# Hledání cesty v mřížce

## Algoritmy umělé inteligence - VAI

Vypracoval: Datum:

Jiří Pakr 22.05.2022

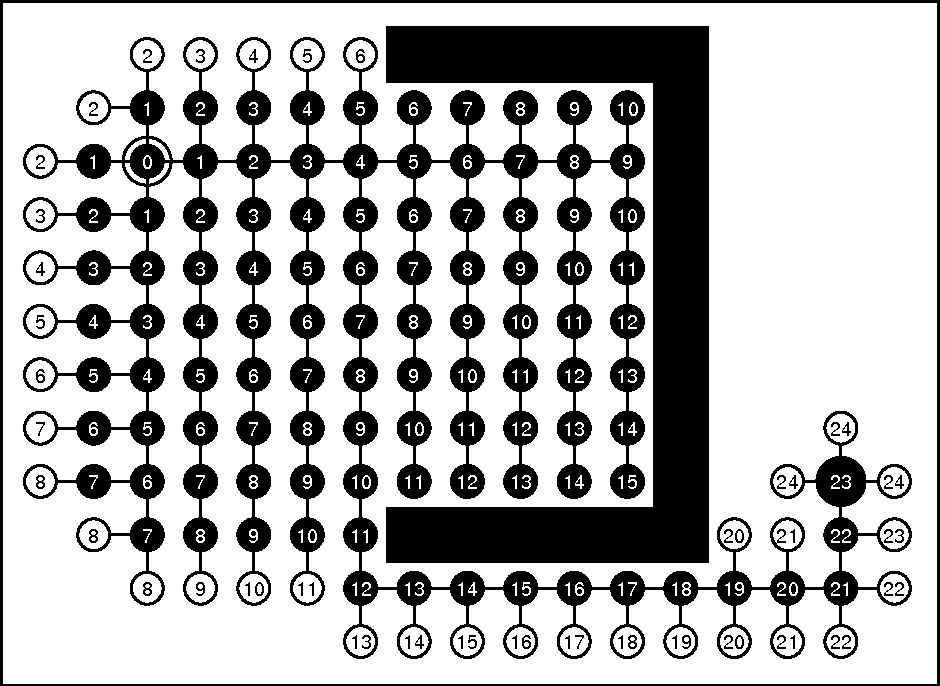
# Popis úlohy

Zvolená úloha tvoří mřížku 1000x1000 uzlů. Cílem je této úlohy je možnost najít cestu z libovolného startovního bodu do libovolného cílového bodu této mřížky. Ke zkomplikování nalezení optimální cesty a lepší možnosti otestování použitých algoritmů slouží možnost přidání libovolných překážek mezi zvoleným startovním a cílovým bodem.

Hlavním implementovaným algoritmem k hledání cesty byl zvolen algoritmus pro informované prohledávání A\*. Pro porovnání byli přidány i algoritmy pro neinformované prohledávání - prohledávání do hloubky (depth-first search) a prohledávání do šířky (breadth-first search).

# Teoreticky popis zvolených algoritmů

Algoritmus A\* je informovaný metodou prohledávání stavového prostoru. Jedná se o algoritmus best-first search s hodnotící funkcí f(i) = g(i) + h(i), kde i je expandovaný stav. Výsledné hodnoticí funkce se skládá se dvou částí. První funkce g(i) je cena přechodu do stavu i, počítaná dle g(i) = g(j) + c(j,i) je cena přechodu ze stavu j do stavu i. Druhá funkce h(i) je heuristická funkce, která odhaduje cestu s nejnižší cenu z bodu i do cílového bodu. Jako heuristická funkce byla zvolena Manhattanská vzdálenost.



*Obr.1 Příklad funkce algoritmu A\**

Algoritmy depth-first search (dále dfs) a breadth-first search (bfs) jsou neinformované metody prohledávání. Liší se od sebe z hlediska strategie podle, které jsou stavy expandovány. Dfs nejdříve expanduje uzel s největší hloubkou, zato bfs nejdříve expanduje uzel s nejmenší hloubkou.

Všechny zmíněné algoritmy využívají v nějaké formě seznamy CLOSED (seznam expandovaných stavů) a OPEN (seznam stavů, čekajících na expanzi). Tyto seznamy jsou v programu znázorněny zelenou barvou (seznam OPEN) a červenou (CLOSED).

# Popis vytvořeného sw

Pro vytvoření programu pro zvolenou úlohy byl použit programovací jazyk Python. K vytvoření vizualizace byla zvolena knihovna pygame a pro datové struktury využívané v prohledávacích algoritmech knihovna queue.

Pro uzly, které jsou reprezentovány mřížkou byla vytvořena třída node, která obsahuje potřebnou režii pro manipulaci a vstupuje do prohledávacích algoritmů. Poté pomocí pygame je vykreslována mřížka a zobrazovaný jednotlivé body tak aby to odpovídalo jednotlivým uzlům.

Algoritmus A\* a jeho seznam OPEN je realizován pomocí prioritní fronty (priority queue). Na začátku je do fronty přidán startovní uzel, jehož g\_score je inicializováno na 0 a h\_score je na manttahanský odhad vzdálenosti k cíli. Poté je z fronty vybrán uzel s nejmenším f\_score, který je přesunut do CLOSED a proběhne prozkoumání jeho sousedů, vypočítání jejich f\_score a následné zařazení do fronty. Tento proces se opakuje, dokud není nalezen cílový bod, pak se algoritmus ukončí.

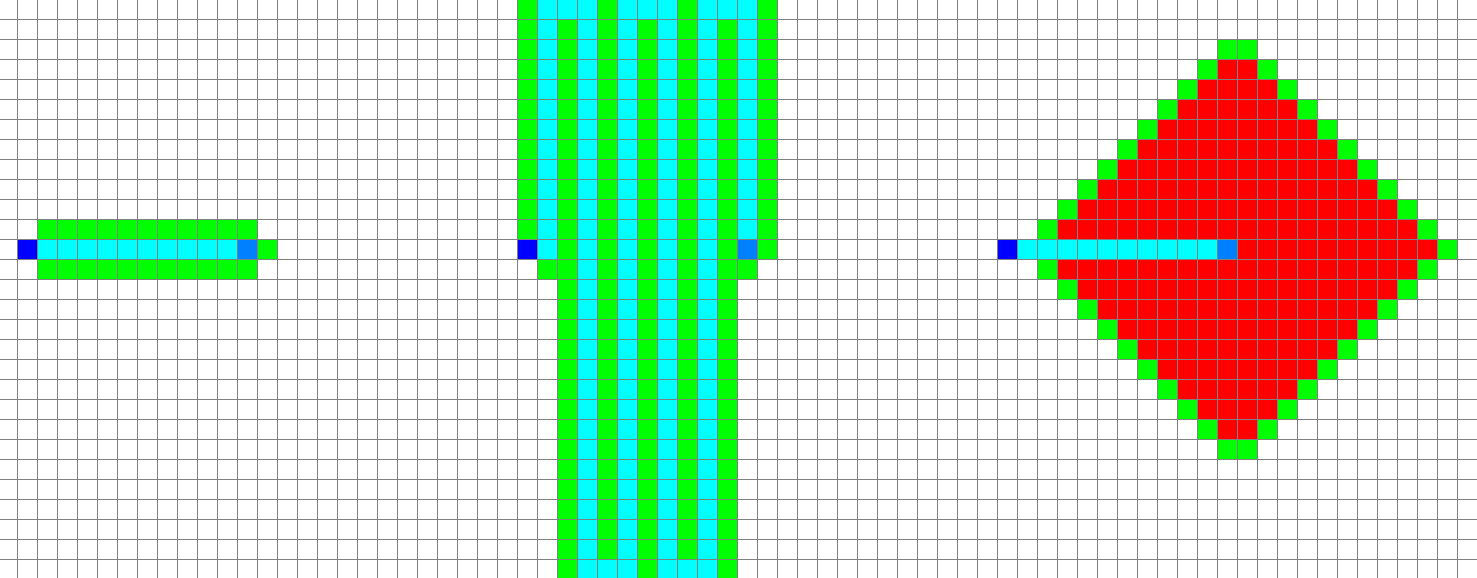
Algoritmy dfs a bfs jsou implementovány podobným stylem, jen dfs využívá zásobník (stack) a bfs frontu (queue) jako seznam OPEN. Do zásobníku (fronty) je inicializován startovní uzel. Algoritmus získá s příslušného seznamu OPEN uzel prozkoumá jeho sousedy a ty zapíše do seznamu OPEN a získaný uzel do CLOSED. Dále prochází seznam OPEN, dokud nenajde cíl, poté se algoritmus ukončí.

Dále jsou-li uzly již v seznamu OPEN algoritmy jej znova nepřidají (pokud teda nemá lepší f\_score). Ke kontrole, zda-li není soused již v seznamu slouží open\_set\_hash (popř stack\_hash nebo queue\_hash). Pro rekonstrukci cest slouží set came\_from, do kterého se ukládají procházené uzly.

K ovládání vizualizace jsou využity eventy v pygamu, pomocí kterých se spouští algoritmy a vybírají uzly jako start, end nebo překážka. Popis ovládání je v komentáři na začátku kódu programu.

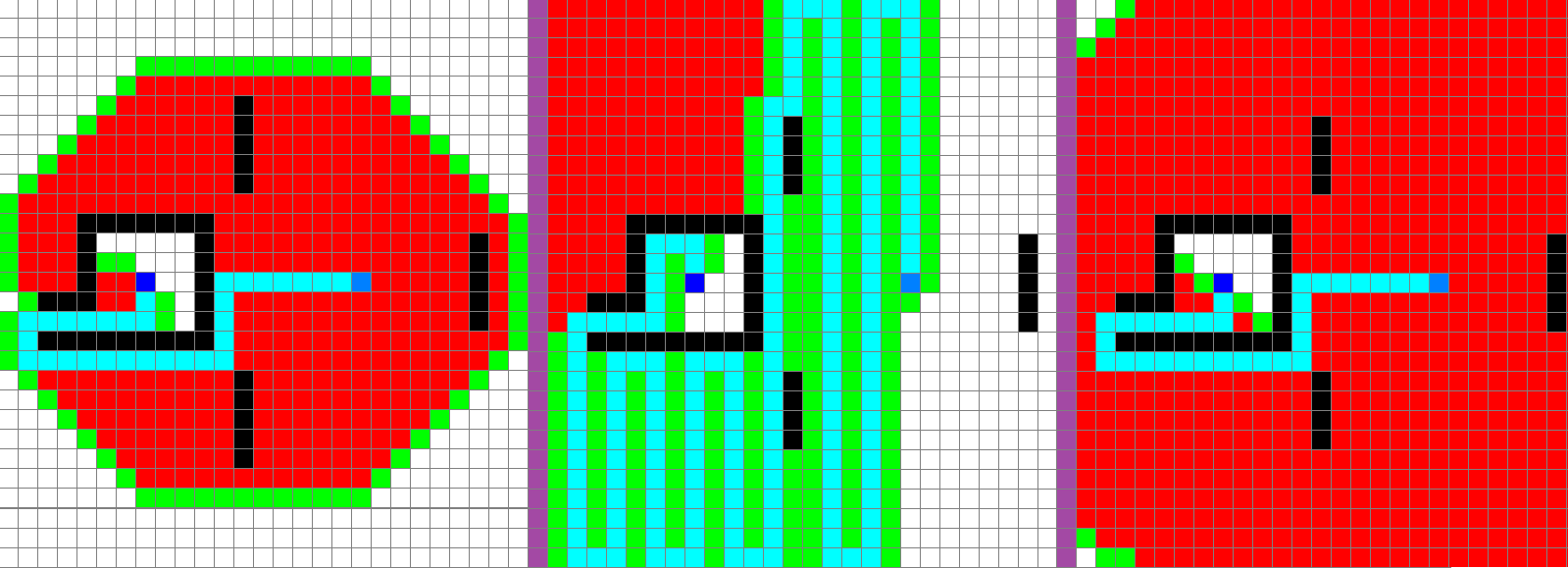
# Porovnání jednotlivých algoritmů

Pro porovnání jednotlivých algoritmů bylo zvoleno několik případů hledání cesty. Porovnáván bude jejich výkon, dle počtu potřebných kroků k nalezení cesty a délka výsledné cesty.



*Obr 2. Porovnání A\* (vlevo) dfs (uprostřed) bfs (vpravo)*

Na obrázku jde vidět, že A\* prošel nejméně polí a našel optimální cestu. Bfs také našel optimální cestu, jen prošel více polí a dfs našel jen cestu, ale optimální rozhodně ne. Počet kroků byl 34 pro A\*, 242 pro bfs a 526 pro dfs (s tím že je obrázek lehce ořezaný a upravený tedy pro dfs to vypadá že měl lepší výkon).



*Obr 3. Porovnání A\* (vlevo) dfs (uprostřed) bfs (vpravo) odděleny fialovým pruhem*

Na tomto obrázku lze vidět, cestu menším bludištěm. A\* našel optimální cestu za 437 kroků, bfs za 1686 a dfs našel jen nějakou cestu oklikou za 1097 kroků. Nutno dodat, že na obrázcích nejde vidět celé prozkoumané pole algoritmy bfs a dfs.

# Závěr

Algoritmus A\* určitě prošel testem jako nejlepší, což se ale dalo čekat, protože se jedná o informovanou metodu. Algoritmus bfs se ukázal, že také umí najít optimální cestu, jen mu to trvá poměrně delší hlavně v případě, když má v cestě mnoho překážek. Algoritmus dfs se v tomto případě ukázal jako nevhodný pro hledání cesty v mřížce, a to ještě kdyby byli pozice startu a konce prohozeny trvalo by mu to při této implementaci ještě mnohem déle.

Na provedeném srovnání lze vidět i proč A\* je dosti populární, lze ho poměrně lehce implementovat, najde optimální cestu, jeho výkon je dosti dobrý a nejspíš by byl ještě lepší při použití lepší heuristiky. Jeho největší nevýhodou jsou hlavně paměťové nároky, které ale v této implementaci nejsou problém. Kdyby to byl problém je nutné sáhnout po nějaké jeho upravené implementaci třeba po IDA\* nebo MA\*.

# Zdroje:

<https://www.geeksforgeeks.org/a-search-algorithm/>

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

<https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm>

<https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>

<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

[https://www.semanticscholar.org/paper/Extensions-and-applications-of-the-A\*-algorithm-Autere/19fa9e809e01655ef3340030e6f092e42d8c1c7b](https://www.semanticscholar.org/paper/Extensions-and-applications-of-the-A*-algorithm-Autere/19fa9e809e01655ef3340030e6f092e42d8c1c7b)