Evolučné programovanie*

Matej Pakán
ID: 110867
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xpakan@stuba.sk

12. novembra 2021

*Riešenie 3. zadanie - Evolučné programovanie - hľadač pokladov – v predmete umelá inteligencia, ak. rok 2021/22, cvičiaci: Ing. Ivan Kapustík

Obsah

1 Zadanie

Úlohou je nájsť riešenie hlavolamu Bláznivá križovatka. Hlavolam je reprezentovaný mriežkou, ktorá má rozmery 6 krát 6 políčok a obsahuje niekoľko vozidiel (áut a nákladiakov) rozložených na mriežke tak, aby sa neprekrývali. Všetky vozidlá majú šírku 1 políčko, autá sú dlhé 2 a nákladiaky sú dlhé 3 políčka. V prípade, že vozidlo nie je blokované iným vozidlom alebo okrajom mriežky, môže sa posúvať dopredu alebo dozadu, nie však do strany, ani sa nemôže otáčať. V jednom kroku sa môže pohybovať len jedno vozidlo. V prípade, že je pred (za) vozidlom voľných n políčok, môže sa vozidlo pohnúť o 1 až n políčok dopredu (dozadu). Ak sú napríklad pred vozidlom voľné 3 políčka (napr. oranžové vozidlo na počiatočnej pozícii, obr. 1), to sa môže posunúť buď o 1, 2 alebo 3 políčka. Hlavolam je vyriešený, keď je červené auto (v smere jeho jazdy) na okraji križovatky a môže sa z nej dostať von. Predpokladajte, že červené auto je vždy otočené horizontálne a smeruje doprava. Je potrebné nájsť postupnosť posunov vozidiel (nie pre všetky počiatočné pozície táto postupnosť existuje) tak, aby sa červené auto dostalo von z križovatky alebo vypísať, že úloha nemá riešenie. Príklad možnej počiatočnej a cieľovej pozície je zobrazený nižšie:

2 Opis riešenia a podstatných častí

Pred tým, ako som začal riešiť nejaký algoritmus som si musel vytvoriť spôsob na reprezentáciu údajov/stavov. Použil som preto triedu State, to ktorej som zapisoval potrebné údaje v priebehu programu.

```
class State():
    level = 1
    vehicles = []
    grid = []
    operation = ""
    parent = None
    def __init__(self, vehicles):
        self.vehicles = deepcopy(vehicles)

    def getVehicles(self):
        array = []
        for vehicle in self.vehicles:
            array.append( [vehicle.color, vehicle.orientation,
                vehicle.x , vehicle.y] )
        return array

    def changePos(self,changed_vehicle, move, count):
        for vehicle in self.vehicles:
            if(vehicle.color == changed_vehicle.color):
                if(move == "RIGHT"):
                    vehicle.x += count
                elif(move == "LEFT"):
                    vehicle.x -= count
                elif(move == "UP"):
                    vehicle.y -= count
                elif(move == "DOWN"):
                    vehicle.y += count
```

Na zápis stavu používam system mriežky, do ktorej zapisujem časti jednotlivých vozidiel. Túto mriežku v priebehu programu aktualizujem.

Na reprezentáciu vozidiel používam takisto triedu - nazvanú Vehicle, z ktorej dedí trieda Car alebo Truck. Tieto 2 dediace triedy sa líšia iba v dĺžke vozidla.

```
# Abstract class
class Vehicle:
    def __init__(self, orientation, x, y, color):
        self.orientation = orientation
        self.x = x
        self.y = y
        self.color = color

# Blueprints to create vehicles
class Car(Vehicle):
    length = 2

class Truck(Vehicle):
    length = 3
```

2.1 Cyklicky prehlbujúci sa algoritmus

Tento algoritmus je charakteristický svojimi vlastnosťami.

Cyklicky prehlbujúci sa algoritmus má rýchlosť algoritmu prehľadávania do šírky ale zároveň má priestorovú zložitosť vo veľkosti algoritmu prehľadávania do hĺbky. Používa sa ak máme k dispozícii menej pamäte a trochu pomalší algoritmus je akceptovateľný.

Princíp algoritmu spočíva postupnom prehlbovaní - teda začíname v hĺbke 0 (koreň) a postupne prechádzame všetky uzly do hĺbky. Ak dôjdeme na hĺbku 1 (priamy potomok koreňa) a žiadny z týchto potomkov nie je cieľom, zvyšujeme maximálnu hĺbku stromu na 2. Takto pokračujeme až kým nenarazíme na riešenie, alebo kým hĺbka nebude väčšia, ako užívateľom maximálne zvolená.

Obrázok, kde je zobrazené prehlbovanie je na konci tohto dokumentu.

3 Testovanie

Testovanie som realizoval na notebooku s procesorom i5-7300HQ pri frekvencii procesora 3.2 - 3.5GHz.

Výsledky boli premenlivé, v drvivej väčšine sa výsledok podarilo nájsť, niekedy však neskôr ako bol odhad. Spôsobené to je pravdepodobne tým, že vedľajšie funkcie na overenie možnosti vstupu vozidla na nejaké políčko asi nie sú na 100% korektne implementované.

Pri výpise riešenia uvádzam pomocou akých krokov sa k riešeniu podarilo dostať, ako aj vizuálnu reprezentáciu formou mriežky.

Výsledky testovanie môžete nájsť na konci tohto dokumentu.

Zadanie mi priblížilo ako funguje umelá inteligencia v praxi. Páčilo sa mi, že som v zadaní riešil hru, ktorú som v minulosti hrával. Hru možno nájsť pod anglickým názvom 'unblock me' alebo 'rush hour'. Zadanie som nevypracoval na 100%, nakoľko niektoré zložitejšie zadania mi program nevyrieši a padne.

4 Výhody a nevýhody implementácie

Výhodou je rýchla odozva, už po niekoľkých iteráciach vieme zistiť, či je zadanie zložité alebo nie. Ďalšou výhodou môže byť menšie použitie pamäte, ako aj to, že pre rôzne zložitosti a rôznorodosť úloh je toto riešenie optimálnejšie ako prehľadávania do hĺbky/šírky.

Hlavnými nevýhodami sú exponenciálny čas riešenia úloh a zbytočné prechádzanie nižších hĺbok ak sa snažíme dostať k hĺbke vyššej.

5 Možnosti rozšírenia

Moje riešenie by sa dalo rozšíriť o vlastné rozmery mriežky, ďalšie velkosti vozidiel alebo napríklad smerovanie sledovaného vozidla iným smerom.