” ale tablei de joc (care nu conțin bombe) și intuirea locațiilor unor bombe plantate pe câmpul de joc, fără a le detona. Jocul oferă ca indicii numărul de bombe situate pe pozițiile adiacente ale unei celule.

Inițial, grila este alcătuită din celule identice. Unele dintre aceste celule conțin, în mod aleatoriu, bombe, pe care jucătorul trebuie să le evite. În majoritatea cazurilor, dimensiunea grilei și numărul bombelor plantate este ales în prealabil de către jucător.

Jucătorul trebuie să deschidă apoi celule de pe tabla de joc. Dacă o celulă conține o bombă, el pierde jocul. Dacă, în schimb, celula este ”sigură”, în locul acesteia va apărea un număr indicând numărul de celule adiacente care conțin bombe; în cazul în care acest număr este 0, toate celulele adiacente ”sigure” vor fi descoperite, în mod recursiv. Jucătorul folosește aceste informații pentru a deduce conținutul celorlalte celule și pentru a deschide noi celule considerate ca fiind ”sigure”.

În unele versiuni ale jocului (inclusiv cea de față), în cazul în care, la prima mutare a sa, jucătorul deschide o celulă care conține o bombă, aceasta va fi mutată într-o altă celulă a grilei, astfel încât jocul să poată continua.

În plus, jucătorul are posibilitatea de a plasa steguleț care să marcheze celulele ”periculoase”, potențiale locații ale bombelor, ușurând astfel alegerea unei noi celule ce urmează a fi deschisă.

Jucătorul câștigă în momentul în care a deschis toate celulele ”sigure”, cu excepția celor în care au fost plantate bombe, sau când toate stegulețele (egale ca număr cu numărul de bombe) au fost plasate pe tabla de joc.

2. Prezentarea programului

Programul de față își propune simularea jocului Minesweeper, având ca limbaj de programare limbajul C++. El va rula în Code::Blocks IDE 13.12, iar interfața va consta în fereastra de output a platformei, în modul consolă.

Programul pune la dispoziție trei niveluri de dificultate diferite:

- *începător* (grilă de 9x9 celule și 10 bombe plantate);
- *mediu* (grilă de 16x16 și 40 bombe plantate);
- *avansat* (grilă de 24x24 și 99 bombe plantate).

Grila va fi construită pe baza nivelului ales, în funcție de dimensiunea câmpului de joc și a numărului de bombe ce trebuie plantate. Ea va fi simulată cu ajutorul unei matrice, a cărei dimensiune variază în funcție de nivelul de dificultate ales de jucător.

Inițial, grila este alcătuită din celule identice între ele, marcate cu simbolul ”■ ”(fig.1), care pot ascunde fie bombe (plantate cu ajutorul unei funcții **plantare_bombe()**, de tip void, fie numărul de bombe adiacente poziției deschise.

```
322   // functie pentru initiliazarea jocului
323   void initializare(char realBoard[][MAXSIDE], char myBoard[][MAXSIDE])
324   {
325       srand(time (NULL));
326       // initial, nicio celula nu are bomba
327       for (int i=0; i<SIDE; i++)
328           for (int j=0; j<SIDE; j++)
329               myBoard[i][j]=realBoard[i][j]=liber;
330       /*char liber=220; //patratele libere vor avea ca simbol un patratel
331       plin in matricea afisata in consola*/
332   }
```

fig. 1

Înainte de o nouă mutare, se va întreba dacă se dorește plasarea sau ștergerea unui steguleț dintr-o celulă oarecare. Pentru aceasta, am definit funcția **plantare_stegulet()** (vezi fig. 2). Răspunsurile au fost codate cu 0 pentru ”NU” și 1 pentru ”DA”. Jucătorul va trebui să introducă de la tastatură una din aceste valori. Dacă celula respectivă conține deja un steguleț, atunci acesta va fi șters

```
void plantare_stegulet(int x, int y, char myBoard[][MAXSIDE], int &tip)
{
    /* variabila tip:
            1 - se adauga un stegulet
            0 - se sterge un stegulet
    */
    if (myBoard[x][y]==liber)
        myBoard[x][y]=stegulet, tip=1;
    else if (myBoard[x][y]==stegulet)
        myBoard[x][y]=liber, tip=0;
}
```

fig. 2

La fiecare nouă mutare, realizată prin introducerea coordonatelor (linie, coloană) a matricii, în locul simbolului ” ■ ” va apărea fie ” * ”, simbolizând faptul că în locația respectivă a fost plasată o bombă care acum va detona, fie un număr care reprezintă numărul de celule adiacente, care sunt locații ale unor bombe. În cazul în care acest număr este 0, vor fi descoperite toate celulele adiancente ”sigure” (care au, la rândul lor, valoarea 0) și valide (care nu depășesc limitele matricii – fig. 3) ale celulelor sigure, în mod recursiv.

```
20  //verifica daca linia si coloana unei celule date
21  //sunt corecte (nu depasesc limita)
22  bool valid(int lin, int col)
23  {
24      return (lin>=0 && lin<SIDE && col>=0 && col<SIDE);
25  }
```

fig. 3

Pentru cazul în care, la deschiderea unei celule este descoperit un număr, am definit o funcție **nr_bombe_vecine()** (vezi fig. 4)**,** de tip int, care primește ca parametri linia și coloana celulei *(int lin, int col)* pentru care se dorește calcularea numărului bombelor adiacente, matricea *a*, reprezentând tabla de joc în momentul mutării, și o matrice bombe[][2]. Aceasta verifică pentru fiecare dintre cele 8 celule adiacente, în sensul acelor de ceasornic începând cu celula din nord, dacă este bombă (pentru aceasta, am definit funcția booleană **este_bomba()** [fig.5], care verifică dacă un element *(lin,col)* din matrice este ‘*’. Dacă da, atunci funcția va returna 1, sau 0 în caz contrar).

```
61  //functie care sa numere cate bombe se afla printre vecinii unei celule de pe tabla a[][]
62  int nr_bombe_vecine(int lin, int col, int bombe[][2], char a[][MAXSIDE])
63  {
64      int nr=0;
65
66      /*
67               N.V   N   N.E
68                 \   |   /
69                  \  |  /
70               V---Celula--E
71                  /  |  \
72                 /   |   \
73               S.V   S   S.E
74
75
76          N  -->  Nord        (lin-1, col)
77          N.E-->  Nord-est    (lin-1, col+1)
78          E  -->  Est         (lin,   col+1)
79          S.E-->  Sud-est     (lin+1, col+1)
80          S  -->  Sud         (lin+1, col)
81          S.V-->  Sud-vest    (lin+1, col-1)
82          V  -->  Vest        (lin,   col-1)
83          N.V-->  Nord-vest   (lin-1, col-1)
84
85      */
86
87      //---------(1) celula din N----------
88
89      if (valid(lin-1,col)) //daca celula din N este valida
90      {
91          if (este_bomba(lin-1,col,a)) //daca este bomba
92              nr++;
93      }
94      //si analoagele:
```

fig. 4

```
27    // verifica daca o celula contine o bomba
28    bool este_bomba(int lin, int col, char a[][MAXSIDE])
29    {
30        /* 1 - bomba
31           0 - celula sigura
32        */
33        return (a[lin][col]=='*');
34    }
```

fig. 5

Dacă, în cadrul funcției **nr_bombe_vecine()**, o celulă adiacentă celulei (lin,col) conține o bombă, atunci se mărește cu o unitate o variabilă *nr* inițializată local cu valoarea 0. Funcția va returna la sfârșitul apelului numărul de celule din jurul celei deschise care au plantată câte o bombă.

Dacă o celulă deschisă conține cifra 0, atunci se vor descoperi toate celulele care ”sigure” adiacente, în mod recursiv.

Funcția booleană **playMinesweeper()**, definită în acest sens, care reprezintă o mutare în jocul Minesweeper, acoperă toate cazurile posibile (când celula deschisă conține “*” sau un număr natural), inclusiv cazul mai complex în care celula ascunde valoarea 0.

- Dacă celula deschisă ascunde simbolul “*”, înseamnă că în această locație se află plantată o bombă, care, la deschiderea celulei, s-a detonat. Se afișează mesajul ”Ați pierdut!”, se iese din recursie, iar jocul se încheie.
- Dacă celula conține valoarea 0, se calculează numărul de bombe vecine adiacente și se actualizează matricea-interfață, care va primi pe poziția (lin, col) a celulei deschise numărul respectiv. Dacă acest număr este egal cu 0, înseamnă că nu există nicio bombă printre cei 8 vecini ai celului, iar programul verifică pentru fiecare dintre aceștia dacă este valid (adică dacă linia și coloana nu depășesc limitele matricii), iar apoi, dacă este bombă. În caz negativ, se apelează **playMinesweeper()** pentru indicii (lin,col) ai vecinului respectiv, în mod recursiv, până când se ajunge pe o poziție din matrice care conține o valoare nenulă, ca în fig. 6.

```
        Tabla curenta :
    0 1 2 3 4 5 6 7 8
0   ■ ■ ■ ■ ■ ■ ■ ■ ■
1   ■ ■ ■ ■ ■ ■ ■ ■ ■
2   ■ ■ ■ ■ ■ ■ ■ ■ ■
3   ■ ■ ■ ■ ■ ■ ■ ■ ■
4   ■ ■ ■ ■ ■ ■ ■ ■ ■
5   ■ ■ ■ ■ ■ ■ ■ ■ ■
6   ■ ■ ■ ■ ■ ■ ■ ■ ■
7   ■ ■ ■ ■ ■ ■ ■ ■ ■
8   ■ ■ ■ ■ ■ ■ ■ ■ ■

        Doriti sa plantati/stergeti un stegulet?
(1 - DA, 0 - NU)
0
(lin, col): 1 1
        Tabla curenta :
    0 1 2 3 4 5 6 7 8
0   0 0 0 1 ■ ■ ■ ■ ■
1   0 0 0 1 ■ ■ ■ ■ ■
2   1 1 2 1 ■ ■ ■ ■ ■
3   ■ ■ ■ ■ ■ ■ ■ ■ ■
4   ■ ■ ■ ■ ■ ■ ■ ■ ■
5   ■ ■ ■ ■ ■ ■ ■ ■ ■
6   ■ ■ ■ ■ ■ ■ ■ ■ ■
7   ■ ■ ■ ■ ■ ■ ■ ■ ■
8   ■ ■ ■ ■ ■ ■ ■ ■ ■
```

fig. 6

Pentru a ne asigura că niciodată nu vom pierde jocul de la prima mutare, deschizând la începutul jocului o celulă ce conține o bombă, am definit funcția **inlocuieste_bomba()** (vezi fig. 7), care, pentru poziția (lin,col) dată, caută următoarea locație vacantă din tabla de joc unde ar putea planta bomba și o plasează acolo, poziția ocupată inițial de bombă devenind astfel ”sigură”. Această îmbunătățire constituie unul dintre lucrurile care face distincția între o versiune a jocului Minesweeper și o alta. Alte astfel de diferențe sunt:

- folosirea de steguleț care să ușureze mutările ulterioare ale jucătorului (cazul de față);
- câștigarea jocului în momentul în care toate steguleț au fost plasate pe toate pozițiile bombelor, nemaifiind necesară deschiderea tuturor celulelor ”sigure”;
- posibilitatea de a relua același joc, cu aceeași amplasare a bombelor, în cazul în care la prima mutare se detonează o bombă; nu se mai protejează prima celulă deschisă, ca în cazul implementării de față.

```
// functie care sa mute bomba de pe pozitia (lin,col) pe o pozitie vacanta
void inlocuieste_bomba(int lin, int col, char a[][MAXSIDE])
{
    int ok=0;
    for (int i=0; i<SIDE && ok==0; i++)
    {
        for (int j=0; j<SIDE && ok==0; j++)
            {
                // Gaseste prima celula care nu contine o mina si plaseaza o bomba acolo
                if (a[i][j] != '*')
                {
                    a[i][j] = '*';
                    a[lin][col] = liber;
                    ok=1;
                }
            }
    }
}
```

fig. 7

3. Utilizarea programului

Programul folosește ca element de inferfață fereastra de output a platformei Code::Blocks (modul consolă).

Primul pas este alegerea nivelului de dificultate. Utilizatorul are de ales dintre trei niveluri de dificultate diferită, care diferă între ele prin dimensiunile matricii (a câmpului de joc) și prin numărul de bombe, care este cu atât mai mare cu cât este mai ridicat nivelul de dificultate. Se va introduce de la tastatură una dintre valorile 1,2 sau 3, care corespund nivelurilor ÎNCEPĂTOR, MEDIU sau AVANSAT. În cazul în care valoarea introdusă diferă de oricare din valorile 1,2 sau 3, jucătorului i se permite să introducă o nouă valoare, până când aceasta este corectă. Acest lucru este posibil datorită funcției **alege_dificultate()** (vezi fig. 8). Ea funcționează ca un meniu (funcția predefinită switch), care, în funcție de nivelul de dificultate ales, stabilește dimensiunile tablei de joc și numărul de bombe de pe aceasta.

```
void alege_dificultate()
{
    /*NIVEL:
    INCEPATOR -->   9 * 9 = 81      celule si 10 bombe
    MEDIU -->       16 * 16 = 256   celule si 40 de bombe
    AVANSAT -->     24 * 24 = 576   celule si 99 de bombe
    */
    int level;
    cout<<"\tIntroduceti nivelul: \n\n";
    cout<<"Tastati 1 pentru INCEPATOR (9*9 celule si 10 bombe)\n";
    cout<<"Tastati 2 pentru MEDIU (16*16 celule si 40 bombe)\n";
    cout<<"Tastati 3 pentru AVANSAT (24*24 celule si 99 bombe)\n\n";
    // asiguram ca ca nivelul ales este corect
    int introducereNivel=0;
    do
    {
        introducereNivel++;
        if (introducereNivel==1)
            cout<<"Nivelul ales: ";
        else cout<<"Nivelul ales este gresit! Introduceti din nou: ";
        cin>>level;
        cout<<'\n';
    }
    while (level<=0 || level>3);
    switch(level) //in functie de nivel, setarea dimensiunile tablei de joc
    {
    case 1:
        SIDE=9;
        NRBOMBE=10;
        break;
    case 2:
        SIDE=16;
        NRBOMBE=40;
        break;
    case 3:
        SIDE=24;
        NRBOMBE=99;
        break;
    }
}
```

fig. 8

În continuare, programul oferă posibilitatea jucătorului de a plasa pe tabla de joc un steguleț într-o poziție considerată ”periculoasă” sau, dimpotrivă, să șteargă un steguleț deja amplasat într-o poziție oarecare. Jucătorul trebuie să introducă înaintea fiecărei mutări valoarea 1 (”DA”) și 0 (”NU”) (fig. 9).

```
        Doriti sa plantati/stergeti un stegulet?
(1 - DA, 0 - NU)
1
(lin, col): 1 1
        Tabla curenta :
    0 1 2 3 4 5 6 7 8
0   ■ ■ ■ ■ ■ ■ ■ ■ ■
1   ■ ^ ■ ■ ■ ■ ■ ■ ■
2   ■ ■ ■ ■ ■ ■ ■ ■ ■
3   ■ ■ ■ ■ ■ ■ ■ ■ ■
4   ■ ■ ■ ■ ■ ■ ■ ■ ■
5   ■ ■ ■ ■ ■ ■ ■ ■ ■
6   ■ ■ ■ ■ ■ ■ ■ ■ ■
7   ■ ■ ■ ■ ■ ■ ■ ■ ■
8   ■ ■ ■ ■ ■ ■ ■ ■ ■
```

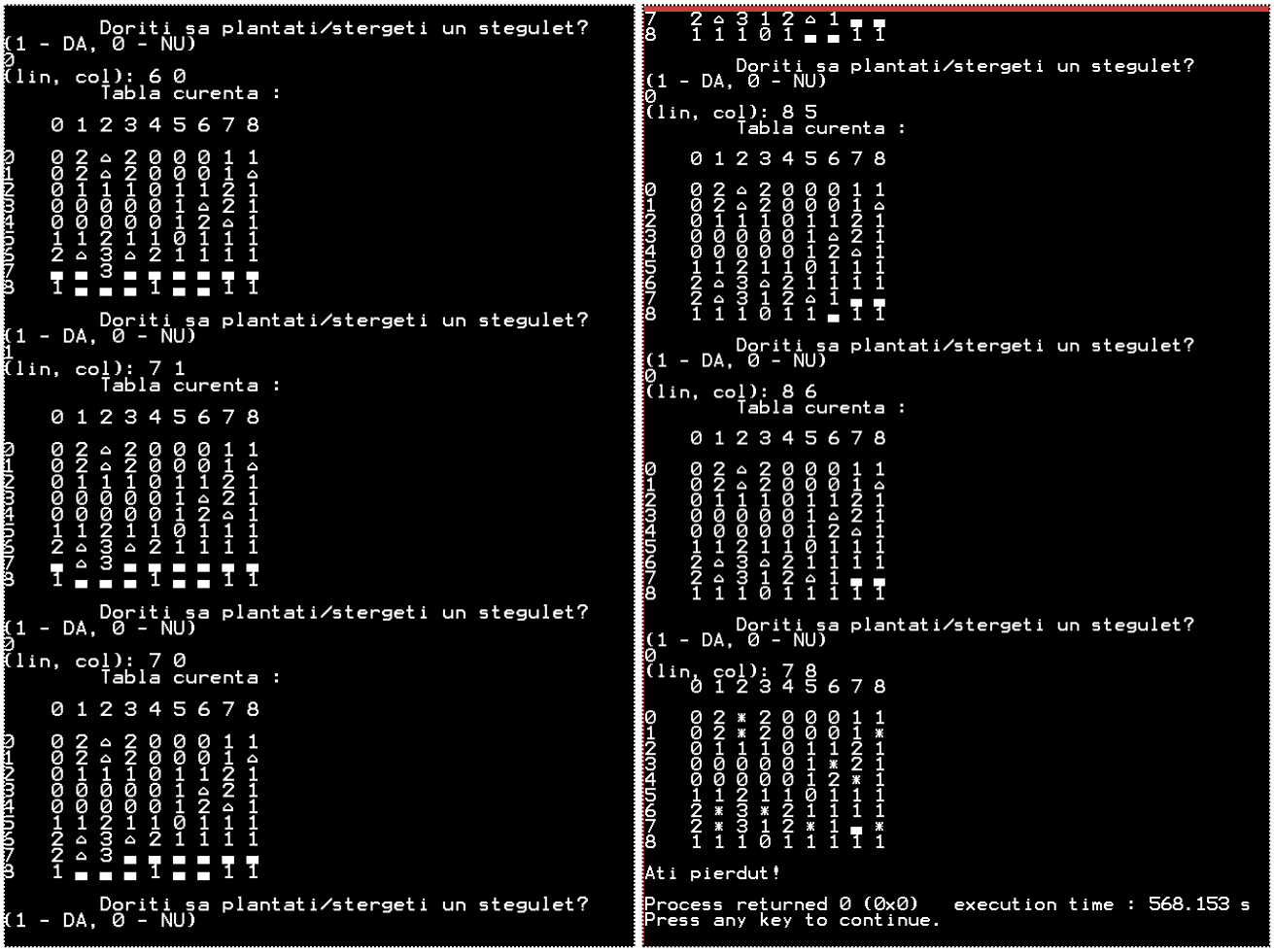
fig. 9

Următoarea etapă este cea a introducerii coordonatelor unei celule ce se dorește a fi descoperită, în formatul (linie, coloană). Dacă celula deschisă conține o bombă, jocul se încheie, iar pe ecran apare mesajul ”Ati pierdut!”.

Pentru a juca efectiv Minesweeper, se apelează în programul principal funcția play_Minesweeper(), de tip void, care rămâne în recursie atâta timp cât numărul mutărilor nu a ajuns la 0 (acesta fiind inițializat la început cu SIDE * SIDE − NRBOMBE, unde SIDE reprezintă dimensiunea matricii, iar NRBOMBE reprezintă numărul de bombe plantate pe câmpul de joc), și cât timp nu s-a detonat nicio bombă.

Jocul va continua, conform algoritmului, până când se va detona o bombă sau atunci când numărul mutărilor se va epuiza. Pe ecran se afișează un mesaj corespunzător:

- ”Ați câștigat! Felicitări!” pentru cazul când jocul a fost câștigat
- ”Ați pierdut!”, dacă jocul a fost pierdut.



IV. Bibliografie

1. https://infoprofa.wikispaces.com: https://infoprofa.wikispaces.com/file/view/Elementele+de+baza+ale+limbajului+C.doc
2. Emanuela Cerchez, M. Ș. (2005). *Programarea în limbajul C/C++ petru liceu.* Iași: POLIROM.
3. *Minesweeper (video game)*. Preluat de pe https://en.wikipedia.org/wiki/Main_Page: 1. https://en.wikipedia.org/wiki/Minesweeper_(video_game)
4. *Minesweeper in C++*. Preluat de pe https://codereview.stackexchange.com: https://codereview.stackexchange.com/questions/141717/minesweeper-in-c