NNUI1

Semestrální práce 1

Michal Struna

1. Instalace a spuštění

Pro běh programu je potřeba překladač pro Python 3. Pokud není nainstalován, lze ho pro Windows nainstalovat přes https://www.python.org/downloads/release/python-380/ (pro 64bitový systém např. `Windows x86-64 web-based installer`). Pro linux postačí:

```
sudo apt install python3
```

V programu jsou využity některé externí moduly, které mohou nebo nemusí být dostupné v závislosti na typu instalace. V případě chybových hlášek je lze doinstalovat podobně jako např. modul numpy z terminálu:

```
python -m pip install numpy
```

Po instalaci je možné spustit program s uvedením cesty k souboru `solve.py` v tomto projektu. V případě, že cesta k Pythonu není v systémové proměnné `PATH`, je nutno uvést celou cestu. Je možné, že nejnovější verze Pythonu bude dostupná pod názvem `python3` namísto `python`:

```
python solve.py
```

V případě linuxu, kdy se překladač nachází na cestě `/usr/bin/python3` lze program spustit i prostým zavoláním názvu souboru. Tento způsob bude použit i v ukázkách dále:

```
solve.py
```

1.1. Parametry

Při spuštění programu lze dosadit i různé parametry:

Parametr	Zkratka	Význam
`--help`	`-h`	Zobrazí nápovědu s použitím všech parametrů.
`--stats`	`-s`	Kromě řešení zobrazí i statistiky (např. dobu běhu programu).
`--plain`	`-p`	Vypíše RAW bludiště namísto semi-grafické verze.
`--input`	`-i`	Nastaví zdrojový soubor (ve výchozím stavu `./area.txt`).

Tabulka 1 – Parametry programu.

Spuštění programu se zobrazením statistik a zdrojovým souborem `../data.txt`:

```
solve.py --stats -i ../data.txt
```

2. Vstup

Vstupem je textový soubor, který obsahuje 2D oblast popsanou pomocí několika druhů znaků oddělených mezerou nebo novým řádkem.

Znak	Význam
`0`	Prázdné pole
`1`	Obsazené pole (nelze vstupovat)
`2`	Počáteční pozice
`3`	Cílová pozice

Tabulka 2 – Struktura vstupní 2D oblasti.

Oblast musí být čtvercová nebo obdélníková – je nutné, aby všechny řádky i sloupce měly stejnou velikost. Součástí projektu je i soubor `area.txt`, který se bude číst ve výchozím stavu.

Při volbě počáteční pozice se předpokládá, že je agent ve výchozím stavu natočen na sever. U cílové pozice naopak natočení nehraje žádnou roli (stěžejní je pozice).

3. Struktura projektu

3.1. solve.py

Vstupním souborem projektu je soubor `solve.py`. Ten parsuje případné vstupní parametry programu a propojuje všechny ostatní soubory.

3.2. io_utils.py

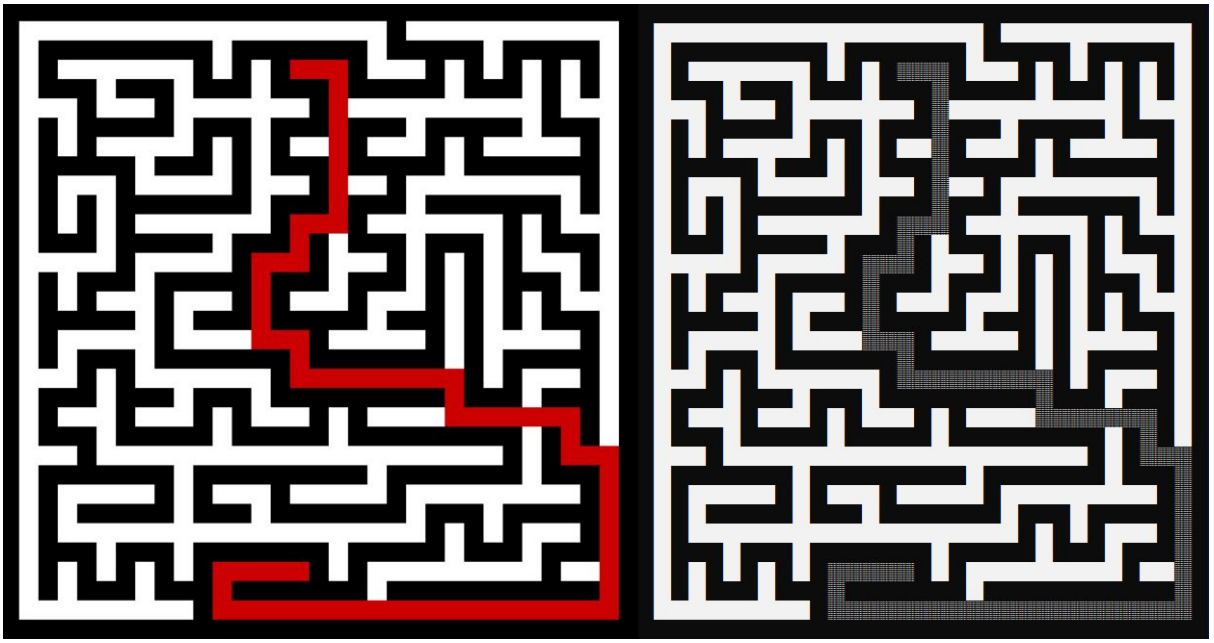
Modul obsahující pomocné třídy pro čtení a psaní.

3.2.1. Reader

Třída umožňující přečíst ze souboru 2D oblast a uložit ji do struktury `numpy.array`.

3.2.2. Formatter

Třída pro formátovaný barevný výpis stavů a cest do terminálu. Vykreslování barevných znaků v CMD ve Windows pomocí ANSI kódů není ve výchozím stavu příliš funkční, proto se v tomto OS používá odlišný způsob vyznačení cesty.



Obrázek 1 – Porovnání semigrafické verze bludiště na Linuxu (vlevo) a na Windows (vpravo).

3.3. solver.py

Modul pro samotné řešení problému bludiště.

3.3.1. PathFinder

Třída pro nalezení optimální cesty.

3.3.2. Node

Pomocná statická třída pro práci s vrcholy.

3.3.3. State

Pomocná statická třída pro práci se stavy.

3.3.4. Path

Pomocná statická třída pro práci s cestou a akcemi.

3.3.5. Array

Pomocná statická třída pro práci s poli.

3.4. area.txt, area2.txt

Předpřipravená bludiště, přičemž první z nich vychází ze zadání.

4. Řešení

Nalezení optimální cesty ve 2D oblasti je převedeno na problém nalezení optimální cesty v kladně ohodnoceném grafu. Problém lze tak řešit algoritmem A*.

4.1. Stav

Stavem je myšleno jednoznačné popsání všech skutečností vzhledem k řešené 2D oblasti. V programu je stav reprezentován strukturou tuple a obsahuje:

Index	Význam	Hodnoty
0	Horizontální souřadnice agenta.	Od `0` do `area_size_y - 1`
1	Vertikální souřadnice agenta.	Od `0` do `area_size_x - 1`
2	Natočení	`0` = nahoru, `1` = doprava, `2` = dolů, `3` = doleva

Tabulka 3 – Struktura stavu.

Příkladem stavu, kdy je agent ve 23. sloupci a 15. řádku, přičemž je natočen doleva, je:

```
state = (14, 22, 3)
```

4.2. Akce

Akce reprezentuje změnu stavu, např. pootočení hráče. Lze ji uplatnit pouze tehdy, pokud výsledkem bude opět stav, v němž se bude agent nacházet na povoleném poli. Akce je stejně jako stav reprezentována strukturou tuple, kde jednotlivé prvky jsou:

Index	Význam	Hodnoty
0	Otočení	`-1 =` doleva, `0` = nic, `1` = doprava, `2` = dozadu
1	Pohyb	`0` = nic, `1` = dopředu

Tabulka 4 – Struktura akce.

Akce pak mohou nabývat hodnot:

Akce	Cena	Význam
`(-1, 0)`	1	Otočit se doprava.
`(1, 0)`	1	Otočit se doleva.
`(2, 0)`	2	Otočit se dozadu.
`(0, 1)`	3	Jít o jedno pole vpřed.

Tabulka 5 – Seznam akcí.

4.3. Vrchol

Vrchol je tuple uchovávající stav a informace, které k tomuto stavu vedly (cena, předek, akce, …). Jednotlivé položky jsou:

Název	Datový typ	Význam
id	`string`	Unikátní identifikátor vrcholu.
state	`numpy.array`	Stav.
parent	`Node`	Předcházející vrchol.
action	`tuple`	Akce, která vedla k dosažení tohoto vrcholu.
path_cost	`int`	Cena cesty od kořene k aktuálnímu vrcholu.
path_eval	`int`	Výsledek heuristické funkce.

Tabulka 6 – Struktura vrcholu.

4.4. Použité datové struktury

4.4.1. Tuple

Pro struktury, které se při běhu programu vytvářejí ve větším množství (stavy a vrcholy) byl zvolen tuple kvůli nejlepší výkonnosti ze všech testovaných struktur.

Struktura	Kód (opakováno v cyklu 10^7krát)	Doba vykonávání
Tuple	`state = (1, 2, 3)`	0,4 s
LIst	`state = [1, 2, 3]`	0,65 s
Dictionary	`state = { 'a': 1, 'b': 2, 'c': 3 }`	1,06 s
Class	`state = State(1, 2, 3)`	2,9 s
Array	`state = numpy.array([1, 2, 3])`	7,1 s

Tabulka 7 – Struktura vrcholu.

4.4.2. Set

Pro seznam všech navštívených vrcholů je nutné zvolit takovou strukturu, která umožňuje vkládat a vyhledávat prvky s co nejmenší časovou složitostí, ideálně `O(1)`. Nabízí se proto použít `dictionary` nebo `set`. Protože není nutné ukládat celé vrcholy, ale pouze jejich ID, pro potřeby programu postačil `set`.

```
explored = set()
explored.add(7)
7 in explored # True
```

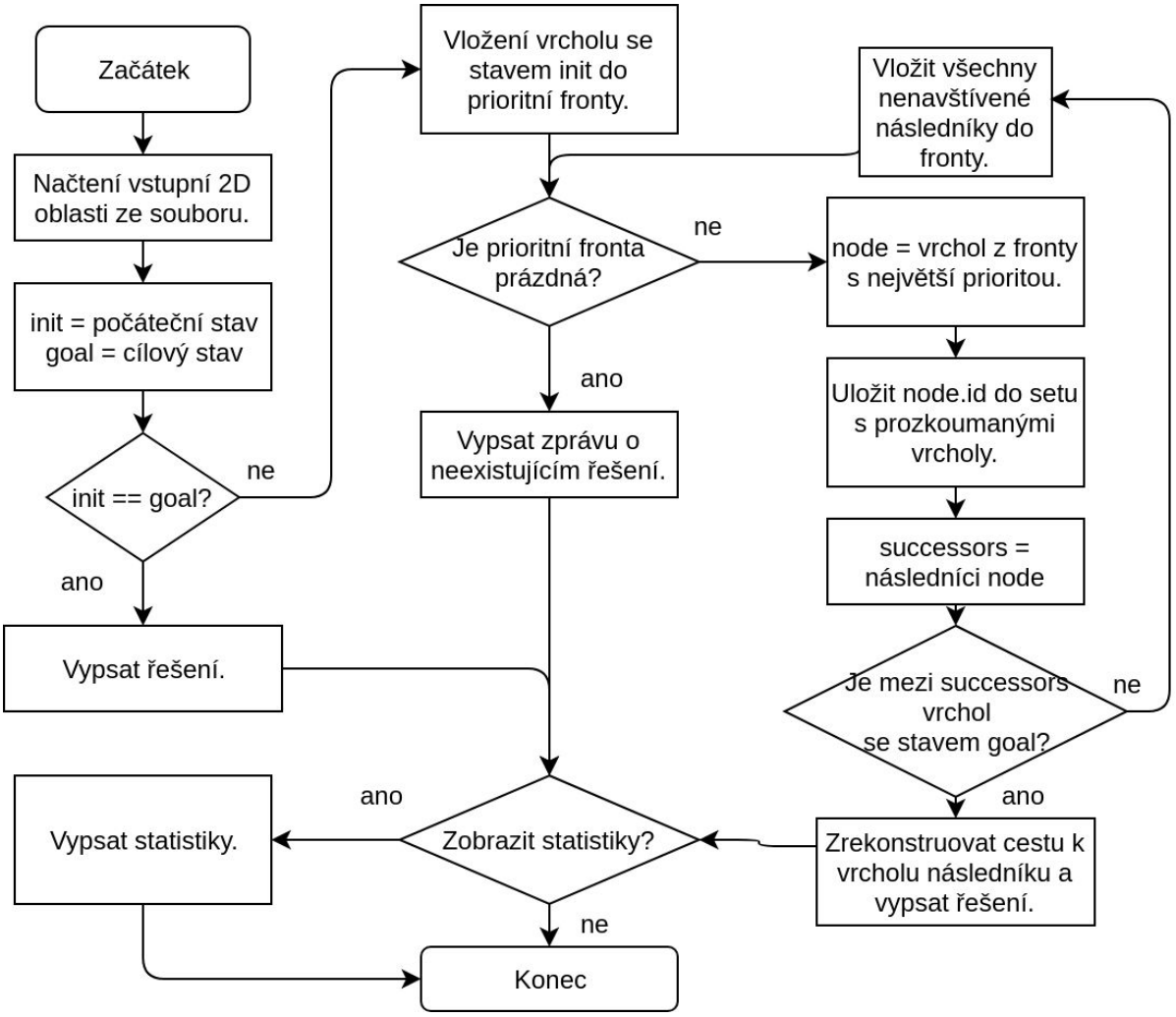
4.4.3. Priority queue

Je třeba si pamatovat ty vrcholy, které mají být prozkoumány, avšak dosud ještě nebyly. Pořadí vrcholu se určuje podle priority (součet vzdálenosti od kořene a výsledku heuristické funkce, menší hodnota znamená vyšší prioritu).

```
fringe = queue.PriorityQueue()
fringe.put((7, "první"))
fringe.put((6, "druhý"))
fringe.put((8, "třetí"))
fringe.get() # (6, "druhý")
```

Nevýhodou výchozí implementace prioritní fronty v Pythonu je nemožnost zjišťovat přítomnost prvku. Ve `fringe` tak mohou být uloženy duplicitní vrcholy.

4.5. Algoritmus



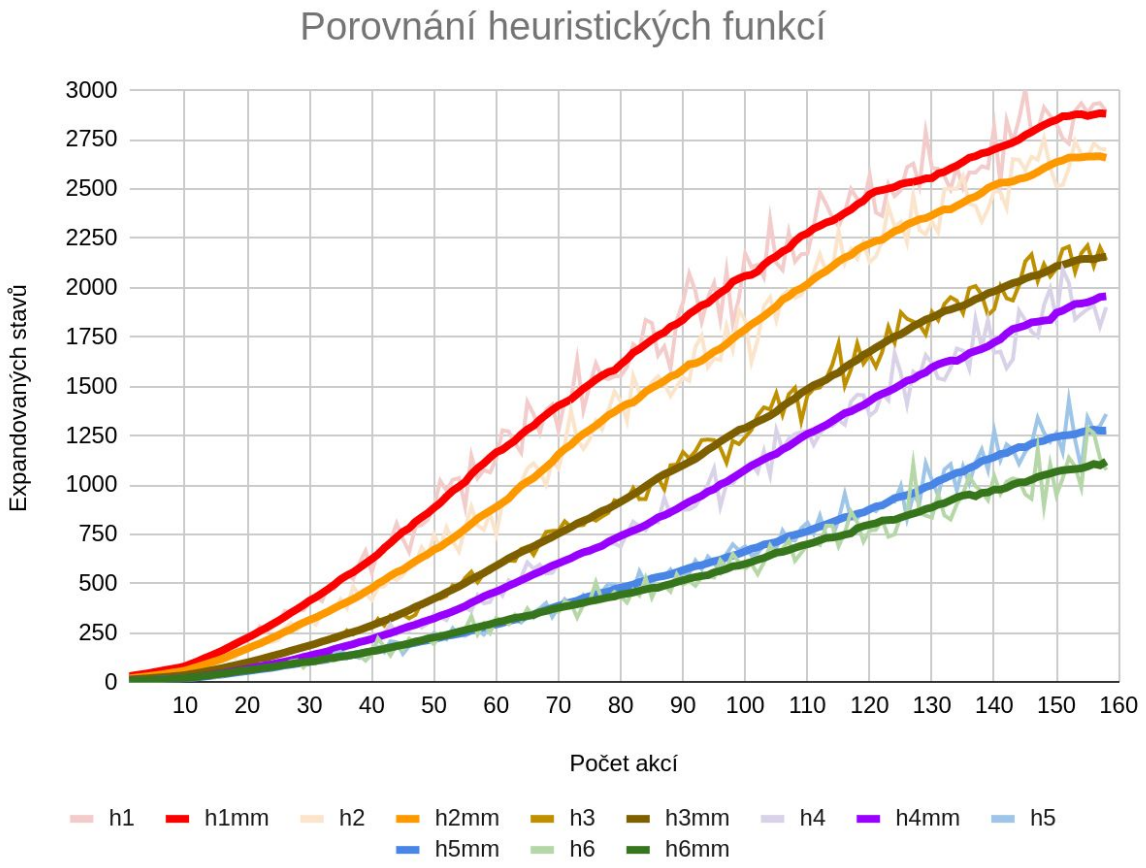
Obrázek 2 – Zjednodušený diagram algoritmu programu.

4.5.1. Heuristická funkce

Hodnota	**N**
$h_1 = 0$	2047
$h_2 = d$	1766
$h_3 = c \times d$	1280
$h_4 = 4 \times d$	1057
$h_5 = 12 \times d$	652
$h_6 = d^2$	592

Tabulka 8 – Porovnání heuristických funkcí. d je Manhattanská vzdálenost aktuálního vrcholu a cíle. c je cena akce (hrany), která k vrcholu vedla. N je počet expandovaných stavů při cestě o délce 100.

Pro volbu vhodné heuristické funkce bylo vygenerováno náhodné bludiště o velikosti 40x40 polí a na něm pro každou heuristickou funkci z tabulky výše 3 000 náhodných zadání pro hledání optimální cesty (celkem tedy 18 000 úloh).

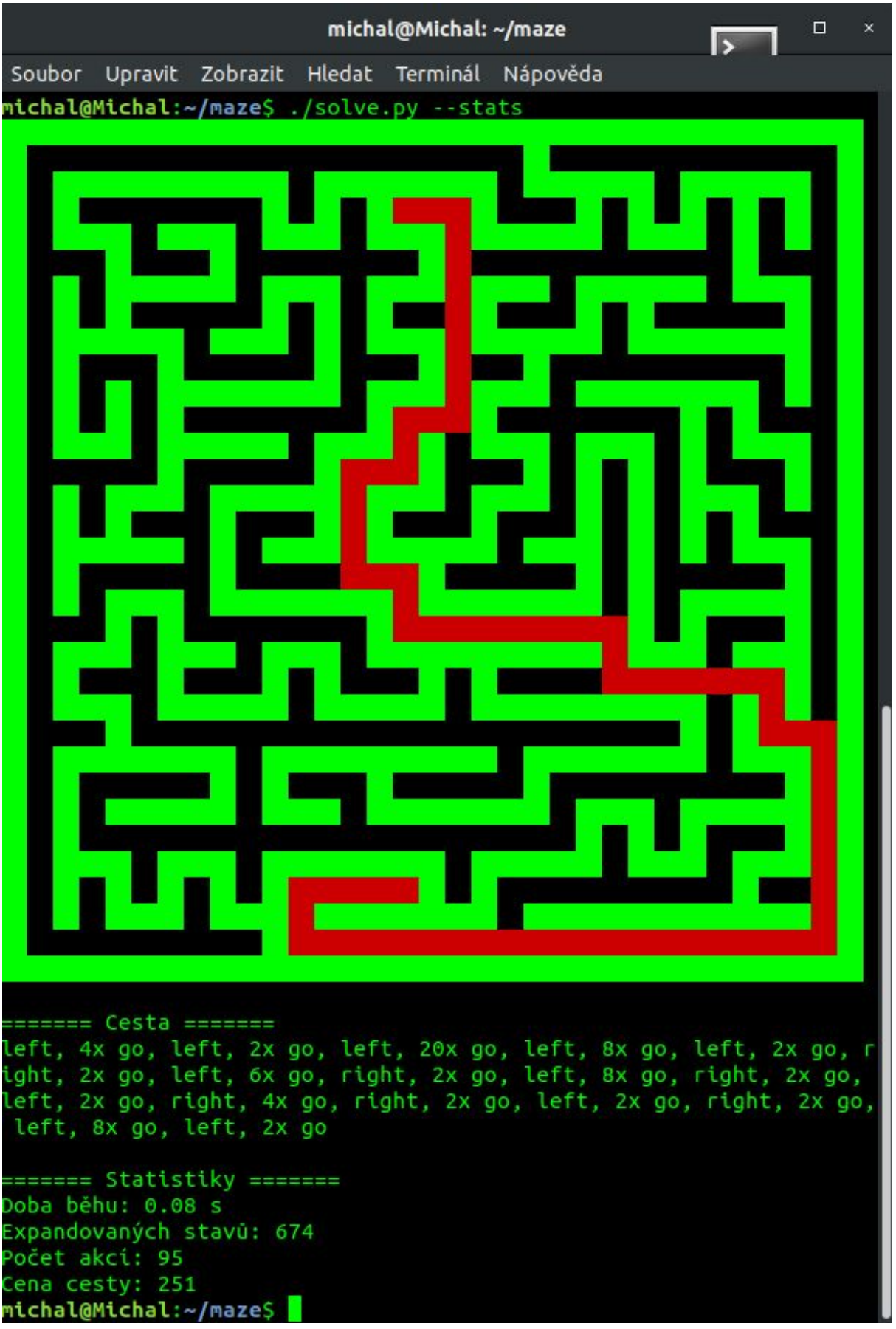


Obrázek 3 – Porovnání heuristických funkcí. U každé funkce jsou zaznamenána konkrétní pozorování a klouzavé průměry (mm - moving mean) – 10 předchozích a 10 následujících pozorování.

Z grafu je patrné, že Manhattanská vzdálenost od cíle má mnohem větší váhu, než cena cesty od kořene. Proto je u funkce určující prioritu vrcholu (f(v) = g(v) + h(v)) nevhodné pouhé sečtení těchto dvou hodnot. Manhattanskou vzdálenost je nutné vynásobit nějakým koeficientem. Ze všech testovaných heuristických funkcí nejlépe vychází druhá mocnina Manhattanské vzdálenosti, a proto je v programu použita h_6.

5. Výsledek

V rámci práce byl vytvořen konzolový program napsaný v jazyce Python, který přečte zadání ze souboru a do terminálu vypíše jeho řešení (pokud existuje). Výstupem může být např.:



Obrázek 4 – Výstup programu.