PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **12** (1984/1985)
Številka 1
Strani 4–7

Roman Rojko:

ŠE O IGRI ŽIVLJENJA

Ključne besede: računalništvo, igra, algoritem.

Elektronska verzija: http://www.presek.si/12/694-Rojko.pdf

ŠE O IGRI ŽIVLJENJA

1. Uvod

Zaradi velikega zanimanja, ki sta ga povzročila članka o igri življenja v Preseku, sem sklenil povedati še kaj več o njunem nastanku.

Za igro življenja sem prvič slišal pred mnogimi leti v družbi prijateljev *Pod lipo*. Pritegnila me je, zato sem podrobnosti poiskal v reviji *Scientific American*, kjer jo je *Martin Gardner* opisal v svoji rubriki *Mathematical Games*. Svoje radovednosti pa s tem nisem potešil, vendar sem jo moral kar več let krotiti, dokler nisem končno dobil dostopa do primernega računalnika in s tem možnosti za raziskovanje zanimivega življenja celic.

2. Izvedba

Mikroračunalnik ID–80 se je izkazal kot nalašč za moje potrebe. Zaslon omogoča prikazovanje 96 krat 160 kvadratkov (celic), poleg tega pa nudi ta računalnik možnost programiranja v zbirniku. To pa je pomembna ugodnost, saj dosežemo s programiranjem v zbirniku največjo hitrost izvajanja. Ostali programski jeziki so večinoma vgrajeni v obliki interpreterjev, ti pa so znani kot zelo počasni.

S sodelavcem Štefanom sva se lotila dela in nastal je program, ki zmore izračunati in narisati nov rod celic v približno dveh sekundah. Tako sem se lahko lotil raziskovanja. Že znane osnove igre življenja in večina novih odkritij pa so našli prostor v obeh člankih. Slike sem narisal tako, da sem kolonijo celic na računalnikovem zaslonu fotografiral na črno-bel film in tako dobljene negative je Presek natisnil kot diapozitive (črne celice na belem ozadju). Zaslon je bil precej temen, da je slika lahko imela zadovoljivo ostrino, zato pa je bil osvetlitveni čas kar okoli dve sekundi. Fotoaparat sem seveda moral pritrditi na stojalo. Ker pa mi ti posnetki še niso zadoščali, sem posnel tudi film s kinokamero (super 8 mm film). Vsak rod celic sem posnel na eno sliko filma. Amaterski filmi se praviloma vrtijo s hitrostjo 18 slik na sekundo, se pravi 18 rodov na sekundo, to pa je že kar fantastična naglica, ki omogoča prav svojevrsten pogled v igro življenja. S tem so soglašali tudi udeleženci seminarja za računalništvo in uporabno matematiko, ko sem jim film predvajal. Hitrost seveda z lahkoto zmanjšamo s počasnejšim predvajanjem ali pa tako, da posnamemo isti rod celic večkrat zapored. Tako lažje spremljamo podrobnosti v razvoju igre.

Naj sedaj priznam, da ne vem za nobeno literaturo, kjer bi bila igra življenja bolj obširno razložena kot v Preseku. To seveda ne pomeni, da take literature ni. Zato prosim vsakogar, ki bi nanjo naletel, da me o tem obvesti preko Presekovega uredništva. Igra življenja me še vedno vznemirja in želim odkriti še katero od njenih skrivnosti. Predvsem bi se rad podal v barvno posplošitev

igre. Zato pa bi potreboval dovolj močan računalnik z barvno grafiko in pa "neizmerne" količine časa. Niti ne vem natančno, kateri od obeh problemov je hujši. Če pa bom pri tem vendarle uspel, bo za to seveda prvi zvedel Presek. (Zato ti svetujem, da ostaneš še dolgo njegov naročnik!)

3. Šolski algoritem

Oglejmo si sedaj algoritem za izračunavanje rodov v igri življenja. Sam postopek je sila enostaven. Potrebujemo dve matriki (imenujmo ju p in q), kamor bomo shranjevali kolonije celic. Vsak element matrike naj ustreza kvadratku (celici) na zaslonu in ga kot po navadi desežemo z navedbo vrstice (i) in stolpca (j), namreč $p[i,j]$. Vrednost elementa naj bo 0, če je kvadratek prazen, in 1, če je tam celica. Sedaj se moramo pomeniti še o tem, kako bomo prikazovali celice na zaslonu. To je odvisno od tega, kakšne ukaze pozna računalnik v te namene. V najenostavnejšem primeru lahko, recimo, prižgemo celico v i-ti vrstici in j-tem stolpcu na zaslonu z ukazom *plot*(i,j), ugasnemo pa jo z *unplot*(i,j). Ta dva ukaza bomo uporabili tudi mi. Na sploh pa vlada na tem področju med računalniki precejšna zmeda in naj bo zato dejanska izvedba prepuščena programerju samemu.

Naj imata matriki p in q mi vrstic in mj stolpcev. Računalniku ju bomo predstavili takole (algoritem bomo opisali v programskem jeziku PASCAL):

```
VAR p, q: ARRAY [1..mi,1..mj] OF 0..1
```

Na tem mestu pa moramo omeniti še robni problem, ki se pojavi pri programiranju. Celice na robu matrike namreč ne morejo imeti polnega števila sosed. V teoriji se s tem problemom nismo ukvarjali, saj smo se igrali življenje na neomejeni ploskvi. Računalnikov zaslov in pomnilnik pa sta še kako omejena. Najenostaveje rešimo ta problem tako, da na robu celic sploh nočemo rojevati niti moriti. Zanki za vrstice in stolpce bosta zato v programu tekli od drugega pa do predzadnjega. Robni problem bi lahko dobil tudi drugačne rešitve, a se zdaj ne bomo ubadali z njimi.

Naslednji del programa bo iz starega rodu (matrika p) izračunal naslednji rod (matrika q) in sproti novi rod narisal na zaslon:

```
FOR i := 2 TO mi - 1 DO
FOR j := 2 TO mj - 1 DO
BEGIN
    (* štetje sosed *)
    a := 10 * p[i,j] +
        p[i-1,j-1] + p[i-1,j] + p[i-1,j+1] +
        p[i+1,j-1] + p[i+1,j] + p[i+1,j+1] +
        p[i,j-1] + p[i,j+1];
```

```
        (★ rojevanje in odmiranje celic ★)
        IF (a = 3) OR (a = 12) OR (a = 13)
        THEN BEGIN q[i,j] := 1; plot(i,j) END
        ELSE BEGIN q[i,j] := 0; unplot(i,j) END
    END;
    (★ prepis novega rodu v matriko p ★)
    p := q
```

Števec sosed (spremenljivka *a*) je dvomesten. Enice povedo, koliko sosed ima kvadratek v *i*-ti vrstici in *j*-tem stolpcu, desetice pa, če v tem kvadratku celica sploh je.

Program mora seveda vsebovati tudi primeren način vnašanja (in popravljanja) kolonij celic na zaslonu. Vnos celic z navajanjem stolpcev in vrstic je zgolj izhod v sili.

Tega programa ni težko predelati za kak drug programski jezik. Opozorim pa naj, da bo ta program na mikroračunalniku zelo počasen. Program v PASCALU na računalniku ID–80 je bil vsak 60-krat počasnejši kot program v zbirniku. To razmerje se z razvojem sicer izboljšuje, vendar je še vedno velika razlika, če čakaš na nov rod dve ali pa dvajset sekund.

4. Izboljšave algoritma

Omenili bomo tri izboljšave. Dve bosta skrajšali čas računanja, tretja pa bo varčevala z računalnikovim pomnilnikom.

Najprej bomo zamenjali matriko z dvema indeksoma z matriko, ki bo imela samo en indeks (vektor), a seveda še vedno *mi* krat *mj* elementov. Primerjajmo oba načina indeksiranja med seboj:

dva indeksa	en indeks
$i-1,j-1$	$k-mi-1$
$i-1,j$	$k-mi$
$i-1,j+1$	$k-mi+1$
$i,j-1$	$k-1$
i,j	k
$i,j+1$	$k+1$
$i+1,j-1$	$k+mi-1$
$i+1,j$	$k+mi$
$i+1,j+1$	$k+mi+1$

Ta ideja ponuja tudi simpatično rešitev problema levega in desnega roba. Edini indeks namreč pri računanju teče od začetka do konca in se ne ozira na vrstice, kar na koncu učinkuje tako, kot da bi življenjski prostor igre zvili v valj,

tako da se vsaka vrstica naravnost nadaljuje v naslednjo spodnjo, levega in desnega roba pa sploh ni. Potnik, ki bi zlezel čez desni rob zaslona, bi se avtomatično pojavil na levi strani. Problem zgornjega in spodnjega roba se seveda ne spremeni.

V prejšnjem algoritmu mora računalnik vsak element matrike poiskati 9 krat (za celico in njenih 8 sosed). To število lahko zmanjšamo na eno tretjino z uvedbo treh števcev (*a*, *b* in *c*). Poskusil bom opisati, kako to izgleda. Izberimo si poljubno celico, ki naj ji pripada indeks *k*. Števec *c* naj prešteje vse celice v kvadratkih *k*, *k*−*mi* in *k*+*mi* (torej celico samo z zgornjo in spodnjo sosedo). Nato prenesemo števec *c* v *b* in se premaknemo na naslednji kvadratek z indeksom *k*+1. Sedaj prištejmo števcu *b* zgornjo in spodnjo sosedo, ga prenesemo v števec *a*, se premaknemo na kvadratek *k*+2 in števcu *a* prištejemo celico z zgornjo in spodnjo sosedo. Na koncu vsebuje števec *a* vsoto vseh sosed kvadratka *k*+1. Za lažje razumevanju zapišimo to v obliki programa:

```
VAR p, q: ARRAY[1..max] OF 0..1;
...
b := 0; c := 0;
FOR k := mi + 1 TO max−mi DO
BEGIN a := b; b := c; c := 0;
    IF p[k−mi] = 1 THEN BEGIN a := a+1; b := b+1; c := c+1 END;
    IF p[k] = 1 THEN BEGIN a := a+1; b := b+10; c := c+1 END;
    IF p[k+mi] = 1 THEN BEGIN a := a+1; b := b+1; c := c+1 END;
    IF (a = 3) OR (a = 12) OR (a = 13)
    THEN q[k−1] := 1
    ELSE q[k−1] := 0
END
```

Oglejmo si sedaj še, kako lahko varčujemo s pomnilnikom. Dve matriki smo potrebovali, da smo lahko iz starega izračunali nov rod, pri tem pa se stari rod ni smel pokrivati. Očitno pa lahko eno matriko zamenjamo s tremi vrsticami, saj potrebujemo za vsako celico naenkrat le njene sosede. Nov rod lahko tako začasno zapisujemo v te tri vrstice, ko pa se pomikamo navzdol, pa lahko nov rod že prepisujemo v matriko nazaj. Če smo prej potrebovali v pomnilniku prostora za 2.*mi.mj* celic, pa zadošča sedaj že *mi.mj*+3.*mi* celic, kar se pri manjših računalnikih kar krepko pozna.

Rado se zgodi in to bomo slej ko prej ugotovili, da je življenjski prostor celic občutno premajhen. Vendar ima zaslon omejene grafične možnosti. Če pa je računalnik dovolj velik in hiter, bi lahko imeli v pomnilniku veliko večji prostor, "skozi" zaslon pa bi opazovali le izbran del prostora. Vedeti pa moramo, da se z večjim življenjskim prostorom poveča čas za izračun novega rodu. In tudi pomnilnik ni ravno neomejeno velik. Morda bo zopet treba zadevo preložiti na kasnejši čas, ko bo tehnologija postregla z močnejšimi in, upajmo, tudi bolj dosegljivimi računalniki.

Roman Rojko