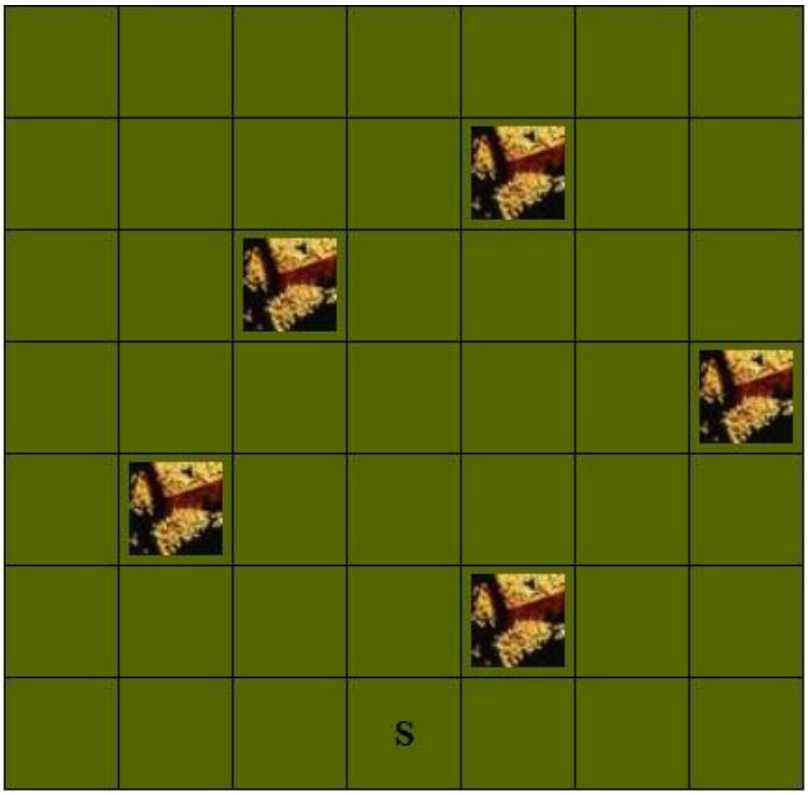
Umelá inteligencia, zadanie 3 - Hľadači pokladov

Marek Oravec

Obsah

Zadanie

Majme hľadača pokladov, ktorý sa pohybuje vo svete definovanom dvojrozmernou mriežkou (viď. obrázok) a zbiera poklady, ktoré nájde po ceste. Začína na políčku označenom písmenom `S` a môže sa pohybovať štyrmi rôznymi smermi: `hore` **H**, `dole` **D**, `doprava` **P** a `doľava` **L**. K dispozícii má konečný počet krokov. Jeho úlohou je nazbierať čo najviac pokladov. Za nájdenie pokladu sa považuje len pozícia, pri ktorej je hľadač aj poklad na tom istom políčku. Susedné políčka sa neberú do úvahy.



Horeuvedenú úlohu riešte prostredníctvom evolučného programovania nad virtuálnym strojom.

Tento špecifický spôsob evolučného programovania využíva spoločnú pamäť pre údaje a inštrukcie. Pamäť je na začiatku vynulovaná a naplnená od prvej bunky inštrukciami. Za programom alebo od určeného miesta sú uložené inicializačné údaje (ak sú nejaké potrebné). Po inicializácii sa začne vykonávať program od prvej pamäťovej bunky. (Prvou je samozrejme bunka s adresou 000000.) Inštrukcie modifikujú pamäťové bunky, môžu realizovať vetvenie, programové skoky, čítať nejaké údaje zo vstupu a prípadne aj zapisovať na výstup. Program sa končí inštrukciou na zastavenie, po stanovenom počte krokov, pri chybnej inštrukcii, po úplnom alebo nesprávnom výstupe. Kvalita programu sa ohodnotí na základe vyprodukovaného výstupu alebo, keď program nezapisuje na výstup, podľa výsledného stavu určených pamäťových buniek.

Virtuálny stroj

V tomto zadaní figuroval aj tzv. virtuálny stroj. Každý z našich hľadačov mal k dispozícii svojich unikátnych 64 pamäťových buniek a tento stroj nimi prechádzal a určoval aký pohyb hľadači budú mať.

Stroj pracoval na základe uvedených inštrukcii:

inštrukcia	tvar
inkrementácia	00XXXXXX
dekrementácia	01XXXXXX
skok	10XXXXXX
výpis	11XXXXXX

Hľadač pokladov

Samotných hľadačov pokladov som mal interpretovaných ako objekty, ktoré v sebe držali rôzne informácie aby sa mi s nimi lepšie pracovalo.

```
class Individual():
    def __init__(self, tape, path, treasures, fitness):
        self.tape = []
        self.tape.extend(tape)
        self.path = path
        self.fitness = fitness
        self.treasures = treasures
```

tape – pole, ktoré obsahuje už spomínané pamäťové bunky

path – pole symbolov ktoré určovali pohyb (D, P, L, H)

fitness – fitness pre konkrétneho hľadača

treasures – počet nájdených pokladov vďaka ktorému sa mi dobre určovala fitness

Prvá generácia

Keďže cieľom tohto zadania bolo demonštrovať vývoj jedincov, na začiatku som pamäťové bunky inicializoval náhodnými hodnotami v rozsahu od 0 – 255. Dôvodom tohto rozsahu je fakt, že máme pamäťové bunky reprezentované 1B a do tohto objemu sa nám zmestí len 256 hodnôt. Treba ešte uviesť, že číslo po prevedení do binárnej sústavy obsahuje len bity s hodnotou 1 a 0, a práve na základe týchto bitov sa určoval pohyb hľadača. Čiže ak dané binárne číslo obsahuje 2 jednotky, pohyb bude **H**, 3 alebo 4 bude **D**, 5 alebo 6 bude **P** a 7 alebo 8 bude **L**.

Spôsob selekcie

V tomto zadaní bola možnosť vybrať si spôsob selekcie jedincov pre ďalšiu generáciu a zároveň nebol vymedzený počet selekcii, ktoré bolo potrebné implementovať. Preto som sa rozhodol pre turnaj. Turnaj funguje na princípe ***k*** náhodne zvolených jedincov, z ktorých sa potom následne vybral najlepší ako rodič ktorý postupuje do ďalšej generácie a zároveň pomocou ktorého sme schopní pri krížení s iným rodičom vytvoriť nového jedinca pre nasledujúcu generáciu. Tiež je ošetrený aj prípad, aby sme nevybrali dva rovnaké objekty (jedincov) ako rodičov.

Kríženie

Kríženie v tomto programe fungovalo na princípe vzatia dvoch rodičov, ktorých som nakopíroval do nových objektov, aby som ich nestratil, a z ich kópii som vytvoril dvoch nových jedincov, ktorým som na náhodnom mieste vymenil gény (pamäťové bunky), pričom som ich následne zaradil k rodičom do novej generácie.

Mutácia

V tomto programe sa však nemôžeme vyhnúť mutácii, pretože tá je dôležitá z dôvodu, aby sme nemali identických jedincov v novej generácii. Keby sme mutáciu v tomto programe nemali, vývoj jedincov by ***nekonvergoval***.

Ovládanie

Program je možné ovládať podľa voľby používateľa. Používateľ si môže určiť veľkosť populácie pre jednu generáciu, a rovnako tak aj počet generácii do skončenia programu.

Postrehy

Osobne som nadobudol pocit, že v tomto programe veľkú úlohu zohráva náhoda. Výsledky sa totiž líšia od používateľom zadaných hodnôt, čiže je rozdiel keď si určíme počet populácie na 20 a generácie na 5000 v porovnaní s populáciou o veľkosti 1000 a počtom generácii skôr. Program totiž pracuje rôzne rýchlo pri zadaných hodnotách, a rovnako tak aj nájde riešenie v rôznom čase (v akej generácii hľadači dokážu nájsť všetky poklady).

Zhodnotenie

Osobne si myslím, že určite by sa dalo ešte mnohými úpravami zefektívniť tento program, avšak ako bolo spomenuté už vyššie, výsledky sa líšia v závislosti od používateľom zadaných hodnôt. Zo skúsenosti dokážem povedať, že nie je veľmi múdre nastaviť príliš malú alebo príliš veľkú mieru mutácie, pretože keď nastavíme mutáciu veľmi malú, jedinci sa budú na seba podobať, a keď nastavíme mutáciu príliš veľkú, jedinci sa budú až veľmi líšiť od predošlých generácii a došlo by k tomu, že by sa riešenie hľadalo podstatne dlhšie. Z toho dôvodu je mutácia nastavená na ***3%.*** Rovnako tak som mal dobré výsledky pri populácii o veľkosti ***100***, hoci minimum a zároveň aj mnou pôvodne zvolená počiatočná populácia bola ***20*** jedincov. Pri 20 jedincoch v populácii som totiž zistil že je potrebné nastaviť podstatne viac generácii. Tieto hodnoty si však používateľ môže nastaviť podľa seba a sám uvidí výsledok.