Umelá inteligencia

Dokumentácia na zadanie 1b – Hľadanie pokladu

Radoslav Muntág

Cvičiaci: Ing. Jakub Abrahoim

1. Úloha

Úlohou je vytvoriť genetický algoritmus pre hľadača pokladov. Na vstupe je veľkosť mapy a rozmiestnenie pokladov. Každý hľadač má 64 buniek ktoré udržiavajú 8 bitové hodnoty. Prvé 2 bity v bunke informáciu o operácií a ostatných 6 je pointer na bunku s hodnotou, s ktorou má operácia pracovať. Operácie sú inkrementácia, dekrementácia, skok, a pohyb. Subjekt sa môže pohybovať doprava, doľava, hore a dole.

Subjekt skončí pohyb test ak vyčerpal svoje množstvo krokov, vyšiel von z mapy alebo našiel všetky poklady.

Program skončí ak sa nejaký subjekt nájde všetky poklady.

2. Návod

Pred spustením je možné zmeniť map_config.json a machine_config.json.

2.1. map_config.json

`"size": [7, 7],` udáva veľkosť mapy. V tomto prípade dĺžka riadka je 8 a dĺžka stĺpca je tiež 8.

`"start_position": [3, 6],` začiatočná pozícia.

`"treasure_count": 5,` počet pokladov.

`"treasure_coordinates": [[4, 1], [2, 2], [6, 3], [1, 4], [4, 5]]` pozície pokladov. Je dôležité aby ich počet bol rovnaký ako treasure_count.

2.2. machine_config.json

`"population_size": 30,` počet subjektov na generáciu.

`"elitism_size": 1,` počet subjektov vybraných pomocou elitizmu pre ďalšiu generáciu. Tieto subjekty budú zaradené do kríženia a zároveň ich bunky budú nezmenené do ďalšej generácie.

`"roulette_size": 1,` počet subjektov vybraných náhodne pre ďalšiu generáciu. Tieto subjekty budú zaradené do kríženia, ale na rozdiel od elitizmu ich bunky neprežijú do ďalšej generácie.

`"tournament_size": 3,` počet subjektov vybraných tournament výberom. Tieto subjekty budú zaradené do kríženia a zároveň ich bunky budú nezmenené do ďalšej generácie.

`"mutation_rate": 0.05,` šanca na mutáciu jedného bitu pre bunku.

`"mutation_rate_all_bits": 0.01,` šanca na invertovanie všetkých bitov pre bunku.

`"dynamic_mutation": true,`
`"dynamic_mutation_constant": 4,` experimentálna funkcia. K hodnote mutation_rate sa pripočíta inverzná funkcia doteraz najlepšej fitnes hodnoty * dynamic_mutation_rate.

```
dynamic_mutation = (
    (1 / (self.best_fitness * self.machine_config["dynamic_mutation_constant"] + 1))
    if self.machine_config["dynamic_mutation"] else 0
)

if random.random() < self.machine_config["mutation_rate"] + dynamic_mutation:
    new_cell = flip_a_random_bit(new_cell)
```

Snaží sa znížiť šancu na mutáciu keď je najlepší subjekt blízko nájdenia všetkých pokladov. Pri zapnutí tejto funkcie je dobré znížiť mutation_rate na číslo okolo 0.05.

`"winner_vizualization": true,` zapne vizualizáciu výhercu keď nájde všetky poklady.

`"end_program_when_winner": true,` ukončí program keď sa nájde subjekt, ktorý objaví všetky poklady (alebo sa dosiahne limit generácií). Pri false sa bude pokračovať s generáciami dokým sa nedosiahne limit generácií.

```
"extended_data_print": true,
```
detailnejší výpis pre každého jedinca.

```
"generations_count_limit": 1000
```
maximálny počet generácií.

2.3. Výber configu

Pri spustení program vyzve používateľa či chce default_config, ktorý je natvrdo zakódovaný v kóde (ten by sa nemal meniť požívateľom, aby bol zachovaný nejaký backup) alebo custom_config kedy sa načíta config z .json súboru.

```
0 - Default config
1 - Load custom config from file
Input 0 or 1:
```

3. Štruktúra programu

Štruktúra je rozdelená do štyroch .py súborov:
main.py,
virtual_machine.py,
suubject.py,
end_vizualization.py

Každý okrem main.py má v sebe jednu triedu, ktorá vykonáva jej účel.

3.1. Inicializácia

Inicializácia nastáva v main.py kde sa načíta config a vytvorí prvá populácia subjektov každý s náhodnými hodnotami od 0-255 pre jeho každú bunku (64 buniek).

Potom nastane inicializácia virtuálneho stroja, ktorému sú zaslané subjekty a config.

Začne sa hlavný cyklus v ktorom sa spustí spúšťa virtuálny stroj a následne inicializuje nová generácia kým sa program neskončí.

3.2. Testovanie subjektov

Po zapnutí stroja prejde každý subjekt testovacou miestnosťou.

3.2.1. Priebeh testu jedného subjektu

Na začiatku hlavný pointer ukazuje na prvú bunku v genóme daného subjektu. Na hodnotu v tejto bunke sa uplatní bitový posun o 6 miest do pravá, aby sa získala informácia o operácií, ktorá sa má uskutočniť.

Keďže v bunke je hodnota od 0-255, očakáva sa že po bitovom posune dostaneme hodnotu od 0-3. Hodnota teda prislúcha jednej zo štyroch operácií ktorá sa má vykonať na bunku, na ktorú ukazuje zvyšných 6 bitov:

0 – increment

1 – decrement

2 – jump to

4 – move

Operácia dostane zvyšných 6 bitov aby vedela na ktorú bunku má operáciu uplatniť.

3.2.2. Operácia inkrement

Inkerementuje hodnotu v bunke, na ktorú dostala pointer.

Inkrementuje sa hlavný pointer, aby sa program posunul na novú bunku.

3.2.3. Operácia dekrement

Dekrementuje hodnotu v bunke, na ktorú dostala pointer.

Inkrementuje sa hlavný pointer, aby sa program posunul na novú bunku.

3.2.4. Operácia jump to

Zmení hlavný pointer na pointer, ktorí dostala.

3.2.5. Operácia move

Podľa počtu bitových 1 v bunke, na ktorú dostala pointer. Zapíše pohyb a zmení polohu subjektu:

Ak je počet bitových 1 <= 3 je iniciovaný pohyb hore.

Ak je počet bitových 1 == 4 je iniciovaný pohyb dole.

Ak je počet bitových 1 == 5 je iniciovaný pohyb vpravo.

Ak je počet bitových 1 > 5 je iniciovaný pohyb vľavo.

(takéto rozloženie na prvý pohľad môže prísť neintuitívne, lebo to vyzerá, že subjekt pôjde častejšie hore a vľavo ako dole a doprava. Toto rozloženie som zvolil hlavne z dôvodu efektu normálnej distribúcie, kedy má hodnota v bunke väčšiu šancu mať 4 alebo 5 bitových jednotiek ako 1,2 alebo 7,8.)

3.2.6. Kontrola pokladu

Ak nastala operácia move, tak sa skontroluje či sa náhodou subjekt nenachádza na mieste pokladu. Ak hej tento poklad je odstránený z listu možných nájdených pokladov.

3.2.7. Vypočítanie fitness hodnoty

Prvé počítanie fitness hodnoty nastáva v operácií move, kde sa odoberajú malé body za to, keď sa subjekt pohne na políčko kde už raz bol.

Po prebehnutí testu, ktorí sa skončí ak subjekt vyšiel z mapy, dosiahol maximálny počet krokov alebo našiel všetky poklady, sa vypočíta fitness hodnota.

K fitness hodnote sa pripočíta počet nájdených pokladov. Tie sú priradené rovnako (3 poklady = 3 fitness). Táto hodnota je zďaleka najvýraznejšia a ovplyvňuje fitness najviac.

Potom sa pripočíta bonus. Ten je vypočítaný ako počet pohybov/200 ak je subjekt našiel menej ako 3 poklady (táto hodnota uprednostňuje subjekty, ktoré sa veľa hýbu, dôležité hlavne na začiatku) a 1/(počet pohybov + 1) ak subjekt našiel 3 a viac pokladov

(po dosiahnutí určitého počtu pokladov chceme aby sa uprednostňovali subjekty, ktoré dokážu nájsť poklady rýchlejšie)

3.3. Výberový proces po testovaní

Subjekty sú zoradené podľa hodnoty fitness.

Elitizmom sa vyberú kandidáti zhora a sú odstránený z ďalšieho výberu a zaradený medzi elite_subjects.

Ruletov sa vyberú náhodne, tiež sú odstránený z ďalšieho výberu, ale sú zaradený medzi survived_subjects.

Turnamentom sa vyberú kandidáti po dvoch a z každej dvojice sa vybere lepší, ktorí sú odstránený z ďalšieho výberu a zaradený medzi elite_subjects.

Elite_subjects a survived_subjects sú zaslaný na kríženie.

3.4. Kríženie

V krížení sa spoja kategórie elite a survived do jedného listu parents (elite a survived stále ostávajú samostatné len ich obsah bude nakopírovaný).

Z parents sa vyberajú po dvoch subjekty, ktorí podstúpia kríženie, kým sa nenaplní populácia.

Z každej dvojice rodičov sa vytvorí jeden potomok krížení buniek s náhodným predelom, kde predel môže byť v intervale buniek 10 – 53

Pre každú bunku sa náhodne vyhodnotí mutácia (otoč jeden náhodny bit alebo otoč vešetky bity)

Nový subjekt sa zaradí medzi elite_subjects (survived_subjects ironicky nepreżijú) a keď sa naplní začne sa nové testovanie.

4. Vizualizácia

Ak bola nastavená True hodnota pre vizualizáciu v machine_config.json, tak sa zobrazí krátka animácia ako postupne subjekt našiel všetky poklady.