Hra života (Game of Life)

Vojtěch Brtník

Zápočtový program z předmětu NPRG005
Neprocedurální programování
Akademický rok 2008/2009
letní semestr

Obsah

1 Uživatelská dokumentace

> **Cíl.** *Naprogramujte v jazyce Prolog simulaci Hry života, resp. jednu její variantu s možností drobné konfigurace. Program bude kompletně v češtině, včetně zdrojového kódu.*

1.1 Hra života

Hra života byla představena v roce 1970 Johnem Conwayem a někdy se jí proto také říká *Conway's Game of Life*. Ve skutečnosti se jedná o konečný celulární automat. Výpočet je neinteraktivní, probíhá bez aktivní účasti hráče a je závislý pouze na vstupních parametrech. Ve 2-D variantě této hry, kterou se my budeme zabývat, se simulace odvíjí na obdélníkové mřížce, kterou si je možno představit jako čtverečkovaný papír. V každé buňce se může, a nemusí nacházet živá bytost, přičemž v kroku hry dojde v každé buňce k právě jedné z následujících čtyř možností:

- živá bytost zemře,
- živá bytost přežije,
- v prázdné buňce se narodí nová bytost, nebo
- buňka zůstane prázdná.

Která z těchto možností nastane je určeno jednak momentální obsazeností dané buňky, ale také počtem obsazených buněk v přímém sousedství (každá neokrajová buňka má osm sousedů). Hra je motivována takto: pokud je sousedů příliš málo, buňka zemře na osamění. Pokud je sousedů přiměřeně, buňka přežije nebo se dokonce narodí nová. Pokud je sousedů příliš, buňka zemře na přelidnění. Při vhodném počátečním rozmístění živých bytostí a volbě kritických konstant osamění, přelidnění a narození se prostředí chová velmi nečekaně a vytváří obrazce, díky kterým je Hra života známá. Na okrajích mají buňky menší počet sousedů, a proto se chovají odlišně, než uprostřed hracího pole.

Pojďme se na hru podívat jako na konečný automat. Kritické konstanty určují přechodovou funkci, v závislosti na nich se totiž rozhodne, do jakého stavu se buňka v každém kroku přesune. Počáteční stav je dán seznamem obsazených buněk na začátku hry. Množina všech stavů je dána všemi možnými kombinacemi (ne)obsazenosti buněk, do kterých se teoreticky může výpočet dostat. Protože každé políčko je vždy buď obsazené, nebo neobsazené, je počet možných stavů celé mřížky

$$2^{\text{Počet sloupců} \times \text{Počet Řádků}}.$$

Automat je deterministický. Množina koncových stavů automatu je prázdná.

1.2 Vstup

1. počet sloupců hrací mřížky
2. počet řádků hrací mřížky
3. seznam obsazených buněk v počátečním stavu, buňka se adresuje dvojicí [sloupec, řádek]
4. počet generací, které mají být spočítány a vytisknuty na obrazovku
5. konstanty udávající počet sousedů, při kterých se v neobsazené buňce narodí buňka nová
6. konstanty udávající počet sousedů, při kterých obsazená buňka nezahyne

1.3 Výstup

Výstupem je výpis jednotlivých generací na obrazovku. Programování v jazyce Prolog bohužel není příliš interaktivní a ani rozhraní není děláno pro krásu. Tomu také odpovídá výstup našeho programu. Čtenář si s tím musí poradit sám. Obvyklým postupem bývá zpracování výsledků Prologu nějakým procedurálním jazykem a jejich zobrazení v přívětivější podobě. Tímto se zde ale zabývat nebudeme.

1.4 Varianty hry

Standardní a nejvíce prozkoumanou variantou je situace, v níž jsou konstanty rozděleny následovně:

- V neobsazené buňce se narodí nová bytost právě tehdy, když je počet obsazených sousedů roven třem.
- Bytost v buňce přežije, pokud je počet obsazených sousedů roven dvěma nebo třem.
- V ostatních případech tedy buňka zůstane neobsazená, případně živá bytost zahyne.

Hra má mnoho modifikací a čtenář se může setkat se spoustou variant. Pro ilustraci uveďme například:

- Vícerozměrná varianta na unitárním prostoru "$\mathbb{X}^n$" (zjednodušeně).
- Varianta, kde si buňky pamatují historii a jejich přežití je závislé na tom, jak vypadala situace v několika předchozích generacích.
- Varianta, v níž hra neprobíhá na dvourozměrné ploše, ale na ploše vyšší dimenze, například tedy na kouli, toru či kleinově láhvi.

Žádná z těchto modifikací ale nedoznala takové slávy a úspěchu jako základní, či chcete-li standardní varianta popsaná výše.

1.5 Volání programu

Kořenový predikát `life` se volá s parametry, jak bylo popsáno v sekci Vstup.

2 Programátorská dokumentace

Jednotlivé funkce jsou velmi detailně komentovány ve zdrojovém textu, včetně vstupních a výstupních parametrů. vysokoúrovňová funkce je jedna a byla zmíněna v uživatelské dokumentaci. Zde pouze nastíníme celkovou architekturu programu a v dodatku připojíme výpis zdrojového kódu.

Program je ve výsledku velmi procedurální a není příliš ukázkovým a typickým příkladem pro jazyk Prolog. Nevyhneme se situacím, kdy musí být hrací pole několikrát procházeno v cyklu, či if-else konstrukcím.

2.1 Reprezentace hrací plochy

Jedna buňka je reprezentována kartézskými souřadnicemi, dvouprvkovým seznamem ve formátu [Sloupec, Řádek].

Hrací plocha je určena svými rozměry a seznamem obsazených buněk. Není nijak setříděna. Její setřídění a následné programování s ohledem na toto by snížilo asymptotickou složitost, může to být námět na následné vylepšení programu. Prozatím jsme, vzhledem k tomu, že hra probíhá často na velmi malé ploše a pro enormní plochy nemá valný smysl, od třídění upustili.

Výhoda neprocedurálního jazyka a výše popsané reprezentace je zřejmá. Není potřeba kontrolovat přetečení hrací plochy či speciálně ošetřovat kraje a rohy.

2.2 Reprezentace přechodové funkce

Přechodová funkce je definována dvěma seznamy. Seznam `Narození` ovlivňuje mrtvé buňky; udává kolik musí mít daná buňka aktivních sousedů, aby se v ní mohla narodit živá bytost. Tedy počet aktivních sousedů musí být libovolné číslo ze zmíněného seznamu. V opačném případě se neaktivní buňka nestane aktivní. Druhým seznamem je `Život`, ten definuje stavový přechod pro živé buňky. Udává, kolik musí mít daná buňka aktivních sousedů, aby přežila. Jinak zemře.

Výskyt v těchto seznamech se vyhledává pomocí funkcí `konfiguraceNarozeni`, `konfiguracePoSmrti`, `konfiguraceZivot`, `konfiguraceUmreni`. Ty volají jed-

noduché funkce `vSeznamu` a `vSeznamuNeni`. Samotné testování podmínky pak zajišťují funkce `testNarozeni`, `testPoSmrti`, `testZivota` a `testUmreni`. Celý blok zastřešuje funkce `generacePolicka`.

2.3 Výpočet

Výpočet je přímočarý. Pro každou buňku hrací plochy vygenerujeme jejich osm sousedů (funkce `pocetSousedu`) a podíváme se, kolik z těchto osmi sousedů je obsazených (funkce `spoctiZive`). Na základě uživatelem nastavených kritických konstant pak rozhodneme, co s buňkou budeme dělat (funkce `umri`, `narod`) a provedeme jednu ze čtyř přechodových akcí, jak bylo popsáno v uživatelské dokumentaci.

Tento postup opakujeme pro každou buňku hracího pole (funkce `generacePolicka`) a pak znova pro každou novou generaci (funkce `dalsiGenerace`). Po výpočtu každé generace vytiskneme výstup (funkce `tiskni`)

3 Dodatek A, Seznam funkcí

Seznam funkcí a jejich stručný popis. V dodatku chybí diakritika, neboť jde o kopie zdrojového kódu.

`life(+Sloupce, +Radky, +Mrizka, +PocetGeneraci, +Narozeni, +Zivot)`
: hraje hru Game of Life, Spocita a vytiskne prvnich PocetGeneraci generaci. hraje se na Mrizce, ktera ma rozmery Sloupce x Radky a prechodova funkce je dana polem Narozeni a Zivot

`dalsiGenerace/6(+Mrizka, +Sloupce, +Radky, -MrizkaNova, Narozeni, Zivot)`

`dalsiGenerace/8(+[Sloupec,Radek], +PocetSloupcu, +PocetRadku, +TestMrizka, ...)`
: spocita dalsi generaci zivota pro vsechny policka v Mrizce vraci MrizkuNova

`generacePolicka(+[Sloupec, Radek], +TestMrizka, +StaraMrizka, -NovaMrizka)`
: spocita dalsi generaci policka [Sloupec,Radek] pro testovani poctu sousedu pouziva TestMrizku pro urceni stavu policka pouziva StarouMrizku po zmene stavu policka je vracena NovaMrizka

`vSeznamu(+Prvek, +Seznam)` naprosto obecny predikat, ktery uspeje, pokud Prvek je v Seznamu.

`vSeznamuNeni(+Prvek, Seznam)` naprosto obecny predikat, ktery uspeje, pokud Prvek v Seznamu neni

`konfiguraceNarozeni(+PocetSousedu, Narozeni)` zjisti, jestli cislo PocetSousedu je v seznamu Narozeni. Selze, pokud neni.

`konfiguracePoSmrti(+PocetSousedu, Narozeni)` uspeje, pokud cislo PocetSousedu se nenachazi v seznamu narozeni. Selze, pokud se nachazi.

`konfiguraceZivot(+PocetSousedu, Zivot)` zjisti, jestli cislo PocetSousedu je v seznamu zivot. Selze pokud neni.

`konfiguraceUmreni(+PocetSousedu, Zivot)` uspeje, pokud cislo PocetSousedu se nenachazi v seznamu Zivot. Selze, pokud se nachazi.

`testNarozeni(+[Sloupec,Radek], +Mrizka, +Narozeni)` otestuje, jestli mrtva bunka na miste [Sloupec, Radek] v Mrizce nema byt narozena Narozeni udava konfiguraci seznamu Narozeni

`testPoSmrti(+[Sloupec,Radek], +Mrizka, +Narozeni)` otestuje, jestli mrtva bunka na miste [Sloupec, Radek] v Mrizce ma byt nadale mrtva Narozeni udava konfiguraci seznamu Narozeni

`testZivota(+[Sloupec,Radek], +Mrizka, +Zivot)` otestuje, jestli ziva bunka na miste [Sloupec, Radek] v Mrizce ma nadale zit Zivot udava konfiguraci seznamu Zivot

`testUmreni(+[Sloupec,Radek], +Mrizka, +Zivot)` otestuje, jestli ziva bunka na miste [Sloupec,Radek] v Mrizce ma umrit Zivot udava konfiguraci seznamu Zivot

`umri(+[Sloupec, Radek], +Mrizka, -NovaMrizka)` Pokud je v Mrizce na pozici [Sloupec,Radek] prvek, tak je znicen, jinak se nic nestane

`narod(+[Sloupec, Radek], +Mrizka, -NovaMrizka)` Pokud neni v Mrizce na pozici [Sloupec,Radek] prvek, je vytvoren, jinak se nic nestane

`tiskni/3(+PocetSloupcu, +PocetRadku, +Mrizka)` vytiskne Mrizku na vystup

`tiskni/5(+Sloupec, +Radek, +PocetSloupcu, +PocetRadku, +Mrizka)` vytiskne na obrazovku stav pozice [Sloupec, Radek] v mrizce

`zije(+[Sloupec,Radek], +Mrizka)` zjisti, jestli v Mrizce je na pozici [Sloupec,Radek] ziva bytost pokud neni, predikat selze, jinak vraci true

`nezije(+[Sloupec,Radek], +Mrizka)` zjisti, jestli v Mrizce je na pozici [Sloupec,Radek] volno pokud je tam ziva bunka, predikat selze

`spoctiZive/3(+Seznam, +Mrizka, -Vysledek)` `spoctiZive/5(+Seznam, +Mrizka,, ?Acc, -Vysledek)` spocte pocet zivych bytosti na pozicich v Seznamu Acc je potreba, aby fungovala aritmetika

`pocetSousedu(+[Sloupec,Radek],+Mrizka, -PocetSousedu)` spocte v Mrizce na pozici [Sloupec,Radek] pocet zivych bytosti okolo

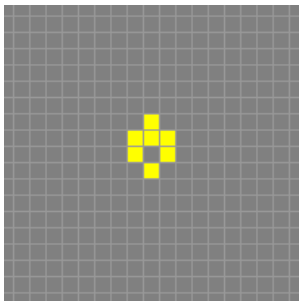
4 Dodatek B, Testovací data

Pro jednoduchou ukázku funkčnosti programu jsou přímo ve zdrojovém kódu čtyři testovací funkce, tedy inicializované konečné automaty; uživatel pouze zavolá danou metodu a zvolí, kolik generací má být vypsáné na obrazovku. Přechodové funkce jsou zvoleny standardně, tedy buňka přežije, pokud má dva či tři aktivní sousedy a narodí se, pokud má právě tři.

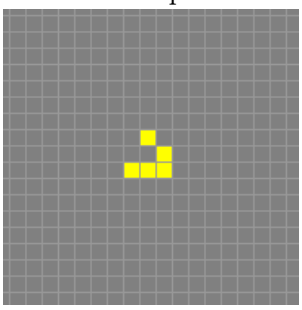
```
glider(+PocetGeneraci)
smallExploder(+PocetGeneraci)
exploder(+PocetGeneraci)
row(+PocetGeneraci)
```

Níže jsou uvedeny počáteční stavy připravených testovacích dat.

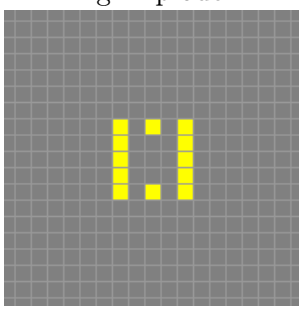
Glider



Small Exploder



Big Exploder



Row