FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Plánování evakuace založené na lokálních technikách kooperativního hledání cest
Student:	Róbert Selvek
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Úloha evakuace spočívá v plánování přesunu osob z ohrožené oblasti do bezpečí [1]. Při grafové interpretaci úlohy jde o nalezení nekonfliktních cest v neorientovaném grafu vedoucí evakuované osoby z ohrožených vrcholů do vrcholů bezpečných. Úlohu lze modelovat a řešit optimálně jako hledání maximálního toku v časově expandovaném grafu, ovšem tento přístup vyžaduje centrální plánování, které není při evakuaci často realizovatelné. V rámci navrhovaného tématu je cílem prozkoumat možnosti úpravy lokálních algoritmů hledání cest [2], jako je například LRA* tak, aby produkovaly řešení zohledňující cíle evakuace (co nejvíce zachráněných osob, v co nejkratším čase atd.).

Předpokládaným výsledkem práce bude nový algoritmus či modifikace existujícího algoritmu. Výsledný algoritmus bude implementován jako softwarový prototyp a dále bude teoreticky a experimentálně vyhodnocen.

Seznam odborné literatury

[1] Ajay Gupta, Nandlal L. Sarda: Efficient Evacuation Planning for Large Cities. Database and Expert Systems Applications - 25th International Conference (DEXA (1) 2014), pp. 211-225, LNCS 8644, Spinger, 2014

[2] David Silver: Cooperative Pathfinding. Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005), pp. 117-122, AAAI Press, 2005

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 14. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Plánovanie evakuácie založené na lokálnych technikách kooperatívneho hľadania ciest

Róbert Selvek

Katedra teoretické informatiky
Vedúci práce: doc. RNDr. Pavel Surynek, Ph.D.

15. mája 2019

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k užívaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo užívať. Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené.

V Prahe 15. mája 2019 ……

Odkaz na túto prácu

Selvek, Róbert. *Plánovanie evakuácie založené na lokálnych technikách kooperatívneho hľadania ciest.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

V tejto práci sa venujeme otázke evakuácie z budov alebo priestranstiev z perspektívy algoritmov na kooperatívne hľadanie ciest. Definujeme problém nazvaný multi-agentná evakuácia, ktorý sa skladá z neorientovaného grafu, v ktorého vrcholoch sa nachádzajú agenti (napríklad ľudia, roboty alebo postavy v počítačovej hre). Vrcholy v grafe sú označené ako ohrozené alebo bezpečné a úlohou je naplánovať presun agentov z ohrozených do bezpečných vrcholov v čo najkratšom čase.

Existujú centralizované algoritmy založené na modelovaní tokov v sieťach, ktoré dokážu tento problém riešiť optimálne vzhľadom na rôzne ukazovatele. V reálnom svete sa však tieto algoritmy dajú aplikovať len problematicky, pretože skutoční agenti nie sú schopní nasledovať centrálne generovaný plán a musia reagovať na meniace sa situácie, ako napríklad nekooperujúcich agentov. Preto sme navrhli a implementovali algoritmus na plánovanie evakuácie založený na technikách lokálneho kooperatívneho hľadania ciest. Simulácie na viacerých realistických situáciách ukazujú, že riešenia generované týmto algoritmom majú kvalitu podobnú riešeniam z centralizovaných algoritmov.

Kľúčové slová evakuácia, WHCA*, kooperatívne hľadanie ciest, multi-agentné hľadanie ciest, Python, tokové algoritmy

Abstract

This thesis addresses the problem of evacuation from buildings and open areas from the perspective of cooperative path-finding algorithms. We define a problem called Multi-Agent Evacuation based on undirected graphs with agents (such as people, robots, or game characters) located in their vertices. Each vertex can either be marked as safe or as endangered. The task consists of moving the agents from endangered to safe vertices as quickly as possible.

There are centralized algorithms for solving this problem that are optimal with respect to various objectives. Such algorithms are hardly applicable to real-world scenarios because real agents are unable to follow the centrally generated plan and must react to changes in the environment. Therefore, we designed and implemented an evacuation planning algorithm based on local cooperative path finding techniques. Simulations on various realistic situations show that solutions generated by this algorithm are of a quality similar to the solutions from centralized algorithms.

Keywords evacuation, WHCA*, cooperative path finding, multi-agent path finding, Python, flow algorithms

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Kooperatívne a multi-agentné hľadanie ciest je významná výskumná téma v rozličných oblastiach umelej inteligencie a návrhu softvéru, od plánovania pohybu robotov až po vývoj hier. Typicky sa definuje ako hľadanie ciest pre viacerých agentov z ich počiatočných lokácií do ich cieľov. V tomto kontexte môže byť agent napríklad robot, človek alebo postava v počítačovej hre. Existujú rôzne algoritmy, ktoré tento problém s rôznymi výhodami a nevýhodami riešia.

Bežná definícia kooperatívneho hľadania ciest však vyžaduje, aby agenti mali definované ciele, na rozdiel od problému *evakuácie*, kde síce vieme, na akých pozíciách by sa agenti nachádzať nemali, určiť však, kam konkrétne sa majú snažiť dostať, nie je triviálne.

Existujú algoritmy, ktoré problém evakuácie prostredníctvom rôznych prístupov riešia. Pri jeho modelovaní však robia rôzne kompromisy, ktoré znižujú ich realizmus. Príkladom je predpoklad, že všetci agenti sú rovnako informovaní, majú rovnaké schopnosti pohybu, a že sú ochotní nasledovať centrálne generovaný plán. Niektoré algoritmy zas predpokladajú, že sa agenti dajú rozdeliť do skupín, ktoré sa v priebehu evakuácie dajú spoľahlivo vnímať ako jedna jednotka.

Ďalší kompromis môže nastať pri modelovaní prostredia, v ktorom evakuácia prebieha. Pri transformácií skutočného prostredia na graf volia niektoré algoritmy príliš veľkú mieru abstrakcie, pri ktorej sa miestnosti, v ktorých môžu byť stovky agentov, reprezentujú ako jediný vrchol v grafe.

Evakuačné plány generované algoritmami, ktoré robia vyššie spomenuté kompromisy, sú užitočné na plánovanie a simuláciu evakuácie vo veľkej mierke, na úrovni miest alebo ich častí. Na simuláciu evakuácie budov alebo priestranstiev sú však použiteľné len ťažko, pretože takáto simulácia by bola veľmi nepresná a nerealistická.

V tejto práci preto navrhujeme rozšírenie existujúceho algoritmu decentralizovaného kooperatívneho hľadania ciest, tj. takého, v ktorom agenti sami

robia rozhodnutia o svojej budúcej ceste, tak, aby efektívne riešil problém evakuácie. Vďaka takémuto individuálnemu plánovaniu a detailnému modelovaniu prostredia je náš algoritmus vhodný na modelovanie a plánovanie evakuačných situácií, v ktorých je žiadúce sledovať správanie sa evakuovaných agentov na úrovni jednotlivcov v dave. Keďže algoritmus dokáže simulovať agentov s rôznou úrovňou racionality správania, výsledky experimentálneho overovania môžu vypovedať aj o správaní sa evakuovaného davu, v ktorom sa nachádzajú ľudia, ktorí sa kvôli panike nesprávajú rozumne.

V teoretickej časti práce uvádzame prehľad existujúcej literatúry a algoritmov, ktoré riešia problém multi-agentného hľadania ciest. Veľkú pozornosť venujeme algoritmu WHCA*, ktorý tvorí základ nášho algoritmu.

V praktickej časti diskutujeme rozšírenie algoritmu WHCA* na problém evakuácie a jeho implementáciu v jazyku Python. Práca je zakončená výsledkami overenia algoritmu na rôznych mapách a analýzou, ako ovplyvňujú rôzne pomery agentov so zníženou mierou racionálneho správania dĺžku evakuácie.

Kapitola 1

Cieľ práce

Cieľom tejto práce je návrh a experimentálne overenie lokálneho algoritmu na plánovanie evakuácie viacerých agentov. Na jeho splnenie je nutné vykonať viacero čiastkových úloh.

Prvým čiastkovým cieľom je analýza aktuálnej literatúry týkajúcej sa multi-agentného a kooperatívneho hľadania ciest. Okrem toho je nutné získať prehľad a kriticky zhodnotiť aktuálne algoritmy používané na počítačové plánovanie evakuácie a ich využiteľnosť na rôzne typy tejto úlohy.

Druhý čiastkový cieľ, ktorý je nutné splniť, je samotné vytvorenie algoritmu použiteľného na plánovanie evakuácie. Tento algoritmus musí vedieť plánovať evakuáciu agentov nachádzajúcich sa na mriežke, ktorej každé políčko môže byť označené ako bezpečné alebo nebezpečné. Okrem toho má byť algoritmus lokálny, respektíve decentralizovaný, a teda každý agent musí riešiť problém evakuácie nezávisle na ostatných, s výnimkou riešenia kolízií na lokálnej úrovni.

Aby sa algoritmus dal experimentálne overiť, je nutné ho implementovať. Táto implementácia musí byť dostatočne flexibilná, aby sa rôzne časti algoritmu dali zmeniť alebo konfigurovať pre rôznych agentov nachádzajúcich sa na mape. Musí naviac umožňovať mapy, na ktorých plánovanie prebieha jednoducho vytvárať a editovať. Plány, ktoré algoritmus vygeneruje sa musia dať pomocou nástrojov dodaných s implementáciou vizualizovať.

Napokon je nutné pri experimentálnom overení stanoviť metriky, na základe ktorých sa algoritmus dá hodnotiť a pomocou ktorých sa dá diskutovať jeho výkon a kvalita generovaných plánov. Treba overiť, ako sa správa na mapách rôznej veľkosti, s rôznymi útvarmi a počtom a počiatočným rozmiestnením agentov. V rámci tohto experimentálneho overenia sa taktiež overia výsledky evakuácie v závislosti na rôznych pomeroch agentov s rôzne inteligentným správaním.

Kapitola 2

Teoretické východiská práce

Vzhľadom na to, že problém multi-agentnej evakuácie je modifikáciou problému multi-agentného hľadania ciest, začína táto kapitola s popisom multi-agentného hľadania ciest a algoritmov používaných na riešenie, tohto problému.

2.1 Multi-agentné hľadanie ciest

Typické algoritmy hľadania ciest, ako napríklad A*, sa zaoberajú problémom naplánovania cesty medzi dvoma vrcholmi u, v grafu $G = (V, E)$ [1]. To zodpovedá situácií, kedy sa robot alebo osoba snažia premiestniť medzi dvoma bodmi v priestore, ktorý je modelovaný grafom G.

Tento problém sa dá zovšeobecniť na situáciu, v ktorej sa v priestore nachádza na rôznych pozíciách viacero robotov alebo osôb (ďalej len *agentov*), ktoré sa chcú dostať na rozličné cieľové pozície. Naivné riešenie by mohlo pre postupnosti vrcholov $u_0, \ldots, u_k$ a $v_0, \ldots, v_k$, zodpovedajúce počiatočným a koncovým pozíciám agentov, plánovať pre každé $0 \geq i \geq k$ cestu z u_i do v_i prostredníctvom ľubovoľného algoritmu hľadania ciest.

Toto naivné riešenie neberie do úvahy polohy a cesty ostatných agentov. Nie je teda vylúčená situácia, v ktorej sa dvaja agenti budú nachádzať na tom istom vrchole grafu G v tom istom čase. Takýto stret agentov v niektorých prípadoch nie je problémom. V počítačových hrách môže viesť ku neestetickému prechodu počítačom riadených postáv cez seba, avšak môže aj nepriaznivo ovplyvniť mechaniku hry. V prípade, že agenti sú v skutočnosti roboty, ktoré nasledujú naplánované cesty, môže spôsobiť až zrážku a ich poškodenie.

Surynek definuje v [2] dva pohybové problémy, ktoré sa dajú interpretovať ako základné varianty problému multi-agentného hľadania ciest: *Problém pohybu kameňa po grafe* a *problém plánovania ciest pre viaceré roboty.*

Definícia 1. *Problém pohybu kameňa po grafe*
Nech $G = (V, E)$ je neorientovaný graf. Ďalej nech je $P = \{\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_\mu\}$,

kde $\mu < |V|$ množina kameňov. Graf modeluje prostredie, v ktorom sa kamene pohybujú. Počiatočná konfigurácia *kameňov je definovaná prostým zobrazením $S_P^0 : P \longrightarrow V$ (teda, $S_P^0(p) \neq S_P^0(q)$ pre každé $p, q \in P$ kde $p \neq q$).* Cieľová konfigurácia *kameňov je definovaná ďalším prostým zobrazením $S_P^+ : P \longrightarrow V$ (teda, $S_P^+(p) \neq S_P^+(q)$ pre každé $p, q \in P$, kde $p \neq q$). Problém* pohybu kameňov po grafe *je úloha spočívajúca v nájdení čísla ξ a postupnosti $\mathcal{S}_P = [S_P^0, S_P^1, \ldots, S_P^\xi]$, kde $S_P^k : P \longrightarrow V$ je pre každé $k = 1, 2, \ldots, \xi$ prosté zobrazenie. Postupnosť $\mathcal{S}_P$ naviac musí spĺňať nasledovné podmienky:*

(i) $S_P^\xi = S_P^+$, *teda všetky kamene sa dostanú do svojich cieľových vrcholov*

(ii) Pre každé $p \in P$ a $k = 1, 2, \ldots, \xi - 1$ platí, že buď $S_P^k(p) = S_P^{k+1}(p)$ alebo $\{S_P^k(p), S_P^{k+1}(p)\} \in E$. To znamená, že medzi každými dvoma časovými krokmi môže kameň buď ostať vo vrchole alebo sa presunúť do susedného vrcholu.

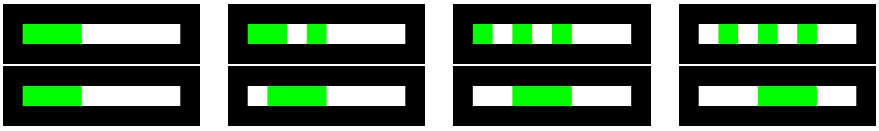
(iii) Ak $S_P^k(p) \neq S_P^{k+1}(p)$ (a teda sa kameň p medzi časovými krokmi k a $k+1$ pohol), potom $S_P^k(q) \neq S_P^{k+1}(p)$ $\forall q \in P$ také, že $p \neq q$ musí platiť pre každé $p \in P$ a $k = 1, 2, \ldots, \xi - 1$. Teda kameň sa môže presunúť iba do neobsadeného susedného vrcholu. Táto podmienka spolu s prostotou zobrazení tvoriacich $\mathcal{S}_P$ implikuje, že žiadne dva kamene nemôžu v jednom časovom kroku vstúpiť do toho istého vrcholu.

Formálne je inštancia problému pohybu kameňa po grafe štvorica $\Pi = (G, P, S_P^0, S_P^+)$. Niekedy sa riešenie problému Π dá označiť ako $\mathcal{S}_P(\Pi) = [S_P^0, S_P^1, \ldots, S_P^\xi]$.

Definícia 2. *Problém plánovania ciest pre viaceré roboty*
Nech $G = (V, E)$ je opäť neorientovaný graf. Ďalej majme namiesto množiny kameňov množinu robotov $R = \{\overline{r}_1, \overline{r}_2, \ldots, \overline{r}_v\}$, kde $v < |V|$. Podobne, graf modeluje prostredie, v ktorom sa roboty pohybujú. Počiatočná konfigurácia *robotov je definovaná prostým zobrazením $S_R^0 : R \longrightarrow V$ (teda, $S_R^0(r) \neq S_R^0(s)$ pre každé $r, s \in R$ kde $r \neq s$).* Cieľová konfigurácia *robotov je definovaná ďalším prostým zobrazením $S_R^+ : R \longrightarrow V$ (teda, $S_R^+(r) \neq S_R^+(s)$ pre každé $r, s \in R$, kde $r \neq s$). Problém* plánovania ciest pre viaceré roboty *je úloha spočívajúca v nájdení čísla ζ a postupnosti $\mathcal{S}_R = [S_R^0, S_R^1, \ldots, S_R^\zeta]$, kde $S_R^k : R \longrightarrow V$ je pre každé $k = 1, 2, \ldots, \zeta$ prosté zobrazenie. Postupnosť $\mathcal{S}_R$ naviac musí spĺňať nasledovné podmienky:*

(i) $S_R^\zeta = S_R^+$, *teda všetky roboty sa dostanú do svojich cieľových vrcholov*

(ii) Pre každé $r \in R$ a $k = 1, 2, \ldots, \zeta - 1$ platí, že buď $S_R^k(r) = S_R^{k+1}(r)$ alebo $\{S_R^k(r), S_R^{k+1}(r)\} \in E$. To znamená, že medzi každými dvoma časovými krokmi môže robot buď ostať vo vrchole alebo sa presunúť do susedného vrcholu.



Obr. 2.1: Kontrast medzi pohybom agentov (označených zelenou farbou) podľa pravidiel z definície 1, na prvom riadku, a podľa pravidiel z definície 2, na druhom riadku

(iii) *Ak* $S_R^k(r) \neq S_R^{k+1}(r)$ *(a teda sa robot* r *medzi časovými krokmi* k *a* $k+1$ *pohol) a* $S_R^k(s) \neq S_R^{k+1}(r)\ \forall s \in R$ *také, že* $s \neq r$ *(a teda v čase* k *nie je cieľový vrchol obsadený žiadnym iným robotom* s*), tak pohyb* r *v čase* k *nazývame* povolený *(teda sa robot* r *presúva do neobsadeného susedného vrcholu - je* vedúcim *robotom). Ak* $S_R^k(r) \neq S_R^{k+1}(r)$ *a existuje* $s \in R$ *také, že* $s \neq r \wedge S_R^k(s) = S_R^{k+1}(r) \wedge S_R^k(s) \neq S_R^{k+1}(s)$ *(a teda sa robot* r *presúva do vrcholu, z ktorého robot* s *odchádza) a pohyb robota* s *v čase* k *je povolený, potom aj pohyb* r *v čase* k *je povolený. Všetky pohyby robotov v každom čase* musia byť povolené. *Analogicky, táto podmienka spolu s prostotou zobrazení tvoriacich* $\mathcal{S}_R$ *implikuje, že žiadne dva roboty nemôžu v jednom časovom kroku vstúpiť do toho istého vrcholu.*

Formálne je inštancia problému plánovania ciest pre viaceré roboty štvorica $\Sigma = (G, R, S_R^0, S_R^+)$. *Niekedy sa riešenie problému* Σ *dá označiť ako* $\mathcal{S}_R(\Sigma) = [S_R^0, S_R^1, \ldots, S_R^\zeta]$.

V kontexte problému pohybu kameňov po grafe sú kamene agenti. V probléme plánovania ciest pre viaceré roboty sú agenti roboty. Rozdiel medzi týmito dvoma problémami je iba v pravidlách pre presun agentov medzi vrcholmi, ilustrovaný na obrázku 2.1. V prípade pohybu kameňov po grafe (ilustrovanom na prvom riadku obrázku) sa agenti môžu posúvať iba do vrcholu, ktorý bol v predošlej časovej jednotke prázdny. Nemôžu teda vytvárať súvislé prúdy agentov. Toto obmedzenie je v probléme plánovania ciest pre viacerých robotov uvoľnené, takže agenti sa môžu posúvať ako jedna jednotka, bez medzier medzi sebou, ako vidieť na druhom riadku obrázku.

Ako Surynek ďalej dokazuje v [2], riešenie inštancie Π problému pohybu kameňov po grafe je zároveň aj riešenie inštancie Σ problému plánovania ciest viacerých robotov nad tým istým grafom, s množinou robotov rovnakou ako množinou kameňov a s rovnakými počiatočnými a cieľovými konfiguráciami.

Cesty naplánované pre jednotlivých agentov v rámci multi-agentného hľadania ciest bývajú dlhšie, ako najkratšie cesty do ich cieľa, keďže sa agenti od svojej najkratšej cesty musia často odkloniť alebo zastaviť, aby sa vyhli zrážke [3].

Silver v [4] definuje tri typy multi-agentného hľadania ciest. Pri *kooperatívnom* hľadaní ciest vie každý agent o ostatných agentoch a o ich plánovaných

cestách. Pri *nekooperatívnom* hľadaní ciest agenti o plánoch ostatných agentov nevedia a musia ich pohyby predpovedať. Napokon, v *antagonistickom* hľadaní ciest sa agenti snažia dosiahnuť svoj cieľ a pritom ostatným agentom v jeho dosiahnutí zabrániť.

2.1.1 Algoritmy multi-agentného hľadania ciest

Spôsoby, akými sa problém multi-agentného hľadania ciest dá riešiť sa dajú rozdeliť na dva základné prístupy. V *centralizovanom* prístupe sa spoločne plánujú cesty všetkých agentov, kým v *decentralizovanom* sa každý agent považuje za autonómnu jednotku, ktorej cesta sa plánuje samostatne, pričom sa do úvahy berú pohyby ostatných autonómnych agentov na mape [5]. V nasledujúcich odsekoch popíšeme vybrané algoritmy používané na multi-agentné hľadanie ciest.

2.1.1.1 Centralizované algoritmy

Jedným z typov centralizovaných algoritmov multi-agentného hľadania ciest sú *algoritmy založené na pravidlách*. Každý z týchto algoritmov definuje množinu operácií, ktoré sa dajú vykonať nad grafom, v ktorom sú agenti. Pomocou opakovanej aplikácie týchto operácií potom agentom nachádza cesty.

Typickým príkladom algoritmu založeného na pravidlách je algoritmus *Push and Swap*, popísaný Lunom a Bekrisom v [6]. Jeho základné operácie sú podľa [7]:

- *Push*, ktorá agent, nad ktorým je zavolaná posúva po najkratšej ceste ku jeho cieľu, pričom z jeho cesty odsúva agentov, ktoré ho blokujú. Takto odsunúť však môže len agentov, ktorí zatiaľ nie sú na svojom cieľovom vrchole. Ak sa agent ďalej nemôže posunúť bližšie ku cieľu, je nutné aplikovať operáciu swap.

- *Swap*, ktorá vymení pozíciu agenta a s agentom b, ktorý susedí s jeho aktuálnou pozíciou a leží na najkratšej ceste ku cieľu. Pri tom môže byť každý agent, ktorý sa v grafe nachádza, posunutý. Tieto posunutia však musia byť pred dokončením operácie Swap zvrátené tak, aby jediní agenti, ktorých pozície ostanú zmenené, boli a a b.

Algoritmus Push and Swap bol jeho autormi prezentovaný ako úplný pre všetky grafy, ktoré obsahujú aspoň 2 vrcholy neobsadené agentom. V [8] však De Wilde a kol. poukázali na 4 nedostatky, ktoré spôsobujú, že to tak nie je a navrhli rozšírenie Push and Swap nazvané *Push and Rotate*, ktoré ich napravuje.

Ďalší algoritmus založený na pravidlách je *Bibox*, popísaný Surynkom v [9]. Pre svoj beh vyžaduje 2-súvislé grafy, ktoré dekomponuje na hlavný cyklus

a uchá. V jeho prípade je základnou operáciou rotácia, pomocou ktorej dokáže roboty nasúvať do úch a presúvať ich tak na ľubovoľné miesta v hlavnom cykle.

Na odlišnom prístupe, *prehľadávaní* namiesto pravidiel, je založený algoritmus *Increasing Cost Tree Search* (ICTS) od Sharona a kol. [10]. Využíva dve úrovne prehľadávania. Na vyššej úrovni prehľadáva tzv. strom zvyšujúcich sa cien (Increasing cost tree), ktorý má v každom vrchole uložený vektor, v ktorom je pre každého agenta uložená dĺžka jeho cesty do cieľa. Deti vrcholu obsahujú vektory, v ktorých je pre jedného z agentov dĺžka cesty zvýšená o 1. Cieľ je nájsť taký vrchol, v ktorom bude pre každého agenta cesta danej dĺžky existovať a súčet týchto dĺžok bude minimálny. Existencia ciest sa zisťuje pomocou prehľadávania nižšej úrovne.

Napokon, prístup navrhnutý Standleym [11], spočíva v algoritme, ktorý v inštancií problému hľadania ciest pre viacerých robotov deteguje hrozbu kolízií medzi robotmi. Na základe tejto detekcie dekomponuje danú inštanciu problému na niekoľko nezávislých podproblémov. Tie sa potom riešia prehľadávacím algoritmom, ako A* alebo ICTS. Na podproblémy, ktoré sa takto dekomponovať nedajú, používa globálne vyhľadávanie.

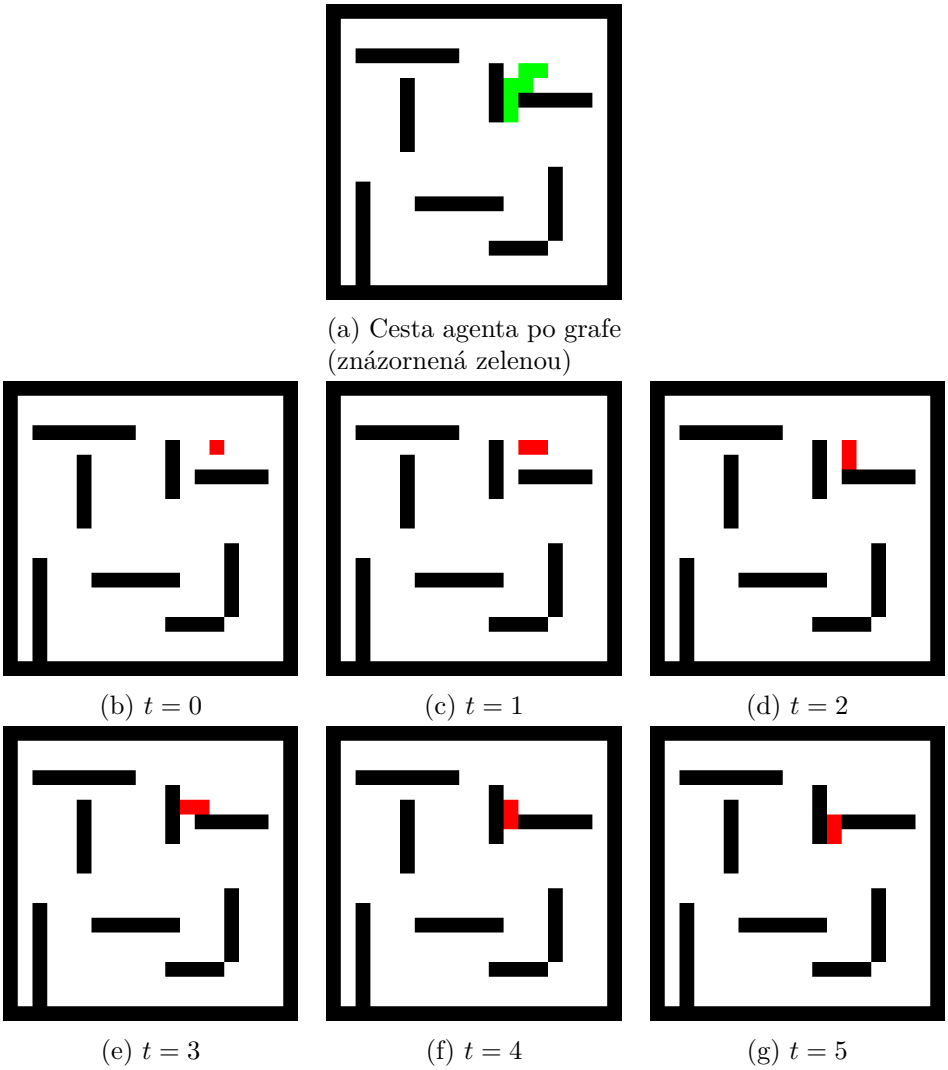
2.1.1.2 Decentralizované algoritmy

Štandardný spôsob, ako decentralizovane riešiť problém multi-agentného hľadania ciest je použiť algoritmus *A* s lokálnymi opravami* (LRA*). Ten je založený na fakte, že algoritmus A* sa dá upraviť tak, že v prípade, že hrozí kolízia s iným agentom, sa vykoná *lokálna oprava*. Pri nej je agent, s ktorým hrozí kolízia si uložený do mapy ako prekážka a cesta do cieľa sa naplánuje znova [12]. Takýto prístup funguje dobre na mapách s málo agentmi a veľkým podielom voľného miesta. Na mapách s väčšou hustotou agentov a úzkymi priechodmi však zlyháva a spôsobuje ich uviaznutie [4].

Silver [4] vyvinul alternatívu ku A* s lokálnymi opravami, algoritmus *Windowed Hierarchical Cooperative A** (WHCA*). Jeho rozlišujúcou charakteristikou je, že agenti nehľadajú cestu do cieľa iba na dvojrozmernej mriežke, ale v trojrozmernom časopriestore. Vždy, keď agent vyhľadá cestu, vrcholy v časopriestore (teda dvojice vrcholu v pôvodnom grafe a času), ktorými na nej prejde sú zaznamenané do tzv. *rezervačnej tabuľky*. Tieto vrcholy potom ostatní agenti považujú za prekážky.

Na obrázku 2.2 je vidieť spôsob, akým sa rezervácie pre cestu po grafe vytvárajú. Agent si pre každú časovú jednotku musí rezervovať vrchol, na ktorom sa v nej bude nachádzať a zároveň vrchol, na ktorom sa nachádzal v predošlej časovej jednotke. Táto dvojitá rezervácia zamedzuje výmenám agentov, teda situácií, kedy sa v časovej jednotke i nachádza agent a_1 na vrchole u a agent a_2 na vrchole v a v časovej jednotke $i + 1$ je a_1 na vrchole v a a_2 na u.

Pri tomto spôsobe rezervovania rieši algoritmus WHCA* problém pohybu kameňov po grafe z definície 1. V prípade, že je potrebné riešiť problém plá-



(a) Cesta agenta po grafe (znázornená zelenou)

(b) $t = 0$ (c) $t = 1$ (d) $t = 2$

(e) $t = 3$ (f) $t = 4$ (g) $t = 5$

Obr. 2.2: Cesta po grafe (a) a rezervácie, ktoré pre ňu museli v jednotlivých časoch vzniknúť (b-g)

novania ciest pre viacerých robotov (definícia 2), dá sa robiť rezervácia iba jedna a výmenám agentov brániť úpravou plánovacieho algoritmu.

Pre výkon algoritmu A* je veľmi dôležitý výber heuristiky. Algoritmus WHCA* je založený na rozšírení A* nazvanom *Hierarchical A** [13] a používa jednoduchú abstrakciu stavového priestoru, ktorá ignoruje časový rozmer a rezervačnú tabuľku. Ako heuristika časopriestorových vzdialeností bodov sa teda používajú skutočné vzdialenosti týchto bodov na pôvodnej mriežke. Tieto vzdialenosti sa vypočítavajú algoritmom *Reverse Resumable A** (RRA*). Ten vykonáva A* vyhľadávanie začínajúce v cieli agenta a smerujúce ku jeho polohe. Vyhľadávanie sa ukončuje v okamihu, keď je expandovaný vrchol, ktorého

heuristická vzdialenosť bola vyžiadaná. Vnútorný stav využitého A* algoritmu sa ukladá, takže ak je žiadaný vrchol v množine uzavretých vrcholov, je možné jeho vzdialenosť vrátiť okamžite, bez opätovného výpočtu.

V prípade, že by agenti plánovali celú svoju cestu do cieľa, algoritmus by bol veľmi citlivý na ich poradie (a teda prioritu, ktorá im tým implicitne bola priradená). Preto kooperatívne vyhľadávanie prebieha iba do určitej hĺbky, tzv. *vyhľadávacieho okna*. Aby však agenti smerovali správnym smerom, hĺbka abstraktného vyhľadávania (tj. RRA*) nie je obmedzená. Aj potom, čo agent dosiahne svoj cieľ, pokračuje v hľadaní ciest a vytváraní rezervácií. Vďaka tomu nenastáva situácia, kedy agent neumožní za ním nasledujúcim agentom dosiahnuť ich cieľ. Tie totiž v určitom momente majú možnosť zapísať svoju cestu do rezervačnej tabuľky pred agentom, ktorý už je v cieli a tým ho prinútia vyhnúť sa im.

2.2 Evakuácia

V prípade, že máme priestor, ktorý je rozdelený na ohrozené a bezpečné zóny, *evakuácia* spočíva v presune agentov z ohrozených do bezpečných zón. Ide o problém, ktorý je v odbore umelej inteligencie často riešený a ku ktorému existujú rôzne prístupy, napríklad modelovanie ako toky v sieťach [14, 15, 16].

2.2.1 Kapacitne obmedzené plánovanie ciest

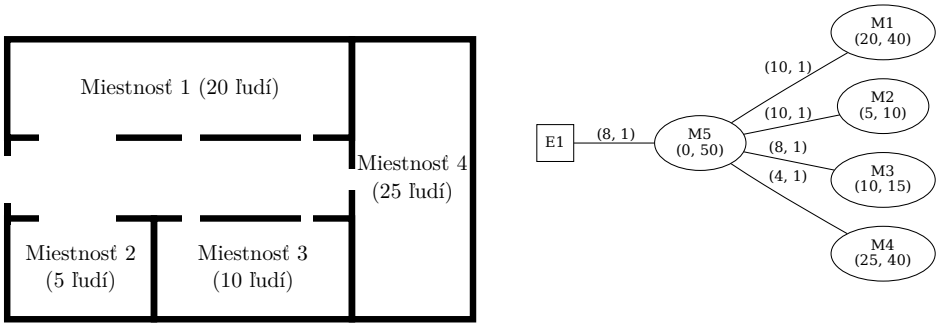
Jeden zo spôsobov, akým sa dá evakuácia modelovať predstavili Lu a kol. v [17] na použitie s algoritmami kapacitne obmedzeného plánovania ciest.

Priestor (budova, mesto), z ktorého prebieha evakuácia, je transformovaný na graf. Príklad takejto transformácie je na obrázku 2.3. Uzavreté podpriestory, ako napríklad miestnosti, chodby, ulice, budovy alebo námestia sú reprezentované vrcholmi. Každý vrchol má maximálnu kapacitu a počiatočnú obsadenosť. Zvláštne vrcholy sú východy. Nemajú maximálnu kapacitu a po ich dosiahnutí sa agenti považujú za evakuovaných a ďalej ich algoritmus neberie do úvahy.

Prechody medzi všetkými podpriestormi sú reprezentované hranami, ktoré sú označené svojou maximálnou kapacitou a dĺžkou prechodu.

Algoritmus *Single-Route Capacity Constrained Planner* (jedno-cestný kapacitou obmedzený plánovač) nájde cesty z každého vrcholu do každého z východov. Potom z každého vrcholu pošle všetkých agentov do východu, do ktorého z neho vedie najkratšia cesta. Pritom sú agenti rozdelení do menších skupín tak, aby v žiadnom vrchole nebolo a po žiadnej hrane nešlo naraz viac agentov, než sú ich kapacity.

Modifikácia tohto algoritmu nazvaná *Multiple-Route Capacity Constrained Planner* (viac-cestný kapacitou obmedzený plánovač) dokáže jednotlivým skupinám, do ktorých sú agenti rozdelení, priradiť rôzne cesty tak, aby neboli zdržované čakaním na uvoľnenie vrcholov a hrán.



(a) Priestor, z ktorého evakuácia prebieha

(b) Transformácia tohto priestoru na graf. Pri vrcholoch je uvedená aktuálna obsadenosť miestnosti a jej kapacita, pri hranách kapacita prechodu a časové trvanie evakuácie cez neho

Obr. 2.3: Transformácia ohrozeného priestoru na graf pre kapacitou obmedzené plánovače

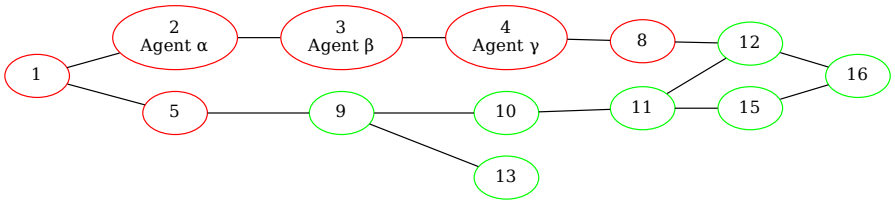
Medzi nevýhody tohto prístupu k modelovaniu evakuácie patrí fakt, že predpokladá, že všetci agenti majú tie isté schopnosti, takže napríklad prechod medzi dvoma miestnosťami trvá všetkým skupinám taký istý čas. Naviac ignoruje agentov, ktorí sa dostali do bezpečia. To je však nerealistické, pretože aj bezpečná zóna má len konečnú veľkosť a agenti sa musia ďalej presúvať, aby uvoľnili miesto pre tých, ktorí ďalej prichádzajú z ohrozenej zóny.

2.2.2 Multi-agentná evakuácia

Ďalší spôsob, ktorým sa problém evakuácie dá modelovať je inšpirovaný multi-agentným hľadaním ciest. Namiesto plánovania ciest pre skupiny agentov v abstrahovanom priestore, kde vrcholy reprezentujú miestnosti sa mapa priestoru premietne na mriežku, z ktorej sa následne vygeneruje graf.

Abstraktne sa *problém multi-agentnej evakuácie* dá popísať prostredníctvom neorientovaného grafu, $G = (V, E)$, v ktorom chceme evakuovať množinu agentov $A = (a_1, \ldots, a_k)$ z ohrozenej do bezpečnej zóny. Každý agent sa nachádza vo vrchole grafu G tak, aby v každom vrchole bol nanajvýš jeden agent. Konfigurácia agentov v grafe v čase t sa označuje ako $c_t : A \to V$. Agent sa môže do susedného vrcholu presunúť v prípade, že je prázdny. Takýto presun zaberie práve jednu časovú jednotku. Viacero agentov sa môže pohybovať naraz, v prípade, že medzi nimi nenastane kolízia. Tato formulovaný problém je veľmi podobný problému kooperatívneho hľadania ciest, riešenému algoritmom WHCA*.

(a) Mapa na ktorej prebieha evakuácia



(b) Graf, na ktorý je mapa prevedená

Obr. 2.4: Príklad prevodu mriežkovej mapy na problém multi-agentnej evakuácie

Definícia 3. *Multi-agentná evakuácia (MAE) je usporiadaná 5-ica*

$$\epsilon = (G = (V, E), A, c_0, D, S)$$

v ktorej G označuje prostredie, v ktorom sa agenti nachádzajú, $A = (a_1, \ldots, a_k)$ je množina agentov, $c_0 : A \to V$ je ich počiatočná konfigurácia a D a S množiny ohrozených (D) a bezpečných (S) vrcholov také, že $D \subset V$, $S \subset V$, $D \cup S = V$, $D \cap S = \emptyset$ a $|S| > k$.

Cieľom multi-agentnej evakuácie je presunúť všetkých agentov na bezpečné vrcholy, teda nájsť plán $\pi = (c_0, c_1, \ldots, c_m)$ taký, že $\forall a_i \in A, c_m(a_i) \in S$.

Pre každé $0 < i \leq k$ a $0 \leq t < m$ musia platiť tieto podmienky:

1. *$c_t(i) = c_{t+1}(i) \vee \{c_t(i), c_{t+1}(i)\} \in E$, teda agent musí v každej časovej jednotke ostať na tom istom vrchole alebo sa presunúť do jedného zo susedných vrcholov.*

2. *Ak $c_t(i) \neq c_{t+1}(i)$, potom $c_t(j) \neq c_{t+1}(i) \forall (0 < i \leq k)$, kde $i \neq k$. To znamená, že agenti sa môžu presúvať len do prázdnych vrcholov.*

Čas m, v ktorom posledný agent dosiahne bezpečnú zónu označíme ako *celkový čas evakuácie* (v anglickej literatúre označovaný ako *makespan*). Je nutné zdôrazniť, že agenti z grafu po vstupe do bezpečnej zóny nemiznú. Je teda

potrebné naplánovať aj ich pohyby v bezpečnej zóne tak, aby neprekážali evakuácií ostatných agentov.

Takto formulovaný problém multi-agentnej evakuácie je analogický ku problému pohybu kameňov po grafe, popísanému v definícií 1. Problém, v ktorom sú pravidlá presunu medzi vrcholmi uvoľnené tak, že agent môže vstúpiť aj do vrcholu, ktorý je v danom čase opúšťaný iným agentom (analogicky ku problému plánovania ciest pre viaceré roboty z definície 2) označíme ako *relaxovaný* problém multi-agentnej evakuácie.

Jednoduchý príklad mapy, nad ktorou sa dá multi-agentná evakuácia plánovať, je na obrázku 2.4a. Farby štvorcov sú popísané v legende pri obrázku 3.1. Horná polovica mapy je ohrozená a nachádzajú sa v nej traja agenti, ktorých bude nutné presunúť do spodnej polovice. Tento problém bude prevedený na graf z obrázku 2.4b a formálne sa dá zapísať takto:

$$V = \{1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 15, 16\}$$

$$E = \{\{1,2\},\{2,3\},\{3,4\},\{1,5\},\{4,8\},\{5,9\},\{8,12\},\{9,10\},$$
$$\{10,11\},\{11,12\},\{9,13\},\{11,15\},\{12,16\},\{15,16\}\}$$

$$A = \{\alpha, \beta, \gamma\}$$

$$c_0(\alpha) = 2, c_0(\beta) = 3, c_0(\gamma) = 4$$

$$D = \{1, 2, 3, 4, 5, 8\}$$

$$S = \{9, 10, 11, 12, 13, 15, 16\}$$

$$\epsilon = ((V, E), A, c_0, D, S)$$

Optimálne riešenie relaxovaného problému multi-agentnej evakuácie sa dá v polynomiálnom čase nájsť prostredníctvom techník používaných na riešenie problému *maximálneho toku v sieti* [16, 18]. Algoritmus, ktorý túto vlastnosť využíva popisujeme v kapitole 3.4.

Centralizovaný charakter tokových algoritmov je ale aj nevýhodou. Predpokladajú, že agenti budú centrálne vygenerovaný plán schopní bezchybne nasledovať. Tento predpoklad však v reálnych evakuačných situáciách neplatí [19]. V riešeniach relaxovaného problému multi-agentnej evakuácie naviac môžu byť veľkým zhlukom agentov naplánované nevykonateľné pohyby. Príkladom môže byť posunutie skupiny sto agentov o jeden vrchol v tom istom smere v jednej časovej jednotke. Skutočné davy takýchto pohybov nie sú schopné, čo ďalej znižuje znižuje realizmus a možnosť aplikácie týchto algoritmov.

Plán z týchto algoritmov nie je možné používať predtým, ako je kompletne vygenerovaný pre všetkých agentov. Rovnako nie je adaptovateľný na nečakané situácie, ako napríklad agentov, ktorí plán nenasledujú. Preto v nasledujúcej kapitole popisujeme algoritmus LC-MAE, založený na algoritme WHCA*, ktorý týmito slabinami netrpí.

Kapitola 3

Návrh algoritmu

Na riešenie problému multi-agentnej evakuácie sme vytvorili nový algoritmus, nazvaný *LC-MAE*. Táto kapitola začína v podkapitole 3.1 vysvetlením základných požiadaviek, ktoré LC-MAE musí spĺňať a potom popisuje jednotlivé aspekty jeho fungovania.

Aby bolo možné overiť výkon LC-MAE a kvalitu vytváraných plánov, implementovali sme algoritmus, ktorý na relaxovaný problém multi-agentnej evakuácie dáva optimálne riešenia. Tento algoritmus je popísaný v podkapitole 3.4. Dodatočným spracovaním riešení, ktoré tento algoritmus produkuje, popísaným v kapitole 3.5, sme schopní ich transformovať na riešenia samotného problému multi-agentnej evakuácie.

3.1 Zadanie problému

Táto podkapitola abstraktne popisuje vstup a výstup algoritmu a pravidlá, ktorými sa pri plánovaní evakuácie musí riadiť. Nepreberajú sa tu konkrétne formáty súborov alebo rozhrania, ktorými sa s algoritmom komunikuje. Pre informácie o nich viz. kapitolu 4.1.

3.1.1 Vstup

Na vstupe dostane algoritmus mapu, reprezentovanú mriežkou o rozmeroch $m \times n$. Každé políčko v mriežke môže byť voľné alebo sa na ňom môže nachádzať stena. Voľné políčka môžu byť označené ako nebezpečné, čo indikuje, že sa nachádzajú v zóne, z ktorej sa agenti majú snažiť dostať. Na mape môže byť viacero nesúvislých nebezpečných zón.

Každé voľné políčko, či už bezpečné alebo nebezpečné, môže byť počiatočnou polohou agenta. Ku každej počiatočnej polohe je špecifikovaný typ agenta, ktorý určuje jeho správanie a prípadné parametre, ktoré ho ďalej ovplyvňujú. Agenti sa môžu pohybovať iba medzi voľnými políčkami, na ktorých sa nena-

chádzajú iní agenti. Každý pohyb medzi dvoma políčkami trvá jednu časovú jednotku.

Existujú tieto typy agentov, zoradené vzostupne podľa ich racionality:

- *Panikáriaci agent*, ktorého správanie nemusí vykazovať žiadne známky racionality a pohybuje sa náhodne
- *Staticky cielený agent*, ktorého cieľ evakuácie je určený v čase vytvárania mapy a snaží sa ku nemu dostať nezávisle na možnej existencií alternatívnych (a prípadne lepších) prechodov do bezpečnej zóny
- *Prechod hľadajúci agent*, ktorý si pred samotným začiatkom svojej evakuácie heuristicky vyberie najlepší prechod do bezpečia a k tomu sa snaží dostať, až kým ním neprejde na bezpečné políčko
- *Opakovane zameriavajúci agent*, ktorý sa správa podobne ako predošlý typ, ale svoju snahu dostať sa ku prechodu hodnotí podľa nejakých kritérií a v určitom momente môže výber zopakovať a prípadne hľadať cestu ku inému prechodu

3.1.2 Výstup

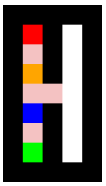
Výstup algoritmu je pre každého agenta zoznam m pozícií, na ktorých sa v každom okamihu evakuácie nachádza, označených indexmi 0 až $m - 1$. Tieto pozície musia spĺňať, že

- v čase 0 sa každý agent nachádza na svojej počiatočnej pozícií špecifikovanej mapou;
- v každom čase sa na jednom políčku nachádza iba jeden agent;
- agenti sa môžu v čase i nachádzať iba na voľnom políčku, ktoré je nad, pod, napravo alebo naľavo od políčka, na ktorom sa nachádzali v čase $i - 1$ a
- ak sa na políčku nachádzal v čase $i - 1$ agent a, v čase i sa na ňom môže nachádzať buď opäť agent a alebo musí ostať voľné.

Konkrétna hodnota m ostáva nešpecifikovaná. Malo by ísť o čas, v ktorom sa evakuuje čo najviac agentov, ideálne všetky. V prípade vzájomného uviaznutia agentov alebo prítomnosti vysokého počtu panikáriacich agentov však táto situácia nemusí nastať vôbec a preto algoritmus môže ukončiť plánovanie aj bez toho, aby sa všetci agenti nachádzali na bezpečnom políčku.

3.2 Vysvetlenie vizualizácií

V tejto práci sa na ilustráciu rôznych situácií budú často používať nákresy podobné tomu na obrázku 3.1. Významy jednotlivých farieb sú popísané v tabuľke.



Obr. 3.1: Príklad vizualizácie mapy evakuačného problému

Biela	Bezpečný voľný vrchol
Ružová	Ohrozený voľný vrchol
Čierna	Stena (v grafe vrchol neexistuje)
Červená	Panikáriaci agent
Oranžová	Staticky cielený agent
Modrá	Prechod hľadajúci agent
Zelená	Opakovane zameriavajúci agent

3.3 Algoritmus LC-MAE

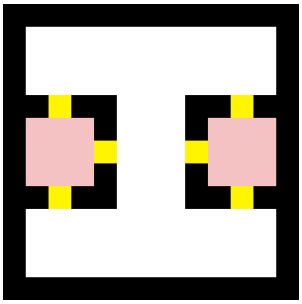
Algoritmus, ktorý rieši problém evakuácie, sme nazvali LC-MAE, čo je skratka z anglického *Local Cooperative Multi Agent Evacuation*. Ako je zjavné z názvu, problém evakuácie v tomto algoritme modelujeme ako problém kooperatívneho multi-agentného hľadania ciest (viz. definíciu v kapitole 2.1). V prípade, že implementácia algoritmu použije na hľadanie ciest dostatočne malé vyhľadávacie okno, ide o dobrú aproximáciu reality, v ktorej sú agenti schopní predpovedať akcie ostatných agentov na niekoľko časových jednotiek do budúcnosti.

Na úspešné naplánovanie evakuácie musí algoritmus vyriešiť nasledujúce čiastkové problémy:

1. *Výber evakuačného cieľa:* Každý agent sa musí na základe svojich znalostí mapy rozhodnúť, do ktorého vrcholu patriacemu do bezpečnej zóny sa vydá.

2. *Plánovanie cesty do cieľa:* Agent sa musí do do vybraného vrcholu cez ohrozenú zónu presunúť, bez kolízií s inými agentmi.

3. *Správanie sa v bezpečnej zóne:* Po dosiahnutí bezpečnej zóny agent nezmizne, takže sa v nej musí správať tak, aby neprekážal ostatným ohrozeným agentom v evakuácií.



Obr. 3.2: Ilustrácia hranice

3.3.1 Výber evakuačného cieľa

Dátovou štruktúrou používanou pri výbere cieľa je *hranica*, značená $F \subseteq S$. Hranica je množina bezpečných vrcholov, ktoré priamo susedia s ohrozenými vrcholmi a teda oddeľujú bezpečné zóny od nebezpečných. Ak sa má agent evakuovať z nebezpečnej zóny, musí prejsť jedným z vrcholov patriacich do hranice. Formálne, pre problém multi-agentnej evakuácie $\epsilon = (G = (V, E), A, c_0, D, S)$, platí:

$$F = \{u \in S | (\exists v \in D, \exists e \in E)(e = \{u, v\})\}$$

Na obrázku 3.2 sú žltou označené vrcholy patriace do množiny F. Táto množina je, rovnako ako celá mapa, známa všetkým agentom a je vygenerovaná algoritmom 1 pri inicializácií LC-MAE.

Algoritmus 1: Inicializácia množiny F

```
1 frontier (E)
2     F ← ∅
3     for e = {u, v} in E do
4         if u ∈ S ∧ v ∈ D then
5             F ← F ∪ {u}
6         if u ∈ D ∧ v ∈ S then
7             F ← F ∪ {v}
8     return F
```

Samotný výber cieľa evakuácie využíva modifikovaný A* algoritmus inšpirovaný algoritmom RRA*, ktorý popísal Silver [4]. Ako cieľ vyhľadávania je nastavená aktuálna poloha agenta a všetky vrcholy patriace do F sú vložené do počiatočnej množiny otvorených vrcholov. Hľadanie ciest sa potom nechá prebehnúť, pričom ako heuristika sa použije *Manhattanská vzdialenosť*.

Výsledkom výberu cieľa je taký vrchol z F, ktorý sa nachádza v najkratšej vzdialenosti od aktuálnej pozície agenta a zároveň je týmto agentom dosiahnuteľný. Takéto správanie je podobné mnohým skutočným situáciám, v ktorých

prebieha evakuácia a evakuované osoby priestor, v ktorom sa nachádzajú poznajú, takže vedia, kde je umiestnený najbližší východ [20]. Okrem samotného vrcholu je vrátená aj jeho vzdialenosť od agenta.

Opakovane zameriavajúci agenti sledujú v priebehu evakuácie prejdenú vzdialenosť. Keďže vzdialenosť, ktorú by v optimálnom prípade (bez nutnosti vyhýbať sa kolíziám s inými agentmi) prešli, aby sa dostali do cieľa je známa, môžu tieto dve hodnoty porovnať. Ak je prejdený počet krokov výrazne väčší ako očakávaný, môže to indikovať, že agent bol donútený veľmi sa vzdialiť od optimálnej trasy do vybraného cieľa a je možné, že sa priblížil ku inému prechodu do bezpečia. Preto proces výberu cieľa zopakuje a prípadne svoj cieľ zmení.

3.3.2 Rezervačná tabuľka

LC-MAE používa štruktúru rezervačnej tabuľky, ktorá sa podobá na rezervačnú tabuľku používanú v algoritme WHCA*, ale ku každej rezervácií ukladá aj jej prioritu. Prakticky teda ide o zobrazenie z množiny dvojíc $(v \in V, t \in \mathbb{N}_0)$ do množiny dvojíc $(a \in A, p \in \mathbb{N})$, kde a je agent, ktorý rezerváciu vytvoril a p je jej priorita. Agent sa v čase t môže presunúť iba na vrchol, na ktorý má pre daný čas rezerváciu. Vytvoriť si rezerváciu na vrchol v v čase t môže agent ak:

- na vrchol v v čase t neexistuje rezervácia, alebo
- na vrchol v v čase t existuje rezervácia s nižšou prioritou, alebo
- má vrchol v rezervovaný v čase $t - 1$.

Vďaka poslednej uvedenej podmienke môžu algoritmy hľadania ciest používané v LC-MAE vždy vykonať aspoň jednu akciu, státie na mieste, bez toho, aby sa poradie, v ktorom sa agentom plánujú cesty muselo dynamicky meniť. Taktiež sa tým vyhýbame situácií, v ktorej by boli agenti nútení vykonávať neplatné akcie, napríklad vstupovať na obsadené vrcholy.

3.3.3 Plánovanie cesty do cieľa evakuácie

Po tom, čo si agent vyberie cieľ evakuácie, začne doň plánovať svoju cestu pomocou algoritmu WHCA* s RRA* heuristikou. Pseudokód WHCA* je uvedený v algoritme 2.

Vzhľadom na to, že tento algoritmus neplánuje celú cestu do cieľa, ale len určitý počet nasledujúcich krokov, plánovanie ciest je prekladané s ich nasledovaním. Počet krokov, ktorý sa plánuje je určený veľkosťou vyhľadávacieho okna, označenou l. Tá sa dá interpretovať aj ako počet časových jednotiek do budúcnosti, pre ktoré vie agent vďaka rezervačnej tabuľke plánované cesty ostatných agentov. Ako už bolo spomenuté, aby bola simulácia realistická, táto konštanta by mala byť malá.

Algoritmus 2: Pseudokód algoritmu WHCA* používaného pri plánovaní cesty do bezpečia. Na vstupe je mu predaný časopriestorový graf G, rezervačná tabuľka *res*, ID agenta, ktorý hľadá cestu a, počiatočný vrchol v časopriestore S a počet časových jednotiek, na ktoré má plánovať w.

```
1  whca (G, res, a, S, goal, w)
2      open ← Heap()
3      open.insert(s, rra(S, goal))
4      closed ← ∅
5      S.g ← 0
6      while open.size() > 0 do
7          V ← pop(open)
8          closed ← closed ∪ {V}
9          if V.t = S.t + w then
10             return reconstruct_path(V)
11         foreach N, c in neighbors(G, res, a, V) do
12             g ← V.g + c
13             if n ∈ closed ∨ g > N.g then
14                 continue
15             N.g ← g
16             N.pred ← V
17             f = g+rra(N, goal)
18             if N ∉ open then
19                 open.insert(N, f)
20             else
21                 open.decrease_key(N, f)
22     return false
23 neighbors (G, res, a, V)
24     n ← ∅
25     foreach N in G.neighbors(V) do
26         if reservable(res, a, N) then
27             n ← n ∪ {(N, 1)}
28     c = 2
29     V.t ← V.t + 1
30     if reservable(res, a, V) then
31         c ← 1
32     n ← n ∪ {(V, c)}
33     return n
34 reservable (res, a, V)
35     r0 ← res[V]
36     V.t ← V.t + 1
37     r1 ← res[V]
38     p0 ← r0 = NULL ∨ r0.a = a ∨ r0.priority < 2
39     p1 ← r1 = NULL ∨ r1.a = a ∨ r1.priority < 2
40     return p0 ∧ p1
```

V každej časovej jednotke musia všetci agenti v náhodnom poradí oznámiť, na aký vrchol sa počas nej presunú. V reakcií na tento dopyt agent naplánuje svojich nasledujúcich l krokov, ak spĺňa akúkoľvek z týchto podmienok:

- *Robí svoj prvý krok:* Agent, ktorý ešte neurobil žiaden krok si musí zvoliť cieľ, do ktorého sa bude evakuovať a naplánovať prvých l krokov smerom k nemu.
- *Z naplánovanej cesty už vykonal polovicu krokov:* Keď agenti nasledujú svoju naplánovanú cestu, postupne sa stávajú menej kooperatívnymi. Algoritmus WHCA* je preto vhodné upraviť tak, aby agenti využili iba polovicu zo svojej naplánovanej cesty a potom plánovanie zopakovali [21].
- *Zmenil svoj cieľ:* Ak opakovane zameriavajúci agent zopakoval proces výberu cieľa evakuácie a našiel bližší cieľ, musí do neho aj naplánovať cestu.
- *Stratil rezerváciu na vrchol na svojej ceste:* V krajnej situácií sa môžu agenti rozhodnúť stáť na mieste a zrušiť tak rezerváciu iného agenta, ako vidieť v algoritme 2 na riadku 32. Pred vykonaním kroku preto musí agent skontrolovať cestu, ktorú má naplánovanú a ak na nejaký z vrcholov na nej stratil rezerváciu, proces plánovania sa musí zopakovať.

Keď má agent cestu naplánovanú, vytvorí v rezervačnej tabuľke rezervácie vrcholov, ktorými bude prechádzať. Rezervácie agentov nachádzajúcich sa v nebezpečnej zóne sú vytvárané s maximálnou prioritou.

Ak agent žiadnu z týchto podmienok nespĺňa, znamená to, že má naplánovaný aspoň jeden nasledujúci krok a tento krok nebude viesť ku kolízií. Ako odpoveď na dopyt ho teda vráti bez toho, aby musel plánovať.

3.3.4 Správanie sa v bezpečnej zóne

Pre úspech evakuácie je kritické, aby agenti nachádzajúci sa v bezpečnej zóne neprekážali ohrozeným agentom v evakuácií. Možné riešenie by bolo, keby sa agenti snažili dostať od vrcholov z množiny D tak ďaleko, ako to je možné. Zistiť však, či sa agent od vrcholov, ktoré sa môžu nachádzať v rôznych častiach mapy, vzďaľuje je pre decentralizovaný algoritmus problematické. Nútená snaha dostať sa do maximálnej vzdialenosti od ohrozenej zóny môže byť kontraproduktívna aj z hľadiska dĺžky ciest, ktoré agenti prejdú. Agent, ktorý sa sám nachádza v odľahlej časti mapy môže byť jediný, ktorý určitým východom z ohrozenej zóny prejde. Vďaka tomu sa môže zastaviť aj na prvom bezpečnom vrchole bez toho, aby inému agentovi prekážal.

Algoritmus, ktorý riadi správanie sa agentov v bezpečnej zóne je označený ako *surfovanie*, pretože sa snaží docieliť, aby sa agent nechal tlačiť vlnou agentov, ktorí ho nasledujú. K tomuto účelu sa využíva flexibilita algoritmu

Algoritmus 3: Algoritmus, ktorým sa vypočítava počet agentov nasledujúcich agenta a

```
1  previous-reserved (E, π, a, t)
2      let π = [c_0, c_1, ..., c_t]
3      V_a ← {c_{t−b}(a) | b = 0, ..., l/2}
4      for v ∈ V_a do
5          if reserved(v, t) then
6              r ← r + 1
7      return r
```

Algoritmus 4: Výpočet cien pre rôzne akcie, ktoré môže agent a v bezpečnej zóne vykonať. t špecifikuje čas, v ktorom sa akcia vykoná, kým t_c čas, v ktorom prebieha plánovanie.

```
1  neighbors (E, π, a, t, v, t_c)
2      let π = [c_0, c_1, ..., c_t]
3      A_p ← previous-reserved(E,π,a,t_c)
4      costs ← []
5      foreach u ∈ S | {v, u} ∈ E do
6          if reservable(u, t + 1)∧u ∈ S then
7              if u ∈ {c_t(a) | t = 0, 1, ..., t} then
8                  costs ← costs ∪{(u, 3)}
9              else
10                 costs ← costs ∪{(u, 2)}
11     b ← max(1, |A_p| − (t − t_c))
12     if reservable(v, t + 1) then
13         costs ← costs ∪{(u, 1 ∗ b)}
14     else
15         costs ← costs ∪{(u, 4 ∗ b)}
16     return costs
```

WHCA*. Rozširuje sa počet akcií, ktoré môže agent v každom kroku vykonať a ich cena sa vypočítava dynamicky.

V každej časovej jednotke môže agent v bezpečnej zóne vykonať nasledujúce akcie:

- Stáť na mieste
- Prejsť na nový bezpečný vrchol
- Prejsť na už navštívený bezpečný vrchol

Ceny jednotlivých akcií sa odvíjajú od počtu agentov, ktorí *nasledujú* plánujúceho agenta. Výpočet ich počtu je vidieť v algoritme 3 a spočíva v kontrole,

na koľko z $\frac{l}{2}$ predošlých pozícií agenta existuje v aktuálnej časovej jednotke rezervácia.

Konkrétne ceny sú určené algoritmom 4. V prípade, že agent nie je nasledovaný žiadnymi inými agentmi, je pre neho najlacnejšie ostať na mieste. So zvyšujúcim sa počtom nasledujúcich agentov sa zvyšujú ceny všetkých akcií okrem presunu na susedný rezervovateľný vrchol. Faktor, o ktorý sa zvyšujú je naviac na riadku 11 upravený podľa toho, ako ďaleko do budúcnosti sa akcia plánuje, keďže plánujúci agent nemá úplné informácie o budúcich krokoch iných agentov.

Prechod na vrchol, ktorý agent ešte nenavštívil je za všetkých okolností lacnejší, ako prechod na už raz navštívený vrchol. Vďaka tomu sa agenti viac rozptyľujú po bezpečnej zóne a nechávajú miesto pre novo prichádzajúcich agentov.

Konkrétne ceny jednotlivých akcií (uvedené v algoritme 4 na riadkoch 8, 10, 13 a 15) nemajú žiaden špeciálny význam. Slúžia iba na to, aby v algoritme A*, určili to, aké želané sú z hľadiska agenta jednotlivé akcie. Preto boli ich hodnoty zvolené ako malé prirodzené čísla, ktoré priamo odrážajú to, aká je akcia pre agenta dobrá v čase, keď ho nenasledujú žiadni iní agenti.

Mechanizmus plánovania akcií je veľmi podobný, ako pre agentov nachádzajúcich sa v nebezpečnej zóne. Plán akcií je vygenerovaný na l časových jednotiek dopredu a generuje sa, ak agent žiaden plán zatiaľ nemá, ak z existujúceho plánu už vykonal polovicu krokov alebo ak prišiel o rezerváciu.

Agenti v bezpečnej zóne oznamujú svoje akcie až potom, čo ich oznámia agenti z nebezpečných zón. Vďaka tomu majú ohrození agenti možnosť rezervovať svoje cesty ako prvé a tým donútiť surfujúcich agentov posunúť sa hlbšie do bezpečnej zóny. Ohrození agenti sú preferovaní aj vďaka tomu, že iba prvá polovica rezervácií vytvorených surfujúcimi agentmi má maximálnu prioritu. Rezervácie na druhú polovicu cesty majú zníženú prioritu a ohrození agenti ich teda vedia zrušiť a nahradiť svojimi.

3.3.5 Lokalita a decentralizácia algoritmu

Ako bolo popísané vyššie, jediný mechanizmus komunikácie medzi agentmi je rezervačná tabuľka. Táto globálna dátová štruktúra agentom poskytuje dva druhy informácií. Prvý je informácia o obsadenosti vrcholov v aktuálnej časovej jednotke. Druhý je prehľad o plánovaných pohyboch ostatných agentov na nanajvýš l časových jednotiek do budúcnosti. Mohlo by sa zdať, že najmä kvôli druhému typu poskytovaných informácií v skutočnosti LC-MAE nie je decentralizované a ani lokálne. Tak to však nie je.

Definícia decentralizovaného algoritmu sa netýka spôsobu, akým agenti komunikujú, ale toho, či sa nejakým spôsobom (napríklad globálnou optimalizáciou) plánujú cesty všetkým agentom spoločne. To sa v LC-MAE nedeje. Samotné plánovanie, v zmysle výberu akcií, ktoré agenti vykonajú, prebieha

na úrovni jednotlivých agentov a globálna je len rezervačná tabuľka a koordinácia času, v ktorom tieto individuálne plánovacie algoritmy prebiehajú.

Veľkosť vyhľadávacieho okna l naviac zásadne limituje mieru globálnej komunikácie, ktorá prebieha. Agenti vidia iba také budúce akcie iných agentov, ktoré sa odohrajú v okolí l časových a priestorových jednotiek od ich aktuálnej pozície. V prípade, že je hodnota l malá, takýto výhľad do budúcnosti napodobňuje správanie sa ľudí, ktorí dokážu obmedzene predvídať pohyby osôb, ktoré sa nachádzajú v ich bezprostrednej blízkosti.

V prípade, že je l nastavené v implementácií na veľmi vysokú hodnotu, v algoritme fakticky prebieha globálna komunikácia medzi agentmi. Na zvolenie takejto vysokej hodnoty však nie je dôvod. Vyhľadávacie okno nebolo do algoritmu WHCA* zavedené ako umelý mechanizmus slúžiaci na obmedzenie globálnej komunikácie, ale preto, že zlepšuje výsledky algoritmu znížením jeho citlivosti na poradie, v ktorom agenti cesty vyhľadávajú.

Evakuácia, ktorú teda LC-MAE simuluje je situácia, v ktorej sa agenti nachádzajú na mape, ktorá im je do istej miery známa, aspoň do tej miery, aby vedeli odhadnúť vzdialenosť k únikovému východu. Počas svojej evakuácie dokážu obmedzene predvídať pohyby iných agentov, ktorí sa nachádzajú v ich bezprostrednom okolí.

Konkrétna hodnota konštanty l by mala byť zvolená s ohľadom na prostredie a charakteristiky agentov, ktorí sa evakuujú. Pri plánovaní evakuácie v málo členitom prostredí, v ktorom sú agenti, ktorí dokážu vnímať svoje široké okolie sa dá nastaviť na vyššie hodnoty, ako napríklad 20 časových jednotiek. Naopak, v komplikovaných a stiesnených priestoroch, alebo ak majú simulovaní agenti problémy s vnímaním okolia je vhodné ju znížiť na 4-6 časových jednotiek. Vzhľadom na cieľ poskytovať primerane kvalitné plány pre veľké množstvo rozličných scenárov sme v implementácií popísanej v kapitole 4.2.1 na základe predbežných experimentov zvolili $n = 10$.

3.4 Tokový algoritmus

Problémy multi-agentného hľadania ciest sa dajú riešiť tým, že sa modelujú ako tok v sieti, ako to urobili Yu a LaValle v [18]. Aby bolo možné porovnávať riešenia, ktoré generuje algoritmus LC-MAE s riešeniami, ktoré taktiež simulujú evakuáciu na úrovni jednotlivých agentov, ale boli generované centralizovaným spôsobom, navrhli sme aj algoritmus, ktorý rieši problém relaxovanej multi-agentnej evakuácie pomocou hľadania tokov v sieti.

Toto riešenie je založené na konštrukcií časovo expandovaného grafu, v ktorom sa nachádza m kópií grafu G. V tejto sieti existuje tok o veľkosti $|A|$ práve vtedy, ak má problém *relaxovanej* multi-agentnej evakuácie na grafe riešenie v čase m.

Tvorba časovo expandovaného orientovaného grafu G_m z neorientovaného grafu $G = (V = \{v_1, \ldots, v_n\}, E)$ prebieha nasledovne:

1. Do grafu G_m sa pridajú vrcholy z a s, globálny zdroj a stok.

2. Pre každý vrchol v_j z G sa do G_m pridajú dva vrcholy i_j^0 a o_j^0 a hrana (i_j^0, o_j^0). Pre každé j také, že sa vo v_j nachádza agent je do G_m pridaná hrana (z, i_j^0).

3. Do G_m sa pre každé $t \in 1, \ldots, m-1$ a každý vrchol v_j pridajú dva vrcholy, i_j^t a o_j^t spojené hranou. Pre každú hranu $e = \{v_x, v_y\} \in E$ sa do G_m pridajú hrany $(o_x^{t-1}, i_y^t), (o_y^{t-1}, i_x^t)$.

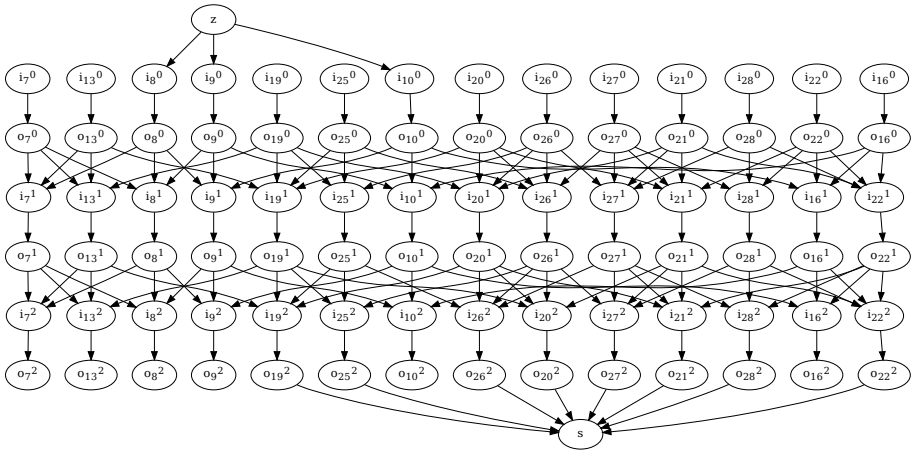
4. Pre o_j^{m-1} také, že v_j je v pôvodnom grafe bezpečný vrchol je pridaná do G_m hrana (o_j^{m-1}, s).

5. Všetkým hranám v G_m sa nastaví kapacita 1.

Algoritmus 5: Algoritmus na nájdenie minimálneho času evakuácie

```
1  minimum-makespan (E)
2      f ← 0
3      m ← |A|
4      w_hi ← 0
5      while f ≠ |A| do
6          E_e ← expand(E, m)
7          f ← maximum-flow(G_e)
8          if f ≠ |A| then
9              w_hi ← m
10             m ← m * 3/2
11     while true do
12         m_new ← w_hi + ⌊(m − w_hi)/2⌋
13         E_e ← expand(E, m_new)
14         f ←maximum-flow(G_e)
15         if f = |A| then
16             m ← m_new
17             if m = w_hi + 1 then
18                 break
19         else
20             w_hi = m_new
21             if m_new = m − 1 then
22                 break
23     return m
```

Výsledok behu tohto algoritmu na grafe z obrázku 2.4 je vidieť na obrázku 3.3. Do troch vrcholov s agentmi idú hrany zo zdroja a z bezpečných vrcholov idú v tretej časovej jednotke hrany do stoku. Z výstupných vrcholov (označených o) sa dá prejsť len do vrcholov patriacich ku nasledujúcej časovej jednotke.



Obr. 3.3: Graf evakuačného problému z obrázku 2.4 expandovaný na 3 časové jednotky

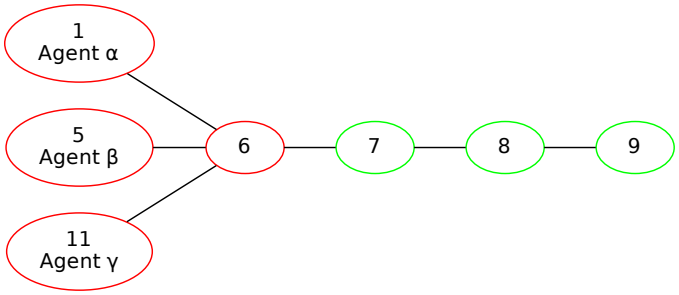
Keďže miera expanzie grafu priamo určuje dĺžku evakuácie v riešení, je nutné toto optimálne riešenie nájsť. To sa docieli tým, že algoritmus začne s $m = |A|$, teda graf sa expanduje na rovnaký počet časových jednotiek, ako je počet agentov. Ak sa pri takomto m riešenie nenájde, m sa zvýši o polovicu. To sa opakuje pokiaľ riešenie nie je nájdené. Optimálna dĺžka evakuácie, teda najnižšie m pri ktorom v grafe existuje tok o veľkosti $|A|$ sa potom hľadá upraveným algoritmom binárneho vyhľadávania, ako je popísaný v algoritme 5 na riadkoch 11 až 22.

Riešenie relaxovanej multi-agentnej evakuácie sa dá dodatočne spracovať na riešenie multi-agentnej evakuácie. Nazvime algoritmus, ktorý toto dodatočné spracovanie vykoná ako *POST-MAE*.

3.5 POST-MAE

Algoritmus POST-MAE je založený na vyššie popísanom tokovom algoritme na riešenie problému evakuácie, ale pridáva k nemu krok dodatočného spracovania, v ktorom sa z riešenia problému relaxovanej multi-agentnej evakuácie vytvorí riešenie problému multi-agentnej evakuácie.

Toto dodatočné spracovanie je založené na rezervačnej tabuľke podobnej tej použitej v algoritme LC-MAE. Pri dodatočnom spracovaní sa z (relaxovanej) cesty každého agenta vytvorí fronta. Pre každú časovú jednotku t sa potom iteruje cez fronty všetkých agentov a overuje sa, či na vrchol, ktorý je vo fronte prvý, neexistuje v čase t rezervácia. Ak existuje, agent ostane stáť na mieste. Ak rezervácia neexistuje, vrchol sa z fronty odstráni a vytvorí sa naň rezervácia pre časy t a $t + 1$.



Obr. 3.4: Počiatočná konfigurácia agentov, v ktorej môže pri triviálnej transformácií nastať vzájomné zablokovanie agentov

V mnohých prípadoch tento jednoduchý algoritmus zlyháva. Ide o situácie, v ktorých potrebujú cez ten istý vrchol prejsť agenti z dvoch rôznych smerov a z dôvodu blokovania sa naň dostanú v poradí odlišnom od poradia, ktoré im určil tokový algoritmus. V prípade, že jeden z agentov sa následne dostane na svoj cieľový vrchol, môže druhému zabrániť dostať sa do cieľa a tým znemožniť získanie výsledku (nastane vzájomné blokovanie, *deadlock*).

Príklad situácie, v ktorej môže nastať vzájomné zablokovanie agentov je vyobrazený na obrázku 3.4. Tokový algoritmus pre agentov naplánuje tieto cesty:

α	1	1	1	6	7
β	5	5	6	7	8
γ	11	6	7	8	9

Pri dodatočnom spracovaní majú však prednosť pohyby, ktoré chce urobiť agent α, takže sa vygenerujú tieto cesty:

α	1	1	1	6	7	7	7	7	...
β	5	5	5	5	5	6	6	6	...
γ	11	6	7	8	9	9	9	9	...

Po piatich pohyboch sa žiaden z agentov nemôže posunúť. α a γ preto, že na ich cestách neleží viac vrcholov, β preto, že ju blokuje α. Tento problém rieši algoritmus POST-MAE *výmenou agentov*.

Pri výmene agentov ostanú obidvaja agenti v danej časovej jednotke na svojom mieste, ale vymenia si svoje fronty budúcich pohybov. Plán pre agenta α bol ostať na vrchole 7, kým plán pre agenta β bol presunúť sa na vrchol 7 a potom na vrchol 8. Potom, čo sa vymenia, sa bude agent α snažiť dostať na

vrchol 8, čo sa mu aj podarí, keďže ten je prázdny a agent β sa len posunie na uvoľnený vrchol 7.

Algoritmus, ktorý implementuje takéto dodatočné spracovanie riešenia relaxovanej multi-agentnej evakuácie pomocou zdržiavania agentov a výmen front pohybov je popísaný vo funkcií **postprocess-round** v algoritme 6.

Algoritmus 6: Algoritmus, ktorý vykoná jedno kolo dodatočného spracovania ciest z tokového algoritmu

```
1  postprocess-round (π)
2      foreach p_i ∈ π do
3          q[i] ← Queue(p_i)
4          P_p[i] ← []
5      r ← Dictionary()
6      t ← 0
7      while ∃q[i], q[i].len > 0 do
8          for a ∈ {0, 1, ..., π.len}) do
9              m ← false
10             if q[a].len = 0 then
11                 P_p[a][t] ← P_p[a][t − 1]
12                 m ← true
13             else if r[(q[a].head, t)] ∈ {NULL, a} then
14                 P_p[a][t] ← q[a].pop()
15                 m ← true
16             if m then
17                 r[(P_p[a][t], t)] ← a
18                 r[(P_p[a][t], t + 1)] ← a
19             else
20                 handle-blocked(q, P_p, r, a, t)
21         t ← t + 1
22     return P_p
23 handle-blocked (q, P_p, r, a, t)
24     a′ ← r[(q[a].head, t)]
25     d ← q[a′].len > 0 ∧ q[a′].head = P_p[a][t − 1] ∧ q[a] = P_p[a′].last
26     if d then
27         swap(q[a], q[a′])
28     else if q[a′].len = 0 ∧ q[a].head = P_p[a′].last then
29         q[a].pop()
30         q[a′] ← q[a]
31         q[a] ← Queue(P_p[a′].last)
32     P_p[a][t] = P_p[a][t − 1]
33     r[(P_p[a][t], t)] ← a
34     r[(P_p[a][t], t + 1)] ← a
```

Veta 1. *Pre každé riešenie problému relaxovanej multi-agentnej evakuácie vráti funkcia* ***postprocess-round*** *z algoritmu 6 korektné riešenie problému multi-agentnej evakuácie.*

Dôkaz. Podľa definície 2.2.2, aby bolo riešenie korektné, musí každý agent prejsť zo svojej počiatočnej pozície na nejaký bezpečný vrchol tak, aby pri tom spĺňal pravidlá pohybu.

Dokážme najskôr, že agenti skončia na bezpečných vrcholoch. Predpokladajme, že nejaký z agentov neskončí v riešení vrátenom **postprocess-round** na bezpečnom vrchole. Funkcia však v žiadnom momente nemení cesty vygenerované tokovým algoritmom, ale iba vymieňa medzi agentmi ich jednotlivé úseky. Aby teda agent skončil na ohrozenom vrchole, jedna z ciest, ktoré boli na vstupe do algoritmu musela končiť na ohrozenom vrchole, čo je spor s tým, že na vstupe dostal algoritmus riešenie problému relaxovanej multi-agentnej evakuácie.

Ďalej dokážeme, že vygenerované cesty dodržujú pravidlá pohybu. Prvé pravidlo je, že ak agent nezostáva na mieste, môže sa pohnúť iba do susedných vrcholov. V prípade, že agent nasleduje svoju cestu, je to zaručené tým, že na vstupe je riešenie relaxovaného problému MAE. Výmeny ciest prebiehajú na riadkoch 27 a 29-31. Po obidvoch z nich majú obaja agenti vo fronte pohybov na prvom mieste pohyb na vrchol, na ktorom sa už nachádzajú a za ním nasleduje korektná cesta, ktorá začína v tomto vrchole.

Podľa druhého pravidla sa agenti môžu presúvať iba do prázdnych vrcholov. Presúvanie sa agentov je implementované na riadkoch 13-15. Aby sa agent mohol presunúť, vrchol nesmie byť rezervovaný alebo ho musí mať rezervovaný sám. Keďže na všetkých miestach, kde sa vytvára rezervácia (riadky 17-18 a 33-34) sa rezervácia vytvára na časy t aj $t+1$, rezervácia vyprší až keď je agent, ktorý ju vytvoril, na inom vrchole. V čase, keď doň iný agent vstúpil už teda vrchol musel byť jednu časovú jednotku prázdny.

Posledné, čo je nutné dokázať je, že algoritmus vždy skončí. Všetky cesty, ktoré dostáva na vstupe sú konečné a sú z nich vytvorené fronty pohybov jednotlivých agentov. Keď sú všetky fronty prázdne, algoritmus skončí. Existujú dve situácie v ktorých agent v jednej iterácií slučky začínajúcej na riadku 8 neskráti svoju frontu:

- Agenti si chcú vymeniť miesta, tj. agent α stojaci na vrchole i sa chce pohnúť na vrchol j, na ktorom stojí agent β, ktorý sa chce presunúť na vrchol i. Namiesto vykonania pohybov si navzájom vymenia fronty pohybov, ale ani jedna z nich nie je skrátená. Pre obidvoch agentov je však na prvom mieste v ich novej fronte pohybov presun na vrchol, na ktorom aktuálne stoja ($q[\alpha].\text{head} = i, q[\beta].\text{head} = j$), takže v ďalšej časovej jednotke určite svoje fronty skrátia.

- Agent α je zablokovaný agentom β, ktorý ešte vo fronte má pohyby. V takom prípade agent α jednoducho čaká, kým β neopustí vrchol, alebo nenastane taký reťazec výmen front a pohybov, pri ktorom sa fronta agenta β vyprázdni. V takom prípade vykoná výmenu front popísanú na riadkoch 28-31 a pritom skráti svoju frontu.

V každej časovej jednotke sa teda buď niektorým agentom priamo skrátia fronty, alebo sa vykoná operácia, ktorá vedie k tomu, že sa im fronty skrátia v ďalšej časovej jednotke. Keďže všetky fronty sú konečné, algoritmus eventuálne skončí a vráti výsledok. □

Tento algoritmus dokáže vygenerovať korektné riešenia problému multi-agentnej evakuácie, značne však predlžuje trvanie evakuácie, keďže agenti po výmene čakajú do ďalšej časovej jednotky, kým vykonajú svoje pohyby. Posun n agentov o jeden vrchol, ktorý v relaxovanej multi-agentnej evakuácií trvá 1 časovú jednotku sa po korektnom spracovaní môže predĺžiť až na n časových jednotiek. V prípade, že sa títo agenti posúvajú pomocou výmeny pohybov, môže trvať až $2n$. n časových jednotiek trvá, než sa vykonajú výmeny a n jednotiek trvá vykonanie samotných pohybov.

Na vyriešenie problému predlžovania evakuácie využívame fakt, že v spracovanom pláne sú pôvodne naplánované cesty agentov medzi nimi vymenené tak, aby sa navzájom neblokovali. Na zistenie, ktoré zdržania sú nepotrebné sa dá opäť použiť funkcia **postprocess-round**.

Algoritmus POST-MAE preto z plánov odstráni všetky vyčkávania, tj. pobyty na vrcholoch trvajúce viac, ako jednu časovú jednotku (z plánu prechádzajúceho po vrcholoch $1, 2, 2, 3, 3, 2, 2$ by napríklad vznikol plán $1, 2, 3, 2$). Na takto upravených plánoch potom opäť spustí algoritmus dodatočného spracovania. Tento proces opakuje, až kým sa cesty agentov nestabilizujú, tj. neprestanú sa po odstránení vyčkávania a opätovnom spracovaní meniť.

Takýto algoritmus vráti pre každé riešenie relaxovaného problému multi-agentnej evakuácie riešenie problému multi-agentnej evakuácie. Po konečnom počte iterácií totiž vymení medzi agentmi úseky ciest tak, aby poradie, v ktorom prichádzajú do cieľov súhlasilo s poradím, v akom sú ich pohyby vyberané z front, ktoré teda nikdy nemusia byť vymieňané a tým zdržovať pohyby agentov.

Kapitola 4

Implementácia

Popísané algoritmy boli implementované v jazyku Python, v rámci programu nazvaného *evacsim*. Ide o program spúšťaný cez príkazový riadok, ktorý má z pohľadu užívateľa štyri základné funkcie:

- *Plánovanie evakuácie:* Pre danú mapu a scenár sa dá vygenerovať plán evakuácie pomocou algoritmu LC-MAE, tokového algoritmu alebo algoritmu POST-MAE.
- *Kontrola plánov:* Kontrola, či vygenerovaný plán spĺňa pravidlá pohybu agentov a nasleduje zadaný scenár.
- *Vizualizácia:* Zobrazenie grafického rozhrania, v ktorom užívateľ môže upravovať mapy a scenáre a vizualizovať pohyb agentov podľa vygenerovaného plánu.
- *Overovanie výkonu:* Užívateľ môže zadefinovať sadu máp a scenárov, pre ktoré sa majú (voliteľne paralelne) naplánovať evakuácie. Nad vygenerovanými plánmi sa vypočítajú rôzne štatistiky.

4.1 Vstupy a výstupy

Na načítavanie vstupu a ukladanie výstupu používa *evacsim* celkovo štyri formáty súborov. Formát pre mapy, scenáre, vygenerované plány a pre sady máp a scenárov používané na overovanie výkonu.

Mapový súbor obsahuje informácie o rozmere mapy, na ktorej sa evakuácia odohráva a o polohách stien a voľných vrcholov. Tento mapový súbor je napísaný vo formáte používanom na 2D mriežkové mapy v benchmarkoch od Moving AI Lab Nathana Sturtevanta [22], keďže predtým, než sme navrhli pre účely tejto práce realistické mapy, používali sme práve mapy z tohto zdroja, konkrétne z hry Warcraft 3. Príklad textového zápisu je na obrázku 4.1.

```
type octile
height 9
width 5
map
@@@@@
@.@.@
@.@.@
@.@.@
@...@
@.@.@
@.@.@
@.@.@
@@@@@
```

Obr. 4.1: Textový zápis mapy z obrázku 3.1

```
6 11 16 21 22 26 31 36
6p 16s18 26f 36r
```

Obr. 4.2: Textový zápis scenára z obrázku 3.1

Súbor začína hlavičkou o štyroch riadkoch. Prvý je `type octile`, keďže na mapách AI Lab sa medzi štvorcami v mriežke dalo presúvať aj diagonálne, hoci *evacsim* tieto pohyby nepodporuje. Ďalšie dva riadky, `height H` a `width W` udávajú výšku a šírku mapy. Hlavičku ukončuje riadok, na ktorom je uvedené iba slovo `map`. Samotná mapa je zložená z H riadkov, na každom z nich je W znakov. Znak `@` označuje stenu a znak `.` voľné miesto.

Súbor so scenárom sa skladá z dvoch riadkov. Na prvom sú medzerami oddelené ID ohrozených vrcholov. Na druhom sú medzerou oddelené popisy agentov. Bežný popis agenta sa skladá z ID vrcholu, na ktorom sa agent nachádza a jedného znaku, určujúceho jeho typ (`p` pre panikáriacich agentov, `f` pre prechod hľadajúcich agentov a `r` pre opakovane zameriavajúcich agentov). Staticky cielení agenti majú za svojim typom (`s`) uvedené ešte jedno číslo, ktoré udáva ich cieľ.

ID vrcholu je vypočítané ako $r * W + s$, kde r je riadok, v ktorom sa vrchol v mriežke nachádza, W je počet stĺpcov v mriežke a s je stĺpec, v ktorom sa nachádza. Koordináty začínajú nulou, nachádzajúcou sa v ľavom hornom rohu mriežky.

Načítavanie máp a scenárov je implementované v module `level`, ktorý pre ostatné moduly poskytuje triedy *Level* a *Scenario*, umožňujúce pristupovať ku načítaným dátam. Mapy sú reprezentované ako neorientovaný graf implementovaný dátovou štruktúrou z knižnice networkx [23].

Výstupom algoritmov je textový súbor, ktorý má na každom riadku zoznam ID vrcholov, na ktorých sa postupne nachádzal agent, ktorý bol v scenári uvedený na pozícií zhodnej s číslom riadku. Pozície prvého agenta sú uvedené

na prvom riadku, druhého na druhom a pod. Pre všetkých agentov je uvedený rovnaký počet vrcholov, každý stĺpec tohto súboru sa teda dá použiť na zistenie polôh všetkých agentov v danom čase.

4.2 Plánovanie

```
evacsim plan --algorithm=lcmae office.map office.scen
```

Obr. 4.3: Príklad spustenia plánovania nad mapou a scenárom office

Užívateľ, ktorý chce naplánovať evakuáciu dá programu cesty ku súborom s mapou a scenárom a zvolí si algoritmus. Po načítaní jednotlivých súborov sa inštancia triedy *Level* predá funkcií `plan_evacuation` modulu zvoleného algoritmu.

4.2.1 Implementácia algoritmu LC-MAE

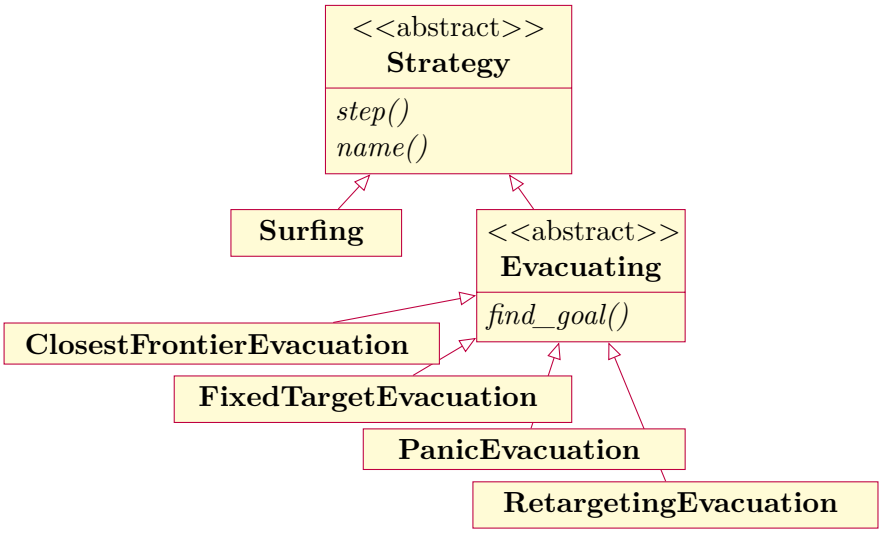
Po predaní inštancie triedy *Level* do funkcie `plan_evacuation` vytvorí koordinačná časť algoritmu rezervačnú tabuľku a inicializuje množiny ohrozených a neohrozených agentov. Kým sú v grafe ohrození agenti, zisťuje, aké sú nasledujúce kroky všetkých agentov. Poradie, v akom postupuje je pre každé kolo zisťovania náhodné, ale najskôr sú spracované všetci ohrození agenti a až potom agenti v bezpečných zónach. Po každom kole sa aktualizujú množiny ohrozených a bezpečných agentov.

Základom implementácie algoritmu LC-MAE je trieda *Agent*. Pri inicializácií jej je predaná referencia na graf, na ktorom evakuácia prebieha, na rezervačnú tabuľku a na evakuačnú stratégiu.

Konkrétne správanie sa agenta záleží na tom, či sa nachádza v ohrozenej alebo bezpečnej zóne a zároveň na jeho type, ktorý je špecifikovaný v scenári. Takto variabilné správanie je vhodné implementovať pomocou objektového návrhového vzoru *Strategy*, popísaného Gammom a kol. v [24]. Diagram jeho implementácie v algoritme LC-MAE je vidieť na obrázku 4.4. Využitie tohto vzoru umožňuje jednoduché pridanie ďalších typov agentov, ktoré by napríklad umožňovali pokročilejšiu simuláciu rôznych situácií.

Abstraktná trieda *Strategy* má dve metódy, ktoré musia jej potomkovia implementovať. Už spomenutú metódu `step`, ktorá určuje nasledujúci krok agenta a metódu `name`, ktorej návratová hodnota sa používa pri výpisoch pre účely ladenia.

Keďže tri zo štyroch evakuačných stratégií sa medzi sebou líšia iba spôsobom, akým vyberajú cieľ, je algoritmus WHCA*, používaný na navigáciu do cieľa, implementovaný v abstraktnej triede *Evacuating*, ktorej potomkovia musia povinne implementovať iba metódu `find_goal`. To, nezabraňuje implementácií úplne odlišných evakuačných stratégií, ako napríklad *PanicEvacuation*, ktorá namiesto informovanej evakuácie s agentom pohybuje náhodne.



Obr. 4.4: Diagram tried implementujúcich správanie agentov

Medzi stratégiou *Surfing* a dedičom stratégie *Evacuating* predaným pri inicializácií sa prepína sám agent, podľa typu vrcholu, na ktorom sa nachádza.

Napriek tomu, že sa to deje veľmi zriedkavo, pri lokálnej multi-agentnej evakuácií sa nedá vylúčiť, že sa agenti navzájom zablokujú. Pre tento účel sleduje riadiaca časť algoritmu, či nejakí agenti vykonali pohyb. V prípade, že v grafe sa ešte nachádza ohrozený agent, ale viac ako 15 časových jednotiek sa žiaden agent nepohol, algoritmus sa ukončí.

4.2.2 Implementácia tokového algoritmu a POST-MAE

Vďaka architektúre podporných modulov a požiadavkám kladeným na tokový algoritmus mohla byť jeho implementácia, nachádzajúca sa v module `expansion`, pomerne jednoduchá. Po transformácií vstupu na graf, ktorú prevedie modul `level` je graf expandovaný podľa pravidiel popísaných v podkapitole 3.4. Vzhľadom na centralizované plánovanie, ktoré prebieha v tokovom evakuačnom algoritme sa rôzne typy agentov neberú do úvahy.

Na nájdenie maximálneho toku v grafe je použitá funkcia `maximum_flow` z knižnice `networkx`. Tá je založená na algoritme *preflow-push*, označovanom aj ako *push-relabel*. Z výsledku tvoreného hodnotami tokov pre všetky hrany, sú rekonštruované cesty jednotlivých agentov.

V závislosti na tom, či užívateľ chcel výsledky iba tokového algoritmu, alebo algoritmu POST-MAE (tj. tokového algoritmu, ktorého výsledky sú dodatočne spracované, viz. kapitola 3.5), môžu byť výsledky tokového algoritmu dodatočne spracované. Využíva sa pritom tá istá implementácia grafu s rezervačnou tabuľkou, ako v algoritme LC-MAE. Logika, podľa ktorej sa rozhoduje

o tom, či agent vykoná pohyb, odloží ho, alebo či sa jeho fronta pohybov vymení s iným agentom, je implementovaná v triede *FlowAgent*.

4.3 Grafické rozhranie

```
evacsim gui office.map office.scen office.sol
```

Obr. 4.5: Príklad spustenia grafického rozhrania pripraveného vizualizovať riešenie naplánované pre mapu a scenár office

Počas vývoja algoritmu je možnosť rýchlo si zobraziť generované plány a tým intuitívne zhodnotiť ich kvalitu veľmi užitočná. Preto bolo jednoduché grafické rozhranie a editor na scenáre a mapy jedna z prvých častí tejto práce, ktorú sme implementovali. Založené je na knižnici `arcade` [25], používanej na jednoduchý vývoj 2D hier. Vidieť ho je možné na už uvedených obrázkoch, napríklad 3.1.

Keď je rozhranie spustené s mapových súborom a súborom scenáru, umožňuje ich upravovať. Pomocou klávesnice užívateľ volí typ políčka, ktoré vytvára (stena alebo rôzne typy agentov), ľavým tlačidlom myši kreslí a pravým maže. Po dokončení úprav je možné zmeny uložiť do nového súboru. Ak sa pri spustení zadá aj cesta k výstupu jedného z algoritmov na plánovanie evakuácie, je možné pomocou medzerníka spustiť vizualizáciu.

4.4 Kontrola plánov

```
evacsim check office.map office.scen office.sol
```

Obr. 4.6: Príklad kontroly riešenia naplánovaného pre mapu a scenár office

Pre zjednodušenie vývoja sme implementovali sadu základných kontrol výstupu plánovacích algoritmov. Po spustení modulu `check` sa overí, či riešenie spĺňa nasledujúce podmienky:

1. Cesty všetkých agentov sú rovnako dlhé, tj. pre každého agenta je v súbore m pozícií.
2. Na každej pozícií sa v jednej časovej jednotke vyskytuje len jeden agent.
3. Agenti vstupujú iba do prázdnych vrcholov.
4. V čase 0 sa všetci agenti nachádzajú na pozícií, ktorá je im zadaná scenárom.
5. Agenti sa pohybujú iba medzi susednými vrcholmi.

4.5 Overovanie výkonu

```
evacsim benchmark bench_suite/benchmarks.txt
```

Obr. 4.7: Príklad spustenia sady benchmarkov zo zložky `bench_suite`

Aby bolo možné sledovať zmeny vo výkone algoritmu počas jeho vývoja, implementovali sme modul na *benchmarking*. Ten načíta zo súboru zoznam máp a scenárov, nad ktorými sa má spustiť plánovanie evakuácie. Pre každý plán potom vypočíta nasledujúce metriky:

- Počet agentov rôznych typov na mape
- Čas, ktorý algoritmu generovanie plánu zabralo
- Celkový čas evakuácie
- Časy, v ktorých boli evakuované poslední agenti jednotlivých typov
- Časy, v ktorých bolo evakuovaných 25, 50, 75, 95 a 99% agentov
- Priemerné časy evakuácie pre všetky typy agentov nachádzajúce sa na mape
- Celkový čas evakuácie podľa evakuačného plánu generovaného tokovým algoritmom
- Počet agentov, ktoré sa evakuovať nepodarilo

Keďže plánovacie algoritmy nie sú paralelizované, plánujú sa evakuácie pre niekoľko scenárov naraz, tak, aby boli využité všetky jadrá procesora. Tokový algoritmus sa vzhľadom na jeho časovú náročnosť spúšťa iba v prípade, že je modulu predaný pri spustení na príkazovom riadku parameter `-e`.

Pre účely priebežného overovania výkonu a funkčnosti sme počas vývoja využívali fakultný systém GitLab a jeho podporu *priebežnej integrácie* (CI). Pomocou konfigurácie, ktorú vidno na obrázku 4.8 sme docielili, že po každej zmene v Git repozitári tejto práce sa kvalita plánov generovaných algoritmom a jeho výkon overili. Výsledky boli uložené a boli z nich vygenerované *grafy priebehu evakuácie*, ktoré je možné vidieť v nasledujúcej kapitole.

```
image: python

before_script:
  - cd proto
  - pip install .
  - mkdir -p solutions

benchmark:
  script:
    - evacsim benchmark --p solutions -d solutions bench_suite/benchmarks_socs.txt
  artifacts:
    paths:
      - proto/bench_solutions
```

Obr. 4.8: `.gitlab_ci.yml` používaný na overenie výkonu algoritmu po každej zmene v Git repozitári práce

Kapitola 5

Experimentálne výsledky

V tejto kapitole porovnávame kvalitu plánov a výkon algoritmu LC-MAE s plánmi, ktoré sú generované tokovým algoritmom a algoritmom POST-MAE. Taktiež uvádzame výsledky experimentov, v ktorých sú na mapách rozmiestnení agenti rôznych typov s rozličnou mierou informovanosti. Farby použité na vizualizáciách v tejto kapitole sú vysvetlené v kapitole 3.2.

5.1 Testovacie scenáre

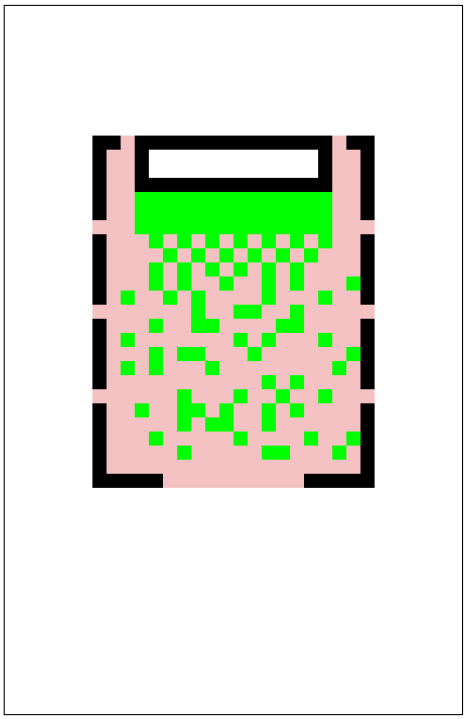
Pri overovaní algoritmu LC-MAE sme využili štyri mapy, zobrazené na obrázku 5.1.

- *Koncert* (Obrázok 5.1a) je inšpirovaný koncertnou sálou, v ktorej je nerovnomerne rozmiestnených 118 agentov. Má veľký hlavný únikový východ a po stranách sa nachádzajú núdzové východy, ktoré ale vedú na stiesnené priestranstvo.

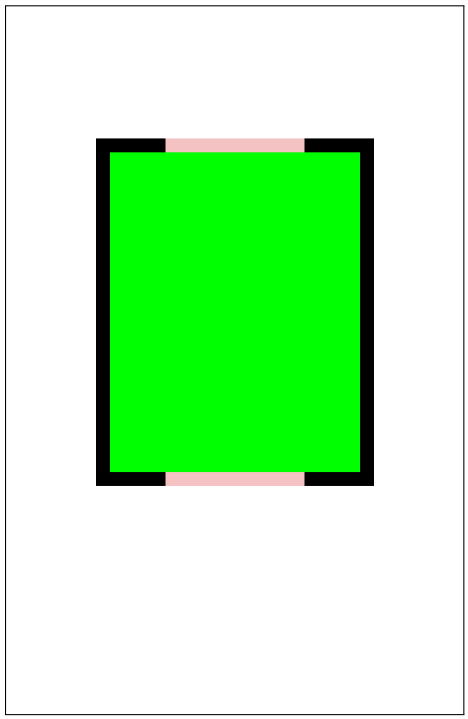
- *Kancelárie* (Obrázok 5.1c) ukazuje chodbu v kancelárskej budove, ktorá má z oboch strán malé kancelárie, s dvoma agentmi v každej, celkovo 80 agentov. Na obidvoch stranách na začiatku prázdnej chodby sa nachádzajú únikové východy.

- *Nákupné centrum* (Obrázok 5.1d) je inšpirované rozvrhnutím obchodov v nákupnom centre. Toto rozvrhnutie je komplikované a budova má množstvo malých núdzových východov. Na mape sa nachádza 299 agentov, tak v obchodoch, ako na chodbách centra.

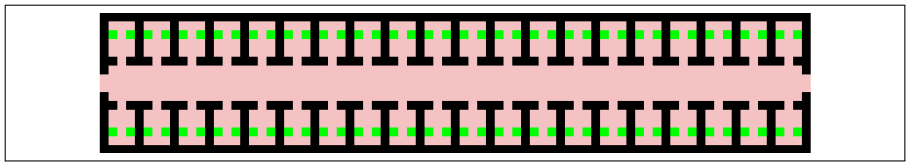
- *Blokovanie* (Obrázok 5.1b) je nerealistická mapa, na ktorej sa nachádza miestnosť s dvoma núdzovými východmi. Na každom políčku v miestnosti sa nachádza agent, celkovo ich je 414.



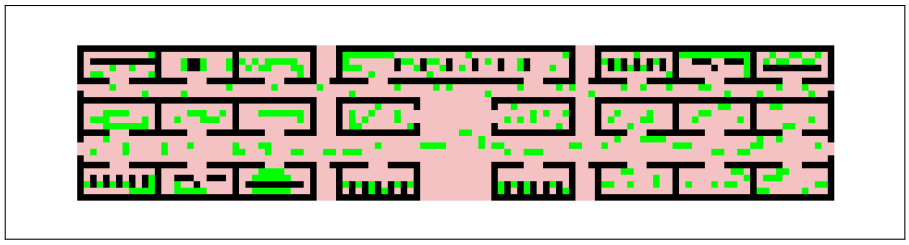
(a) Koncert



(b) Blokovanie



(c) Kancelárie



(d) Nákupné centrum

Obr. 5.1: Mapy s realistickými evakuačnými situáciami

5.2 Porovnanie LC-MAE, POST-MAE a tokového algoritmu

Pre všetky štyri scenáre sme vygenerovali evakuačné plány pomocou algoritmu LC-MAE, pomocou tokového algoritmu a pomocou algoritmu POST-MAE. Výsledky tohto porovnania sa nachádzajú v tabuľke 5.1 spolu s informáciou,

Scenár	LCMAE	Tok	POSTMAE	Predĺženie
Koncert	90	17	33	2,73×
Kancelárie	94	47	62	1,52×
Nákupné centrum	129	36	75	1,72×
Blokovanie	146	23	69	2,12×

Tabuľka 5.1: Porovnanie dĺžok plánov medzi LCMAE, tokovým algoritmom a algoritmom POSTMAE

koľkonásobne dlhšie sú plány algoritmu LC-MAE oproti algoritmu POST-MAE. Celkový čas evakuácie v evakuačných plánoch, ktoré generuje LC-MAE je 1,89 až 5,91-krát dlhší, ako celkový čas evakuácie v plánoch generovaných tokovým algoritmom a 1,52 až 2,73-krát dlhší, ako v plánoch generovaných algoritmom POST-MAE. Plány, ktoré vzniknú spracovaním riešenia relaxovanej multi-agentnej evakuácie sú zas 1,3 až 3-krát dlhšie, ako nespracované plány.

Algoritmus POST-MAE generuje plány, ktoré sú založené na optimálnom riešení relaxovaného problému multi-agentnej evakuácie vyprodukovanom tokovým algoritmom. Vzhľadom na to, že plány generované LC-MAE sú dlhšie len o malý faktor, môžeme tvrdiť, že majú kvalitu blízku optimu. Naviac sú realistickejšie, keďže medzi agentmi vyžadujú iba lokálnu komunikáciu a dávajú im možnosť reagovať na prípadné nečakané situácie, ako napríklad zmeny v prostredí alebo nepredvídateľných agentov nachádzajúcich sa na mape.

Implementácia algoritmu LC-MAE generuje plány výrazne rýchlejšie, ako implementácie tokového algoritmu a algoritmu POST-MAE, ako sa dá vidieť v tabuľke 5.2.

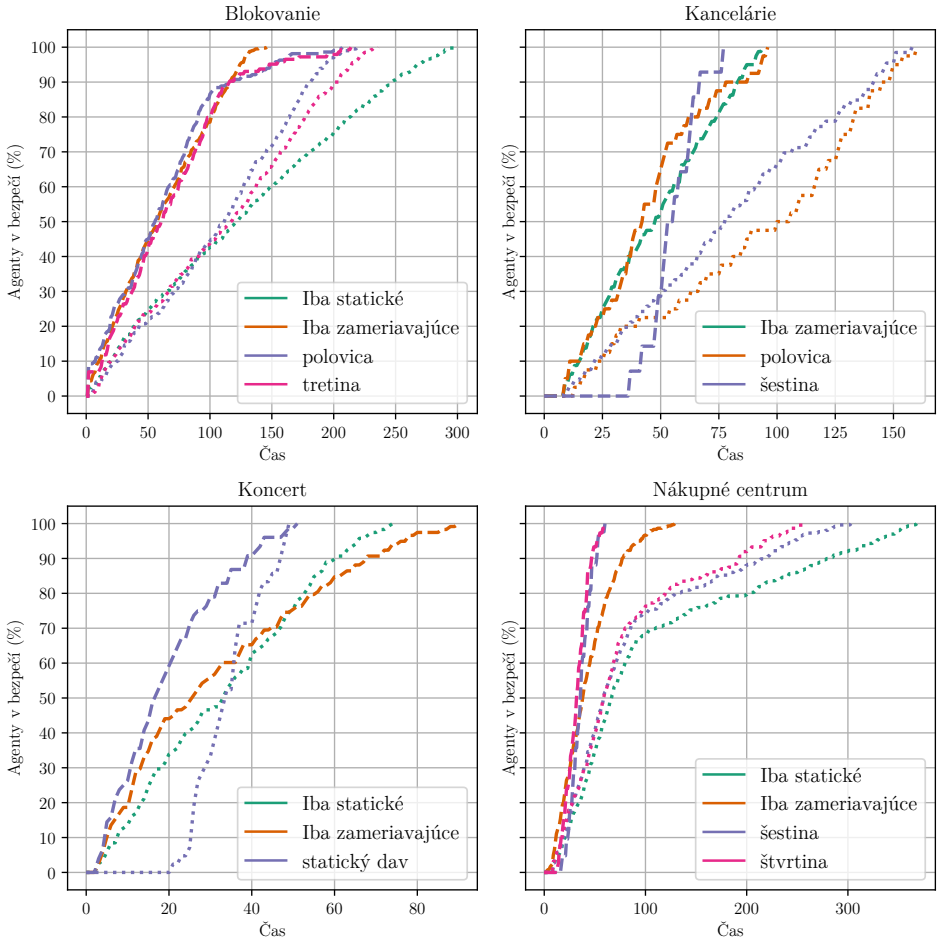
Scenár	LCMAE	Tok	POSTMAE	Zrýchlenie
Koncert	7	52	62	8.86×
Kancelárie	4	57	61	15.25×
Nákupné centrum	20	379	403	20.15×
Blokovanie	27	220	243	9×

Tabuľka 5.2: Trvanie plánovania v sekundách pre algoritmus LCMAE a pre tokový algoritmus

5.3 Experimenty s rôznymi typmi agentov

Pre lepšie porozumenie procesu evakuácie, v ktorom je spolu zmiešaných viacero typov agentov s rôznou mierou informovanosti sme vykonali sériu experimentov, založených na upravených scenároch z kapitoly 5.1.

Z každého základného scenáru sme vytvorili niekoľko verzií a v každej z nich sme časti agentov zmenili typ na statický. Takto upravení statickí agenti



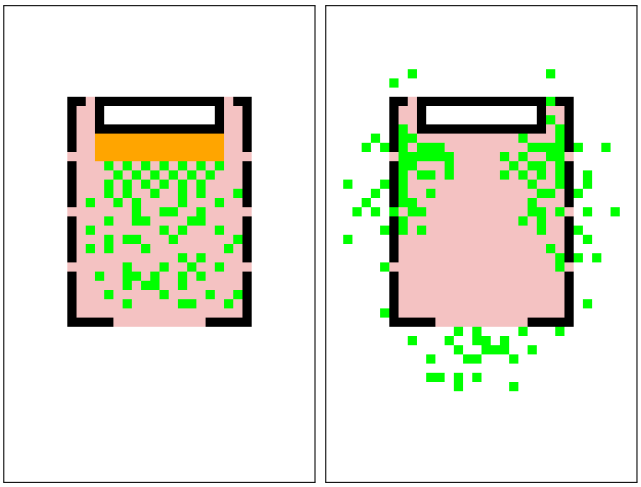
Obr. 5.2: Grafy priebehu evakuácie pre jednotlivé mapy a scenáre na nich. Čiarkované čiary označujú opakovane zameriavajúcich agentov, bodkované označujú staticky cielených agentov.

sa budú snažiť uniknúť najväčším otvorom medzi ohrozenou a bezpečnou zónou (ktorý sa dá interpretovať ako hlavný vchod a východ) a budú ignorovať ostatné východy z ohrozenej zóny. Preferovanie známej cesty pred optimálnejšími evakuačnými cestami je správanie, ktoré je pri evakuácií reálnych osôb typické [20]. Z tohto dôvodu je takýchto agentov možné považovať za menej informované.

Grafy priebehu evakuácie pre všetky mapy a ich jednotlivé scenáre sa nachádzajú na obrázku 5.2. Na ose X je počet časových jednotiek od začiatku evakuácie a na ose Y je vyznačené, koľko percent agentov daného typu už bolo evakuovaných do bezpečia. Jednotlivé scenáre sú odlíšené farbami. Bodkované čiary označujú priebeh evakuácie pre staticky cielených agentov na

mape, čiarkované pre opakovane zameriavajúcich agentov.

5.3.1 Koncert



(a) Scenár *statický dav* mapy Koncert

(b) Upchatie bočných východov v scenári *iba zameriavajúce* mapy Koncert

Obr. 5.3: Situácie z mapy Koncert

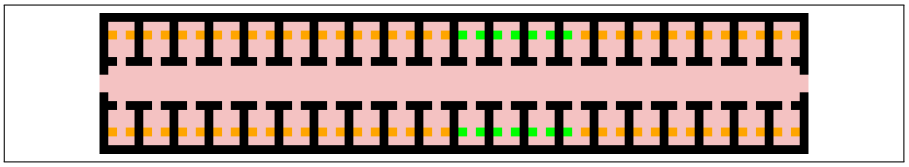
Scenár, v ktorých bol rozdiel medzi dĺžkami evakuácií naplánovaných algoritmom POST-MAE a algoritmom LC-MAE najväčší, bol scenár koncert. Naša hypotéza bola, že spomalenie spôsobuje to, že postranné núdzové východy sú veľmi úzke a za nimi je len málo bezpečného priestoru, takže sa rýchlo upchajú, podobne, ako to je vidieť na obrázku 5.3b.

Na overenie tejto hypotézy sme vytvorili dva ďalšie scenáre, nazvané *statický dav* a *iba statické*. V scenári *statický dav* je statických 42 agentov stojacich pred priestorom pripomínajúcim pódium v hornej časti mapy, viz. obrázok 5.3a. Unikať sa pokúšajú veľkým východom v spodnej časti mapy. V scenári *iba statické* týmto východom unikajú všetci agenti.

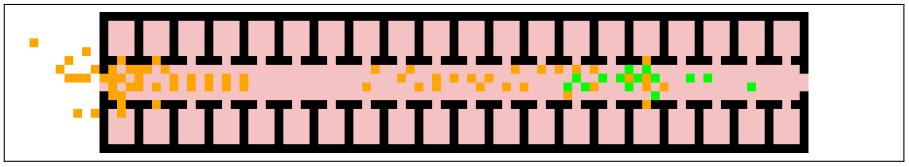
Hypotéza sa nám potvrdila. Zatiaľ, čo opakovane zameriavajúcim agentom (pôvodný scenár je označený ako *iba zameriavajúce*) trvá evakuácia 90 časových jednotiek, ak sú všetci agenti statickí, evakuácia trvá iba 74 časových jednotiek. Najrýchlejšie, v 51 časových jednotkách, prebehne evakuácia scenáru *statický dav*. Informovaní agenti využijú postranné východy, ktoré sa však neupchajú a cestou ku nim uvoľnia stred mapy pre dav unikajúci hlavným východom.

	Opakovane zameriavajúce		Staticky cielené	
Scenár	**Počet**	**Trvanie evakuácie**	**Počet**	**Trvanie evakuácie**
Iba zameriavajúce	118	90	0	
Statický dav	76	51	42	49
Iba statické	0		118	74

Tabuľka 5.3: Počty agentov a trvanie evakuácie pre scenáre na mape Koncert



(a) Počiatočný stav



(b) Stav počas zrážky opakovane zameriavajúcich a statických agentov v chodbe

Obr. 5.4: Scenár *šestina* mapy Kancelárie

5.3.2 Kancelárie

Pre mapu kancelárií sme vytvorili dva modifikované scenáre, *polovica* a *šestina*, ktorých názvy určujú, aký zlomok agentov bol ponechaný ako opakovane zameriavajúci. Ostatní agenti boli zmenení na staticky cieliacich a ako únikový východ im bol určený východ na ľavej strane mapy.

V scenári *šestina* sme 14 opakovane zameriavajúcich agentov umiestnili do siedmich stĺpcov nachádzajúcich sa napravo od stredu mapy (viz. obrázok 5.4a). Očakávali sme, že staticky cielení agenti unikajúci smerom doľava a informovaní agenti unikajúci ku východu napravo si budú v úzkej chodbe kancelárií prekážať. Táto hypotéza sa potvrdila. Evakuácia 14 informovaných agentov trvala 77 časových jednotiek, čo je len o 17 časových jednotiek menej, ako evakuácia 80 agentov v pôvodnom scenári, kde boli všetci agenti opakovane zameriavajúci a teda informovaní. Staticky cielení agenti boli postihnutí týmto stretom v chodbe, ale aj veľkým davom, ktorý sa skúšal evakuovať cez ľavý východ a ich evakuácia trvala celkovo 158 časových jednotiek. Na obrázku 5.4b je vidieť, ako vyzerala situácia na mape v čase, keď už zhruba polovica statických agentov z pravej strany mapy prekonala stret s opakovane zameriavajúcimi agentmi.

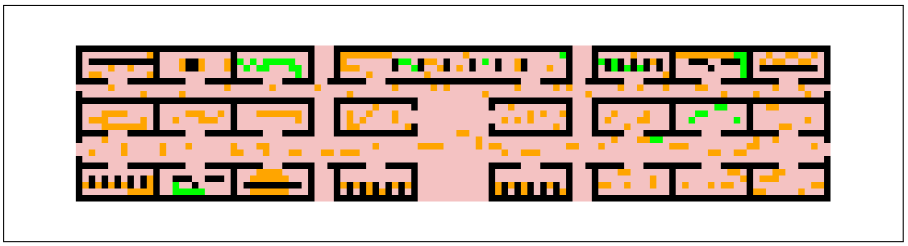
V scenári *polovica* boli agenti rozmiestnení tak, aby bol v každej kancelárií jeden staticky cielený a jeden opakovane zameriavajúci agent. Na grafe priebehu evakuácie pre opakovane zameriavajúcich agentov v tomto scenári je

vidieť spomalenie tempa evakuácie od času zhruba 60. Dovtedy unikali agenti ktorí sa evakuovali pravým východom alebo prišli ku ľavému východu skôr, ako sa pri ňom vytvoril dav. V časovej jednotke 60 už všetci ohrození agenti stoja v jednom dave pred ľavým východom, takže staticky cielení aj opakovane zameriavajúci agenti sa do bezpečia dostávajú zhruba rovnakým tempom.

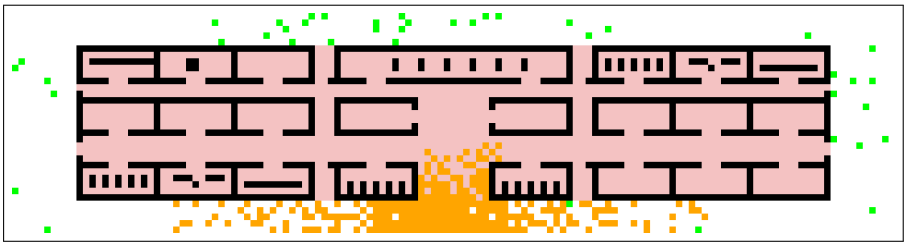
	Opakovane zameriavajúce		**Staticky cielené**	
Scenár	**Počet**	**Trvanie evakuácie**	**Počet**	**Trvanie evakuácie**
Iba zameriavajúce	80	94	0	
Polovica	40	96	40	160
Šestina	14	77	66	158

Tabuľka 5.4: Počty agentov a trvanie evakuácie pre scenáre na mape Kancelárie

5.3.3 Nákupné centrum



(a) Počiatočný stav



(b) Preplnenie priestranstva pred hlavným vchodom

Obr. 5.5: Scenár *šestina* mapy Nákupné centrum

Pre mapu nákupného centra sme vytvorili tri modifikované scenáre, nazvané *štvrtina*, *šestina* a *iba statické*, podľa počtu opakovane zameriavajúcich agentov, ktorí sa na nich nachádzajú. Statickí agenti sa snažia uniknúť hlavným východom umiestneným v spodnej časti mapy. Rozdelenie agentov medzi opakovane zameriavajúcich a statických je vo všetkých scenároch náhodné, takže obidva typy sú po mape rozptýlené rovnomerne.

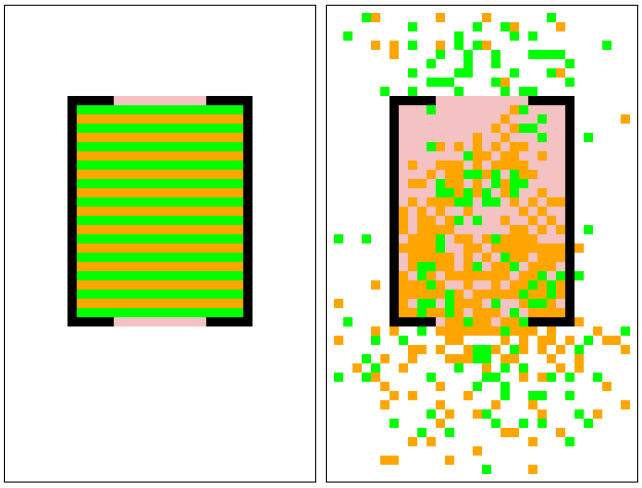
Okolo celej mapy je bezpečná zóna pomerne úzka, preto je kritické, aby agenti využívali čo najviac únikových východov. Inak sa evakuácia spomalí,

kvôli preplneniu bezpečnej zóny. Ako vidieť na výsledkoch simulácie pre rôzne scenáre, zníženie informovanosti agentov evakuáciu neúmerne predlžuje. Na grafe vidno vo všetkých scenároch okolo časovej jednotky 90 extrémne spomalenie priebehu evakuácie. Spôsobuje ho naplnenie priestranstva pred hlavným východom z budovy, ktoré kvôli nedokonalej koordinácií agenti nestíhajú opúšťať. Táto situácia je ilustrovaná pre scenár *šestina* na obrázku 5.5b.

	Opakovane zameriavajúce		**Staticky cielené**	
Scenár	**Počet**	**Trvanie evakuácie**	**Počet**	**Trvanie evakuácie**
Iba zameriavajúce	299	129	0	
Štvrtina	75	61	224	257
Šestina	42	60	257	303
Iba statické	0		299	368

Tabuľka 5.5: Počty agentov a trvanie evakuácie pre scenáre na mape Nákupné centrum

5.3.4 Blokovanie



(a) Scenár *polovica* (b) Priebeh evakuácie v scenári *tretina*

Obr. 5.6: Situácie z mapy Blokovanie

Mapa Blokovanie je špecifická tým, že ide o nerealistickú mapu, používanú na testovanie správania agentov v úplne zaplnenom priestore. Vytvorili sme na nej tri modifikované scenáre, nazvané *polovica*, *tretina* a *iba statické*, podľa počtu opakovane zameriavajúcich agentov, ktorí sa na nich nachádzajú. Statickí agenti sa pokúšajú uniknúť východom na spodnej strane mapy. V jednom riadku mapy je vždy iba jeden typ agentov (viz. obrázok 5.6a).

Vzhľadom na pravidelnosť mapy nie sú výsledky na nej prekvapivé. Evakuácia zo scenára *iba statické*, pri ktorej sa využíva iba jeden vchod z dvoch trvá 2,06× dlhšie, ako evakuácia, pri ktorej plne informovaní agenti využívajú obidva východy. Výsledky sú podobné výsledkom z mapy nákupného centra, vrátane efektu spomalenia evakuácie opakovane zameriavajúcich agentov v čase, keď sú zaseknutí v dave čakajúcom na prechod hlavným východom. Na tejto mape sa však výraznejšie prejavuje pri opakovane zameriavajúcich agentoch (v čase 100).

Zaujímavú situáciu je vidieť na obrázku 5.6b. Keďže si opakovane zameriavajúci agenti z hornej polovice vybrali na evakuáciu horný východ, snažili sa ku nemu dostať a vytvorili si pre svoje cesty rezervácie. Tie zablokovali cestu statických agentov k dolnému východu a donútili ich uhnúť. Preto sa napokon aj časť statických agentov (ktorí mali všetci ísť dolným východom) do bezpečia dostala cez horný.

Okrem toho je okolo stredu obrázku vidieť aj zhluk opakovane zameriavajúcich agentov, ktorí boli zablokovaní statickými agentmi a snažia sa cez nich dostať ku hornému východu.

	Opakovane zameriavajúce		**Staticky cielené**	
Scenár	**Počet**	**Trvanie evakuácie**	**Počet**	**Trvanie evakuácie**
Iba zameriavajúce	414	146	0	
Polovica	216	207	198	219
Tretina	144	214	270	236
Iba statické	0		414	301

Tabuľka 5.6: Počty agentov a trvanie evakuácie pre scenáre na mape Blokovanie

Záver

Na základe rešerše aktuálnej literatúry týkajúcej sa algoritmov multi-agentného hľadania ciest a algoritmov na plánovanie evakuácie sme v tejto práci navrhli a implementovali nový lokálny algoritmus na plánovanie evakuácie. Experimentálne overenie našej implementácie ukazuje, že plány, ktoré generuje, majú kvalitu (meranú ako celkové trvanie evakuácie) podobnú evakuačným plánom, ktoré generujú centralizované algoritmy.

Plány, ktoré generuje algoritmus LC-MAE sú realistickejšie, než plány z centralizovaných algoritmov a dajú sa použiť pri simulácií evakuácie z rôznych budov alebo priestranstiev. Nie sú totiž založené na predpokladoch o homogenite agentov alebo možnosti ich rozdelenia na skupiny. Cesta každého agenta sa plánuje individuálne, na základe obmedzených znalostí o jeho okolí a plánoch ostatných agentov. Evakuácia sa naviac plánuje na detailnej mape prostredia, ktorá do úvahy berie aj malé prekážky, ktoré sa v ňom nachádzajú a správanie sa agentov po tom, čo sú evakuovaní do bezpečia.

Fakt, že na mapy sa dajú umiestniť agenti rôznych typov ďalej vylepšuje realizmus simulácií. Umožňuje napríklad manuálne špecifikovať únikový východ, ktorý časť agentov použije a tým overiť jeho kapacitu a značenie, ktoré by sa malo aplikovať. Flexibilita implementácie umožňuje jednoduché pridanie ďalších typov agentov, ktorí sa od už existujúcich môžu líšiť len výberom cieľa alebo aj úplne inou implementáciou pohybu a hľadania ciest.

Vďaka implementácií LC-MAE, ktorú sme v rámci práce vypracovali, je možné ho na simuláciu evakuácie okamžite používať. Vo vizualizačnom module je jednoduché zobraziť si výsledok takejto simulácie a modul na overovanie výkonu umožňuje o priebehu evakuácie automaticky vypočítať množstvo štatistík a jednoducho porovnávať rôzne návrhy budov a možné rozloženia osôb.

Okrem využitia implementácie algoritmu na simuláciu evakuácie by algoritmus LC-MAE mohol nájsť uplatnenie aj v hrách. Herní vývojári, ktorí potrebujú do správania sa postáv pridať útek z ohrozených miest by mohli algoritmus LC-MAE jednoducho využiť, keďže je založený na algoritme WHCA*,

ktorý už je v mnohých hrách implementovaný.

V nadväzujúcich prácach by sa algoritmus LC-MAE dal vylepšiť zlepšením výberu evakuačného cieľa. Jedna z ciest, ktorou by sa výskum mohol uberať sú techniky, ktoré by možným cieľom prideľovali sofistikovanejšie skóre, ako iba ich vzdialenosť od agenta.

Ďalej by sa správanie agentov v bezpečnej zóne dalo vylepšiť heuristikou, ktorá by sa ich snažila dostať do nej čo najhlbšie. Ako však už bolo spomenuté v práci, ide o komplikovaný problém bez jednoznačného riešenia.

Okrem toho by sa dal na základe aktuálnych poznatkov z oblasti davového správania a správania sa evakuovaných osôb ďalej spresňovať spôsob, akým agenti vyhľadávajú a nasledujú svoje cesty. Tým by sa mohlo dosiahnuť zlepšenie realizmu a teda aj užitočnosti algoritmov, ktoré sme popísali v tejto práci.

Bibliografia

1. HART, Peter E.; NILSSON, Nils J.; RAPHAEL, Bertram. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Systems Science and Cybernetics*. 1968, roč. 4, č. 2, s. 100–107. Dostupné z DOI: 10.1109/TSSC.1968.300136.
2. SURYNEK, Pavel. *Abstract Path Planning for Multiple Robots: A Theoretical Study*. 2010. Dostupné tiež z: https://iti.mff.cuni.cz/series/2010/503.pdf. Technická správa. Institut teoretické informatiky, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze.
3. RYAN, Malcolm R. K. Graph Decomposition for Efficient Multi-Robot Path Planning. In: VELOSO, Manuela M. (ed.). *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. 2007, s. 2003–2008. Dostupné tiež z: http://ijcai.org/Proceedings/07/Papers/323.pdf.
4. SILVER, David. Cooperative Pathfinding. In: YOUNG, R. Michael; LAIRD, John E. (ed.). *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA*. AAAI Press, 2005, s. 117–122. ISBN 1-57735-235-1.
5. ERDMANN, Michael A.; LOZANO-PÉREZ, Tomás. On Multiple Moving Objects. *Algorithmica*. 1987, roč. 2, s. 477–521. Dostupné z DOI: 10.1007/BF01840371.
6. LUNA, Ryan; BEKRIS, Kostas E. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. In: WALSH, Toby (ed.). *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. IJCAI/AAAI, 2011, s. 294–300. ISBN 978-1-57735-516-8. Dostupné z DOI: 10.5591/978-1-57735-516-8/IJCAI11-059.

7. LUNA, Ryan; BEKRIS, Kostas E. Efficient and Complete Centralized Multi-Robot Path Planning. In: BORRAJO, Daniel; LIKHACHEV, Maxim; LINARES LÓPEZ, Carlos (ed.). *Proceedings of the Fourth Annual Symposium on Combinatorial Search, SOCS 2011, Castell de Cardona, Barcelona, Spain, July 15.16, 2011*. AAAI Press, 2011. Dostupné tiež z: http://www.aaai.org/ocs/index.php/SOCS/SOCS11/paper/view/4010.

8. WILDE, Boris de; MORS, Adriaan ter; WITTEVEEN, Cees. Push and rotate: cooperative multi-agent path planning. In: GINI, Maria L.; SHEHORY, Onn; ITO, Takayuki; JONKER, Catholijn M. (ed.). *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*. IFAAMAS, 2013, s. 87–94. ISBN 978-1-4503-1993-5. Dostupné tiež z: http://dl.acm.org/citation.cfm?id=2484938.

9. SURYNEK, Pavel. A novel approach to path planning for multiple robots in bi-connected graphs. In: *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*. IEEE, 2009, s. 3613–3619. Dostupné z DOI: 10.1109/ROBOT.2009.5152326.

10. SHARON, Guni; STERN, Roni; GOLDENBERG, Meir; FELNER, Ariel. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. In: WALSH, Toby (ed.). *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. IJCAI/AAAI, 2011, s. 662–667. ISBN 978-1-57735-516-8. Dostupné z DOI: 10.5591/978-1-57735-516-8/IJCAI11-117.

11. STANDLEY, Trevor Scott. Finding Optimal Solutions to Cooperative Pathfinding Problems. In: FOX, Maria; POOLE, David (ed.). *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010. Dostupné tiež z: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1926.

12. ZELINSKY, Alexander. A mobile robot exploration algorithm. *IEEE Trans. Robotics and Automation*. 1992, roč. 8, č. 6, s. 707–717. Dostupné z DOI: 10.1109/70.182671.

13. HOLTE, Robert C.; PEREZ, M. B.; ZIMMER, Robert M.; MACDONALD, Alan J. Hierarchical A*: Searching Abstraction Hierarchies Efficiently. In: CLANCEY, William J.; WELD, Daniel S. (ed.). *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 1*. AAAI Press / The MIT Press, 1996, s. 530–535. ISBN 0-262-51091-X.

Dostupné tiež z: http://www.aaai.org/Library/AAAI/1996/aaai96-079.php.

14. CHALMET, L. G.; FRANCIS, R. L.; SAUNDERS, P. B. Network models for building evacuation. *Fire Technology*. 1982, roč. 18, č. 1, s. 90–113. ISSN 1572-8099.

15. MISHRA, Gopinath; MAZUMDAR, Subhra; PAL, Arindam. Improved Algorithms for the Evacuation Route Planning Problem. In: LU, Zaixin; KIM, Donghyun; WU, Weili; LI, Wei; DU, Ding-Zhu (ed.). *Combinatorial Optimization and Applications - 9th International Conference, COCOA 2015, Houston, TX, USA, December 18-20, 2015, Proceedings*. Springer, 2015, zv. 9486, s. 3–19. Lecture Notes in Computer Science. ISBN 978-3-319-26625-1. Dostupné z DOI: 10.1007/978-3-319-26626-8_1.

16. ARBIB, Claudio; MUCCINI, Henry; MOGHADDAM, Mahyar Tourchi. Applying a network flow model to quick and safe evacuation of people from a building: a real case. In: *Proceedings of the GEOSAFE Workshop on Robust Solutions for Fire Fighting, RSFF 2018, L'Aquila, Italy, July 19-20, 2018*. 2018, s. 50–61.

17. LU, Qingsong; HUANG, Yan; SHEKHAR, Shashi. Evacuation Planning: A Capacity Constrained Routing Approach. In: CHEN, Hsinchun; MIRANDA, Richard; ZENG, Daniel Dajun; DEMCHAK, Chris C.; SCHROEDER, Jennifer; MADHUSUDAN, Therani (ed.). *Intelligence and Security Informatics, First NSF/NIJ Symposium, ISI 2003, Tucson, AZ, USA, June 2-3, 2003, Proceedings*. Springer, 2003, zv. 2665, s. 111–125. Lecture Notes in Computer Science. ISBN 3-540-40189-X. Dostupné z DOI: 10.1007/3-540-44853-5_9.

18. YU, Jingjin; LAVALLE, Steven M. Multi-agent Path Planning and Network Flow. In: *Algorithmic Foundations of Robotics X - Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics, WAFR 2012*. 2012, s. 157–173.

19. FOUDIL, Cherif; DJEDI, Noureddine; SANZA, Cédric; DUTHEN, Yves. Path finding and Collision Avoidance in Crowd Simulation. *CIT*. 2009, roč. 17, s. 217–228.

20. KURDI, Heba A.; AL-MEGREN, Shiroq; ALTHUNYAN, Reham; AL-MULIFI, Asma. Effect of exit placement on evacuation plans. *European Journal of Operational Research*. 2018, roč. 269, č. 2, s. 749–759.

21. SILVER, David. Cooperative Pathfinding. In: *AI Game Programming Wisdom 3*. Ed. RABIN, Steve. Charles River Media, 2006, kap. 2.1. ISBN 1584504579.

22. STURTEVANT, N. Benchmarks for Grid-Based Pathfinding. *Transactions on Computational Intelligence and AI in Games*. 2012, roč. 4, č. 2, s. 144–148. Dostupné tiež z: `http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf`.

23. HAGBERG, Aric A.; SCHULT, Daniel A.; SWART, Pieter J. Exploring Network Structure, Dynamics, and Function using NetworkX. In: *Proceedings of the 7th Python in Science Conference*. 2008, s. 11–15.

24. GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design Patterns: Elements of Reusable Object-Oriented Software. In: Pearson Education, 1994, kap. Strategy, s. 315. Addison-Wesley Professional Computing Series. ISBN 9780321700698. Dostupné tiež z: `https://books.google.cz/books?id=6oHuKQe3TjQC`.

25. CRAVEN, Paul Vincent. *The Python Arcade Library* [`http://arcade.academy`]. 2019.

26. WALSH, Toby (ed.). *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. IJCAI/AAAI, 2011. ISBN 978-1-57735-516-8. Dostupné tiež z: `http://ijcai.org/proceedings/2011`.

Dodatok A

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `README.md` ..... Súbor s informáciami o programe evacsim
- `text` ..... adresár s LaTeX zdrojovými kódmi textu práce
  - `BP_Selvek_Robert_2019.pdf` ..... PDF súbor s prácou