Fakulta informatiky a informačných technológií
Predmet: Umelá inteligencia(UI)

Zadanie č. 2. : 8-hlavolam

Autor: **Patrik Gallik**

Zadanie úlohy:
Našou úlohou je nájsť riešenie 8-hlavolamu. Hlavolam je zložený z 8 očíslovaných políčok a jedného prázdneho miesta. Políčka je možné presúvať hore, dole, vľavo alebo vpravo, ale len ak je tým smerom medzera. Je vždy daná nejaká východisková a nejaká cieľová pozícia a je potrebné nájsť postupnosť krokov, ktoré vedú z jednej pozície do druhej.

Opis riešenia:
Na riešenie problému bol implementovaný A* algoritmus. Ako prostredie bol zvolený NodeJS (JavaScript), algoritmus však beží aj pri spustení pod prehliadačom. Na demoštráciu algoritmu bolo vytvorené jednoduché GUI, ktoré dovoľuje používateľovi zamiešať algoritmus podľa svojho uváženia, vybrať heuristiku a spustiť riešenie.

Na spustenie GUI spustite súbor index.html v adresári projektu. Na spustenie iba simulácie, spustite súbor solve.js (je potrebné mať nainštalovaný NodeJS). GUI používa ten istý súbor, samotný algoritmus sa teda nachádza iba v jednom súbore. Príkaz na spustenie riešenia cez konzolu:

```
$ node js/solve.js
```

Reprezentácia údajov problému

Uzol v grafe (trieda Node) obsahuje nasledujúce údaje:
state – stav hlavolamu (reprezentovaný dvojrozmerným poľom)
h – hodnota heuristickej funkcie v danom stave
parent – referencia na rodiča (použitie pri spätnom konštruovaní výsledku)
lastOperand – operand, ktorý bol použitý pri prechode do daného stavu (použité, aby sa zbytočne nevytváral uzol do strany, z ktorej sa prišlo, takisto pri konšktruovaní výsledku)
price – hĺbka/cena – počet krokov, ktoré musia byť vykonané, aby sa dosiahol tento stav
hprice – súčet heuristiky a ceny – používa sa pri zoraďovaní v prioritnej fronte

Rad uzlov je implementovaný v triede Queue. Poskytuje metódu push, a pop. Ak je rad prázdny, algoritmus končí bez úspechu.

Na ošetrenie cyklenia pri prechádzaní grafom slúži pole stavov, ktoré už boli navštívené. Toto pole je hashované pomocou jednoduchej funkcie, aby sa otestovanie, či bol aktuálny stav už navštívaný, uskutočnilo čo najrýchlejšie.

Samotný krok algoritmu je vo funkcii iteration(), ktorá sa nachádza v globálnej funkcii run(). Algoritmus prebieha nasledovne:

1. Z radu sa vyberie uzol. Ak je rad prázdny, algoritmus sa neúspešne ukončí.
```
if (node = queue.pop()) {
```

2. Ak je heuristika uzla 0, jedna sa o koncový stav, ak nie, ideme ďalej
```
if (node.h == 0) {
```

3. Pre stav v uzle sa pokúsi vytvoriť dcérske stavy použitým každého operandu. Ak je to možné, vytvorí nový uzol.
```
if ((node.lastOperand != 'down') && (state = go(node.state, 'up'))) {
  var newNode = new Node(node, state, 'up');
  …
```

4. Zároveň sa pre každý nový vrchol použije funkcia checkHashAndPushNode, ktorá skontroluje, či daný stav už bol navštívený, ak nie, tak pridá uzol do radu.
```
function checkHashAndPushNode(node, state) {
    var hash = hashState(state);
    if (!visitedStates[hash]) {
      visitedStates[hash] = {
        'state': state,
        'node': node
      };
      queue.push(node);
    } else {
      if (debug) console.log("Rovnaky stav");
      node = null;
    }
  }
```

5. Takto algoritmus pokračuje, až kým sa nenájde riešenie, alebo sa v rade nenachádza žiaden uzol.

Spôsob testovania

Základné otestovanie algoritmu je možné pomocou GUI, v ktorom sa dá otestovať algoritmus na veľkosti problému 3x3, z definovaným konečnýcm stavom (1,2, 3 atď.) Je možné zvoliť heuristiku a výsledky porovnať. Heuristika 2 dáva lepšie výsledky (menej preskúmaných vrcholov, tj rýchlejšie).

Podrobnejšie testovanie je možné uskutočniť spustení cez konzolu. V súbore solve.js sa dá testovanie rôznych vstupov uskutočniť zmenou nasledujúcich premenných:

M – šírka hlavolamu
N – výška hlavolamu
h – heuristika – môže mať hodnotu ‘h1’ alebo ‘h2’
initState – počiatočný stav (dvojrozmerné pole)
finalState – konečný stav (dvojrozmerné pole)

Rôzne testy, ktoré sú overené, sa nachádzajú v súbore js/tests.js. Spustiť tieto testy je možné nakopírovaním daných hodnôt do solve.js.

Zhodnotenie riešenia
Hlavným nedostatkom riešenia je pomalosť implementácie. Použitím Profilera v prehliadači Chrome som zistil, že najviac času vykonávania zaberá funkcia sortFn, ktorá zoraďuje Rad uzlov. Tento nedostatok sa dá vyriešiť implementovaním radu ako haldy, alebo podobne rýchleho spôsobu.

Ďalším nedostatkom je, že riešenie neobsahuje overenie, či zadaný stav je možné vyriešiť. Pri takomto stave bude algoritmus hľadať riešenie, až kým nevyskúša všetky možné stavy.

Rýchlosť algoritmu bola vylepšená nasledujúcimi spôsobmi: pri heuristike číslo 2, sa namiesto prechádzania celého stavu a hľadania riadku a stĺpca daného čísla, sa na začiatku výpočtu vygeneruje zo stavu objekt (*h2list*), ktorý obsahuje hodnoty stĺpcov a riadkov každého čísla. Ďaľší zrýchlením je ukladanie už navštívených zahashovaných stavov do objektu *visitedStates.* Na zistenie, či už bol stav navštívený, stačí zaheshovať stav pomocou funkcie *hashState*, a zistiť, či existuje *visitedStates.zaheshovany_stav.*