**A close up of a logo

Description automatically generated**

Umelá inteligencia

Problém 2e)

Riešenie 8-hlavolamu pomocou A\* algoritmu

Jakub Šimko

AIS ID: 103146

2021/2022

Cvičiaci: Ing. Ivan Kapustík

Čas cvičení: Streda 13:00

Obsah

[1. Zadanie úlohy 3](#_Toc86148401)

[2. Používateľské zozhranie 5](#_Toc86148402)

[3. Reprezentácia údajov problému 7](#_Toc86148403)

[4. Implementácia A\* algoritmu 8](#_Toc86148404)

[4.1. Teoretické vlastnosti algoritmu 8](#_Toc86148405)

[4.2. Heuristika 1 – počet políčok ktoré nie su na svojom mieste 9](#_Toc86148406)

[4.3. Heuristika 2 – súčet vzdialeností jednotlivých políčok od ich cieľovej pozície 9](#_Toc86148407)

[4.4. A\* algoritmus 10](#_Toc86148408)

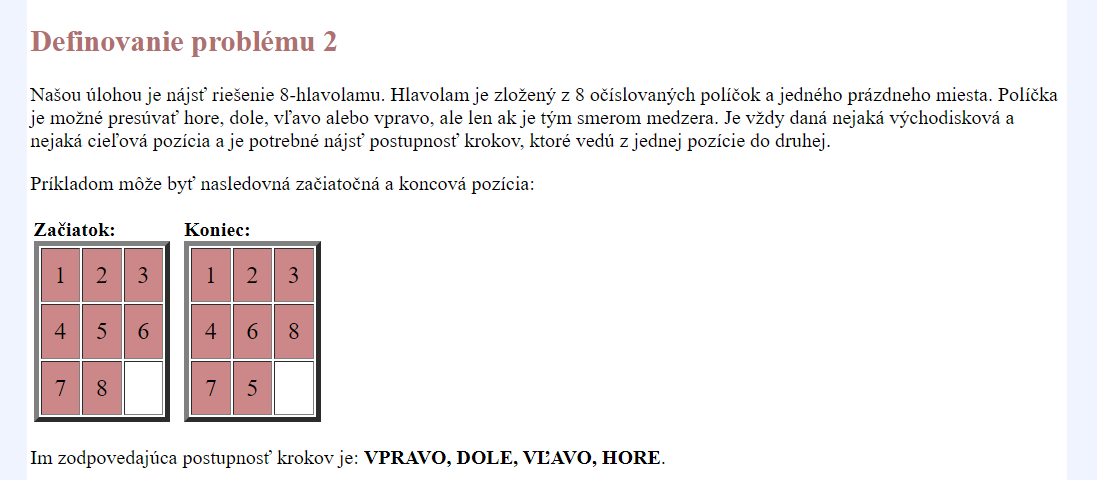
[5. Zhodnotenie riešenia 18](#_Toc86148409)

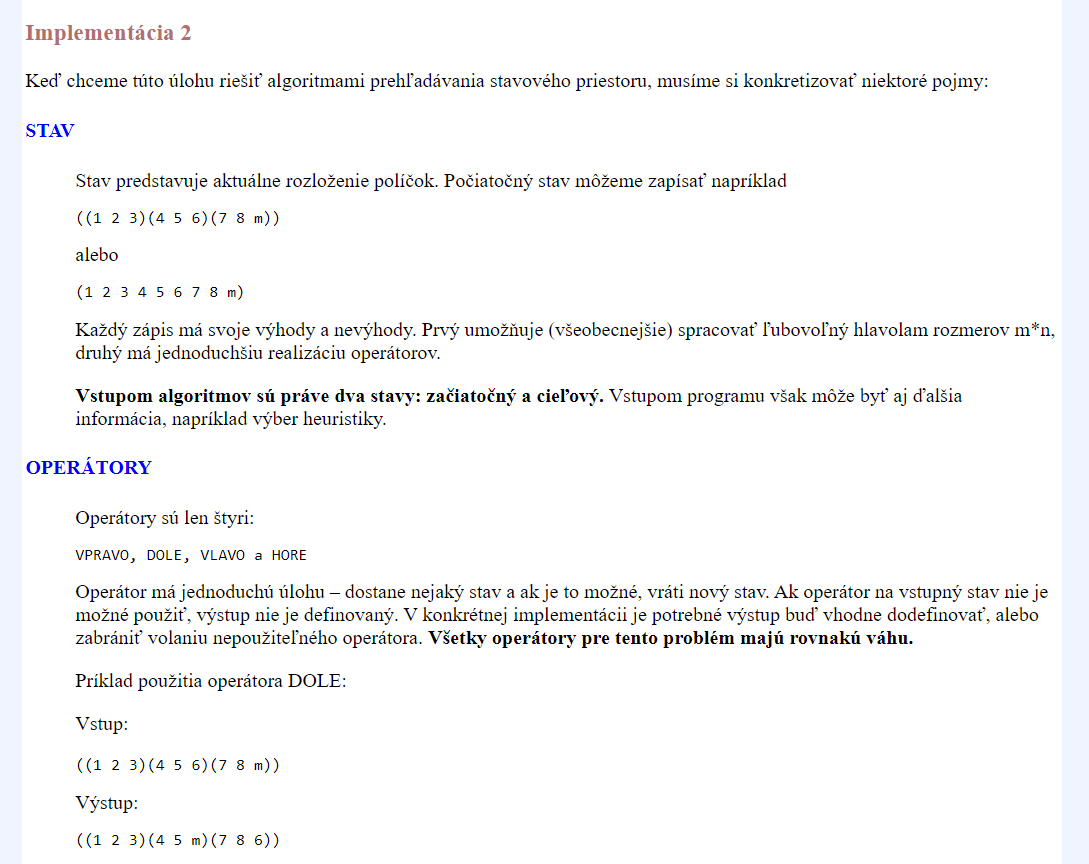
[5.1. Možnosti rozšírenia a optimalizácie 18](#_Toc86148410)

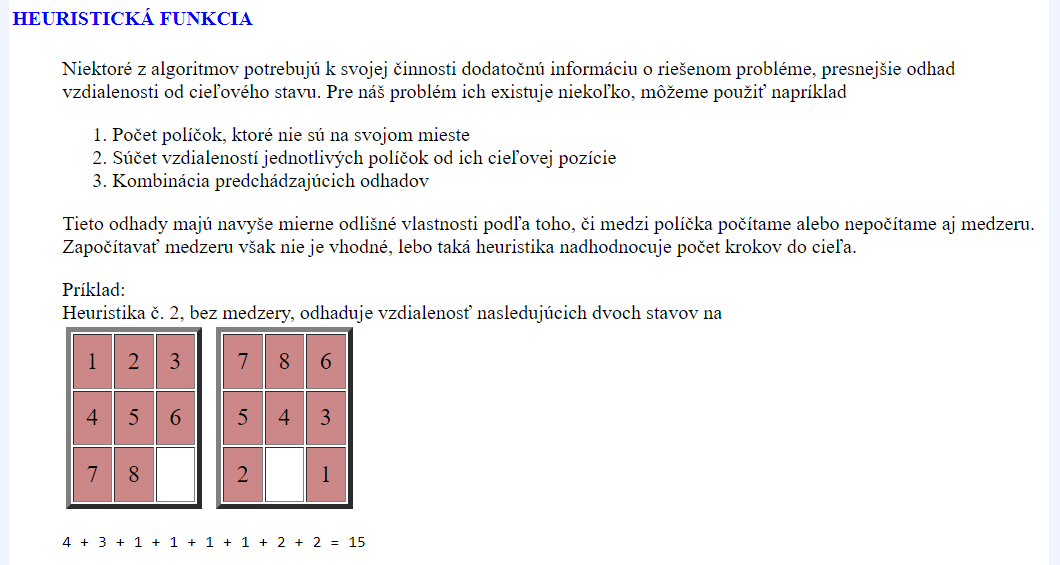
[6. Výsledky testovania 18](#_Toc86148411)

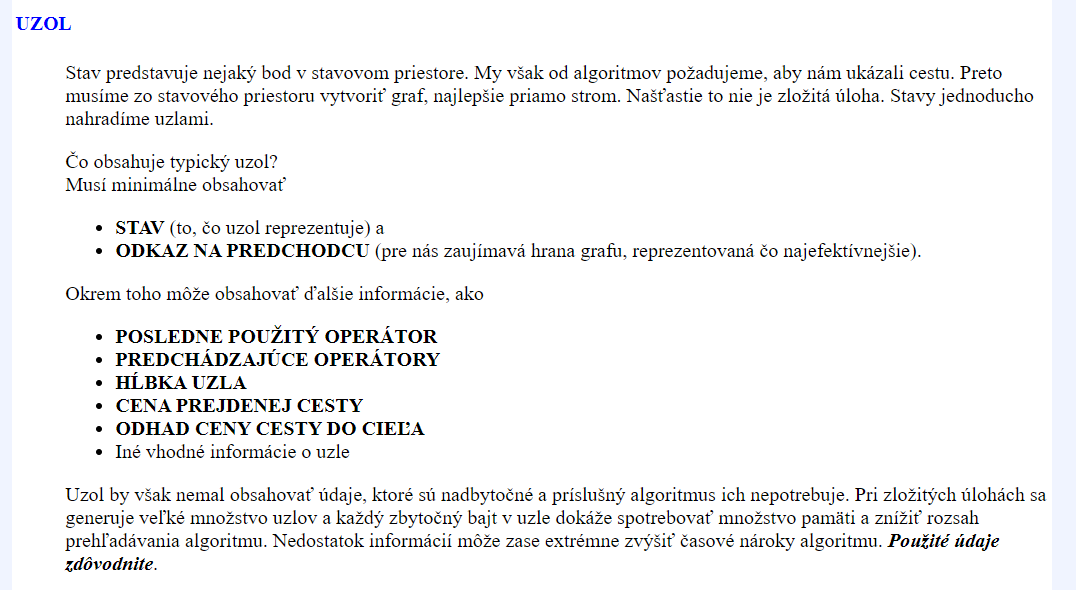
[6.1. Porovnanie vlastností použitých metód pre rôznu veľkosť 8-hlavolamu 21](#_Toc86148412)

# Zadanie úlohy

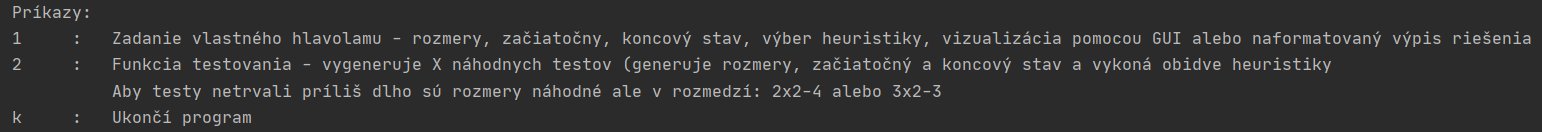






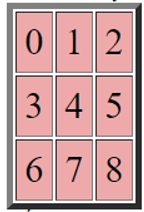
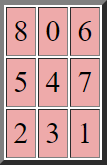


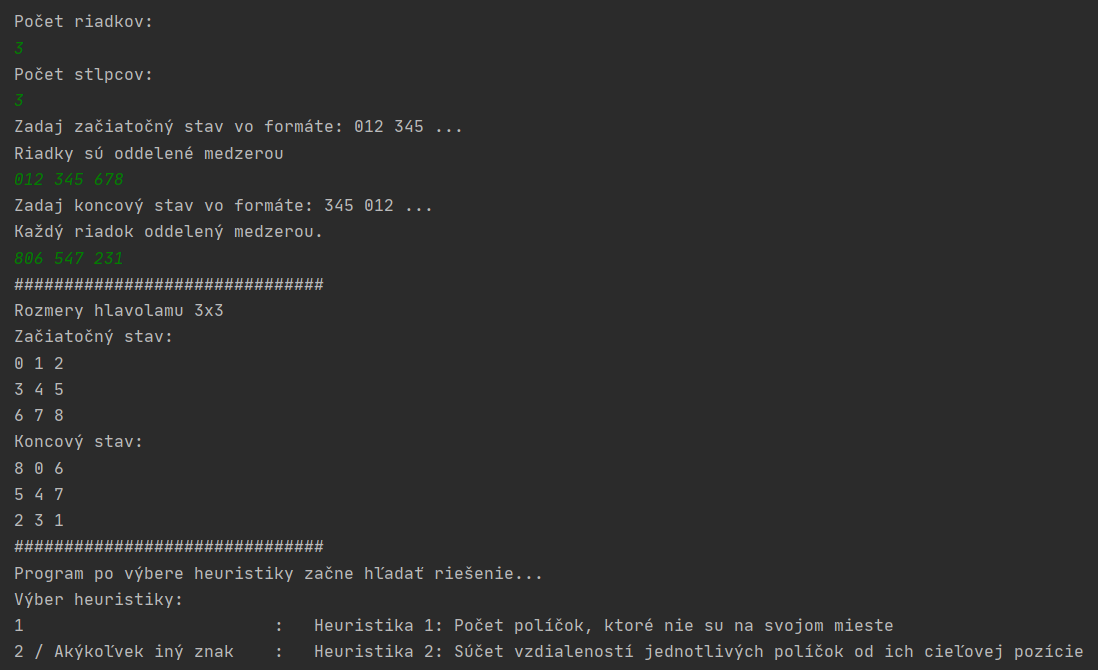
# Používateľské zozhranie

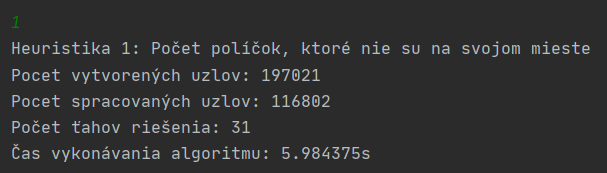


Príklad použitia príkazu 1 používateľského rozhrania:

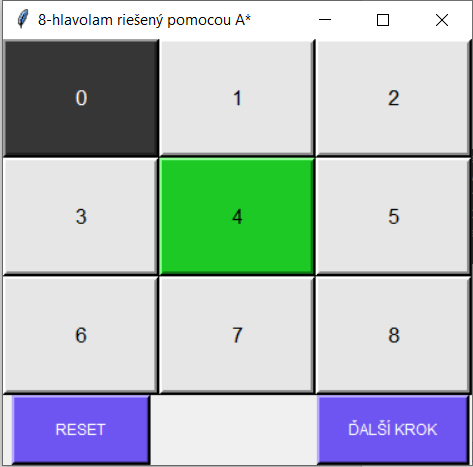
Začiatočný stav Koncový stav

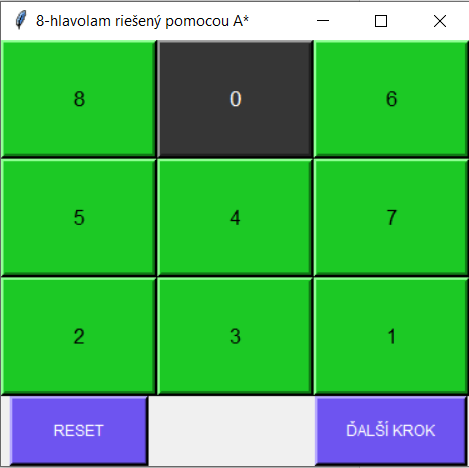
 



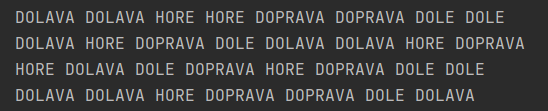


GUI vizualizácia



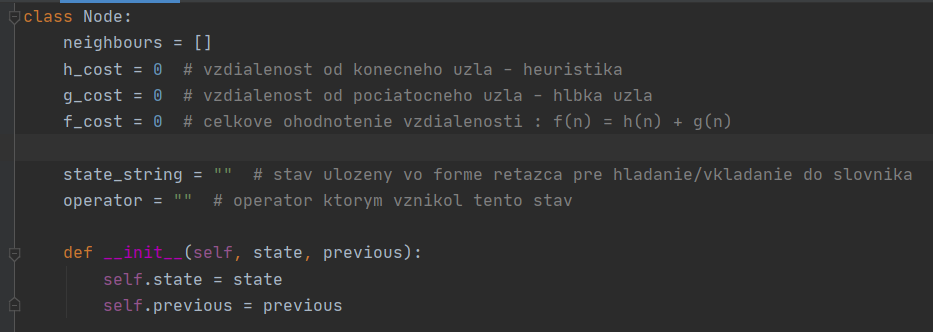


CLI formatovaný výpis

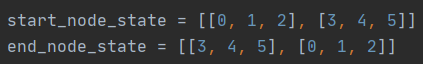


# Reprezentácia údajov problému

Trieda uzla:



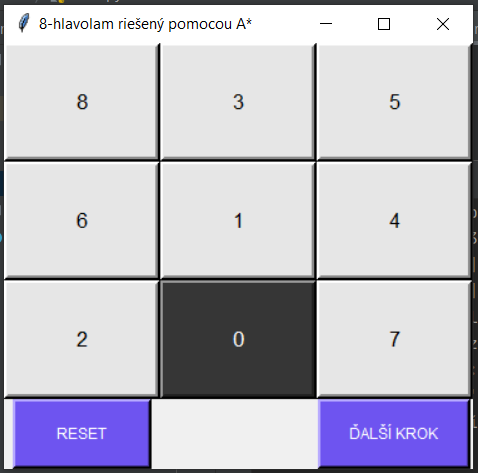
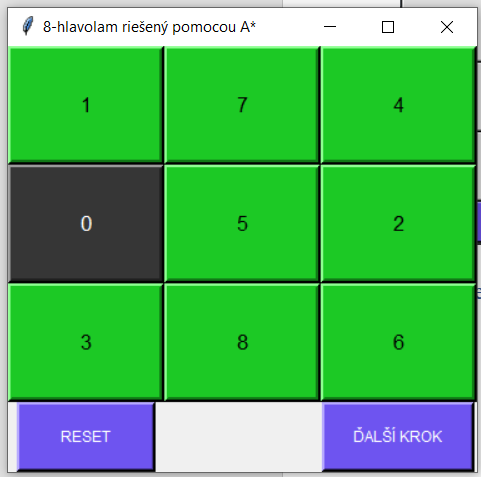
Reprezentácia stavov je realizovaná pomocou multidimenzionálneho poľa:



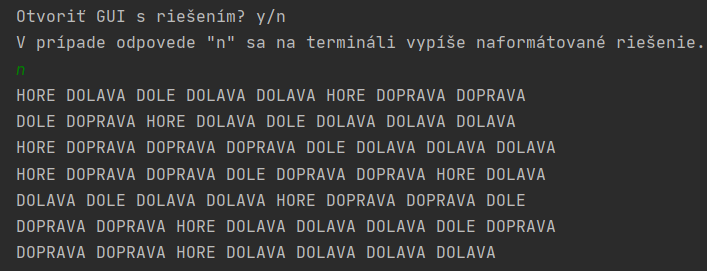
Reprezentácia výstupu / riešenia:



Program taktiež ponúka možnosť vizualizácie riešenia:

Alebo naformátovanie výpisu riešenia na terminál:



# Implementácia A\* algoritmu

## Teoretické vlastnosti algoritmu

A\* je počítačový algoritmus používaný na vyhľadávanie optimálnych ciest v kladne ohodnotených grafoch. Používa rovnaké princípy ako Djikstrov algoritmus ale pridáva naviac heuristický prvok.

A\* používa hladný princíp na nájdenie optimálnej cesty z daného počiatočného uzla do koncového. Optimálnou cestou rozumieme najkratšiu/najrýchlejšiu/najlacnejšiu cestu v závislosti na reprezentácií hodnôt hrán v grafe.

K tomu používa funkciu obvykle označenú ako f(x), ktorá ohodnocuje jednotlivé uzly na určenie ich poraia v ktorom sa majú prechádzať. Táto funkcia sa skladá z dvoch funkcií: f(x) = g(x) + h(x).

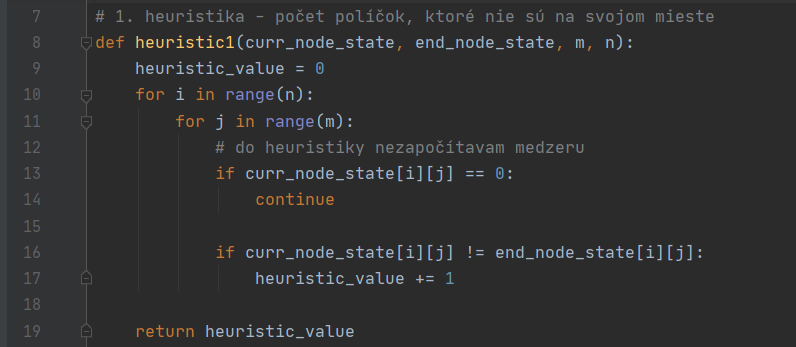
Funkcia g(x) je funkcia predstavujúca vzdialenosť medzi počiatočným a daným uzlom – v prípade problému 8-hlavolamu predstavuje g(x) hĺbku uzla.

Funkcia h(x) predstavuje heuristicku funkciu. Táto funkcia odhaduje správnosť postupu pri vyhľadávaní optimálnej cesty za pomoci vzdialenosti z aktuálneho uzlu do koncového. Funkcia musí byť prípustná, teda nesmie nahodnocovať vzdialenosť k cieľu. V prípade 8hlavolamu teda nechceme započítavať do výpočtu prazdné miesto.

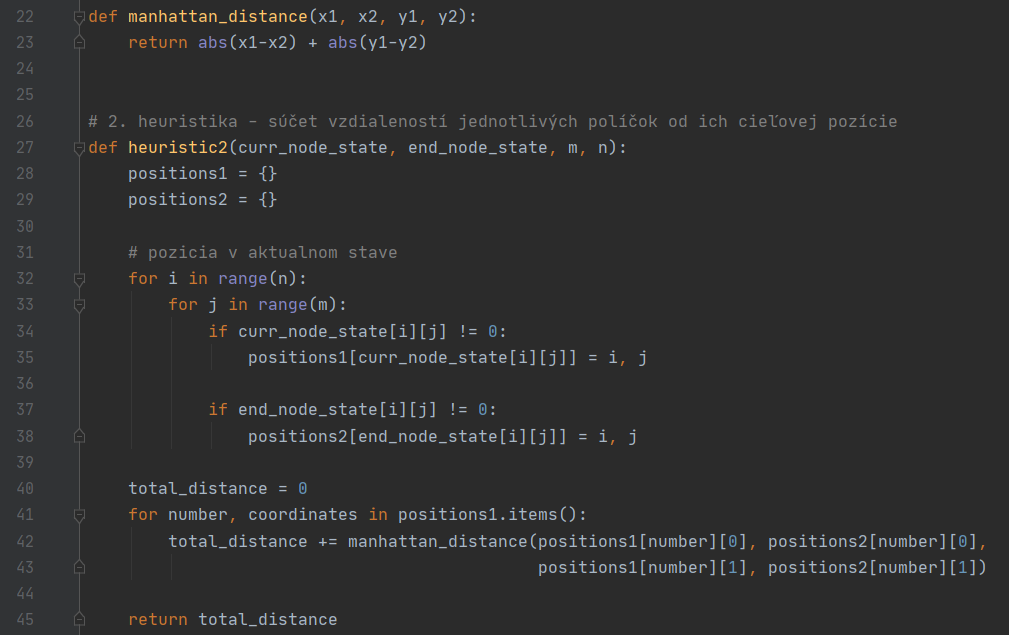
Samotný algoritmus prebieha nasledovne. Je vytvorená a udržovaná prioritný front otvorených (nenavštevených uzlov). Čím menšia je hodnota f(x) pre daný uzol tým vyššiu ma prioritu. V každom kroku algoritmu je uzol s najvyšou prioritou odobraný z prioritnej fronty a sú vypočítané hodnoty f a h pre jeho susedné uzly. Tieto uzly sú potom pridané do prioritnej fronty alebo sú znížené ich hodnoty ak sa tam už nachádzajú a nové hodnoty sú nižšie. Algoritmus pokračuje dokedy nemá konečný uzol menšiu hodnotu f, než ľubovoľný iný uzol z fronty alebo je tento front prázdny. Hodnota f koncového uzla je potom dĺžkou najkratšej cesty grafom. Ak je potrebné poznať aj konkrétnu cestu, je nutné udržovať aj zoznam uzlov na tejto ceste. Pre udržiavanie cesty si stačí pamätať v každom uzle jeho predchodcu na najkratšej ceste.

Zdroj: [https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)

## Heuristika 1 – počet políčok ktoré nie su na svojom mieste



## Heuristika 2 – súčet vzdialeností jednotlivých políčok od ich cieľovej pozície

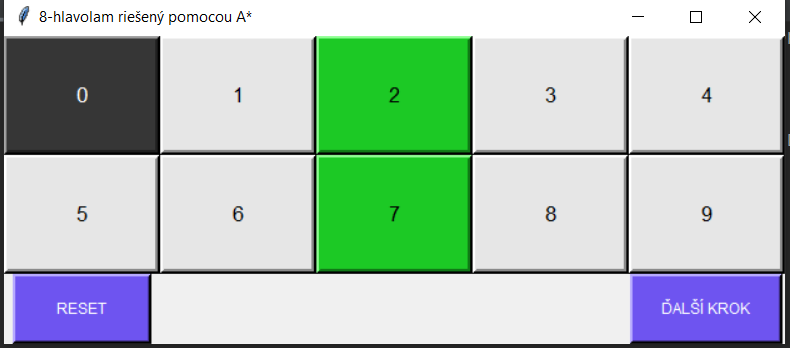


## A\* algoritmus

Na konci programu sa vypíše na termináli počet spracovaných uzlov a taktiež počet vytvorených uzlov. Program vypíše čas vykonávania daného príkladu a ak nemá riešenie tak program vypíše, že nemá riešenie. Uvediem 2 konkretné príklady, ďalšie sa dajú nájsť v sekcii testovania.

1. **Hlavolam 5\*2 (Heuristika: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície)**
   * Počet vytvorených uzlov 1 674 337
   * Počet spracovaných uzlov 1 051 455
   * Čas vykonávania 78.87s

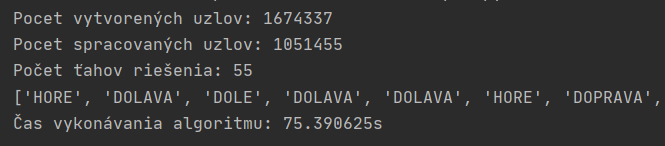
Počiatočný stav 01234 56789

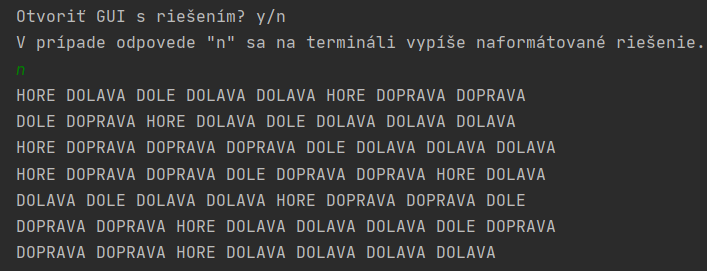


Koncový stav 43261 98750

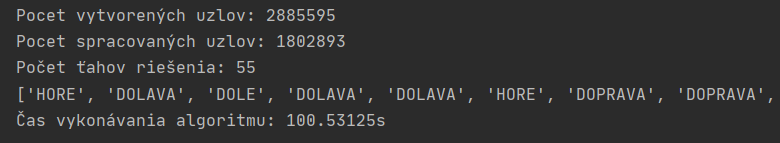


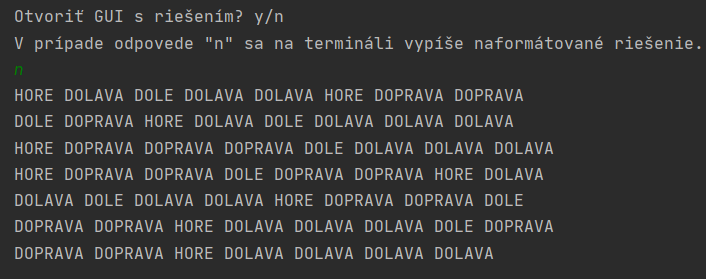
Program na konci vypíše všetky dôležité informácie spolu s riešením.





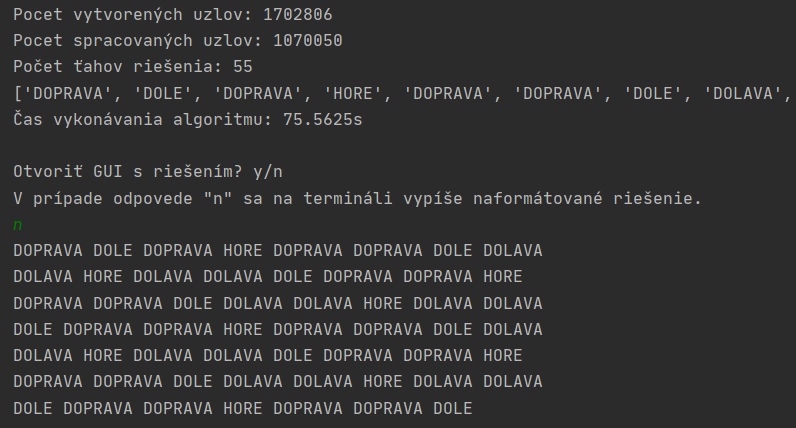
1. Rovnaký hlavolam 5\*2 **(Heuristika:** **Počet políčok, ktoré nie sú na svojom mieste)**
   * Počet vytvorených uzlov 2 885 595
   * Počet spracovaných uzlov 1 802 893
   * Čas vykonávania 100.5s





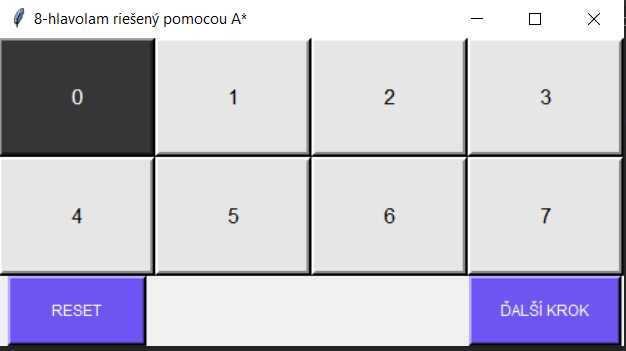
V prípade tohto konkrétneho hlavolamu a výmeny začiatočného a koncového stavu (**Heuristika: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície)** sú výsledky následovné:

Čas vykonania je takmer úplne rovnaký. Rozdiel môže byť spôsobený procesmi PC na pozadí.



1. Hlavolam 4\*2 **(Heuristika: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície)**
   * Počet vytvorených uzlov 17 937
   * Počet spracovaných uzlov 11 985
   * Čas vykonávania 0.6875s

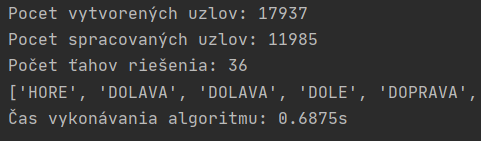
Počiatočný stav 0123 4567

****

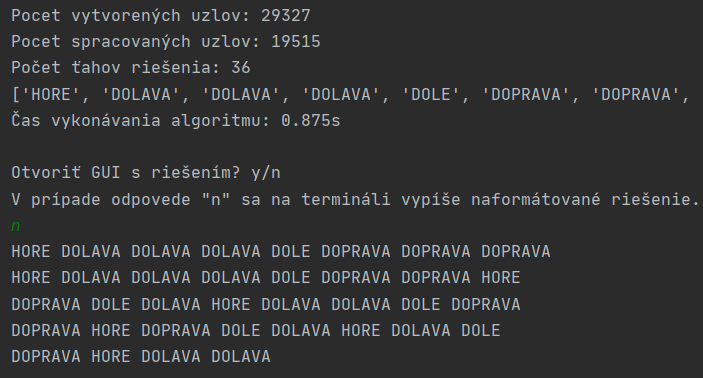
Koncový stav 3254 7610



Riešenie:

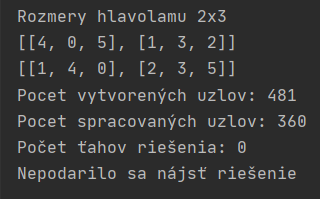


1. Rovnaký hlavolam 4\*2 **(Heuristika:** **Počet políčok, ktoré nie sú na svojom mieste)**
   * Počet vytvorených uzlov 29 327
   * Počet spracovaných uzlov 19 515
   * Čas vykonávania 0.875s



Vačšina príkladov 3x3 a menšie sa vykonajú zväčša okamžite.

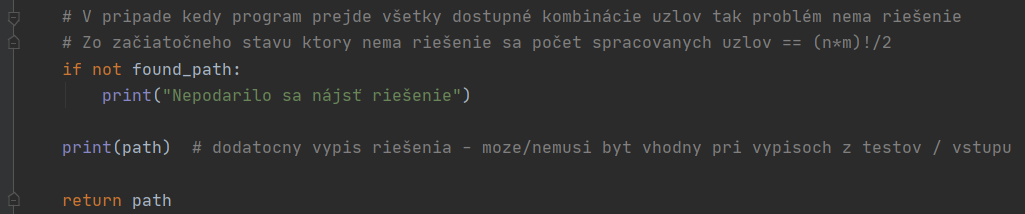
Príklady, ktoré nemajú riešenie sa počet spracovaných uzlov rovná (n\*m)!/2



A\* algoritmus je implementovaný vo funkcii a\_star. Argumentmi tejto funkcie je heuristika (referencia na funkciu), začiatočný stav, koncový stav a rozmery hlavolamu.



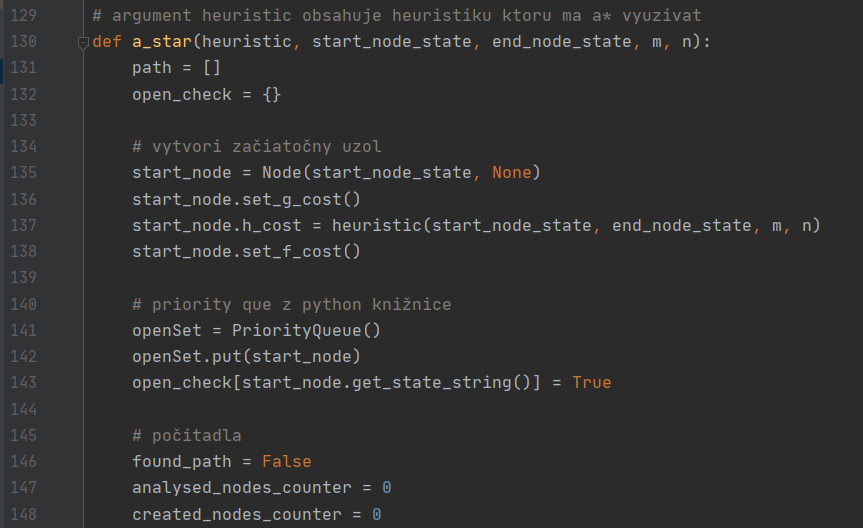
Ak je problém vyriešiteľný, tak funkcia vráti riešenie v podobe poľa operátorov. V opačnom prípade funkcia vráti prázdne pole.



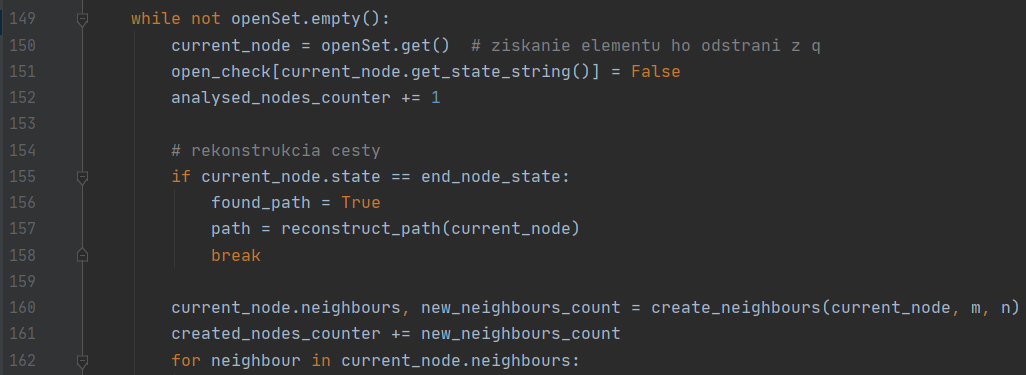
Na začiatku funkcie si program vytvorí pole na uloženie cesty a slovník, ktorý slúži na zisťovanie či program daný uzol/stav analyzoval.

Ďalej sa vytvorí začiatočný uzol s počiatočným stavom, nastavia sa jednotlivé g, h, f ceny.

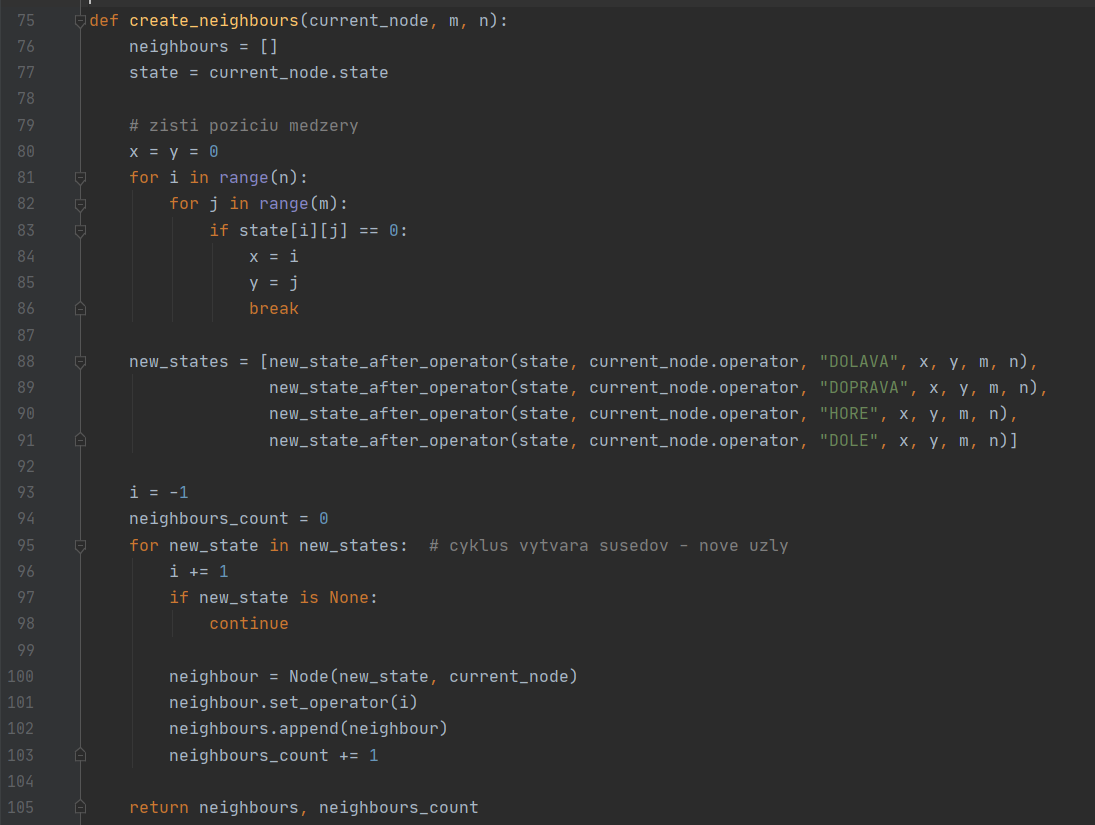
Uzly, ktoré ešte neboli analyzované sú uložené v openSet. Ten je realizovaný pomocu datovej štruktúry priorityQue, ktorá zoradzuje uzly podľa f\_cost. To je hĺbka uzla (g\_cost) + ohodnotenie heuristiky (h\_cost). Vďaka tejto implementácii prebieha výber ďalšieho uzla s najnižším f\_cost v O(1). Nie je nutné hľadať uzol s najnižším f\_cost ako by to bolo pri implementácií pomocou poľa/listu.



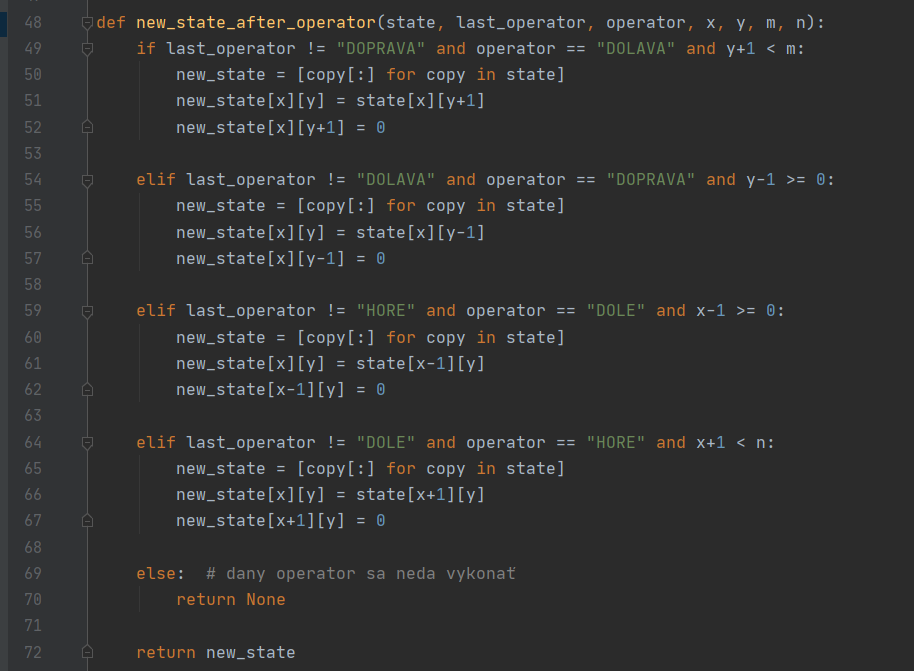
Hlavný cyklus A\* algoritmu beží dokedy sa nenašlo riešenie alebo neboli všetky uzly z openSet spracované. Pre každý spracovaný uzol sa nastaví hodnota v slovníku open\_check na False aby bolo jasné, že tento uzol sa už nenachádza v openSet – bol spracovaný.  
Ak aktuálny uzol nie je riešením tak sa vytvoria jeho susedia.



Program zistí súradnice medzery a následne vytvorí nové stavy, ktoré budú uložené do uzlov.



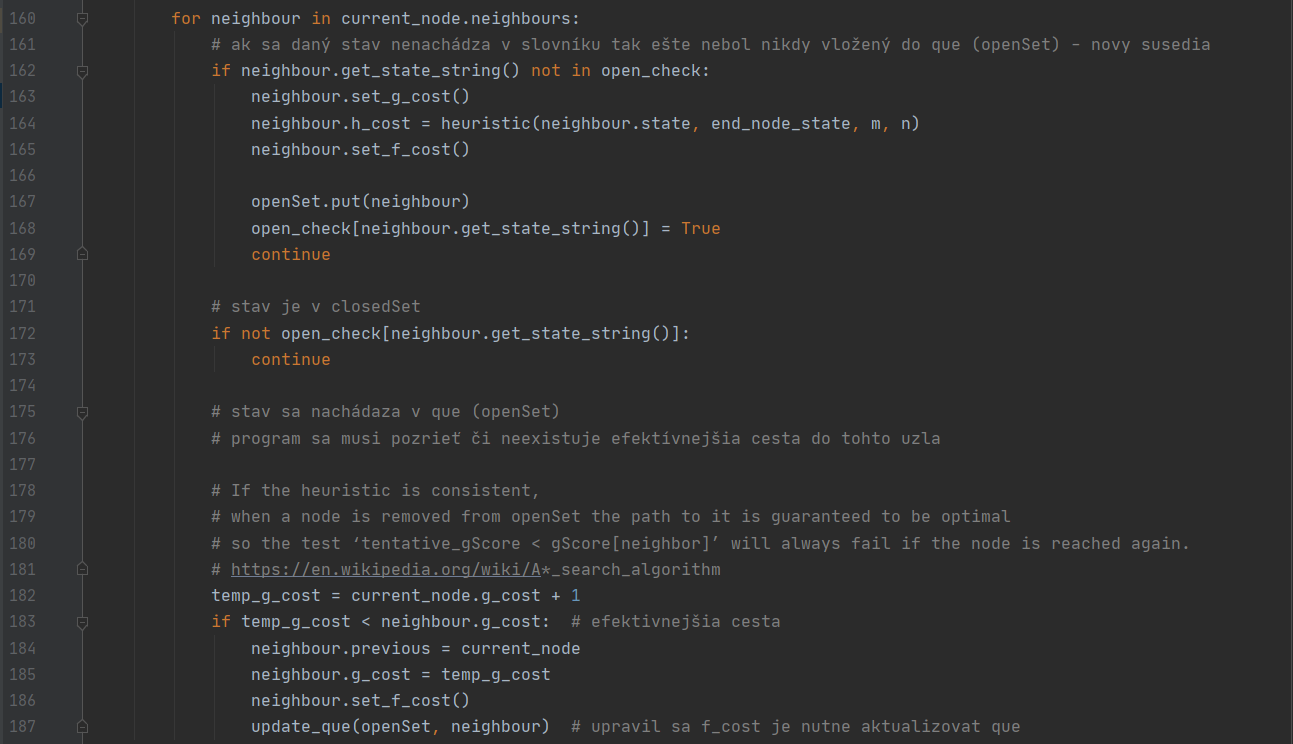
Pri generovaní stavov som program optimalizoval tak aby negeneroval spätný ťah.



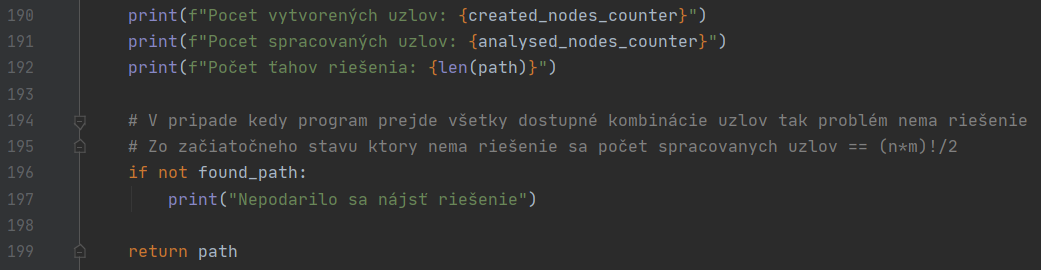
Po vytvorení susedov program zisťuje či daný sused ešte nebol v openSet ak nebol tak ho tam vloží a nastaví mu f, g, h ceny. Ak sa daný stav/sused nachádza v openSet tak program skontroluje či sa nenašla efektívnejšia cesta do tohto uzla.

Pri implementovaní tejto časti algoritmu som postupoval podľa inštrukcií a pseudo kódu zo stránok:

1. [https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)
2. <https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2>



Koniec funkcie:



# Zhodnotenie riešenia

## Možnosti rozšírenia a optimalizácie

Vrámci mojej implementácie som vykonal tieto optimalizácie:

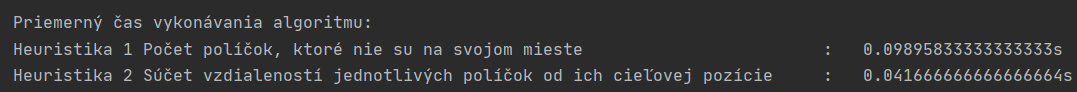
1. Vytvorené uzly s novými stavmi sú vkladané do priorityQue, aby boli zoradené podľa ich f\_cost a nebolo nutné vykonávať vyhľadávanie najmenšieho prvku.
2. Spracované uzly a uzly ktoré sa nachádzajú v openSet sú označené v slovníku open\_check. Není teda nutné zbytočne ukládať uzly, ktoré už boli spracované a taktiež nie je nutné vykonávať ich vyhľadanie.
3. Algoritmus negeneruje spätný ťah. To znamená, že napríklad ak rodičovský uzol bol vytvorený zo svojho predchodcu posunom políčka doľava, nebude sa generovať jeho potomok posunom doprava. Ušetrí sa tým zbytočné vytváranie uzla a kontrola stavu, ktorý už bol spracovaný.

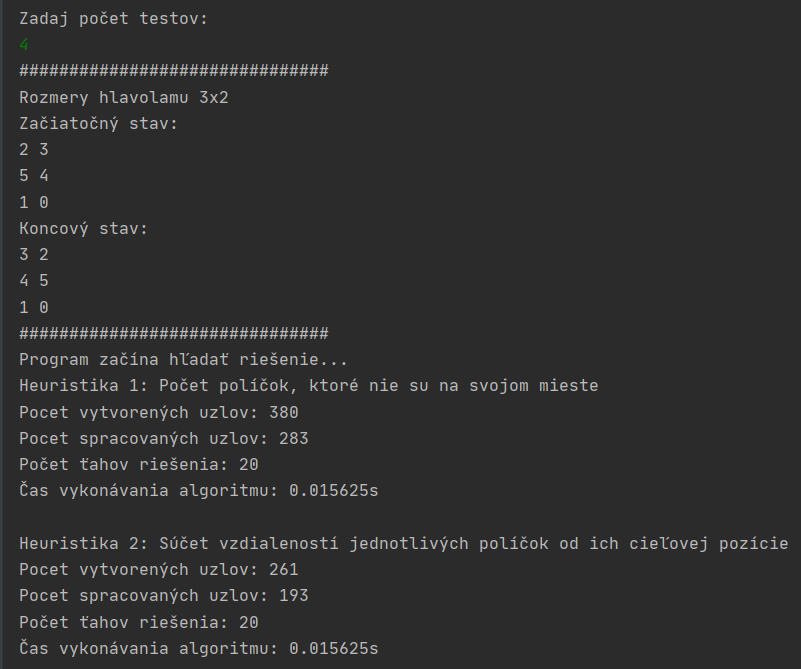
Vrámci optimalizácie som rozmýšľal ešte nad ďalšími možnosťami ale vzhľadom na moje obmedzené skúsenosti s jazykom Python mi už nič iné nenapadlo.

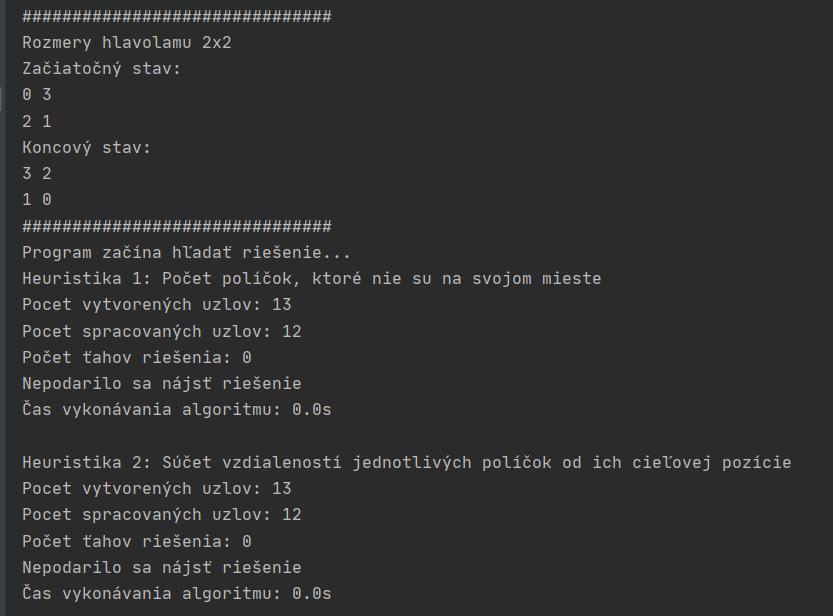
# Výsledky testovania

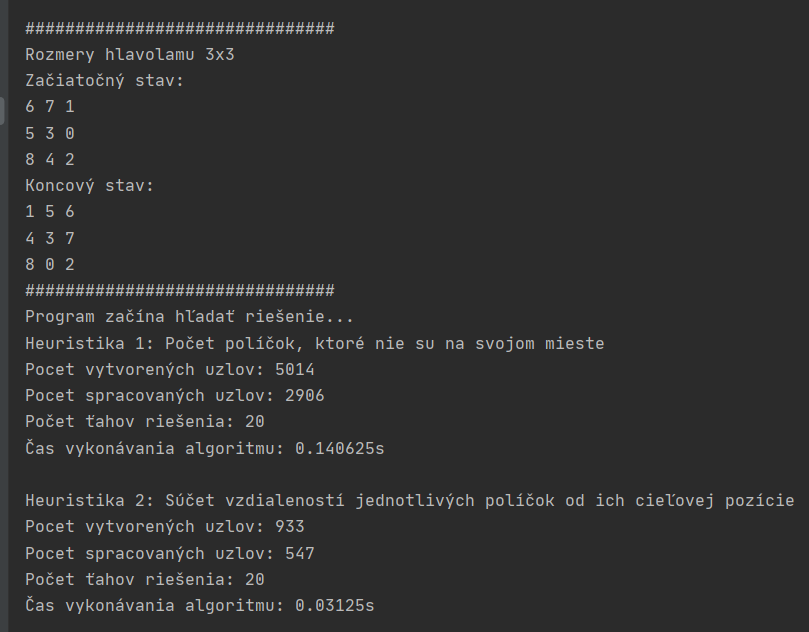
Program poskytuje možnosť testovania pomocou náhodneho generovania hlavolamov. Generuje aj hlavolamy ktoré nie sú riešiteľne, v takomto prípade sa čas vykonávania funkcie nezapočitáva do priemeru. V prípade ak je začiatočný stav rovný konečnému tak čas vykonania by bol 0s a teda ošetrenie v rámci testovania nespôsobí skreslenie výsledkov.

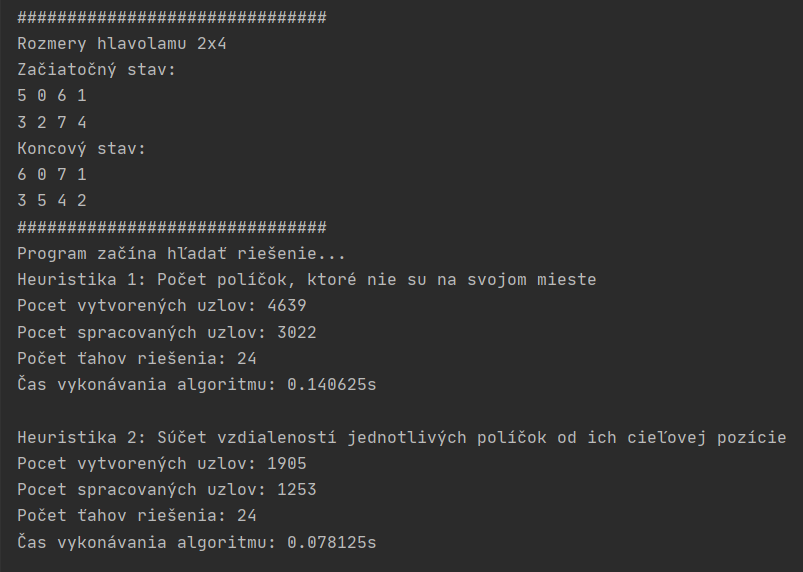
Na konci testovania z X testov program vypíše ich priemerný čas vykonania pre jednotlivé heuristiky. Z výpisov jednotlivých testov sa dá taktiež určiť koľko uzlov, ktorá heuristik musela vytvoriť a spracovať.









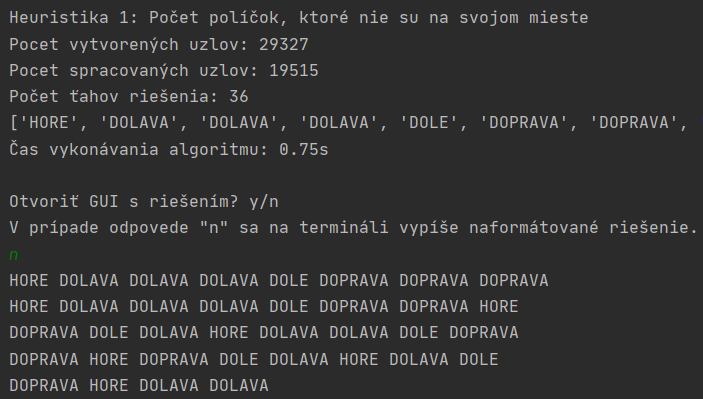


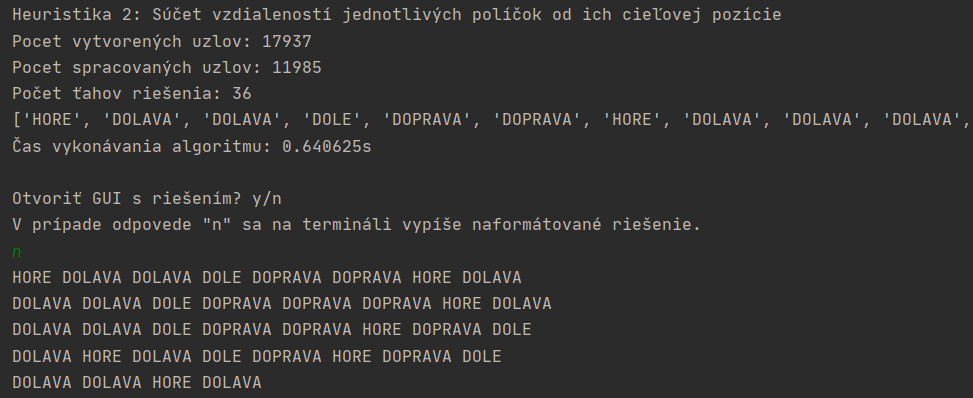
Vo všetkých testoch sa heuristika 2 ukázala ako efektívnejšia a rýchlejšia varianta. Čas vykonania je nižší, rozvíja menej uzlov a menej ich aj spracuje. V prípade akýchkoľvek ďalších testov stačí len spustiť možnosť “2“ v programe.

## Porovnanie vlastností použitých metód pre rôznu veľkosť 8-hlavolamu

Hlavolam 2x4:

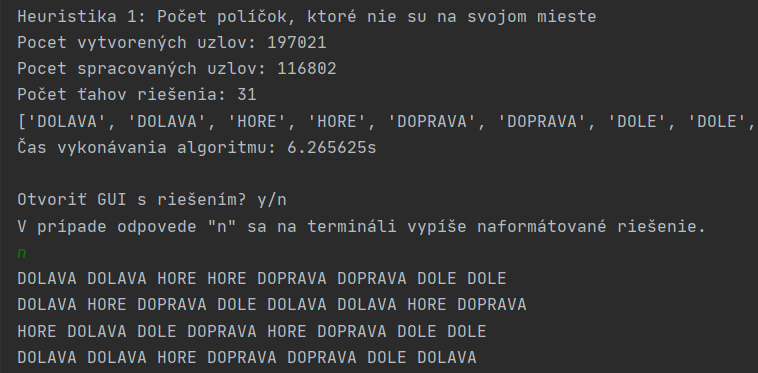
* začiatočný stav – 0123 4567
* koncový stav – 3254 7610

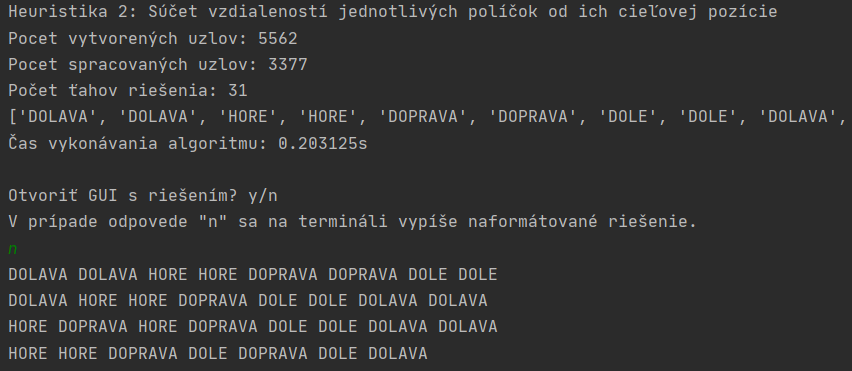




Hlavolam 3x3:

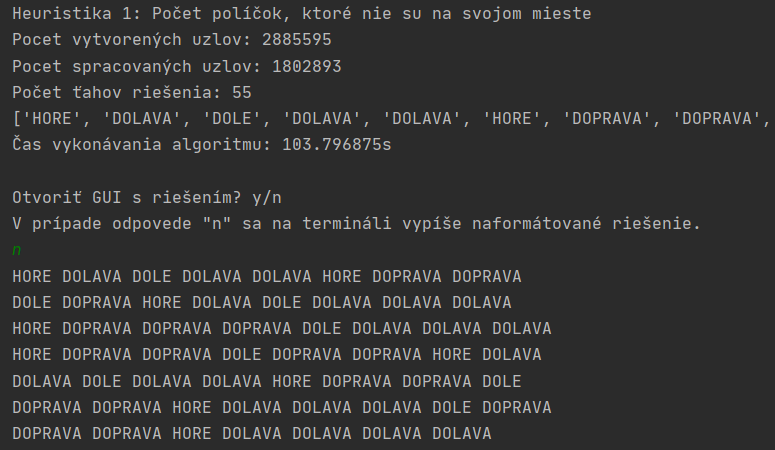
* začiatočný stav – 012 345 678
* koncový stav - 806 547 231

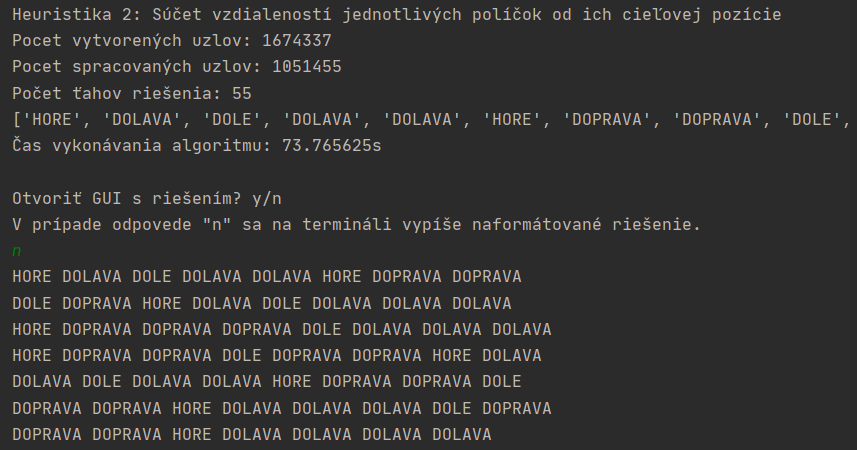




Hlavolam 2x5:

* začiatočný stav – 01234 56789
* koncový stav - 43261 98750

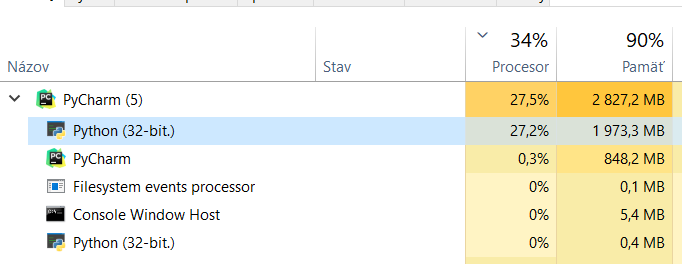


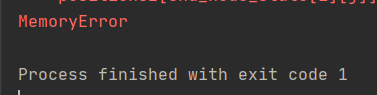


Hlavolam 3x4:

* začiatočný stav – 0,1,2,3 4,5,6,7 8,9,10,11
* koncový stav - 3,2,10 11,6,5,4 7,10,9,8

Môj notebook s 3GB voľnej RAM nestačil.





Vo všetkých prípadoch narastá počet vytvorených, spracovaných uzlov spolu s časom potrebným na získanie riešenia. Aj keď počet stavov závisí hlavne od rozmerov hlavolamu taktiež závisí od toho aký je začiatočný a koncový stav. Napríklad v prípade hlavolamu 3x4 kedy sa riešenie nájde už po treťom ťahu bude počet vytvorených uzlov menší oproti hlavolamu 3x3 kde sa musí hľadať riešenie pomocou 10 ťahov.

Vrámci zisťovania vlastností ohľadom 8-hlavolamu som taktiež objavil informáciu, že počet dostupných stavov ľubovoľného hlavolamu n\*m je rovný (n\*m)!/2 zatiaľ čo počet možných kombinácií začiatočného/koncového stavu je (n\*m)!. Preto pri problémoch, ktoré nemajú riešenie môj algoritmus spracuje (n\*m)!/2 uzlov a skončí s hláškou, že sa nenašlo riešenie.