01SSI – Miniprojekt číslo 2 2022/2023

Posluchači: **Eliška Pečenková, Soňa Molnárová, Aleksej Gaj**
Zadání: **Vliv polohy sloupů na rychlost evakuace místnosti ve formulaci floor field modelu**

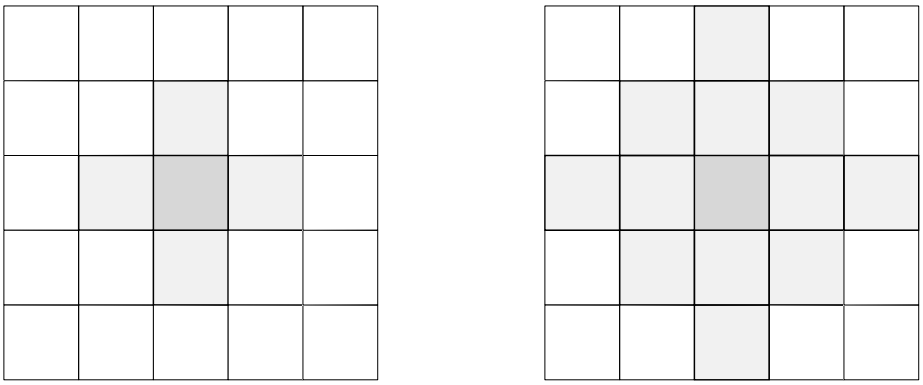
Odevzdáno: Získané body: Finální: ANO/NE

1 Teoretický úvod

V našem projektu jsme se pokusili o simulaci evakuace místnosti s jedním východem. V matematickém modelování se podobné simulace povádí na základě dvourozměrné mřížky buněk, z kterých každá může být obsazena maximálně jedním chodcem. Jedná se o tzv. celulární model, viz [3]. Pro evakuaci chodců jsou však kromě jednotlivých buněk a jejich obsazeností důležité i sousedící buňky. Aby se chodec mohl dostat blíže k východu, potřebuje mít k dispozici volnou buňku, do které se může posunout. Začneme proto popisem okolí buňky.

1.1 Von Neumannovo okolí

Prvním příkladem okolí je tzv. *von Neumannovo okolí*, které je definovávno jako daná buňka spolu se svými ortogonálně sousedícími buňkami. Když si tedy buňku uvnitř mřížky představíme jako čtverec, jeho von Neumannovo okolí bude tvořeno čtyřmi čtverci, které se ho dotýkaji celou jednou stranou. Graficky je tento typ okolí znázorněn na obrázku 1. Tmavě šedě je zde zobrazen chodec a světle šedou je zase zvýrazněno jeho okolí. Rozměr okolí vyjádřuje dosah intekarce. To znamená, že chodec je ovlivněn nejen chodci v jeho bezprostředné vzdálenosti (tj. v naší interpretaci čtyřmi nejbližšími čtverci), ale taky chodci umístěnými o buňku dále.



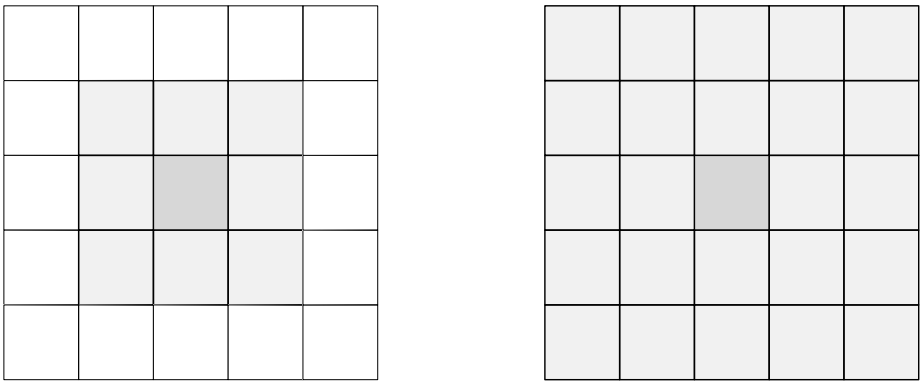
(a) Rozměr 1. (b) Rozměr 2.

Obrázek 1: Von Neumannovo okolí.

V tomto typu okolí je však chodci umožněn jen pohyb vpravo, vlevo, dopředu a dozadu. V reálném světě pohyb ale není diskretizován do čtverců, proto by se nám hodilo nějak postihnout i diagonální směry, kudy je ve skutečnosti chodci pohyb rovněž umožněn. K tomu využijeme následující okolí.

1.2 Moorovo okolí

V geometrii zavedené výše se jedná o buňku obklopenou všemi osmi sousedícími buňkami, jsou tu tak zahrnuty i čtyři diagonální směry, ktoré jsme u von Neumannova okolí neuvažovali. *Moorovo okolí* s dosahem interakce rovné jedné, viz [7], je tedy definováno jako množina buněk, kterých obě souřadnice se liší od souřadnic vybrané buňky maximálně o jedna. Znázorněno je na obrázku 2 níže.



(a) Rozměr 1. (b) Rozměr 2.

Obrázek 2: Moorovo okolí.

Tento popis okolí je bližší skutečnému chování chodců v reálné mistnosti, a proto pro implementaci naší úlohy použijeme toto okolí. Interakce se vzdálenějšími chodci sme zanedbali a rozsah byl tak volený roven jedné.

1.3 Floor Field Model

Pohyb chodce v místnosti je obecně ovlivněn ostatními chodci, polohou východu a infrastrukturou. V naší práci jsme infrastrukturu, kterou obvykle představují zdi, navíc obohatili o různý počet různě rozmístněných sloupů a zkoumali jsme, jak jejich poloha ovlivní rychlost evakuce místnosi. Předtím než se ale hlouběji ponoříme do samotných simulací a jejich výsledků bude vhodné model pohybu chodců popsat o něco víc důkladně.

Jak již bylo řečeno v předešlé sekci, ve *floor field modelu* (FFM) zavedeném v [2], budeme u každého choddce uvažovat Moorovo okolí. To je důležité poznamenat, neboť každý chodec se může ze své momentální obsazené buňky, ozn. x, pohnout jen do další (prázdné) buňky ze svého okolí, ozn. N_x (z anglického neighbourhood). Pokud označíme mřížku reprezentující místnost jako $\mathbb{L}$ (z anglického lattice), můžeme předchozí větu zapsat matematicky formou

$$y \in N_x \cap \mathbb{L}, \tag{1}$$

kde y značí nový stav (polohu) chodce.

Další podmínkou nutnou pro to, aby došlo k pohybu, je, že buňka některá z okolitých buňek musí být prázdná, jinak zůstává chodec stát na svém místě. Tento děj lze opět formulovat matematicky. Množinu přípustných dějů tak budeme značiť M a vyjádříme ji jako

$$M = \{x = y \vee (x \neq y \wedge y \text{ prázdná})\}. \tag{2}$$

Při zavedeném značení můžeme pak pravděpodobnost přechodu do buňky y při dané momentální buňce x zapsat jako

$$P(x \to y) \approx \exp\left(-\sum_F k_F F(y)\right). \tag{3}$$

Zde $k_F \in (0, +\inf)$ představuje parametr modelu a F je označení pole. To může být statické a dynamické, pro naše potřeby však vystačí pole statické, které je jednodušší, a tak se polem dynamickým nebudeme v práci zabývat, jelikož jsme jej při simulacích nijak nevyužili. Označme tedy statické pole symbolem S. Klademe jej rovno minimálnímu počtu kroků potřebných k dosáhnutí cíle - v našem případě východu, tj.

$$S(y) = \mathrm{dist}(y, E), \tag{4}$$

kde E je označení východu.

Zužitkováním znalostí (1), (2) a (4) a následným dosazením do vztahu (3) tak odstáváme tvar pravděpodobnosti přechodu z buňky x do buňky y jako

$$P(x \to y) = \frac{\exp(-k_S \ \mathrm{dist}(y, E))\mathbb{I}_M}{\sum\limits_{y \in N_x \cap \mathbb{L}} \exp(-k_S \ \mathrm{dist}(y, E))\mathbb{I}_M}.$$

FFM byl rozsáhle studován a rozšířen v [4]. Lze tady najít třeba efekt setrvačnosti nebo efekt šířky a počtu východů na rychlost evakuace chodců. Demonstrován je zde také efekt překážek (neboli obstrukcí), na které jsme se podrobněji zaměřili v naší práci. Důkladněji se na tento efekt podíváme v sekci 3 a následně budou naše závěry porovnány se závěry obdrženými z citované práce.

2 Popis použitých algoritmů

V kapitole 1 jsme představili Floor field model po teoretické stránce, v této kapitole se podíváme na implementaci tohoto modelu. V celé simulaci operujeme se třemi vrstami matic, o stejných rozměrech (kterými jsou výška místnosti P a šířka místnosti Q):

- zobrazení poloh agentů v systému (matice nul a jedniček, kde jednička znamená, že v dané buňce se nachází agent, nula naopak),
- zobrazení poloh buněk, kam agenti nesmí vstoupit (matice nul a jedniček, kde jednička vyjadřuje, že v dané buňce je stěna nebo sloup, či jiná obstrukce, nula vyjadřuje buňku, kde agentům je umožněn pohyb),
- gradientní pole (buňky, na které nelze vstoupit jsou reprezentovány hodnotou NaN, ostatní buňky obsahují číselné hodnoty vyjadřující minimální počet kroků, nezbytný pro dojití z dané buňky k východu).

Poznamenejme, že díky hodnotám NaN jsme takto zároveň do gradientního pole zahrnuli právě i polohy stěn a sloupů, takže nám pro popis stačí pouze 2 vrstvy matic, nýbrž i tak v práci pro přehlednost definujeme vrstvy 3.

Veškerá implementace simulace pohybu chodců byla provedena v jazyce MATLAB (verze 2022a) [6].

2.1 Popis spouštěcího skriptu

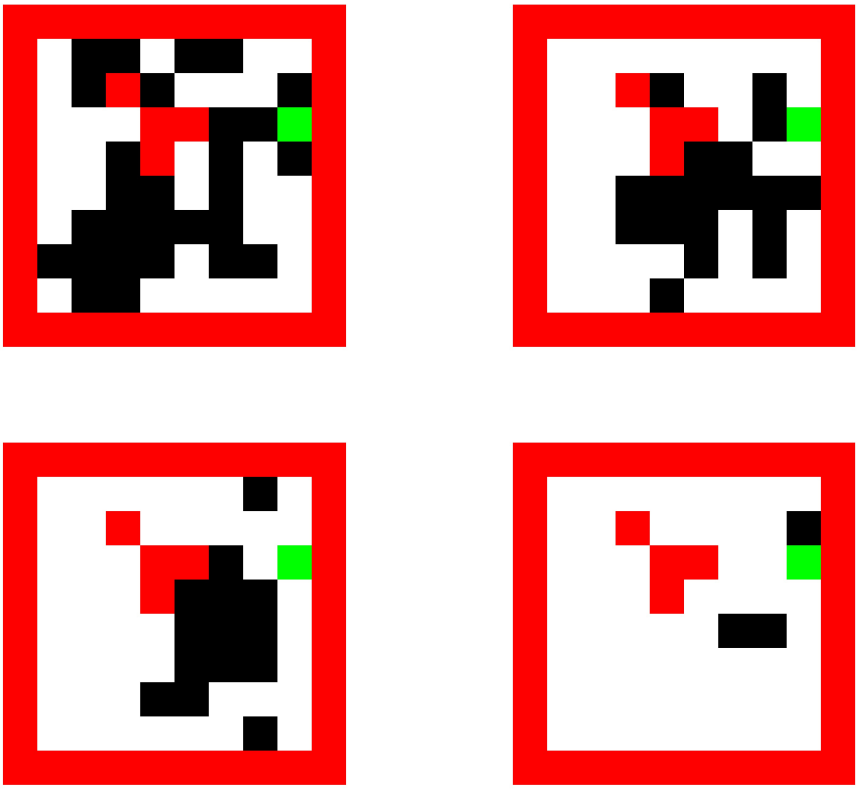
Myšlenka celé simulace je založena na tom, že máme k dispozici matici poloh agentů v systémů, tuto vrstvu jsme definovali v sekci 2. V každém kole vybereme náhodně jednu buňku (generujeme náhodně indexy pole pomocí generátoru náhodných integerů `randi`, který již je v MATLABu implementován) a aktualizujeme okolí této buňky, pomocí pravidel, které jsme zvolili jako vhodná a logická pro pohyb chodců. Tato pravidla budou podrobně okomentována

v podkapitole 2.2, kde popíšeme funkci `DECISION`, jejíž zdrojový kód je v sekci 4. Okolím buňky je myšleno Moorovo okolí s rozsahem interakce rovné jedné.

Bez újmy na obecnosti můžeme předpokládat, že východ se bude nacházet vždy v pravé části mřížky, proto můžeme pro urychlení simulace využít následujícího triku. Vždy nalezneme první neprázdný sloupec mřížky, tj. sloupec, ve kterém se nachází alespoň 1 agent a budeme generovat indexy mřížky počínaje tímto sloupcem směrem doprava. Tímto se tedy ušetří několik iterací pro řídká pole.

Simulaci provádíme tak dlouho, dokud všechny buňky v mřížce nejsou prázdné, tedy všichni chodci opustili místnost. Zároveň pro pozdější použití v každém experimentu zaznamenáme počet iterací, které byl potřeba pro vyprázdnění místnosti.

Vykreslení experimentu provádíme "po pixelech". Černá pole znamenají přítomnost chodce v dané buňce, červená pole označují buňky, kam chodec nesmí vstoupit. Mohou tedy představovat např. stěny místnosti nebo sloupy. Zelenou barvou je vyznačena pozice východu. Poznamenejme, že pozice východu se může nacházet na kraji pole, což by představovalo situaci východu chodců z místnosti, nýbrž i uprostřed. Druhá varianta, kdy východ se nachází uprostřed pole, by mohla odpovídat např. evakuaci nástupiště metra, kde východ mohou být např. eskalátory uprostřed nástupiště. Pro vykreslení jsme použili funkci `cat`, ve které skládáme 3 složky RGB spektra. Pár snímků simulace je zobrazeno na Obr. 7.



Obrázek 7: Simulace pohybu chodců směrem k východu. Černou barvou jsou zobrazeni agenti, červená barva znázorňuje místa, kam agent nesmí vstoupit, zelenou barvou je označen východ.

2.2 Popis rozhodovací funkce

V této sekci popíšeme pravidla pohybu chodců po mřížce směrem k východu. Pohyb chodců je určen funkcí `DECISION`, viz sekce 4, kterou v této části popíšeme. Vstupními parametry

jsou:

- indexy buňky, jejíž okolí budeme v dané iteraci aktualizovat (dvojice souřadnic),
- matice poloh chodců v předchozí iteraci (jednička znamená přítomnost chodce v dané buňce, nula znamená volné místo),
- matice potenciálů (pro zadané rozměry ji získáme jako výstup djiskra algoritmu, podrobněji v podsekci 2.3),
- matice poloh stěn a sloupů (jednička znamená stěna, nula reprezentuje místo možné k pohybu),
- poloha východu (dvojice souřadnic).

Výstupem je nová matice poloh chodců, kterou získáme aktualizací Moorova okolí vybrané buňky.

V každé iteraci simulace je náhodně vylosována buňka, jejíž okolí budeme aktualizovat. Rozlišíme 2 případy.

2.2.1 Vylosování buňky obsazené člověkem

V první řadě vždy nalezneme Mooreovo okolí buňky, kterou chceme aktualizovat. Poté se podíváme, zda je tato buňka obsazená člověkem, nebo je prázdná. V této části popíšeme pravidla pro vylosování buňky obsazené člověkem.

Nalezneme vždy hodnotu nejmenšího potenciálu v okolí buňky, kterou chceme aktualizovat a určíme počet míst s tímto nejmenším potenciálem. Dále se podíváme, kolik z těchto míst s nejnižším potenciálem je volných, tedy na kolika z těchto míst se nenachází žádný člověk.

Označme m počet volných míst s nejnižším potenciálem v okolí buňky, kterou chceme aktualizovat. Je-li $m > 0$, pak náhodně (s uniformním rozdělením) vybereme jedno z m míst a posuneme chodce z buňky, kterou chceme aktualizovat na toto volné místo. Je-li $m = 0$, tedy žádné místo s nejnižším potenciálem není volné, pak nalezneme v okolí buňky, kterou chceme aktualizovat místa s druhým nejnižším potenciálem, opět určíme jejich počet a postupujeme zcela analogicky. Pokud ani zde nejsou volná místa, pak v této iteraci člověk zůstává stát a losujeme indexy nové buňky, jejíž okolí budeme aktualizovat.

Poznamenejme zde, že strukturou gradientního pole, kdy zakázené buňky jsou reprezentovány hodnotou NaN, je dobře zajištěn problém, že chodci nikdy nevstoupí na místo, kde se nachází sloup, nebo stěna.

2.2.2 Vylosování prázdné buňky

Pokud vylosujeme buňku, která není obsazená člověkem, nejprve se podíváme, zda v této buňce je nebo není sloup. Pokud vylosujeme buňku, ve které je sloup, pak v této iteraci chodci zůstávají na svém místě a losujeme novou buňku. Pro prázdou buňku, do které je možné vstoupit, budou platit obdobná pravidla jako v sekci 2.2.1. Budeme se řídit pravidlem, že do prázdné buňky, kterou jsme vylosovali, chceme vždy posunout chodce z míst s nejvyšším potenciálem.

Nalezneme tedy vždy hodnotu nejvyššího potenciálu v okolí buňky, kterou chceme aktualizovat a určíme počet míst s tímto potenciálem. Dále se podíváme, kolik z těchto míst je plných. Označme n počet plných míst s nejvyšším potenciálem v okolí buňky, kterou chceme aktualizovat. Je-li $n > 0$, pak náhodně (opět s uniformním rozdělením) vybereme jedno z n

míst a posuneme chodce z tohoto vybraného místa do buňky, kterou chceme aktualizovat. Toto je hlavní myšlenka řešení konfliktu, tedy pokud chce více chodců vstoupit na jedno místo, tak vždy náhodně vybíráme jednoho z nich. Je-li $n = 0$, tedy žádné místo s nejvyšším potenciálem není obsazené člověkem, začneme hledat člověka v buňkách s druhým nejvyšším potenciálem, případně pak třetím a postupujeme zcela analogicky.

2.3 Výpočet gradientního pole pomocí Dijkstra algoritmu

Gradientní pole (proměnná `'levels'`), které je používáno agentem pro rozhodnutí (proměnná `'DECISION.m'`, Sekce 4), je generováno funkcí `'get_grad_field.m'`, Sekce 4. Funkce je spouštěna pouze jednou, na začátku programu. Vstupní parametry jsou:

- rozměry místnosti (P, Q),
- pozice východu (dvojice souřadnic),
- polohy sloupů (matice $2 \times O$, kde O je počet sloupů, nebo jiných obstrukcí).

Výstupem je proměnná `levels`, což je matice, představující místnost. Buňky, na které nelze vstoupit (stěny a sloupy) jsou reprezentovány hodnotou NaN, ostatní buňky obsahují číslo, představující minimální počet kroků, nezbytný pro dojití z dané buňky do buňky, která představuje východ. Východu je přiřazeno číslo 0. Příslušná nenulová čísla jsou nalezena pomocí tzv. *algoritmu pro nalezení nejkratší cesty* (shortest path algorithm), konkrétně Dijkstrův algoritmus [10].

Dijkstrův algoritmus (nazván podle nizozemského matematika Edsgera W. Dijkstry) je algoritmus sloužící pro nalezení nejkratší cesty na grafu. Vstupem pro algoritmus je matice grafu (tj. reprezentující graf a vzdálenosti z vrcholu do vrcholu) a indexy dvou uzlů grafu, mezi kterými hledáme nejkratší vzdálenost. V našem případě vzdálenosti mezi sousedními buňkami (sousední ve smyslu, že jedna leží v okolí druhé) byly vždy rovny jedné.

Na základě těchto podmínek funkce `'get_grad_field.m'` postupně očíslovala všechny volné[1] buňky od jedné do n (v kódu těmto číslům říkáme *index* buňky; zatímco dvojici čísel, které určují prvek dvojrozměrné matice, říkáme *souřadnice* buňky). Následně byla vytvořena matice o velikosti $n \times n$, kde její (i, j)-tý prvek představoval vzdálenost z buňky s indexem i do buňky s indexem j, tedy v případě, že buňky i a j sousedily, byla hodnota nastavena na 1, a pokud ne, hodnota byla nastavena na 0. Tedy tímto jsme representovali mapu místnosti floor field modelu jako matici grafu.

Nyní úloha hledání nejkratší cesty z buňky do buňky na mapě se převedla na úlohu hledání nejkratší vzdálenosti z uzlu do uzlu na grafu, a tedy bylo možné použít existující implementaci Dijkstrova algoritmu pro grafy. Jádrem funkce je existující implementace Dijkstra algoritmu [1].

3 Simulace

Na závěr ilustrujeme evakuaci místnosti na příkladech s a bez použití překážek a budeme zkoumat jejich vliv na rychlost evakuace chodců z místnosti. Intuitivně bychom čekali, že zpomalení bude nejvýraznější při umístění sloupů (překážek) blízko k východu, čím se ztíží přístup. Porovnáme proto tři případy:

- místnost s pěti sloupy tvořící bariéru přímo před východem a

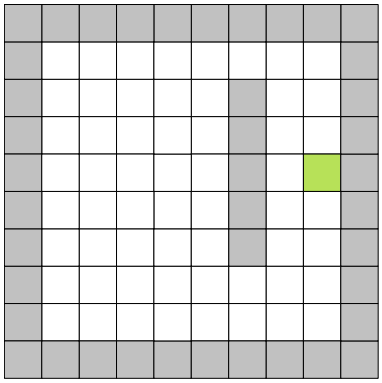
[1] Tj. takové, na kterých chodec může být (není tu zeď ani sloup).

- místnost s pěti sloupy tvořící bariéru dále od východu.

U každého z těchto případů provedeme 30 simulací (vždy se stejnými počátečními polohami agentů) a na základě nich provedeme statistickou analýzu střední hodnoty počtu kroků potřebného k úplné evakuaci místnosti. Počítáme přitom s místností o rozměrech 10×10, kde východ se nachází na pozici $[5, 9]$ mimo zeď. Do takto disponované místnosti následně umístíme sloupy a 30 chodců s náhodně generovanými počátečními polohami.

3.1 Bariéra u východu

V tomto případě jsme umístili 5 sloupů na pozice $[3, 7]$, $[4, 7]$, $[5, 7]$, $[6, 7]$ a $[7, 7]$, což je názorně vykresleno na obr. 8, kde šedou barvou jsou zvýrazněny obstrukce a zelenou je zvýrazněn východ.



Obrázek 8: Konfigurace místnosti pro první případ.

Naměřené hodnoty počtu časových kroků pro jednotlivé simulace jsou zaznamenány v tabulce 1. Předtím než ale přistoupíme k samotné statistické analýze musíme znát potřebnou teorii k odhadu střední hodnoty počtu kroků. Nakolik pro výpočet skutečné střední hodnoty bychom museli přistoupit k integraci přes celý definiční obor, provedeme jenom její odhad na základě aritmetického průměru

$$\bar{N} = \sum_{i=1}^{30} \frac{n_i}{30}. \tag{5}$$

Tady 30 je celkový počet proběhlých simulací a n_i je počet kroků potřebných k evakuaci místnosti při i-té simulaci. To je odhad střední hodnoty počtu kroků. Ten má ale vždy určitou chybu, kterou odhadneme pomocí Čebyševovi nerovnosti [8]. Označme μ jako skutečnou střední hodnotu počtu kroků a α jako hladinu významnosti. Potom z Čebyševovy nerovnosti získáváme odhad chyby střední hodnoty

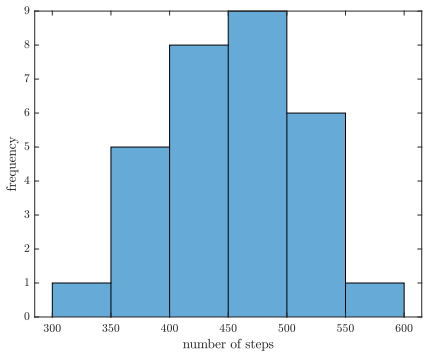
$$\delta = |\mu - \bar{N}| \leq \frac{\sigma}{\sqrt{30\alpha}}, \tag{6}$$

přičemž σ značí směrodatnou odchylku počtu kroků.

Po zvolení hladiny významnosti $\alpha = 0.05$ jsme obdrželi výsledek, že střední počet kroků potřebný pro evakuaci místnosti s 30 chodci je 453 ± 40. Jako součást grafické analýzy zde ještě přiložíme histogram počtu kroků, Obr. 9. Z histogramu se rovněž potvrzuje, že střední hodnota leží v intervalu $[450, 500]$.

	počet kroků		počet kroků		počet kroků
1	391	11	457	21	488
2	399	12	412	22	500
3	463	13	504	23	471
4	470	14	463	24	448
5	426	15	496	25	443
6	509	16	501	26	378
7	418	17	391	27	349
8	440	18	437	28	534
9	388	19	465	29	475
10	501	20	551	30	420

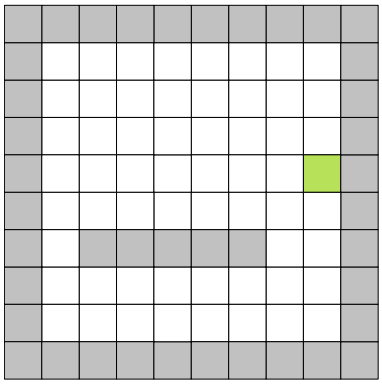
Tabulka 1: Naměřené hodnoty počtu kroků do úplné evakuace místnosti pro simulace s blokovaným východem.



Obrázek 9: Histogram počtu kroků potřebných pro evakuaci místnosti při blokaci východu.

3.2 Bariéra dále od východu

Porovnejme nyní první případ s případem, kde jsou sloupy umístěny na místa s nižším potenciálem dále od východu. Souřadnice sloupů byly voleny jako $[7, 3]$, $[7, 4]$, $[7, 5]$, $[7, 6]$ a $[7, 7]$, což odpovídá obrázku 10 níže.



Obrázek 10: Konfigurace místnosti pro druhý případ.

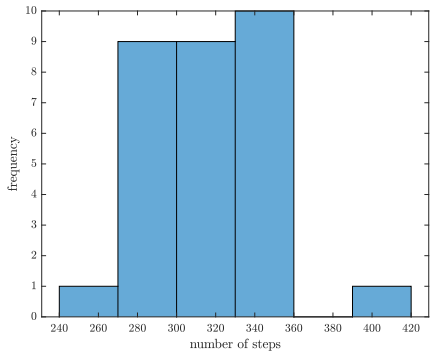
Opět uveďme tabulku pro napozorované hodnoty počtu kroků do evakuace místnosti, které budou sloužit jako data pro další analýzu. Ty uvádíme v tabulce 2.

Využitím vztahů (5) a (6) získáváme odhad střední hodnoty spolu s její chybou jako 315 ± 26. Podobné hodnoty pozorujeme i v histogramu 11. Obdrželi jsme tedy výsledky svědčící o rychlejší evakuaci místnosti při umístění bariér dále od východu. Volnější přístup

	počet kroků		počet kroků		počet kroků
1	310	11	290	21	283
2	335	12	320	22	338
3	331	13	307	23	320
4	350	14	292	24	352
5	320	15	297	25	293
6	321	16	331	26	409
7	302	17	337	27	287
8	279	18	336	28	346
9	333	19	326	29	244
10	272	20	306	30	272

Tabulka 2: Naměřené hodnoty počtu kroků do úplné evakuace místnosti pro simulace s blokací dále od východu.

k východu má tedy pozitivní vliv na rychlost vyprázdnění místnosti, což se shoduje s prací [4] citovanou v sekci 1.



Obrázek 11: Histogram počtu kroků potřebných pro evakuaci místnosti při blokaci dále od východu.

4 Závěr

V práci jsme využili FFM, který jsme nejdříve představili v sekci 1 a následně jej využili k implementaci simulace evakuace chodců z místnosti v části 3. Vycházeli jsme z práce [4], kde jsme se zaměřili na reprodukci části popisující vliv obstrukce východu na rychlost evakuace místnosti. To jsme demonstrovali na případech s jednoduchou přepážkou tvořenou pěti sloupy, kterou jsme umístili na místo blízko u východu a následně dále, čím se zlepšila dostupnost chodců k východu a obdrželi jsme tak očekávané výsledky shodné s prací výše. citovanou. Sekce 2 popisuje použité algoritmy, které jsou k dispozici k nahlédnutí v dodatku níže.

Zdrojový kód

Výpisy zdrojového kódu jsou realizovány pomocí `mcode` prostředí [5].

`Floor_field_model.m` - spouštěcí skript

```
%% FLOOR FIELD MODEL - spousteci skript

%% uklid
close all
clear variables

%% Nastaveni konstant (pomoci fce 'params.m')
% lze vytvorit nekolik ruznych sad pocatecnich parametru, treba fce
% 'params1.m', 'params2.m' atd)

params = parameters(1);
P = params.P;
Q = params.Q;
loc_of_exit = params.exit;
mtx_loc_of_pillars = params.pillars;
num_of_peds = params.pedestrians;

%[num_of_peds, P, Q, mtx_loc_of_pillars, loc_of_exit, animation_show] = ...
    params();
%[num_of_peds, P, Q, mtx_loc_of_pillars, loc_of_exit, animation_show] = ...
    params_big_room_full();  % pozor: Dijkstra bezi cca 4min na tak ...
    velke mape
%[num_of_peds, P, Q, mtx_loc_of_pillars, loc_of_exit, animation_show] = ...
    params_big_room_empty(); % pozor: Dijkstra bezi cca 4min na tak ...
    velke mape

animation_show = true;
%% Nastaveni mapy, rozmisteni sloupu a sten
walls = double(isnan(get_map (P, Q, loc_of_exit, mtx_loc_of_pillars))); ...
     % AG: melo by to vratit mapu, kde na vsech "nevkrocitelnych" polich ...
    je 1, a na volnych 0
% nevim, kde presne se to pouzije, ale jestli je potreba, tak tu je.

%% ------------------------- simulace FFM
n_simulations = 30;                         % pocet simulacii
iterations    = ones(n_simulations, 1);     % pocet krokov do uplnej ...
    evakuacie miestnosti

rng(1)                                      % pevny seed

for i = 1:n_simulations
%% Vypocet potencialu na mape (fce 'get_grad_field.m' od AG)
tStart = tic;
levels = get_grad_field(P, Q, loc_of_exit, mtx_loc_of_pillars);  % ...
    Dijkstra runs inside
tEnd = toc(tStart)
'log: get_grad_field.m successfully done'

%% Umisteni chodcu do mistnosti (fce 'get_pedestrians.m' od SM)
position_people = get_pedestrians(P, Q, mtx_loc_of_pillars, num_of_peds);
'log: get_pedestrians.m successfully done'
```

```
43
44 %% Pomocna matice pro vykresleni polohy exitu zelenou barvou
45 exit_matrix = 0*position_people; %stejne rozmery jako matice position ...
       people
46 exit_matrix(loc_of_exit(1),loc_of_exit(2)) = 1;
47
48 while (sum(sum(position_people)) ≠ 0)
49 [row_num,col_num] = size(position_people);
50 %budu se snazit generovat cisla pouze z te zaplnene casti, abych to
51 %urychlila
52 first_nonempty_col = 1;
53 while sum(position_people(:,first_nonempty_col)) == 0 && ...
       (first_nonempty_col + 1) ≤ (col_num - 1)
54 first_nonempty_col = first_nonempty_col + 1;
55 end
56
57
58 %ind_row a ind_col budou indexy bunky, kterou prave aktualizuju, nesmi
59 %to byt nic z okraju pole
60 ind_row = randi([2,row_num-1]);
61 ind_col = randi([first_nonempty_col,col_num-1]);
62
63 position_people = ...
       DECISION([ind_row,ind_col],position_people,levels,walls,loc_of_exit);
64 R = 255 * (-position_people+1) - 255*exit_matrix;
65 G = 255 * (-position_people+1)-255*walls;
66 B = 255 * (-position_people+1)-255*walls - 255*exit_matrix;
67 res = cat(3,R,G,B);
68
69 %         %plot result
70 if animation_show == true
71 imshow(res,'InitialMagnification',20000)
72 pause(0.07);
73 end
74
75 iterations(i) = iterations(i)+1;
76 end
77 end
78
79 % Stredny pocet krokov potrebny pre evakuaciu miestnosti
80 [mu, Δ] = expectation(iterations, n_simulations);
81 histogram(iterations)
82
83 'Done.'
```

params.m

Funkce, sloužící pro nastavení veškerých konstant při spuštění programu.

```
1 function [num_of_peds, P, Q, mtx_loc_of_pillars, loc_of_exit, ...
       animation_show] = params()
2 %% nastavi pouzivane konstanty - pro pohodlnejsi ladeni
3 % Konstanty:
4 %  - P ... sirka mistnosti (vertikalni smer
5 %  - Q ... delka mistnosti (vodorovny smer)
6 %  - loc_of_exit ... poloha vychodu, tj. vektor o dvou souradnicich. NESMI
7 %  BYT VE ZDI, MUSI BYT V MISTNOSTI U ZDI
8 %  - mtx_loc_of_pillars ... polohy sloupu, tj. matice o velikosti 2xS, kde
```

```
%  S je pocet sloupu (horni radek matice jsou x-ove polohy sloupu, dolni
%  radek jsou y-ove polohy)

num_of_peds = 5;
P = 7;
Q = 8;

mtx_loc_of_pillars = [3, 4, 4, 5;
4, 6, 5, 5];   % tj. sloupy na pozicich (3,4), (4,6) a (5,5)
loc_of_exit = [4, 7];

animation_show = false;
end
```

parameters.m

Funkce, podstatou shodná s 4, obohacena o ukládání dat do struktury.

```
function data = parameters(i)
%% Fixne parametre
%  - P ... sirka mistnosti (vertikalni smer)
%  - Q ... delka mistnosti (vodorovny smer)
%  - exit ... poloha vychodu, tj. vektor o dvou souradnicich. NESMI
%  BYT VE ZDI, MUSI BYT V MISTNOSTI U ZDI
%  - pillars ... polohy sloupu, tj. matice o velikosti 2xS, kde
%  S je pocet sloupu (horni radek matice jsou x-ove polohy sloupu, dolni
%  radek jsou y-ove polohy)

exit   = [4, 9];
n_peds = 30;
P      = 10;
Q      = 10;

%% Rozmiestnenie stlpov v miestnosti
switch i
case 1
pillars = [3, 4, 4, 5;
4, 6, 5, 5];     % povodne nastavenie stlpov

case 2
pillars = [4;
7];              % umiestnenie stlpu pred vychod
end

%% Vytvorenie struktury
data = ...
    struct('P',P,'Q',Q,'exit',exit,'pedestrians',n_peds,'pillars',pillars);
end
```

get_pedestrians.m

Funkce pro počáteční (náhodné) rozmístění zadaného počtu chodců na volné pozice (přidělení počátečních poloh chodcům).

```
function room = get_pedestrians(grid_length, grid_width, pillars, ...
    n_pedestrians)
%% HELP
% get_pedestrians   ... function used to generate initial positions of ...
    a given number of pedestrians
%
% Input arguments:
% grid_length       ... first array dimension (ie. number of rows)
% grid_width        ... second array dimension (ie. number of columns)
% pillars           ... array of pillar coordinates
% n_pedestrians     ... number of pedestrians
%
% Output argument:
% room              ... array with generated pedestrians
%% Code
max_pedestrians = (grid_length - 2)*(grid_width - 2) - size(pillars, ...
    2);% disposable room area (excludes walls and pillars)
pillar_cells    = grid_length*(pillars(2,:) - 1) + pillars(1,:);        ...
    % pillars locations (converted to single subscript indexation)

% Error message
if n_pedestrians > max_pedestrians                                  % two ...
    pedestrians cannot share the same cell
disp("Room capacity too low.")

% Generating pedestrians
else
% Create an empty array
room = zeros(grid_length, grid_width);                          % empty ...
    array to be filled with pedestrians

% Pillars in the same positions as the pedestrians?
while sum(room, "all") ≠ n_pedestrians
pedestrians = zeros(grid_length - 2, grid_width - 2);   % auxiliary ...
    array to be filled with pedestrians (no walls or pillars)
position = randperm(max_pedestrians, n_pedestrians);    % generating ...
    pedestrian positions
pedestrians(position) = 1;                                  % inserting ...
    pedestrians
room(2:grid_length-1, 2:grid_width-1) = pedestrians;    % inserting ...
    pedestrian array inside walls

% Insert pillars
room(pillar_cells) = 0;
end
end
end
```

get_grad_field.m

Funkce, generující gradientní pole v místnosti. Součástí je implementace Dijkstrova algoritmu [1].

```
function grad_field = get_grad_field (P, Q, loc_of_exit, ...
    mtx_loc_of_pillars)
%% GET_GRAD_FIELD (function)
% function that generates gradient field on map, based on given parameters
%
```

```
% INPUT:
%  - Q ... delka mistnosti (vodorovny smer)
%  - P ... sirka mistnosti (vertikalni smer)
%  - loc_of_exit ... poloha vychodu, tj. vektor o dvou souradnicich. NESMI
%  BYT VE ZDI, MUSI BYT V MISTNOSTI U ZDI
%  - mtx_loc_of_pillars ... polohy sloupu, tj. matice o velikosti 2xS, kde
%  S je pocet sloupu
%
% OUTPUT:
%  - grad_field ... matice o velikosti PxQ, kde plati:
%     -- pole s vychodem ma hodnotu 0
%     -- pole, na ktere nelze vstoupit (zed nebo sloup) ma
%        hodnotu NaN
%     -- pole, na ktera vstoupit lze, maji hodnotu,
%        odpovidajici minimalnimu poctu kroku, kterych je
%        potreba pro dosazeni vychodu
%
% KNOWN PROBLEMS:
%  - chybi kontrola vstupu. Ocekava se, ze funkce dostane to, co ocekava.
%  Pridat?...
%  - efektivita vyresena
%
% Created by AG, on 20230110
% Updated by AG, on 20230122

%% code goes below

%% inicializace: na zaklade vstupu vytvorime matici 'map', obsahujici ...
   steny a sloupy
map = nan(P,Q);
for i=2:P-1
for j=2:Q-1
map(i,j) = 0;
end
end

loc_of_exit_x = loc_of_exit(1);
loc_of_exit_y = loc_of_exit(2);
map(loc_of_exit_x, loc_of_exit_y) = 0;

size_mtx_loc_of_pillars = size(mtx_loc_of_pillars);
num_of_pillars = size_mtx_loc_of_pillars(2);

for pillar = 1:num_of_pillars
loc_of_pillar_x = mtx_loc_of_pillars(1,pillar);
loc_of_pillar_y = mtx_loc_of_pillars(2,pillar);
map(loc_of_pillar_x, loc_of_pillar_y) = NaN;
end

%% ocisluji vsechna pole, na ktera lze vstoupit (vsechna pole, ktera ...
   nejsou zed nebo sloup)
num_of_cells = 0;
index_mtx = nan(P,Q);
grad_field = nan(P,Q);
n = 1;

for i=1:P
    for j=1:Q
        if map(i,j) == 0
        num_of_cells = num_of_cells + 1;
```

```
63          index_mtx(i,j) = n;
64          grad_field(i,j) = inf;
65          n = n + 1;
66          end
67      end
68  end
69
70  grad_field(loc_of_exit_x, loc_of_exit_y) = 0;
71
72  % num_of_cells ... pocet bunek, na ktere lze (v principu) vkrocit
73  % index_mtx    ... matice PxQ, vsechny bunky, na ktere lze vkrocit, jsou
74  %                  postupne ocislovany od 1 do num_of_cells
75
76  %% teorie grafu: vytvorime "matici prechodu"
77  % tj. matici o velikosti num_of_cells x num_of_cells, kde (i,j)-ty ...
        prvek je
78  % roven 0, pokud se z bunky 'i' do bunky 'j' nelze dostat jednim ...
        krokem, a
79  % je roven 1, pokud lze
80  % Jinymi slovy: pokud bunka 'i' sousedi s bunkou 'j', pak graph_mtx(i,j)=1.
81  % Pokud nesousedi, graph_mtx(i,j)=0.
82
83  %graph_mtx = nan(num_of_cells, num_of_cells);
84  graph_mtx = [];
85
86  for k=1:num_of_cells   % k probiha vsechny bunky, na ktere lze vkrocit
87      [i,j] = get_coords(k, index_mtx); % pro k-tou bunku najdu jeji ...
            souradnice
88      set_of_neighbours = get_surroundings(i,j, index_mtx); % najdu ...
            mnozinu sousedu bunky s indexem k
89      kth_row_of_graph_mtx = write_row(set_of_neighbours, num_of_cells);
90      graph_mtx = [graph_mtx; kth_row_of_graph_mtx];
91  end
92
93  %% Dijkstra (shortest path algorithm)  ... podle stazene fce
94  grad_field = zeros(P,Q);  % inicializace: vsechny cesty jsou nekonecne
95  l = index_mtx(loc_of_exit_x, loc_of_exit_y); % pozor / je to index :L:, ...
        ne jednicka
96
97  for k=1:num_of_cells
98      [i,j] = get_coords(k, index_mtx); % pro k-tou bunku najdu jeji ...
            souradnice
99      grad_field(i,j) = dijkstra(graph_mtx, k, l);
100 end
101
102 grad_field = grad_field + map; % protoze dijkstra za sebou zanechava
103 % nuly nejen tam, kde byl vychod, ale i
104 % tam, kde jsou steny a sloupy
105
106 end
```

get_coords.m

Funkce, která na základě indexu buňky najde její souřadnice v matici.

```
1  function [i,j] = get_coords(k, index_mtx)
2  %% GET_COORDS (funkce)
3  % pomocna funkce pro funkci 'get_grad_field.m'
```

```
% INPUT:
%   - k ... cislovaci index volne bunky
%   - index_mtx ... matice velikosti i strukturou shodna s realnou mapou
%   mistnosti 'map', tj. PxQ, je vytvarena ve funkci 'get_grad_field.m'
%
% OUTPUT:
%   - i,j ... souradnice, na kterych se nachazi bunka, ocislovana ...
    indexem k
%
% Created by AG, 20230110

%% code goes below
i = NaN;
j = NaN;
size_of_index_mtx = size(index_mtx);

for ii = 1:size_of_index_mtx(1)
    for jj = 1:size_of_index_mtx(2)
        if index_mtx(ii,jj) == k
        %return i,j
        i = ii;
        j = jj;
        end
    end
end

if ( isnan(i) || isnan(j) )
    error('Error in get_coords(): such k was not found in given matrix ...
       index_mtx. Abort.')
end

end
```

`get_surroundings.m`

Funkce, která na základě souřadnic buňky najde indexy buněk v jejím okolí (tj. nejvýše 8 buněk, méně, pokud v okolí je sloup nebo prvek zdi).

```
function set_of_neighbours = get_surroundings(i,j, index_mtx)
%% GET_SURROUNDINGS (function)
% pomocna funce pro funkci 'get_grad_field.m'
% vrati indexy vsech sousedicich dostupnych bunek (tj. jen takovych,
% ktere nejsou stena ani sloup)
%
% INPUT:
%  - i,j ... souradnice bunky B
%  - index_mtx ... matice velikosti i strukturou shodna s realnou mapou
%   mistnosti 'map', tj. PxQ, je vytvarena ve funkci 'get_grad_field.m'
%
% OUTPUT:
%  - set_of_neighbours ... vektor cisel, predstavujici indexy vsech ...
    bunek v
%  okoli bunky B (indexy, ktere jim prirazuje matice 'index_mtx')
%
% Created by AG, 20230110

% Zasadni myslenka: na vstupu bude fce dostavat jen "povolena" i a j.
% To znamena, ze se nemuze stat, ze by skenovaci okoli 3x3 "vypadlo" z nasi
```

```
% matice (ze bych se v kodu nize snazil pristupovat k prvkum matice, ktere
% jsou mimo rozmer matice).

%% code goes here
set_of_neighbours = [];
size_of_index_mtx = size(index_mtx);
%if i == size_of_index_mtx(1) ||
for ii = i-1:i+1
    for jj = j-1:j+1
        if (ii < size_of_index_mtx(1) && jj < size_of_index_mtx(2) )
            if ( ¬isnan(index_mtx(ii,jj)) && index_mtx(ii,jj) ≠ ...
                index_mtx(i,j) )
            set_of_neighbours = [set_of_neighbours, index_mtx(ii,jj)];
            end
        end
    end
end

end
```

write_row.m

```
function kth_row_of_graph_mtx = write_row(set_of_neighbours, num_of_cells)
%% WRITE_ROW
% pomocna funkce pro funkci 'get_grad_field.m'
% INPUT:
%   - set_of_neighbours ... cislovaci index volne bunky
%   - index_mtx ... matice velikosti i strukturou shodna s realnou mapou
%   mistnosti 'map', tj. PxQ, je vytvarena ve funkci 'get_grad_field.m'
%
% OUTPUT:
%   - i,j ... souradnice, na kterych se nachazi bunka, ocislovana ...
    indexem k
%
% Created by AG, 20230110

%% code goes below
kth_row_of_graph_mtx = [];

for j = 1:num_of_cells
if isneighbour(j, set_of_neighbours)
kth_row_of_graph_mtx = [kth_row_of_graph_mtx, 1];
else
kth_row_of_graph_mtx = [kth_row_of_graph_mtx, 0];  % puvodne to ma byt nula
end
end

end
```

decision.m

```
function new_position_people = ...
    DECISION(position_to_be_updated,position_people,levels,walls,exit)
```

```
%INPUT ... %position_to_be_updated ... nahodne vybrany index bunky, ...
    jejiz okoli budeme
%aktualizovat v cele mrizce position_people..... NESMI TO BYT ZADNA Z
%BUNEK NA OKRAJI
%position_people ... tabulka 0 a 1 vyjadrujici polohy lidi na
%mrizce
%levels ... tabulka potencialu
%walls ... tabulka sten a sloupu
%exit ... zvlast vypichnute souradnice vychodu
%OUTPUT ... new_position ... aktualizovana mrizka position_people

new_position_people = position_people; %inicializace

%v patem sloupci promenne neighborhood jsou hodnoty potencialu, pripadne
%NAN pro steny, nebo sloupce
neighborhood = ...
    GET_OKOLI(position_to_be_updated,position_people,levels,walls);

if position_people(position_to_be_updated(1),position_to_be_updated(2)) ...
    == 1 %%%%%%nejdriv resim, pokud je ta vybrana bunka plna
min_pot = min(neighborhood(:,5));

%nalezeni radku s minimalni hodnotou potencialu
indeces_of_min_pot = find(neighborhood(:,5)==min_pot);
%pocet radku s tim minimalnim potencialem
[n_min_pots,¬] = size(indeces_of_min_pot);
%nejriv je snazime tlacit na pole s minimalnim potencialem
is_there_a_man = 0;
is_there_a_wall = 0;
for ind = 1:n_min_pots
is_there_a_man(ind) = neighborhood(indeces_of_min_pot(ind),4);
is_there_a_wall(ind) = neighborhood(indeces_of_min_pot(ind),3);
end
if sum(is_there_a_man) < n_min_pots && sum(is_there_a_wall) < ...
    n_min_pots %pokud bude mozny vubec nejaky pohyb do vrstvy s nizsim ...
    potencialem
%pocet volnych bunek s nejnizsim potencialem
free_spots = n_min_pots - sum(is_there_a_man);
%nahodne vybere jednu v dostupnych volnych pozic
r = randi(free_spots);
%ted vyselektuju, kterou volnou pozici jsme vlastne vybrali
free_spot_position = find(is_there_a_man == 0);
selected_free_spot = free_spot_position(r);
selected_neigh_row = ...
    neighborhood(indeces_of_min_pot(selected_free_spot),:);

new_position_people(position_to_be_updated(1),position_to_be_updated(2)) ...
    = 0;
new_position_people(selected_neigh_row(1),selected_neigh_row(2)) = 1;


elseif  sum(is_there_a_man) == n_min_pots || sum(is_there_a_wall) == ...
    n_min_pots %pokud se nelze pohnout z duvodu preplnenosti nebo sten ...
    do vrstvy s nejnizsim potencialem
sec_min_pot = min(neighborhood(:,5))+1;
%nalezeni radku s druhou minimalni hodnotou potencialu
indeces_of_sec_min_pot = find(neighborhood(:,5)==sec_min_pot);
%pocet radku s timto potencialem
[n_sec_min_pots,¬] = size(indeces_of_sec_min_pot);
%nejdriv je snazime tlacit na pole s minimalnim potencialem
```

```
is_there_a_man = 0;
is_there_a_wall = 0;
for ind = 1:n_sec_min_pots
is_there_a_man(ind) = neighborhood(indeces_of_sec_min_pot(ind),4);
is_there_a_wall(ind) = neighborhood(indeces_of_sec_min_pot(ind),3);
end
if sum(is_there_a_man) < n_sec_min_pots && sum(is_there_a_wall) < ...
    n_sec_min_pots %pokud bude mozny vubec nejaky pohyb
%pocet volnych bunek s nejnizsim potencialem
free_spots = n_sec_min_pots - sum(is_there_a_man);
%nahodne vybere jednu v dostupnych volnych pozic
r = randi(free_spots);
%ted vyselektuju, kterou volnou pozici jsme vlastne vybrali
free_spot_position = find(is_there_a_man == 0);
selected_free_spot = free_spot_position(r);
selected_neigh_row = ...
    neighborhood(indeces_of_sec_min_pot(selected_free_spot),:);

new_position_people(position_to_be_updated(1),position_to_be_updated(2)) ...
    = 0;
new_position_people(selected_neigh_row(1),selected_neigh_row(2)) = 1;
end

end
else %aktualizace prazdne bunky je zalozena na tom, ze se tam ...
    prednostne tlaci spis lidi s vetsim potencialem a az potom ti na ...
    stejne urovni
if isnan(neighborhood(5,5)) %5. radek pole neighborhood odpovida bodu, ...
    ktery prave aktualizujeme, 5. sloupec pokud ma hodnotu NAN znamena, ...
    ze je tam zed/sloup
;
else
max_pot = max(neighborhood(:,5));
%nalezeni radku s maximalni hodnotou potencialu
indeces_of_max_pot = find(neighborhood(:,5)==max_pot);
%pocet radku s tim maximalnim potencialem
[n_max_pots,¬] = size(indeces_of_max_pot);
is_there_a_man = 0;
for ind = 1:n_max_pots
is_there_a_man(ind) = neighborhood(indeces_of_max_pot(ind),4);
end
if sum(is_there_a_man) > 0 %pokud bude mozny vubec nejaky pohyb do ...
    vrstvy s nizsim potencialem
%pocet plnych bunek s nejvyssim potencialem
full_spots = sum(is_there_a_man);
%nahodne vybere jednu z dostupnych plnych pozic
r = randi(full_spots);
%ted vyselektuju, kterou plnou pozici jsme vlastne vybrali
full_spot_position = find(is_there_a_man == 1);
selected_full_spot = full_spot_position(r);
selected_neigh_row = ...
    neighborhood(indeces_of_max_pot(selected_full_spot),:);

new_position_people(position_to_be_updated(1), ...
    position_to_be_updated(2)) = 1;
new_position_people(selected_neigh_row(1), selected_neigh_row(2)) = 0;
elseif sum(is_there_a_man) == 0 %z vrstvy s nejvyssim potencialem nema ...
    kdo sejit niz
sec_max_pot = max(neighborhood(:,5))-1;
%nalezeni radku s druhou nejvyssi hodnotou potencialu
```

```
indeces_of_sec_max_pot = find(neighborhood(:,5)==sec_max_pot);
%pocet radku s timto potencialem
[n_sec_max_pots,¬] = size(indeces_of_sec_max_pot);
%nejdriv je snazime tlacit na pole s minimalnim potencialem
is_there_a_man = 0;
for ind = 1:n_sec_max_pots
is_there_a_man(ind) = neighborhood(indeces_of_sec_max_pot(ind),4);
end
if sum(is_there_a_man) > 0 %pokud bude mozny vubec nejaky pohyb
%pocet plnych bunek s nejvyssim potencialem
full_spots = sum(is_there_a_man);
%nahodne vybere jednu z dostupnych plnych pozic
r = randi(full_spots);
%ted vyselektuju, kterou plnou pozici jsme vlastne vybrali
full_spot_position = find(is_there_a_man == 1);
selected_full_spot = full_spot_position(r);
selected_neigh_row = ...
    neighborhood(indeces_of_sec_max_pot(selected_full_spot),:);

new_position_people(position_to_be_updated(1), ...
    position_to_be_updated(2)) = 1;
new_position_people(selected_neigh_row(1),selected_neigh_row(2)) = 0;
elseif sum(is_there_a_man) == 0 %pokud ani z druheho nejvyssiho ...
    potencialu neni mozne sejit niz
thrd_max_pot = max(neighborhood(:,5))-2;
if thrd_max_pot ≥ neighborhood(5,5) %vezmu jeste treti mozny potencial, ...
    nesmi ale byt nizsi nez ten, ktery prislusi prazdne bunce, kterou se ...
    snazim aktualizovat
%nalezeni radku s treti nejvyssi hodnotou potencialu
indeces_of_thrd_max_pot = find(neighborhood(:,5)==thrd_max_pot);
%pocet radku s timto potencialem
[n_thrd_max_pots,¬] = size(indeces_of_thrd_max_pot);
%nejdriv je snazime tlacit na pole s minimalnim potencialem
is_there_a_man = 0;
for ind = 1:n_thrd_max_pots
is_there_a_man(ind) = neighborhood(indeces_of_thrd_max_pot(ind),4);
end
if sum(is_there_a_man) > 0 %pokud bude mozny vubec nejaky pohyb
%pocet plnych bunek s nejvyssim potencialem
full_spots = sum(is_there_a_man);
%nahodne vybere jednu z dostupnych plnych pozic
r = randi(full_spots);
%ted vyselektuju, kterou plnou pozici jsme vlastne vybrali
full_spot_position = find(is_there_a_man == 1);
selected_full_spot = full_spot_position(r);
selected_neigh_row = ...
    neighborhood(indeces_of_thrd_max_pot(selected_full_spot),:);

new_position_people(position_to_be_updated(1), ...
    position_to_be_updated(2)) = 1;
new_position_people(selected_neigh_row(1), selected_neigh_row(2)) = 0;
end
end
end
end
end

end
%pokud bude plna bunka se souradnicemi "exit", pred vykreslenim ji vzdy
%vyprazdnim
```

```
155
156 if new_position_people(exit(1),exit(2)) == 1
157 new_position_people(exit(1),exit(2)) = 0;
158 end
159
160 end
```

expectation.m

```
1 function [mu, Δ] = expectation(n_steps, n_simulations)
2 sigma = std(n_steps);          % number of steps standard deviance
3 alpha = 0.05;                  % significance level
4
5 %% Expected Value and Its Error
6 mu    = 1/n_simulations*sum(n_steps);          % number of steps ...
      expectation (arithmetic mean)
7 Δ = sigma/sqrt(n_simulations*alpha);       % expectation error
8 end
```

Reference

[1] Dimas Aryo. Dijkstra Algorithm. https://www.mathworks.com/matlabcentral/fileexchange/36140-dijkstra-algorithm. MATLAB Central File Exchange. Retrieved January 22, 2023. 6, 13

[2] A. Schadschneider J. Zittartz C. Bustedde, K. Klauck. Simulation of pedestrian dynamics using a two-dimensional cellular automaton. *Physica A*, 295:507–525, 2001. 2

[3] E. F. Codd. *Cellular automata*. Academic Press, 1968. 1

[4] A. Namazi A. Schadschneider K. Nishinari, A. Kirchner. Extended floor field ca model for evacuation dynamics. *IEICE Transactions on Information and Systems*, 87(3):726–732, 2004. 3, 9

[5] Florian Knorn. M-code latex package. https://www.mathworks.com/matlabcentral/fileexchange/8015-m-code-latex-package, 2023. MATLAB Central File Exchange. Retrieved January 26, 2023. 10

[6] MATLAB. *version 9.13.0. (R2022a)*. The MathWorks Inc., Natick, Massachusetts, 2022. 3

[7] E. F. Moore. Machine models of self-reproduction. *Mathematical Problems in the Biological Sciences*, 14:17–35, 1962. 2

[8] B. van Parys P. B. Stellato, B. J. Goulart. Multivariate chebyshev inequality with estimated mean and variance. *The Americal Statistician*, 71(2):123–127, 2016. 7

[9] Jana Vacková. SSI – přednášky, 2021.

[10] Wikipedia. Dijkstra s algorithm. Accessed on January 28, 2023. 6