Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológii

Zadanie 3

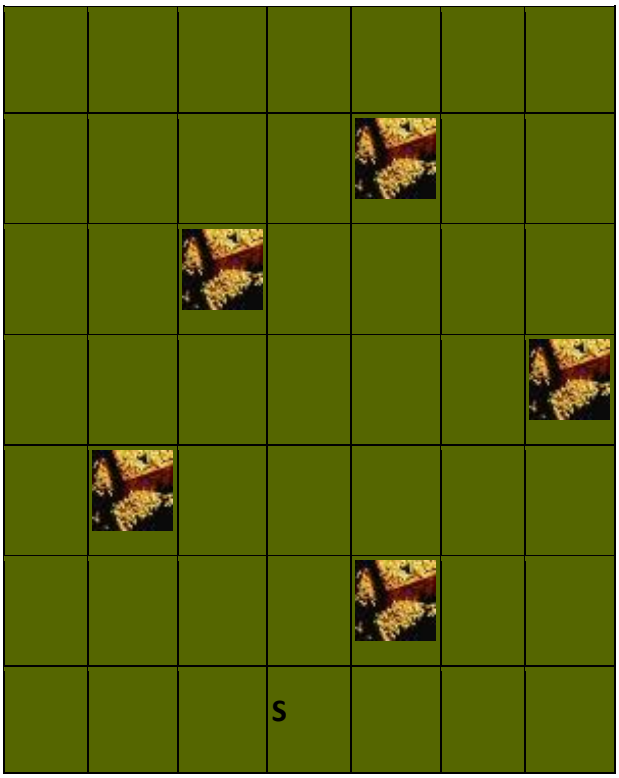
Hľadanie pokladu (3b)

Umelá Inteligencia

Martin Raffáč

Úloha

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 zadaní 3b bolo potrebné taktiež implementovať aj virtuálny stroj. Každí hľadač mal k dispozícii pamäť o veľkosti 64 pamäťových buniek.

Virtuálny stroj poznal štyri inštrukcie: inkrementáciu hodnoty pamäťovej bunky, dekrementáciu hodnoty pamäťovej bunky, skok na adresu a výpis (**H**, **D**, **P** alebo **L**) podľa hodnoty pamäťovej bunky.

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

Hľadač pokladov

Jednotlivý hľadači pokladov boli reprezentovaný triedou Individual ako objekty. Títo hľadači v sebe držali niekoľko informácií potrebných na naimplementovanie tohto zadania.

Boli to:

memory – pole obsahujúce jednotlivé pamäťové bunky

path – pole jednotlivých simulácií krokov (H, D, L, P)

fitness – fitness daného hľadača

treasures – počet nájdených pokladov, potrebných na vypočítanie fitness

```python
class Individual():
    def __init__(self, memory, path, treasures, fitness):

        self.memory = []
        self.memory.extend(memory)
        self.path = path
        self.treasures = treasures
        self.fitness = fitness
```

Začiatočná generácia

Na začiatku programu sa nainicializovala prvá generácia jedincov a to tak, že do jednotlivých pamäťových buniek sa nastavili náhodné hodnoty od 0 po 255. To preto lebo pamäťové bunky sú reprezentované 1B a to znamená že sa do neho zmestí 256 hodnôt. Ešte pred samotným vytvorením jedinca sa pomocou virtuálneho stroja vykonávajú jednotlivé inštrukcie. Medzi ne patrí aj zistenie pohybu hľadača, ten sa vypočíta prevedením čísla pamäťovej bunky z desiatkovej sústavy do binárnej a následne sa spočíta koľko jednotiek dané číslo obsahuje. Po nastavení hodnôt v pamäti sa vytvoril daný jedinec a následne sa pridal do poľa ktoré obsahuje všetkých jedincov danej generácie.

```python
if 0 <= counter <= 2:
    return 'H'
if 2 < counter <= 4:
    return 'D'
if 4 < counter <= 6:
    return 'P'
if 6 < counter <= 8:
    return 'L'
```

Spôsob selekcie

Vo svojom zadaní som sa rozhodol implementovať selekciu turnaj. Táto selekcia funguje na jednoduchom princípe vybratí x náhodných jedincov z generácie a vyberie najlepšieho budúceho rodiča. Tento proces prebehol dva krát keďže na vytvorenie dvoch nových jedincov potrebujeme práve dvoch rodičov. V programe je aj ošetrená podmienka v prípade že sa vyberú dva rovnaké jedince.

Kríženie

Po selekcii dvoch rodičov prichádza samotné kríženie. Najskôr sa vytvoria nové polia ktoré budú reprezentovať pamäťové bunky detí. Potom sa zvolí náhodné číslo z rozsahu počtu pamäťových buniek. Podľa tohto náhodne zvoleného čísla sa určuje ktorá pamäťová bunka rodiča sa uloží do ktorého dieťaťa. Tieto deti sa zaradia do novo vznikajúcej generácie.

Mutácia

Mutácia je v tomto programe dôležitý faktor pretože nechceme aby v generáciách vznikali identické jedince. Bez mutácie by vývoj jedincov nekonvergoval. V novo vytvorenom jedincovi by mali zmutovať zhruba 4% génov.

Ovládanie

Používateľ ovláda program tak, že si zvolí počet jedincov v generácii a počet generácií, ktoré sa majú vygenerovať.

```
Zadajte pocet generacii: 40
Zadajte pocet jedincov pre populaciu: 5000
```

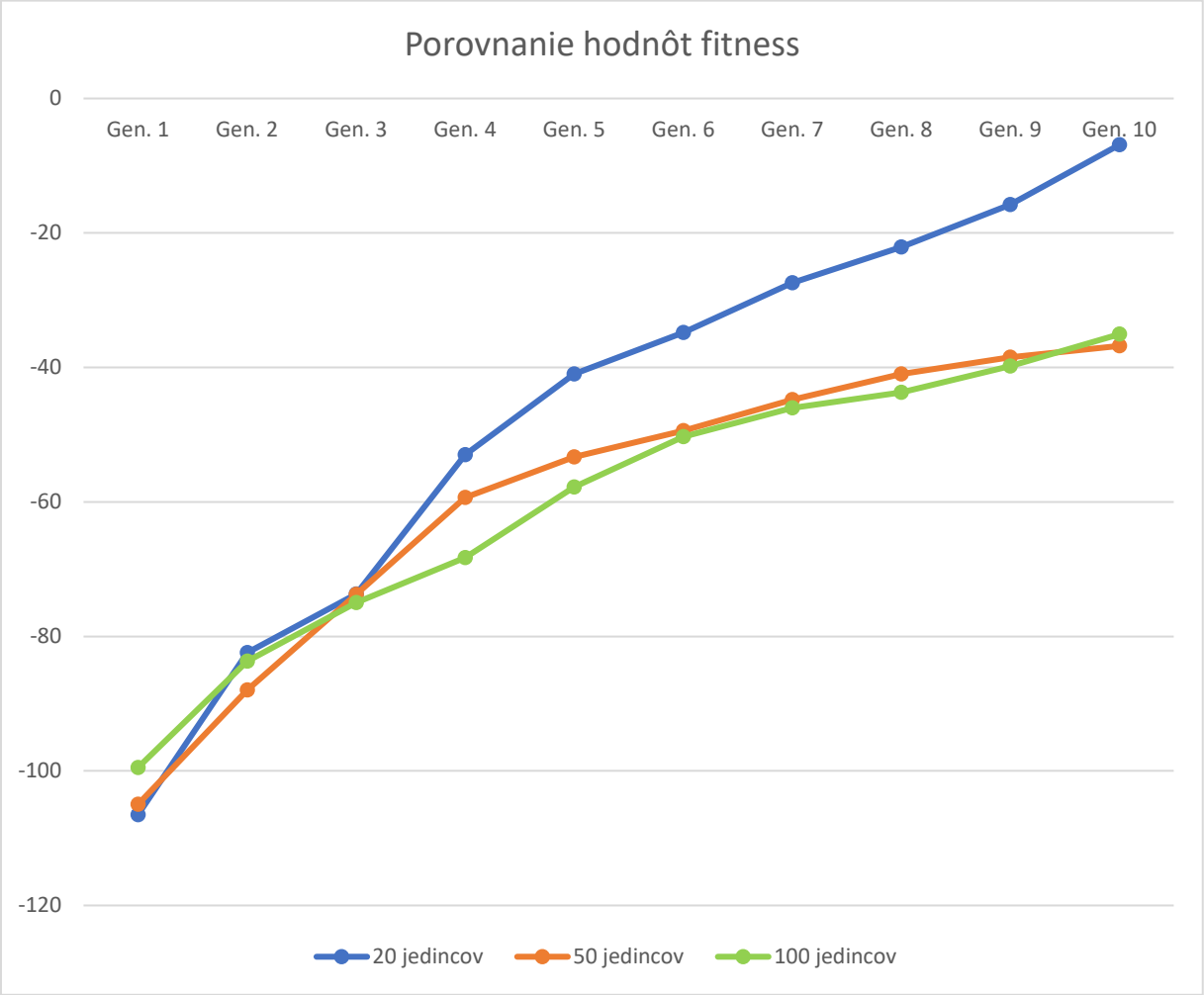
Zistenie

Zaujímavým zistením pre mňa bolo, že v program do veľkej miery závisí len na náhode. Pretože pri skúmaní generovania populácie pre konštantný počet jedincov a konštantný počet generácií vychádzali za každým rozličné výsledky. Tiež som odpozoroval že program pre populáciu o veľkosti 5000 jedincov pre počet generácii 20 nie vždy našiel všetky poklady zatiaľ čo pre populáciu o veľkosti 20 pre 5000 generácii ho našiel.

Zhodnotenie

Môj program je možné rozšíriť a vylepšiť, napríklad tak aby si mohol používateľ sám vybrať koľko percentnú mutáciu alebo aký spôsob výberu jedincov chce. Počas skúšania programu som zistil, že program funguje efektívnejšie keď uprednostníme väčší počet generácii pred väčším počtom jedincov v jednej generácii. Mutáciu som prednastavil v programe na 4% pretože so zväčšujúcou sa mutáciou sa začali deti priveľmi líšiť od rodičov

a hľadanie pokladov trvalo dlhšie.



Na grafe je znázornený vývin priemernej hodnoty fitness pre 10 generácií zo vzorkami jedincov 20, 50 a 100. Z grafu vyplýva, že čím má generácia väčší počet jedincov tým sa hodnota fitness zlepšuje pomalšie. No naopak napríklad pri generácii s 20 jedincami je výsledok pomerne skresľujúci pretože stačí jeden jedinec ktorému sa podarí nájsť viac pokladov a tak dokáže značne zvýšiť priemernú hodnotu fitness.