Meno: Jakub Šimko
AIS ID: 103146

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

Meno: Jakub Šimko
AIS ID: 103146

Obsah

Meno: Jakub Šimko
AIS ID: 103146

1. Zadanie úlohy

Definovanie problému 2

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.

Príkladom môže byť nasledovná začiatočná a koncová pozícia:

Začiatok:

1	2	3
4	5	6
7	8	

Koniec:

1	2	3
4	6	8
7	5	

Im zodpovedajúca postupnosť krokov je: **VPRAVO, DOLE, VĽAVO, HORE**.

Implementácia 2

Keď chceme túto úlohu riešiť algoritmami prehľadávania stavového priestoru, musíme si konkretizovať niektoré pojmy:

STAV

Stav predstavuje aktuálne rozloženie políčok. Počiatočný stav môžeme zapísať napríklad

```
((1 2 3)(4 5 6)(7 8 m))
```

alebo

```
(1 2 3 4 5 6 7 8 m)
```

Každý zápis má svoje výhody a nevýhody. Prvý umožňuje (všeobecnejšie) spracovať ľubovoľný hlavolam rozmerov m*n, druhý má jednoduchšiu realizáciu operátorov.

Vstupom algoritmov sú práve dva stavy: začiatočný a cieľový. Vstupom programu však môže byť aj ďalšia informácia, napríklad výber heuristiky.

OPERÁTORY

Operátory sú len štyri:

```
VPRAVO, DOLE, VLAVO a HORE
```

Operátor má jednoduchú úlohu – dostane nejaký stav a ak je to možné, vráti nový stav. Ak operátor na vstupný stav nie je možné použiť, výstup nie je definovaný. V konkrétnej implementácii je potrebné výstup buď vhodne dodefinovať, alebo zabrániť volaniu nepoužiteľného operátora. **Všetky operátory pre tento problém majú rovnakú váhu.**

Príklad použitia operátora DOLE:

Vstup:

```
((1 2 3)(4 5 6)(7 8 m))
```

Výstup:

```
((1 2 3)(4 5 m)(7 8 6))
```

Meno: Jakub Šimko
AIS ID: 103146

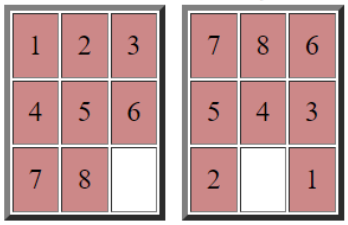
HEURISTICKÁ FUNKCIA

Niektoré z algoritmov potrebujú k svojej činnosti dodatočnú informáciu o riešenom probléme, presnejšie odhad vzdialenosti od cieľového stavu. Pre náš problém ich existuje niekoľko, môžeme použiť napríklad

1. Počet políčok, ktoré nie sú na svojom mieste
2. Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
3. Kombinácia predchádzajúcich odhadov

Tieto odhady majú navyše mierne odlišné vlastnosti podľa toho, či medzi políčka počítame alebo nepočítame aj medzeru. Započítavať medzeru však nie je vhodné, lebo taká heuristika nadhodnocuje počet krokov do cieľa.

Príklad:
Heuristika č. 2, bez medzery, odhaduje vzdialenosť nasledujúcich dvoch stavov na




```
4 + 3 + 1 + 1 + 1 + 1 + 2 + 2 = 15
```

UZOL

Stav predstavuje nejaký bod v stavovom priestore. My však od algoritmov požadujeme, aby nám ukázali cestu. Preto musíme zo stavového priestoru vytvoriť graf, najlepšie priamo strom. Našťastie to nie je zložitá úloha. Stavy jednoducho nahradíme uzlami.

Čo obsahuje typický uzol?
Musí minimálne obsahovať

- **STAV** (to, čo uzol reprezentuje) a
- **ODKAZ NA PREDCHODCU** (pre nás zaujímavá hrana grafu, reprezentovaná čo najefektívnejšie).

Okrem toho môže obsahovať ďalšie informácie, ako

- **POSLEDNE POUŽITÝ OPERÁTOR**
- **PREDCHÁDZAJÚCE OPERÁTORY**
- **HĹBKA UZLA**
- **CENA PREJDENEJ CESTY**
- **ODHAD CENY CESTY DO CIEĽA**
- Iné vhodné informácie o uzle

Uzol by však nemal obsahovať údaje, ktoré sú nadbytočné a príslušný algoritmus ich nepotrebuje. Pri zložitých úlohách sa generuje veľké množstvo uzlov a každý zbytočný bajt v uzle dokáže spotrebovať množstvo pamäti a znížiť rozsah prehľadávania algoritmu. Nedostatok informácií môže zase extrémne zvýšiť časové nároky algoritmu. ***Použité údaje zdôvodnite***.

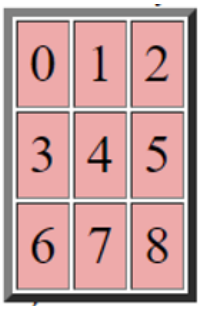
Meno: Jakub Šimko
AIS ID: 103146

2. Používateľské zozhranie

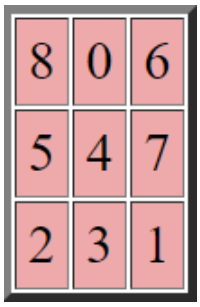
```
Príkazy:
1     :   Zadanie vlastného hlavolamu - rozmery, začiatočny, koncový stav, výber heuristiky, vizualizácia pomocou GUI alebo naformatovaný výpis riešenia
2     :   Funkcia testovania - vygeneruje X náhodnych testov (generuje rozmery, začiatočný a koncový stav a vykoná obidve heuristiky
          Aby testy netrvali príliš dlho sú rozmery náhodné ale v rozmedzí: 2x2-4 alebo 3x2-3
k     :   Ukončí program
```

Príklad použitia príkazu 1 používateľského rozhrania:

Začiatočný stav



Koncový stav

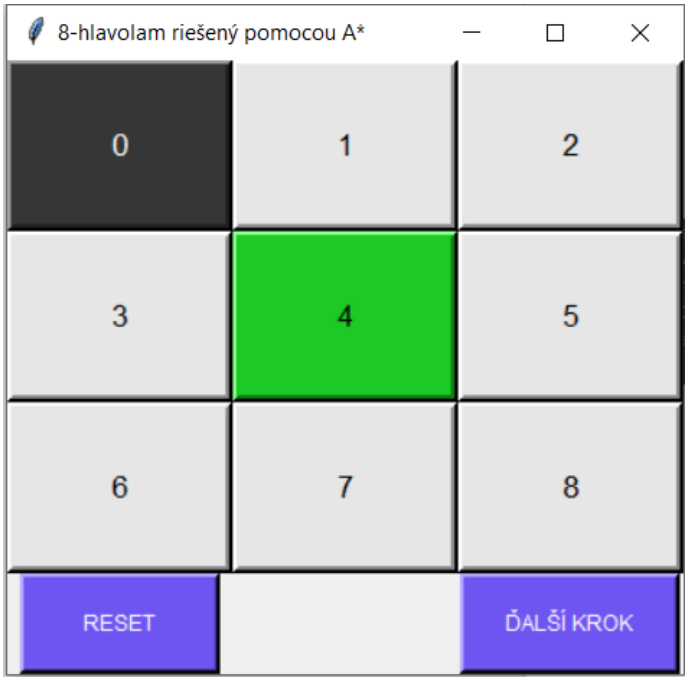


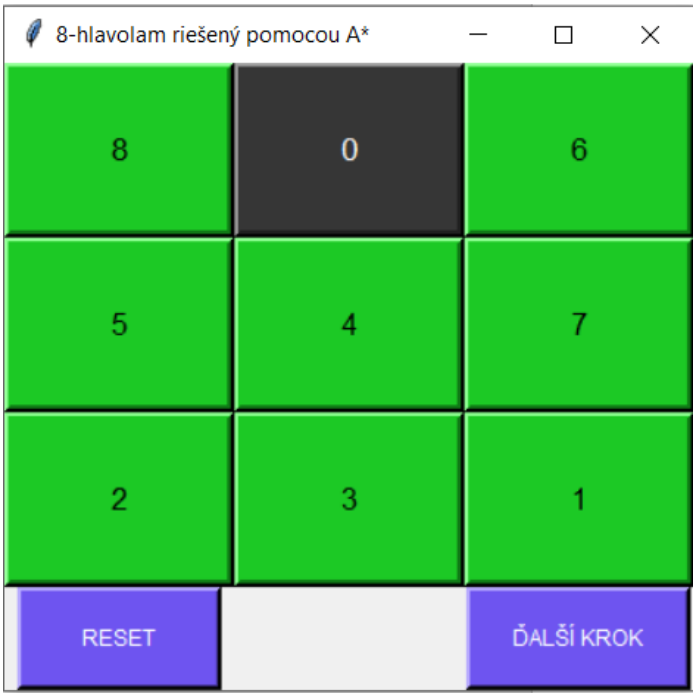

```
Počet riadkov:
3
Počet stlpcov:
3
Zadaj začiatočný stav vo formáte: 012 345 ...
Riadky sú oddelené medzerou
012 345 678
Zadaj koncový stav vo formáte: 345 012 ...
Každý riadok oddelený medzerou.
806 547 231
###############################
Rozmery hlavolamu 3x3
Začiatočný stav:
0 1 2
3 4 5
6 7 8
Koncový stav:
8 0 6
5 4 7
2 3 1
###############################
Program po výbere heuristiky začne hľadať riešenie...
Výber heuristiky:
1                       :   Heuristika 1: Počet políčok, ktoré nie su na svojom mieste
2 / Akýkoľvek iný znak  :   Heuristika 2: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
```

```
1
Heuristika 1: Počet políčok, ktoré nie su na svojom mieste
Pocet vytvorených uzlov: 197021
Pocet spracovaných uzlov: 116802
Počet ťahov riešenia: 31
Čas vykonávania algoritmu: 5.984375s
```

Meno: Jakub Šimko
AIS ID: 103146

GUI vizualizácia





CLI formatovaný výpis

```
DOLAVA DOLAVA HORE HORE DOPRAVA DOPRAVA DOLE DOLE
DOLAVA HORE DOPRAVA DOLE DOLAVA DOLAVA HORE DOPRAVA
HORE DOLAVA DOLE DOPRAVA HORE DOPRAVA DOLE DOLE
DOLAVA DOLAVA HORE DOPRAVA DOPRAVA DOLE DOLAVA
```

Meno: Jakub Šimko
AIS ID: 103146

3. Reprezentácia údajov problému

Trieda uzla:

```
class Node:
    neighbours = []
    h_cost = 0  # vzdialenost od konecneho uzla - heuristika
    g_cost = 0  # vzdialenost od pociatocneho uzla - hlbka uzla
    f_cost = 0  # celkove ohodnotenie vzdialenosti : f(n) = h(n) + g(n)

    state_string = ""  # stav ulozeny vo forme retazca pre hladanie/vkladanie do slovnika
    operator = ""  # operator ktorym vznikol tento stav

    def __init__(self, state, previous):
        self.state = state
        self.previous = previous
```

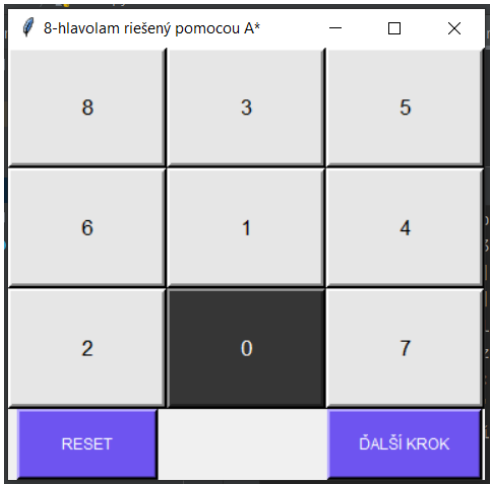
Reprezentácia stavov je realizovaná pomocou multidimenzionálneho poľa:

```
start_node_state = [[0, 1, 2], [3, 4, 5]]
end_node_state = [[3, 4, 5], [0, 1, 2]]
```

Reprezentácia výstupu / riešenia:

```
['DOLE', 'DOLE', 'DOPRAVA', 'HORE', 'DOPRAVA', 'HORE', 'DOLAVA', 'DOLAVA', 'DOLE', 'DOPRAVA', 'DOPRAVA', 'DOLE', 'DOLAVA', 'HORE']
```

Program taktiež ponúka možnosť vizualizácie riešenia:



Alebo naformátovanie výpisu riešenia na terminál:

```
Otvoriť GUI s riešením? y/n
V prípade odpovede "n" sa na termináli vypíše naformátované riešenie.
n
HORE DOLAVA DOLE DOLAVA DOLAVA HORE DOPRAVA DOPRAVA
DOLE DOPRAVA HORE DOLAVA DOLE DOLAVA DOLAVA DOLAVA
HORE DOPRAVA DOPRAVA DOPRAVA DOLE DOLAVA DOLAVA DOLAVA
HORE DOPRAVA DOPRAVA DOLE DOPRAVA DOPRAVA HORE DOLAVA
DOLAVA DOLE DOLAVA DOLAVA HORE DOPRAVA DOPRAVA DOLE
DOPRAVA DOPRAVA HORE DOLAVA DOLAVA DOLAVA DOLE DOPRAVA
DOPRAVA DOPRAVA HORE DOLAVA DOLAVA DOLAVA DOLAVA
```

Meno: Jakub Šimko
AIS ID: 103146

4. Implementácia A* algoritmu

4.1. 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

Meno: Jakub Šimko
AIS ID: 103146

4.2. Heuristika 1 – počet políčok ktoré nie su na svojom mieste

```python
# 1. heuristika - počet políčok, ktoré nie sú na svojom mieste
def heuristic1(curr_node_state, end_node_state, m, n):
    heuristic_value = 0
    for i in range(n):
        for j in range(m):
            # do heuristiky nezapočítavam medzeru
            if curr_node_state[i][j] == 0:
                continue

            if curr_node_state[i][j] != end_node_state[i][j]:
                heuristic_value += 1

    return heuristic_value
```

4.3. Heuristika 2 – súčet vzdialeností jednotlivých políčok od ich cieľovej pozície

```python
def manhattan_distance(x1, x2, y1, y2):
    return abs(x1-x2) + abs(y1-y2)


# 2. heuristika - súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
def heuristic2(curr_node_state, end_node_state, m, n):
    positions1 = {}
    positions2 = {}

    # pozicia v aktualnom stave
    for i in range(n):
        for j in range(m):
            if curr_node_state[i][j] != 0:
                positions1[curr_node_state[i][j]] = i, j

            if end_node_state[i][j] != 0:
                positions2[end_node_state[i][j]] = i, j

    total_distance = 0
    for number, coordinates in positions1.items():
        total_distance += manhattan_distance(positions1[number][0], positions2[number][0],
                                             positions1[number][1], positions2[number][1])

    return total_distance
```

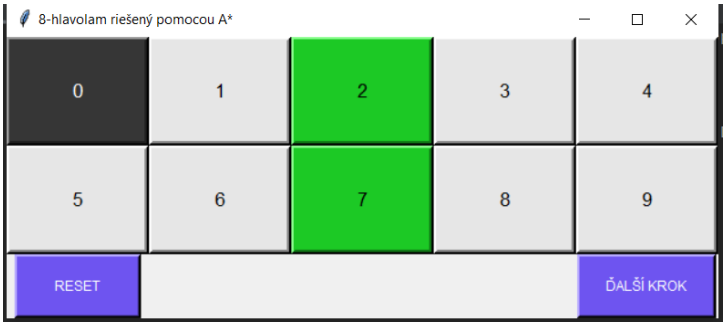
Meno: Jakub Šimko
AIS ID: 103146

4.4. A* algoritmus

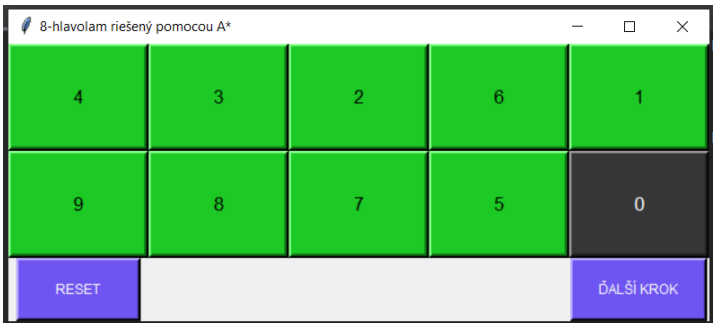
Na konci programu sa vypíše na termináli počet spracovaných uzlov a taktiež počet vytvorených uzlov. Program vypíše čas vykonávania daného príkladu a ak nemá riešenie tak program vypíše, že nemá riešenie. Uvediem 2 konkretné príklady, ďalšie sa dajú nájsť v sekcii testovania.

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.

```
Pocet vytvorených uzlov: 1674337
Pocet spracovaných uzlov: 1051455
Počet ťahov riešenia: 55
['HORE', 'DOLAVA', 'DOLE', 'DOLAVA', 'DOLAVA', 'HORE', 'DOPRAVA',
Čas vykonávania algoritmu: 75.390625s
```

```
Otvoriť GUI s riešením? y/n
V prípade odpovede "n" sa na termináli vypíše naformátované riešenie.
n
HORE DOLAVA DOLE DOLAVA DOLAVA HORE DOPRAVA DOPRAVA
DOLE DOPRAVA HORE DOLAVA DOLE DOLAVA DOLAVA DOLAVA
HORE DOPRAVA DOPRAVA DOPRAVA DOLE DOLAVA DOLAVA DOLAVA
HORE DOPRAVA DOPRAVA DOLE DOPRAVA DOPRAVA HORE DOLAVA
DOLAVA DOLE DOLAVA DOLAVA HORE DOPRAVA DOPRAVA DOLE
DOPRAVA DOPRAVA HORE DOLAVA DOLAVA DOLAVA DOLE DOPRAVA
DOPRAVA DOPRAVA HORE DOLAVA DOLAVA DOLAVA DOLAVA
```

Meno: Jakub Šimko
AIS ID: 103146

2. 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

```
Pocet vytvorených uzlov: 2885595
Pocet spracovaných uzlov: 1802893
Počet ťahov riešenia: 55
['HORE', 'DOLAVA', 'DOLE', 'DOLAVA', 'DOLAVA', 'HORE', 'DOPRAVA', 'DOPRAVA',
Čas vykonávania algoritmu: 100.53125s
```

```
Otvoriť GUI s riešením? y/n
V prípade odpovede "n" sa na termináli vypíše naformátované riešenie.
n
HORE DOLAVA DOLE DOLAVA DOLAVA HORE DOPRAVA DOPRAVA
DOLE DOPRAVA HORE DOLAVA DOLE DOLAVA DOLAVA DOLAVA
HORE DOPRAVA DOPRAVA DOPRAVA DOLE DOLAVA DOLAVA DOLAVA
HORE DOPRAVA DOPRAVA DOLE DOPRAVA DOPRAVA HORE DOLAVA
DOLAVA DOLE DOLAVA DOLAVA HORE DOPRAVA DOPRAVA DOLE
DOPRAVA DOPRAVA HORE DOLAVA DOLAVA DOLAVA DOLE DOPRAVA
DOPRAVA DOPRAVA HORE DOLAVA DOLAVA DOLAVA DOLAVA
```

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.

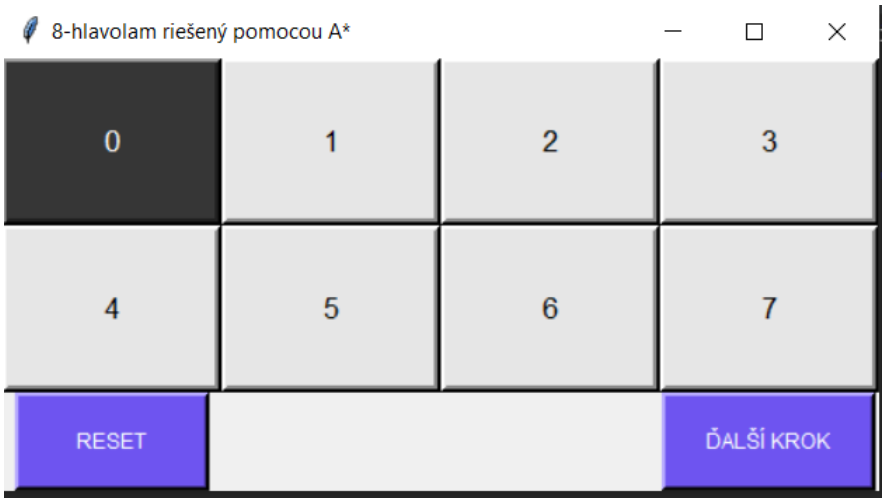
```
Pocet vytvorených uzlov: 1702806
Pocet spracovaných uzlov: 1070050
Počet ťahov riešenia: 55
['DOPRAVA', 'DOLE', 'DOPRAVA', 'HORE', 'DOPRAVA', 'DOPRAVA', 'DOLE', 'DOLAVA',
Čas vykonávania algoritmu: 75.5625s

Otvoriť GUI s riešením? y/n
V prípade odpovede "n" sa na termináli vypíše naformátované riešenie.
n
DOPRAVA DOLE DOPRAVA HORE DOPRAVA DOPRAVA DOLE DOLAVA
DOLAVA HORE DOLAVA DOLAVA DOLE DOPRAVA DOPRAVA HORE
DOPRAVA DOPRAVA DOLE DOLAVA DOLAVA HORE DOLAVA DOLAVA
DOLE DOPRAVA DOPRAVA HORE DOPRAVA DOPRAVA DOLE DOLAVA
DOLAVA HORE DOLAVA DOLAVA DOLE DOPRAVA DOPRAVA HORE
DOPRAVA DOPRAVA DOLE DOLAVA DOLAVA HORE DOLAVA DOLAVA
DOLE DOPRAVA DOPRAVA HORE DOPRAVA DOPRAVA DOLE
```

Meno: Jakub Šimko
AIS ID: 103146

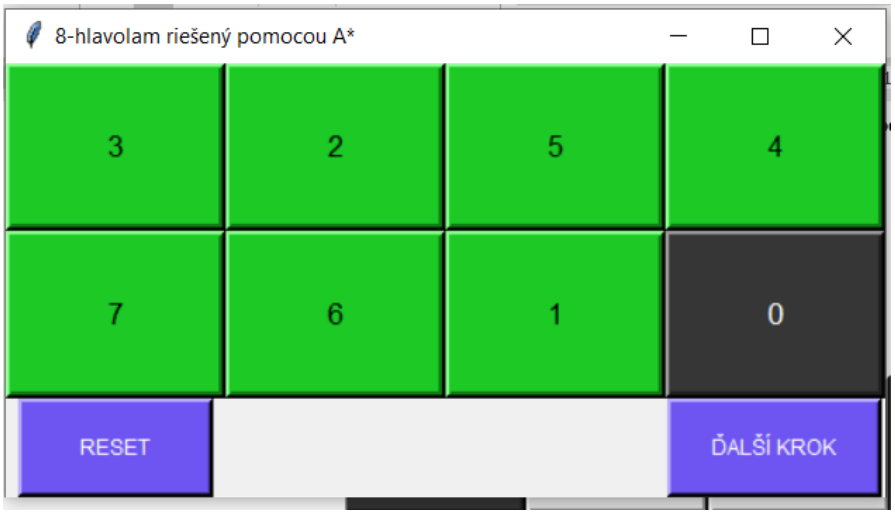
3. Hlavolam 4*2 **(Heuristika: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície)**

- Počet vytvorených uzlov 17 937
- Počet spracovaných uzlov 11 985
- Čas vykonávania 0.6875s

Počiatočný stav 0123 4567



Koncový stav 3254 7610



Riešenie:

```
Pocet vytvorených uzlov: 17937
Pocet spracovaných uzlov: 11985
Počet ťahov riešenia: 36
['HORE', 'DOLAVA', 'DOLAVA', 'DOLE', 'DOPRAVA',
Čas vykonávania algoritmu: 0.6875s
```

Meno: Jakub Šimko
AIS ID: 103146

4. 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

```
Pocet vytvorených uzlov: 29327
Pocet spracovaných uzlov: 19515
Počet ťahov riešenia: 36
['HORE', 'DOLAVA', 'DOLAVA', 'DOLAVA', 'DOLE', 'DOPRAVA', 'DOPRAVA',
Čas vykonávania algoritmu: 0.875s

Otvoriť GUI s riešením? y/n
V prípade odpovede "n" sa na termináli vypíše naformátované riešenie.
n
HORE DOLAVA DOLAVA DOLAVA DOLE DOPRAVA DOPRAVA DOPRAVA
HORE DOLAVA DOLAVA DOLAVA DOLE DOPRAVA DOPRAVA HORE
DOPRAVA DOLE DOLAVA HORE DOLAVA DOLAVA DOLE DOPRAVA
DOPRAVA HORE DOPRAVA DOLE DOLAVA HORE DOLAVA DOLE
DOPRAVA HORE DOLAVA DOLAVA
```

Vačšina príkladov 3x3 a menšie sa vykonajú zväčša okamžite.

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

```
Rozmery hlavolamu 2x3
[[4, 0, 5], [1, 3, 2]]
[[1, 4, 0], [2, 3, 5]]
Pocet vytvorených uzlov: 481
Pocet spracovaných uzlov: 360
Počet ťahov riešenia: 0
Nepodarilo sa nájsť riešenie
```

Meno: Jakub Šimko
AIS ID: 103146

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

```
path = a_star(heuristic2, start_node_state, end_node_state, m, n)
```

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

```
    # V pripade kedy program prejde všetky dostupné kombinácie uzlov tak problém nema riešenie
    # Zo začiatočneho stavu ktory nema riešenie sa počet spracovanych uzlov == (n*m)!/2
    if not found_path:
        print("Nepodarilo sa nájsť riešenie")

    print(path)  # dodatocny vypis riešenia - moze/nemusi byt vhodny pri vypisoch z testov / vstupu

    return path
```

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.

```
129  # argument heuristic obsahuje heuristiku ktoru ma a* vyuzivat
130  def a_star(heuristic, start_node_state, end_node_state, m, n):
131      path = []
132      open_check = {}
133
134      # vytvori začiatočny uzol
135      start_node = Node(start_node_state, None)
136      start_node.set_g_cost()
137      start_node.h_cost = heuristic(start_node_state, end_node_state, m, n)
138      start_node.set_f_cost()
139
140      # priority que z python knižnice
141      openSet = PriorityQueue()
142      openSet.put(start_node)
143      open_check[start_node.get_state_string()] = True
144
145      # počítadla
146      found_path = False
147      analysed_nodes_counter = 0
148      created_nodes_counter = 0
```

Meno: Jakub Šimko
AIS ID: 103146

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.

```
149     while not openSet.empty():
150         current_node = openSet.get()  # ziskanie elementu ho odstrani z q
151         open_check[current_node.get_state_string()] = False
152         analysed_nodes_counter += 1
153 
154         # rekonstrukcia cesty
155         if current_node.state == end_node_state:
156             found_path = True
157             path = reconstruct_path(current_node)
158             break
159 
160         current_node.neighbours, new_neighbours_count = create_neighbours(current_node, m, n)
161         created_nodes_counter += new_neighbours_count
162         for neighbour in current_node.neighbours:
```

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

```
75  def create_neighbours(current_node, m, n):
76      neighbours = []
77      state = current_node.state
78 
79      # zisti poziciu medzery
80      x = y = 0
81      for i in range(n):
82          for j in range(m):
83              if state[i][j] == 0:
84                  x = i
85                  y = j
86                  break
87 
88      new_states = [new_state_after_operator(state, current_node.operator, "DOLAVA", x, y, m, n),
89                    new_state_after_operator(state, current_node.operator, "DOPRAVA", x, y, m, n),
90                    new_state_after_operator(state, current_node.operator, "HORE", x, y, m, n),
91                    new_state_after_operator(state, current_node.operator, "DOLE", x, y, m, n)]
92 
93      i = -1
94      neighbours_count = 0
95      for new_state in new_states:  # cyklus vytvara susedov - nove uzly
96          i += 1
97          if new_state is None:
98              continue
99 
100         neighbour = Node(new_state, current_node)
101         neighbour.set_operator(i)
102         neighbours.append(neighbour)
103         neighbours_count += 1
104 
105     return neighbours, neighbours_count
```

Meno: Jakub Šimko
AIS ID: 103146

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

```
48  def new_state_after_operator(state, last_operator, operator, x, y, m, n):
49      if last_operator != "DOPRAVA" and operator == "DOLAVA" and y+1 < m:
50          new_state = [copy[:] for copy in state]
51          new_state[x][y] = state[x][y+1]
52          new_state[x][y+1] = 0
53
54      elif last_operator != "DOLAVA" and operator == "DOPRAVA" and y-1 >= 0:
55          new_state = [copy[:] for copy in state]
56          new_state[x][y] = state[x][y-1]
57          new_state[x][y-1] = 0
58
59      elif last_operator != "HORE" and operator == "DOLE" and x-1 >= 0:
60          new_state = [copy[:] for copy in state]
61          new_state[x][y] = state[x-1][y]
62          new_state[x-1][y] = 0
63
64      elif last_operator != "DOLE" and operator == "HORE" and x+1 < n:
65          new_state = [copy[:] for copy in state]
66          new_state[x][y] = state[x+1][y]
67          new_state[x+1][y] = 0
68
69      else:  # dany operator sa neda vykonať
70          return None
71
72      return new_state
```

Meno: Jakub Šimko
AIS ID: 103146

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:

a) https://en.wikipedia.org/wiki/A*_search_algorithm
b) https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2

```
160         for neighbour in current_node.neighbours:
161             # ak sa daný stav nenachádza v slovníku tak ešte nebol nikdy vložený do que (openSet) - novy susedia
162             if neighbour.get_state_string() not in open_check:
163                 neighbour.set_g_cost()
164                 neighbour.h_cost = heuristic(neighbour.state, end_node_state, m, n)
165                 neighbour.set_f_cost()
166 
167                 openSet.put(neighbour)
168                 open_check[neighbour.get_state_string()] = True
169                 continue
170 
171             # stav je v closedSet
172             if not open_check[neighbour.get_state_string()]:
173                 continue
174 
175             # stav sa nachádaza v que (openSet)
176             # program sa musi pozrieť či neexistuje efektívnejšia cesta do tohto uzla
177 
178             # If the heuristic is consistent,
179             # when a node is removed from openSet the path to it is guaranteed to be optimal
180             # so the test 'tentative_gScore < gScore[neighbor]' will always fail if the node is reached again.
181             # https://en.wikipedia.org/wiki/A*_search_algorithm
182             temp_g_cost = current_node.g_cost + 1
183             if temp_g_cost < neighbour.g_cost:  # efektivnejšia cesta
184                 neighbour.previous = current_node
185                 neighbour.g_cost = temp_g_cost
186                 neighbour.set_f_cost()
187                 update_que(openSet, neighbour)  # upravil sa f_cost je nutne aktualizovat que
```

Koniec funkcie:

```
190         print(f"Pocet vytvorených uzlov: {created_nodes_counter}")
191         print(f"Pocet spracovaných uzlov: {analysed_nodes_counter}")
192         print(f"Počet ťahov riešenia: {len(path)}")
193 
194         # V pripade kedy program prejde všetky dostupné kombinácie uzlov tak problém nema riešenie
195         # Zo začiatočneho stavu ktory nema riešenie sa počet spracovanych uzlov == (n*m)!/2
196         if not found_path:
197             print("Nepodarilo sa nájsť riešenie")
198 
199         return path
```

Meno: Jakub Šimko
AIS ID: 103146

5. Zhodnotenie riešenia

5.1. 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.

6. 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.

```
Priemerný čas vykonávania algoritmu:
Heuristika 1 Počet políčok, ktoré nie su na svojom mieste                    :   0.09895833333333333s
Heuristika 2 Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície     :   0.41666666666666664s
```

Meno: Jakub Šimko
AIS ID: 103146

```
Zadaj počet testov:
4
##############################
Rozmery hlavolamu 3x2
Začiatočný stav:
2 3
5 4
1 0
Koncový stav:
3 2
4 5
1 0
##############################
Program začína hľadať riešenie...
Heuristika 1: Počet políčok, ktoré nie su na svojom mieste
Pocet vytvorených uzlov: 380
Pocet spracovaných uzlov: 283
Počet ťahov riešenia: 20
Čas vykonávania algoritmu: 0.015625s

Heuristika 2: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
Pocet vytvorených uzlov: 261
Pocet spracovaných uzlov: 193
Počet ťahov riešenia: 20
Čas vykonávania algoritmu: 0.015625s
```

```
##############################
Rozmery hlavolamu 2x2
Začiatočný stav:
0 3
2 1
Koncový stav:
3 2
1 0
##############################
Program začína hľadať riešenie...
Heuristika 1: Počet políčok, ktoré nie su na svojom mieste
Pocet vytvorených uzlov: 13
Pocet spracovaných uzlov: 12
Počet ťahov riešenia: 0
Nepodarilo sa nájsť riešenie
Čas vykonávania algoritmu: 0.0s

Heuristika 2: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
Pocet vytvorených uzlov: 13
Pocet spracovaných uzlov: 12
Počet ťahov riešenia: 0
Nepodarilo sa nájsť riešenie
Čas vykonávania algoritmu: 0.0s
```

Meno: Jakub Šimko
AIS ID: 103146

```
##############################
Rozmery hlavolamu 3x3
Začiatočný stav:
6 7 1
5 3 0
8 4 2
Koncový stav:
1 5 6
4 3 7
8 0 2
##############################
Program začína hľadať riešenie...
Heuristika 1: Počet políčok, ktoré nie su na svojom mieste
Pocet vytvorených uzlov: 5014
Pocet spracovaných uzlov: 2906
Počet ťahov riešenia: 20
Čas vykonávania algoritmu: 0.140625s

Heuristika 2: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
Pocet vytvorených uzlov: 933
Pocet spracovaných uzlov: 547
Počet ťahov riešenia: 20
Čas vykonávania algoritmu: 0.03125s
```

```
##############################
Rozmery hlavolamu 2x4
Začiatočný stav:
5 0 6 1
3 2 7 4
Koncový stav:
6 0 7 1
3 5 4 2
##############################
Program začína hľadať riešenie...
Heuristika 1: Počet políčok, ktoré nie su na svojom mieste
Pocet vytvorených uzlov: 4639
Pocet spracovaných uzlov: 3022
Počet ťahov riešenia: 24
Čas vykonávania algoritmu: 0.140625s

Heuristika 2: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
Pocet vytvorených uzlov: 1905
Pocet spracovaných uzlov: 1253
Počet ťahov riešenia: 24
Čas vykonávania algoritmu: 0.078125s
```

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.

Meno: Jakub Šimko
AIS ID: 103146

6.1. Porovnanie vlastností použitých metód pre rôznu veľkosť 8-hlavolamu

Hlavolam 2x4:

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

```
Heuristika 1: Počet políčok, ktoré nie su na svojom mieste
Pocet vytvorených uzlov: 29327
Pocet spracovaných uzlov: 19515
Počet ťahov riešenia: 36
['HORE', 'DOLAVA', 'DOLAVA', 'DOLAVA', 'DOLE', 'DOPRAVA', 'DOPRAVA',
Čas vykonávania algoritmu: 0.75s

Otvoriť GUI s riešením? y/n
V prípade odpovede "n" sa na termináli vypíše naformátované riešenie.
n
HORE DOLAVA DOLAVA DOLAVA DOLE DOPRAVA DOPRAVA DOPRAVA
HORE DOLAVA DOLAVA DOLAVA DOLE DOPRAVA DOPRAVA HORE
DOPRAVA DOLE DOLAVA HORE DOLAVA DOLAVA DOLE DOPRAVA
DOPRAVA HORE DOPRAVA DOLE DOLAVA HORE DOLAVA DOLE
DOPRAVA HORE DOLAVA DOLAVA
```

```
Heuristika 2: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
Pocet vytvorených uzlov: 17937
Pocet spracovaných uzlov: 11985
Počet ťahov riešenia: 36
['HORE', 'DOLAVA', 'DOLAVA', 'DOLE', 'DOPRAVA', 'DOPRAVA', 'HORE', 'DOLAVA', 'DOLAVA', 'DOLAVA',
Čas vykonávania algoritmu: 0.640625s

Otvoriť GUI s riešením? y/n
V prípade odpovede "n" sa na termináli vypíše naformátované riešenie.
n
HORE DOLAVA DOLAVA DOLE DOPRAVA DOPRAVA HORE DOLAVA
DOLAVA DOLAVA DOLE DOPRAVA DOPRAVA DOPRAVA HORE DOLAVA
DOLAVA DOLAVA DOLE DOPRAVA DOPRAVA HORE DOPRAVA DOLE
DOLAVA HORE DOLAVA DOLE DOPRAVA HORE DOPRAVA DOLE
DOLAVA DOLAVA HORE DOLAVA
```

Meno: Jakub Šimko
AIS ID: 103146

Hlavolam 3x3:

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

```
Heuristika 1: Počet políčok, ktoré nie su na svojom mieste
Pocet vytvorených uzlov: 197021
Pocet spracovaných uzlov: 116802
Počet ťahov riešenia: 31
['DOLAVA', 'DOLAVA', 'HORE', 'HORE', 'DOPRAVA', 'DOPRAVA', 'DOLE', 'DOLE',
Čas vykonávania algoritmu: 6.265625s

Otvoriť GUI s riešením? y/n
V prípade odpovede "n" sa na termináli vypíše naformátované riešenie.
n
DOLAVA DOLAVA HORE HORE DOPRAVA DOPRAVA DOLE DOLE
DOLAVA HORE DOPRAVA DOLE DOLAVA DOLAVA HORE DOPRAVA
HORE DOLAVA DOLE DOPRAVA HORE DOPRAVA DOLE DOLE
DOLAVA DOLAVA HORE DOPRAVA DOPRAVA DOLE DOLAVA
```

```
Heuristika 2: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
Pocet vytvorených uzlov: 5562
Pocet spracovaných uzlov: 3377
Počet ťahov riešenia: 31
['DOLAVA', 'DOLAVA', 'HORE', 'HORE', 'DOPRAVA', 'DOPRAVA', 'DOLE', 'DOLE', 'DOLAVA',
Čas vykonávania algoritmu: 0.203125s

Otvoriť GUI s riešením? y/n
V prípade odpovede "n" sa na termináli vypíše naformátované riešenie.
n
DOLAVA DOLAVA HORE HORE DOPRAVA DOPRAVA DOLE DOLE
DOLAVA HORE HORE DOPRAVA DOLE DOLE DOLAVA DOLAVA
HORE DOPRAVA HORE DOPRAVA DOLE DOLE DOLAVA DOLAVA
HORE HORE DOPRAVA DOLE DOPRAVA DOLE DOLAVA
```

Meno: Jakub Šimko
AIS ID: 103146

Hlavolam 2x5:

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

```
Heuristika 1: Počet políčok, ktoré nie su na svojom mieste
Pocet vytvorených uzlov: 2885595
Pocet spracovaných uzlov: 1802893
Počet ťahov riešenia: 55
['HORE', 'DOLAVA', 'DOLE', 'DOLAVA', 'DOLAVA', 'HORE', 'DOPRAVA', 'DOPRAVA',
Čas vykonávania algoritmu: 103.796875s

Otvoriť GUI s riešením? y/n
V prípade odpovede "n" sa na termináli vypíše naformátované riešenie.
n
HORE DOLAVA DOLE DOLAVA DOLAVA HORE DOPRAVA DOPRAVA
DOLE DOPRAVA HORE DOLAVA DOLE DOLAVA DOLAVA DOLAVA
HORE DOPRAVA DOPRAVA DOPRAVA DOLE DOLAVA DOLAVA DOLAVA
HORE DOPRAVA DOPRAVA DOLE DOPRAVA DOPRAVA HORE DOLAVA
DOLAVA DOLE DOLAVA DOLAVA HORE DOPRAVA DOPRAVA DOLE
DOPRAVA DOPRAVA HORE DOLAVA DOLAVA DOLAVA DOLE DOPRAVA
DOPRAVA DOPRAVA HORE DOLAVA DOLAVA DOLAVA DOLAVA
```

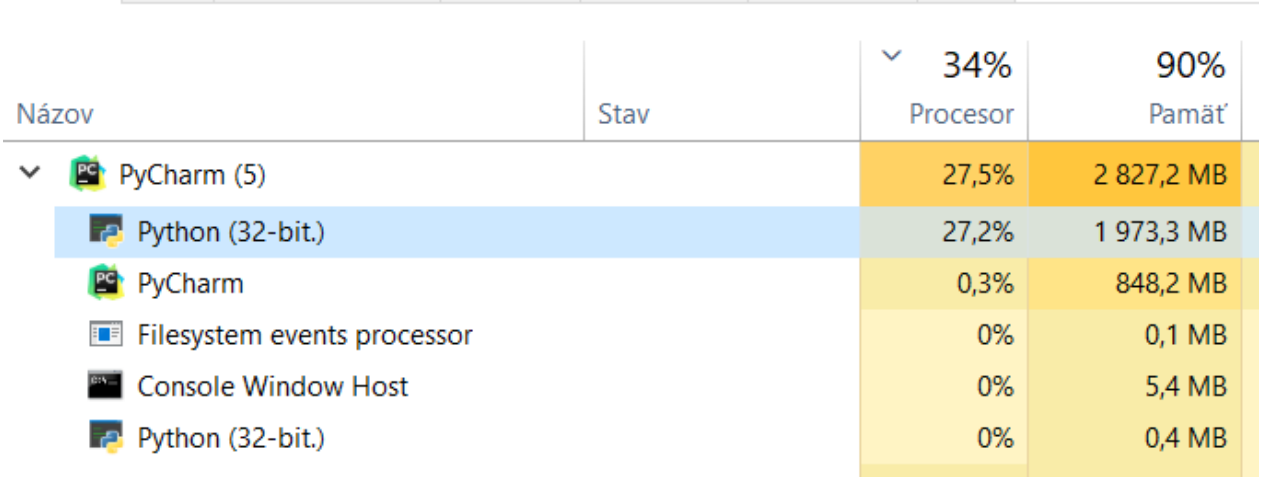
```
Heuristika 2: Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
Pocet vytvorených uzlov: 1674337
Pocet spracovaných uzlov: 1051455
Počet ťahov riešenia: 55
['HORE', 'DOLAVA', 'DOLE', 'DOLAVA', 'DOLAVA', 'HORE', 'DOPRAVA', 'DOPRAVA', 'DOLE',
Čas vykonávania algoritmu: 73.765625s

Otvoriť GUI s riešením? y/n
V prípade odpovede "n" sa na termináli vypíše naformátované riešenie.
n
HORE DOLAVA DOLE DOLAVA DOLAVA HORE DOPRAVA DOPRAVA
DOLE DOPRAVA HORE DOLAVA DOLE DOLAVA DOLAVA DOLAVA
HORE DOPRAVA DOPRAVA DOPRAVA DOLE DOLAVA DOLAVA DOLAVA
HORE DOPRAVA DOPRAVA DOLE DOPRAVA DOPRAVA HORE DOLAVA
DOLAVA DOLE DOLAVA DOLAVA HORE DOPRAVA DOPRAVA DOLE
DOPRAVA DOPRAVA HORE DOLAVA DOLAVA DOLAVA DOLE DOPRAVA
DOPRAVA DOPRAVA HORE DOLAVA DOLAVA DOLAVA DOLAVA
```

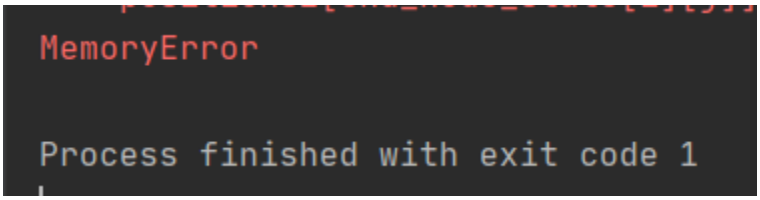
Meno: Jakub Šimko
AIS ID: 103146

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.