FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Vykonávání plánů pro automatický sklad
Student:	Filip Leško
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je otestovat vykonávání plánů pro roboty v automatickém skladu pomocí zmenšeného modelu s malými mobilními roboty (typu Ozobot EVO). Předpokládáme využití předchozích výsledků s vykonáváním plánů pro multi-agetní hledání cest (MAPF). Automatický sklad oproti MAPF představuje výrazné rozšíření, kde se roboti nejen pohybují, ale i převážejí zboží. Úkoly pro uchazeče jsou následující:

1. Seznámí se s literaturou o plánování cest, zejména s řešícími algoritmy a jejich nadstavbami pro navigaci robotů v automatickém skladu.

2. Navrhne a implementuje systém pro testování vykonávání plánů pro sklady v rámci zmenšeného modelu.

3. V rámci navrženého systému provede relevantní testy a provede analýzu problémů, které vykonávání plánů v rámci modelu skladu přináší.

[1]. Ján Chudý, Nestor Popov, Pavel Surynek: Emulating Centralized Control in Multi-Agent Pathfinding Using Decentralized Swarm of Reflex-Based Robots. SMC 2020: 3998-4005

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 17. února 2022 v Praze.

Bakalárska práca

VYKONÁVÁNÍ PLÁNŮ PRO AUTOMATICKÝ SKLAD

Filip Leško

Fakulta informačních technologií
Katedra aplikované matematiky
Vedúci: Prof. RNDr. Pavel Surynek, Ph.D.
2. januára 2023

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam algoritmov

Chcel by som poďakovať vedúcemu práce Prof. RNDr. Pavelovi Surynkovi, Ph.D. za jeho postrehy a návrhy k tejto téme ako aj za požičanie školského vybavenia

Vyhlásenie

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 2. januára 2023 .....

Abstrakt

V této práci byl navržen zmenšený model automatizovaného skladu, na kterém je následně testováno provádění plánů. Plány provádí skupina robotů nazvaná Ozobot Evo. Při řešení tato práce vychází ze znalostí vykonáváním plánů pro multi-agetní hledání cest a algoritmů řešících problém skladu, které práce upravuje a rozšiřuje. Model podporuje jak ruční přidávání úloh, tak jejich generování pro lepší simulaci reálných skladů. Testy byly provedeny jak na upraveném algoritmu, který řeší instance skladu, tak na modelu skladu s robotmi. Během testování algoritmu bylo zjištěno, jak jednotlivé atributy skladu ovlivňují efektivitu řešení algoritmu. Experimenty s roboty porovnávají úspěšnost vykonávaní a následně je diskutováno, co ovlivňuje neúspěšnost provedení plánu.

Klíčová slova MAPF, MAPD, Ozobot, automatický sklad, model automatizovaného skladu

Abstract

In this paper, a scaled down model of an automated warehouse is proposed and the execution of the plans is subsequently tested. The plans are executed by a group of robots called Ozobot Evo. In solving, this work builds on the knowledge of plan execution for multi-agent pathfinding and algorithms solving the warehouse problem, which the work modifies and extends. The model supports both manual job addition and job generation to better simulate real warehouses. Tests were conducted on both the modified algorithm that solves the warehouse instances and the warehouse model with robots. During the testing of the algorithm, it was determined how the different attributes of the warehouse affect the efficiency of the algorithm's solution. Experiments with robots compare the success of execution and then discusses what influences the failure of plan execution.

Keywords MAPF, MAPD, Ozobot, automated warehouse, automated warehouse model

Seznam zkratek

MAPF	Multi-agentné plánovanie
MAPD	Multi-agentné vyzdnihnutie a donáška
CBS	Conflict base search
SMT	Satisfiability modulo theories
TP	Token passing

Kapitola 1
Úvod

V posledných rokoch sa automatické sklady stávajú čím ďalej tým populárnejšími. So zvyšujúcim sa počtom objednávok v logistickej sfére sa zvyšujú aj požiadavky na veľkosť a efektivitu skladov. Tieto požiadavky sa snaží riešiť práve automatizácia logistiky skladov. V mnohých prípadoch tento prístup dokáže odbaviť viac požiadavok za kratší čas a zároveň požaduje menej ľudských zdrojov. Úlohou ako navrhnúť a obsluhovať takýto sklad sa zaoberá problém multi-agentného vyzdvihnutia a donášky.

Problém multi-agentného vyzdvihnutia a donášky modeluje situáciu, kde veľké množstvo agentov vybavuje tok prichádzajúcich požiadavok. Tieto požiadavky tvoria úlohy, ktoré je potrebné vyriešiť v čom najkratšom čase a s čím najväčšou efektivitou. V súčastnosti existuje už niekoľko prístupov ako ho riešiť, no väčšinou sú tieto riešiče testované vo virtuálny prostrediach. Takéto testovanie je však vzdialené od nasadenia agentov do reálnych situácii, ktoré v automatickom sklade môžu nastať.

Hlavným cieľom práce je preto naviazať na predchádzajúci výskum centralizovaného multi-agentného hľadania ciest na fyzických robotoch (Ozobotoch). Výsledky tejto práce, za využitia moderných algoritmov na riešenie problému skladiska, použiť pri vytvorení zmenšeného modelu automatizovaného skladiska. Na tomto modeli následne vykonať testy na Ozobotoch a analyzovať problémy, ktoré sa pri vykonávaní testov vyskytli.

Práca sa však nepokúša vytvoriť nový algoritmus na riešenie inštancií multi-agentného vyzdvihnutia a donášky. Využívať bude už existujúce algoritmy, ktoré môžu byť upravené pre potreby reálneho prostredia.

Štruktúra práce je nasledujúca. Práca v 2. kapitole poskytne teoretické východiská, ktoré budú používané počas celej práce. V 3. kapitole je analyzovaný robot Ozobot Evo, ktorý bol použitý v experimentálnej časti. Kapitola taktiež analyzuje predchádzajúce práce, ktoré sa používali Ozobotov na riešenie podobných problémov. Štvrtá kapitola sa zameria na návrh modelu automatizované skladu. Budú predstavené využitia existujúcich riešení, ich úpravy a rozšírenia. Piata kapitola je venovaná experimentom ako na Ozobotoch, tak aj na vybranom algoritme, ktorý bude riešiť problematiku skladiska. Napokon záver popíše výsledky práce, zhodnotí splnenie cieľov práce a ponúkne možné rozšírenia a skúmania do budúcnosti.

Kapitola 2
Teoretické východiská

V tejto kapitole sa čitateľ oboznámi so základnými teoretickými poznatkami, ktoré sa týkajú automatického skladu a plánovania ciest. Nazačiatku bude definované plánovanie ciest pre skupinu agentov a budú predstavené algoritmy, ktoré ju riešia. V ďalšej sekcii práca popíše problém automatizovaného skladu a taktiež prístupy, ktorými sa nechá riešiť.

2.1 Multi-Agentné plánovanie

Multi-Agentné plánovanie (MAPF) je problém, ktorý hľadá cestu pre skupinu agentov z počiatku do cieľa a zároveň zabezpečuje aby cesta bolo pre všetkých agentov bez kolízií. MAPF má praktické využitie vo viacerých sférach ako videoherný priemysel [1], kontrola dopravnej premávky, či robotika [2]. MAPF tvorí kostru problému automatizovaného skladiska, preto je potrebné túto tematiku popísať.

V sekcii bude formálne zadefinovaný MAPF problém a metriky, ktorými je možné overiť kvalitu riešení problému. Následne budú predstavené možné prístupy riešenia a ich výhody/nevýhody. Koniec sekcie je venovaný algoritmom, ktoré MAPF riešia.

2.1.1 Definícia MAPF

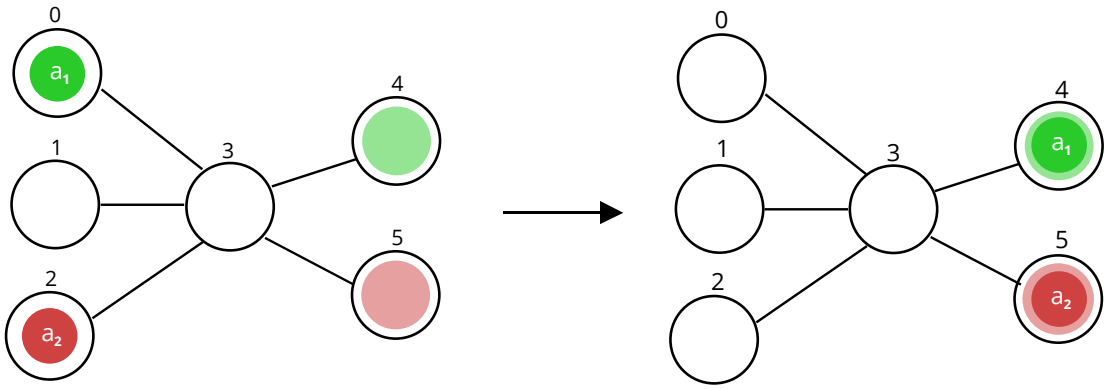
Existuje viacero spôsobov ako formálne zadefinovať MAPF problém. V [3] poskytuje autor *klasickú* definíciu problému.

V klasickom MAPF probléme definujeme k agentov $A = \{a_1, ..., a_k\}$ a trojicu (G, s, t), kde:

- $G = (V, E)$ je neorientovaný graf s usporiadanou dvojicou (V, E). Pričom V sú vrcholy grafu, ktoré predstavujú miesta na ktorých môže agent stáť a E hrany, po ktorých sa môže pohybovať a tým meniť polohu. Každý vrchol môže okupovať práve jeden agent.
- s je funkcia, ktorá mapuje agentov na počiatočné vrcholy.
- t je funkcia, ktorá mapuje agentov na koncové vrcholy.

Čas je braný ako diskrétny. Každú *časovú jednotku* (*timestep*) sa každý agent nachádza na jednom z vrcholov a uskutočňuje pritom jedinú *akciu*. Definujeme 2 druhy akcií: *move* a *wait*. Pri akcii move sa agent premiestni z vrcholu v do susedného vrcholu v'[1]. Wait znamená, že agent ostane stáť na rovnakom vrchole.

[1] Susedný vrchol v' je spojený s vrcholom v hranou



Obr. 2.1 Príklad MAPF problému

Pre agenta a_i môže byť definovaný *plán* ako postupnosť akcií $\pi_i = (k_1, ..., k_n)$ a $\pi_i[x]$ lokácia agenta po vykonaní prvých x krokov. Inštanciu MAPF problému teda ide formálne označiť ako štvoricu $\Sigma = (G, A, s, t)$. *Riešením* problému je potom množina k plánov (pre každého agenta jeden) označená $R_\pi = \{\pi_1, ..., \pi_k\}$.

Príklad jednoduchého MAPF problému je znázornený v ukážke Obr. 2.1.

Riešenie problému je *validné*, ak medzi každými dvoma plánmi v každom časovom okamžiku neexistuje kolízia. Rozličné typy kolízií môžu byť definované pre rôzne variácie MAPF problémov. V [4, 5] autori predstavujú bežné typy konfliktov pre klasické MAPF. Jeden z možných konfliktov je *konflikt na vrchole*. Ten nastáva, keď viac ako jeden agent okupuje jeden vrchol v rovnakom čase. Ďalší možný konflikt je *konflikt na hrane*, ktorý nastáva v prípade, že dvaja alebo viacerí agenti sa snažia použiť rovnakú hranu v rovnaký časový okamžik. Tieto dva základné typy konfliktov sú ilustrované na Obr. 2.2.

Kvalitu validných riešení je možné vyhodnocovať viacerými spôsobmi. Najčastejšie sú účelové funkcie: *makespan* [6] a *sum-of-cost* [7].

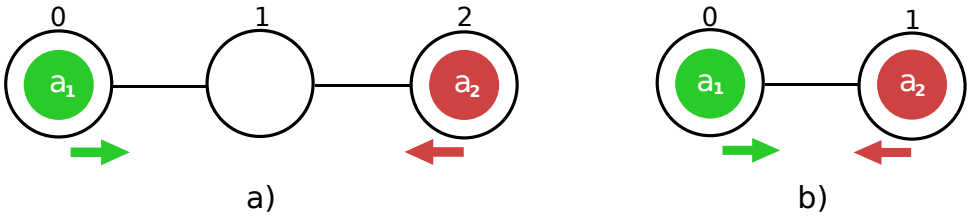
▶ **Definícia 2.1** (**makespan**). *Makespan je počet časových krokov (timestep) potrebný od začiatku vykonávania plánov, až po posledný krok posledného agenta. Označený ako μ, kde* $\mu = \max_{\pi \in R_\pi}(|\pi|)$, *pričom $|\pi|$ je počet akcií agenta v pláne π.*

▶ **Definícia 2.2** (**sum-of-cost**). *Sum-of-cost je súčet všetkých krokov, pre všetkých agentov počas vykonávania plánu. Označený ako ξ, kde* $\xi = \sum_{i=1}^{k} |\pi_i|$, *pričom $|\pi_i|$ je počet akcií agenta v pláne π_i.*

2.1.2 Prístupy k riešeniu MAPF

Nájsť validné, optimálne[2] a bezkolízne riešenie je *NP-ťažký problém* [8]. Preto sa aplikujú dve stratégie pri riešení. Prvou z nich je riešiť problém neoptimálne. Tieto algoritmy majú najhoršiu časovú zložitosť polynomiálnu. Väčšinou ide o relaxované verzie algoritmov, ktoré hľadajú

[2] Účelová funkcia je minimálna



Obr. 2.2 MAPF konflikty. a) konflikt na vrchole b) konflikt na hrane

optimálne riešenie. Využívajú sa v situáciach kedy je potrebné nájsť riešenie rýchlo a zároveň je postačujúce neoptimálne. Príkladom takého algoritmu je *Suboptimal Conflict-Based Search* [9] alebo *Hierarchical Cooperative A** [10]. Taktiež sem patria *Rule-based algoritmy*. Rule-based algoritmy zahrňujú pravidlá pre špecifické pohyby agenta. Ich príkladom je algoritmus *Push-and-Rotate* [11].

Na druhú stranu optimálne algoritmy dokážu za cenu vyšších časových požiadavok poskytnúť riešenie s najnižšou cenou. V posledných rokoch sa objavilo niekoľko spôsobov ako riešiť MAPF optimálne. Tieto algoritmy sa zvyčajne delia do 2 hlavných kategórii: *Search-based* a *Reduction-based*. V Search-based kategórii sa nachadzaju zväčša algoritmy rozširujúce *A** ako *Increasing Cost Tree Search* (ICTS) [12], či *Conflict Based Search* (CBS), priblížený v sekcii 2.1.3. Reduction-based algoritmy rozkladajú MAPF problém na iný už známy problém (SAT, CSP, ...), pre ktorý už existujú efektívne riečiče. Patrí tu napríklad *MDD-SAT*, ktorý práca rozoberá v sekcii 2.1.4.

Nakoľko tieto algoritmy poskytujú optimálne riešenia, ich efektivita s rastúcim počtom agentov a zväčšujúcim sa prostredím rýchlo klesá.

2.1.3 Conflict Based Search (CBS)

Conflict Based Search [13] je optimálny dvoj-vrstvový riešič MAPF problémov. Hlavnou ideou algoritmu hľadať riešenie na postupne rastúce množine obmedzení. Množina je zo začiatku prázdna, a ak je nájdený konflikt, obmedzenie sa pridá do množiny. Práca teda uvádza nasledujúce tvrdenie korektnosti a konečnosti(dôkaz v [13]).

▶ **Tvrdenie 2.3.** *Pre zadanú MAPF inštanciu CBS vráti optimálne riešenie, ak existuje.*

Algoritmus sa skladá z prehľadávania *vysokej úrovni* a *nízkej úrovni*.

Vysoko-úrovňový level prehľadáva *binárny strom s obmedzeniami*. Každý uzol stromu pozostáva z množiny obmedzení, riešenia a celkovej ceny. Omedzenie je usporiadaná trojica (a_i, v, t), kde agent a_i nemôže obsadzovať vrchol v v časovom okamžiku t. V koreňovom uzle je množina prázdna. Potomkovia uzlu dedia túto množinu a pridávajú k nej práve jedno obmedzenie. Riešenie je množina k ciest (pre každého agenta jedna), ktoré sú nájdené v súlade s obmedzeniami v uzle pomocou nízko-úrovňového prehľadávania. Cena je celková cena riešenia v uzle.

Algoritmus 1: High-level search of CBS algorithm

```
1  CBS (Σ)
2      R.constrains ← ∅
3      R.solution ← paths from low-level search
4      R.cost ← calculateCost(R.solution)
5      insert R into OPEN
6      while OPEN is not empty do
7          N ← min(OPEN)
8          eraseMin(OPEN)
9          conflicts ← validate(N.solution)
10         if conflicts is empty then
11             return N.solution
12         C ← first conflict (a_i, a_j, v, t) in conflicts
13         for agent a in C do
14             N' ← new node
15             N'.constrains ← R.constrains ∪ (a, v, t)
16             N'.solution ← R.solution
17             N'.solution ← path from low-level search for a
18             N'.cost ← cost(N'.solution)
19             if N'.cost < ∞ then
20                 insert N' in OPEN
```

Pseudokód vysoko-úrovňového prehľadávania CBS je možné vidieť v ukážke Algoritmus 1. Algoritmus na začiatku inicializuje koreňový uzol R s prázdnu množinu obmedzení. Pre každého agenta nechá nízko-úrovňový prehľadávanie nájsť najkratšiu cestu a z týchto ciest vypočíta cenu (riadok 1-3). Na riadku 4 je koreň vložený do prioritnej fronty OPEN. Ďalšia časť prehľadávania prebieha v cykle, kým je fronta neprázdna. Z fronty je vyňatí prvok s najmenšou cenou. Riešenie, *R.solution*, aktuálne spracovávaného prvku je testované na výskyt kolízií. Ak sa v ňom kolízie nenachádzajú je riešenie vrátené ako výsledok (riadok 10), inak sa uloží prvý konflikt, ktorý bol nájdený (riadok 11). Konflikt je usporiadaná štvorica $(a_i, a_j v, t)$, ktorá značí, že kolízia nastala pre agentov a_i a a_j na vrchole v v časovom okamihu t. Dva nové uzly sa nechajú vytvoriť (pre každého z agentov v kolízii jeden) ako potomkovia aktuálneho uzla. Oba uzly zdedia množinu obmedzení a riešenie, pričom sa pre konfliktných agentov pridá nové obmedzenie a prepočíta sa cesta (riadok 13-17). Následne je prepočítaná cena pre nové riešenie a ak validné, uzly sú vložené do fronty OPEN.

CBS na nízkej úrovni hľadá najkratšiu cestu pre jedného agenta. Dodržia pritom obmedzenia, ktoré mu boli predané. Pre agenta a_i nájde najkratšiu z počiatočnej lokácie do cieľovej, zatiaľ čo v čase t nebude obsadzovať vrchol t. Ako nízko-úrovňové CBS môže byť použitý ľubovoľný optimálny algoritmus hľadania ciest, ktorý je možné modifikovať tak, aby dodržoval spomenuté požiadavky. Použitý by teda mohol byť napríklad algoritmus *A-star* [14].

2.1.4 MDD-SAT

Alternatívou k algoritmom založeným na prehľadávaní sú riešiče, ktoré rozložia MAPF problém do výrokovej formule. Príkladom takéhoto riešiča je aj *MDD-SAT* [15]. Výroková formula $\mathcal{F}(\xi)$ je splniteľná, práve vtedy, keď riešenie MAPF problému existuje pre sum-of-costs ξ. Ak SAT-riešič

vyhodnotí formulu ako splniteľnú, riešenie môže byt rekonštruované z vyhovujúceho priradenia formule. Túto skutočnosť o $\mathcal{F}(\xi)$ zachycuje nasledujúca definícia.

▶ **Definícia 2.4** (**Úplný výrokový model**). *Výroková formula $\mathcal{F}(\xi)$ je úplným výrokovým modelom MAPF Σ, ak platí:*

$\mathcal{F}(\xi)$ je splniteľná $\Leftrightarrow$ Σ má riešenie pre sum-of-costs ξ.

Vďaka tomu, že ξ je monotónna funkcia je možné po sérii dotazov $F(\xi_0)$, $F(\xi_0 + 1), ...$ nájsť MAPF riešenie s optimálnym ξ. Dotazy sú kladené SAT riešiču, kým nebude nájdené uspokojivé priradenie. Základným problémom je teda zakódovanie MAPF problému do výrokovej formule.

Hlavnou myšlienkou ako previesť inštanciu MAPF problému $\Sigma = (G, A, s, t)$ do výrokovej formule $F(\xi)$ je vytvorenie tzv. *time expansion grafu* (TEG) z pôvodného grafu G v Σ. TEG je *orientovaný acyklický graf*. Vzniká skopírovaním si všetkých vrcholov V pre všetky časové kroky $t = 0, 1, 2, ..., \mu$. To znamená, že skopírujeme celú množinu V pre každý časový okamžik z t. μ (makespan) tvorí spodnú hranicu časovej medze, kde sa zmestia všetky riešenia so sum-of-cost menšou alebo rovnou ξ. To znamená, ze všetky riešenia so sum-of-cost ξ musia byť možné pre makespan μ. Následne sú všetky hrany z pôvodného grafu G transformované na orientované hrany prepojujúce časové okamžiky. Hrany smerujú vždy z nižšieho časového okamžiku do vyššieho. Hrany reprezentujú jednotlivé možné akcie. *Wait* akcia je reprezentovaná hranou medzi ekvivalentnými vrcholmi v inom časovom momente. *Move* akcii odpovedajú orientované hrany medzi susediacimi vrcholmi. Orientovaná cesta v TEG odpovedá jednotlivému plánu agenta. Pre každého agenta a_i vytvoríme graf TEG_i. Zavádza sa *výroková premenná* $\mathcal{X}_u^t(a_j)$ pre každý vrchol u v časovom okamžiku t na TEG_i. Premenná je rovná *true* ak sa agent aj nachádza na vrchole v počas časového okamžiku t. Podobne je pre každú orientovanú hranu na TEG_i zavedená premenná $\mathcal{E}_{u,v}^t(a_j)$. Analogicky je hodnota premennej rovná *true* ak agent aj použije orientovanú hranu z u do v medzi časovými okamžikmi t a $t + 1$. Nakoniec sú pridané obmedzenia také, aby pravdivostné priradenia odpovedali platným riešeniam MAPF problému.

Algoritmus 2 ukazuje pseudokód MDD-SAT frameworku na riešenie MAPF problému. Algoritmus nazačiatku pre každého agenta nájde optimálnu cestu z počiatku do konca. Z tohoto riešenia sa vypočíta dolná hranica pre sum-of-cost ξ. ξ budeme postupne inkrementovať až kým nenájdeme vyhovujúce priradenie. Inštancia MAPF sa spoločne s aktuálnym ξ zakóduje do výrokovej formule $F(\xi)$. Formulu položíme ako dotaz SAT-riešiču, ktorý nám vráti priradenie (*assigment*). Následne ak $F(\xi)$ nebola splnená ξ je inkrementované a cyklus začína od znova. Ak bolo nájdené vyhovujúce priradenie, riešenie MAPF problému je vyňaté z tohto priradenia a vrátené z funkcie. V prípade, že MAPF inštancia nie je riešiteľná algoritmus ostane v nekonečnom cykle. Existencia riešenia môže byť zistená dopredu v polynomiálnom čase napríklad algoritmom *Push-and-Rotate* [11]. Výhodou tohoto prístupu je, že najmodernejšie SAT-riešiče môžu byť implementované na vyriešenie výrokových formuly.

Rýchlosť, akou nám SAT-riešič dokáže odpovedať na náš dotaz značne ovplyvňuje veľkosť formule, ktorú mu predávame. Táto veľkosť závisí na veľkosti jednotlivých TEG-ov. MDD-SAT implementuje vylepšenie základného TEG-u. Myšlienkou je pre každého agenta a_i redukovať graf TEG_i pomocou tzv. *multi-value decision diagram* (MDD) štruktúry, ktorá bola už využitá v súvislosti s riešičmi založenými prehľadávaní [12]. Pri prevedení TEG_i na MDD_i pre agenta a_i sa vykonáva prerezávanie vrcholov, pri ktorom sa zachováva správnosť zakódovania. Zatiaľ čo pôvodný TEG_i obsahuje v každom časovom kroku t všetky vrcholy, MDD_i obsahuje iba tie, ktoré sú dosiahnuteľné pod daným sum-of-cost ξ.

2.1.5 SMT-CBS

SMT-CBS [16] kombinuje myšlienku konfliktného vyhľadávania predstavenú v CBS a prístup založený na SAT-e. Narozdiel od MDD-SAT, kde boli zakódované všetky obmedzenia do výrokovej

Algoritmus 2: MDD-SAT framework for MAPF problems

```
1  MDD-SAT (Σ)
2      solution ← shortest paths for every agent
3      ξ ← sum-of-cost(solution)
4      while true do
5          F(ξ) ← encode(Σ, ξ)
6          assignment ← SAT-Solver(F(ξ))
7          if assignment ≠ UNSAT then
8              solution ← extract-Solution(assignment)
9              return paths
10         ξ ← ξ + 1
```

formule, ktorú následne riešil SAT-riešič. SMT-CBS pristupuje k zakódovanie MAPF problému tzv. lenivo (*lazily*). Relaxujeme pritom úplný výrokový model na *neúplný výrokový model*, čo popisuje nasledujúca definícia.

▶ **Definícia 2.5** (**Neúplný výrokový model**). *Výroková formula $\mathcal{H}(\xi)$ je neúplným výrokovým modelom MAPF Σ, ak platí:*

$\mathcal{H}(\xi)$ je splniteľná $\Leftarrow$ Σ má riešenie pre sum-of-costs ξ.

Lenivosť neúplného modelu je teda zakomponovaná v kódovaní obmedzení. Jediné obmedzenia ponechané z pôvodného MDD-SAT budú tie, ktoré zabezpečujú, že priradenie bude vytvárať validnu cestu pre agenta z počiatočnej konfigurácie do koncovej. Preto je pri nájdení vyhovujúceho priradenia nutné previesť *kontrolu validity ciest* extrahovaných z priradenia. Tento prístup znižuje veľkosť výrokovej formule a teda šetrí prácu SAT-rešiča za nutnosť kontroly platnosti MAPF riešenia.

Nahliadnuť na pseudokód je možne v ukážke Algoritmus 3. Pseudokód je rozdelený do 2 fáz, *vyššej* a *nižšej* podobne ako tomu bolo v CBS. *SMT-CBS-High* postupne inkrementuje sum-of-cost podobne ako tomu bolo pri frameworku MDD-SAT. Navyše si však v premennej $conflicts$ udržiavame konflikty, ktoré nám boli nájdené pomocou *SMT-CBS-Low*. SMT-CBS-Low vykonáva riešenie problému na konštantným ξ. Na zakódovanie formule (riadok 11) použijeme funkciu *encodeBasic*, ktorá narozdiel od predošlej *encode* vynecháva zakódovanie obmedzeniam zabraňujúcim kolíziám. Pridáva však kódovanie konfliktov (uložených v $conflicts$), ktoré sa počas riešenia naskytli do formule $H(\xi)$. Následne v cykle pokladáme SAT-riešiču dotaz, či je $H(\xi)$ splniteľná. Ak nie, tak funkcia vráti, že riešenie neexistuje a ukonči sa (riadok 22). Naopak ak splniteľná je, z priradenia získa riešenie, to validuje a kolízie uloží do premennej $collisions$ (riadok 14-16). Ak sa v riešení kolízie nenachádzali, toto riešenie tvorí výsledok algoritmu a je vrátené (riadok 16-18). Ak sú zistené kolízie, formula je upresnená a množina konfliktov a rozšíri o tieto konflikty. Teda ak je napríklad zistená kolízia agentov a_i a a_j na vrchole v v čase t, klauzula $(\neg\mathcal{X}_v^t(a_i) \vee \neg\mathcal{X}_v^t(a_j))$ je pridaná do $H(\xi)$ (riadok 19-21). Algoritmus pokračuje až kým nebude nájdené validné riešenie. SMT-CBS nakoniec môže skonštruovať rovnakú formulu ako MDD-SAT, často sa však môže stať, že riešenie je nájdené skôr a tým sa ušetria prostriedky potrebné na vyriešenie.

2.2 MAPD

Táto kapitola bude pojednávať o MAPD (Multi-agent pickup and delivery) probléme. Nazačiatku práca zadefinuje MAPD problém s použitím definície MAPF z predošlej sekcie. Následne budú predstavené riešiteľné inštancie MAPD problému a prezentované prístupy, ktorými je ich možné riešiť. Nakoniec práca predstaví algoritmus Token Passing, ktorý dokáže riešiť MAPD.

Algoritmus 3: SMT-CBS algorithm for MAPF solving

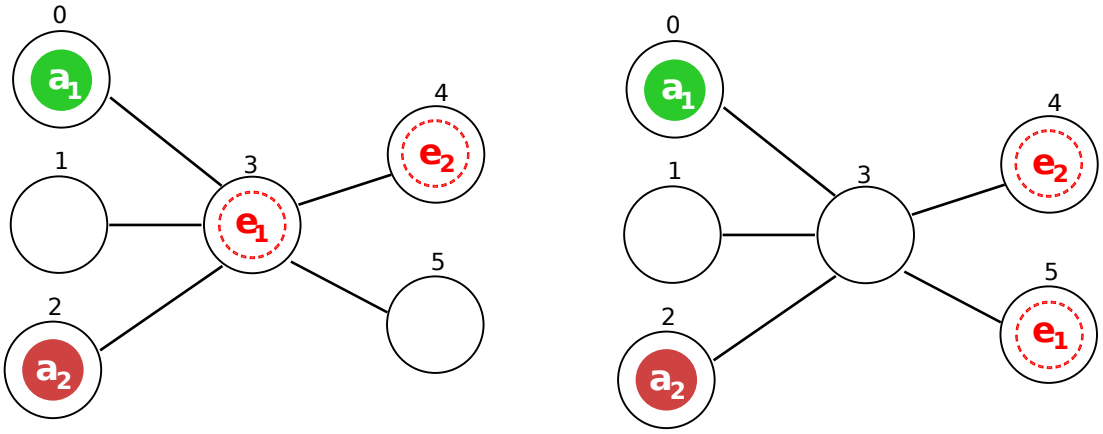
```
1  SMT-CBS-High (Σ)
2      conflicts ← ∅
3      solution ← shortest paths for all agents
4      ξ ← sum-of-cost(solution)
5      while TRUE do
6          (solution, conflicts) ← SMT-CBS-Low(conflicts, ξ, Σ)
7          if solution ≠ UNSAT then
8              return solution
9          ξ ← ξ + 1
10 SMT-CBS-Low(conflicts, ξ, Σ)
11     H(ξ) ← encodeBacic(conflicts, ξ, Σ)
12     while TRUE do
13         assignment ← SAT-Solver(H(ξ))
14         if assignment ≠ UNSAT then
15             solution ← extract-Solution(assignment)
16             collisions ← validate(solution)
17             if collisions = ∅ then
18                 return (paths, conflicts)
19             for each (a_i, a_j, v, t) ∈ collisions do
20                 H(ξ) ← H(ξ) ∪ {¬X_v^t(a_i) ∨ ¬X_v^t(a_j)}
21                 conflicts ← conflicts ∪ {[(a_i, v, t), (a_j, v, t)]}
22         return (UNSAT,conflicts)
```

Pri MAPF mali agenti dopredu určené ako miesta z ktorých štartovali, tak aj koncové destinácie. Na ich riešenie stačilo použiť jednorazove (one-shot) riešiče, ktoré naraz vyriešili inštanciu problému. MAPD predstavuje rozšírenie MAPF problému, kde agenti nemajú predom priradené koncové lokácie ale existujú úlohy, ktoré musia splniť. MAPD problém preto pozostáva z 2 subproblémov: *priraďovanie úloh* a *plánovanie ciest*. Tento prístup mapuje problém automatizovaných skladov.

2.2.1 Definícia MAPD

V sekcii 2.1.1 bol formálne zadefinovaný MAPF problém. MAPD však není možné popísať ako štvoricu (G, A, s, t), keďže agenti nemajú pevne danú koncovú pozíciu. Práca [17] poskytuje zadefinovanie problému ako štvoricu (G, A, s, T), kde T je množina ešte neuskutočnených úloh (Ostatné prvky štvorice ostávajú rovnaké ako v sekcii 2.1.1). Množina môže časom ako rásť tak aj zmenšovať sa. Každá úloha t_j ležiaca v T je charakterizovaná miestom vyzdvihnutia p_j ležiacom vo V a miestom donášky d_j taktiež ležiacom vo V. Jednotlivé úlohy sú priraďované agentom. Agent sa nazýva *voľný*, ak práve nevykonáva žiadnu úlohu. Voľnému agentovi môže byť priradená úloha t_j, ktorá leží v množine T. Po priradení úlohy t_j sa agent presúva zo svojej aktuálnej lokácie do miesta vyzdvihnutia p_j. Po dosiahnutí p_j agent začína vykonávať úlohu. Úloha t_j je následne odstránená z T. Agent, ktorý vykonáva úlohu sa nazýva *obsadený*. Keď agent dorazí do miesta donášky d_j, úloha t_j je označená ako dokončená a agent je označený ako voľný.

Miesta vyzdvihnutia a donášky, štartovacie miesta agentov a prípadné iné parkovacie miesta tvoria *koncové body*. Tie sa delia na: *koncové body úloh* (*task endpoints*), teda miesta vyzdvihnutia



Obr. 2.3 Príklady 2 MAPD inštancií

a donášky a *ostatné koncové body* (*non-task endpont*). Tieto dve množiny sú disjunktné[3]. [17]

2.2.2 Riešiteľné MAPD inštancie

Nie každá MAPD inštancia je riešiteľná. V [18] autori poskytujú podmienky, ktoré musia byť splnené pri tvorení takejto, konkrétne *well-formed*, inštancie. Teda, MAPD inštancia je well-formed, práve vtedy keď:

- Počet úloh je konečný.
- Parkovisko každého agenta sa líši od všetkých koncových bodov úloh.
- Pre každé 2 koncové body musí existovať aspoň 1 cesta, ktorá ich spája a zároveň neprechádza žiadnymi inými koncovými bodmi.
- Agenti môžu ostať na svojich parkovacích miestach tak dlho, ako bude nutné aby sa zabránilo kolíziám s ostatnými agentmi.

Ukážka Obr. 2.3 znázorňuje 2 MAPD inštancie. Inštancie obsahujú 2 štartovacie lokácie agentov a_1 a a_2 (zelené a červené kruhy) a 2 koncové body úloh e_1 a e_2 (červené čiarkované kruhy). Inštancia naľavo nie je well-formed, pretože porušuje pravidlo 2. Pre štartovacie pozície neexistuje cesta, bez koncového bodu, ku koncovému bodu e_2.

2.2.3 Centralizovaný a decentralizovaný pristúp

Podstatnou charakteristikou pri riešení MAPD problému je rozdielnosť prístupov medzi *centralizovaným* a *decentralizovaným*. Rozdiel medzi oboma prístupmi je vtom, kde sú plánované úlohy a cesty pre jednotlivých agentom. Centralizovaný prístup používa *globálny systém*, ktorý vykonáva prerozdeľovanie všetkých úloh a plánovanie ciest všetkých agentov. Agenti následne dostávajú už predom plánovanú cestu a úlohu, ktorú majú vykonať. Algoritmy *TA-Prioritized* a *TA-Hybrid* [19] boli vyvinuté na priraďovanie úloh a ciest. Tieto algoritmy každému agentovi priradia jednu alebo viac úloh a ciest. Akonáhle agentom boli priradené tieto informácie, žiadna ďalšia komunikácie nie je vyžadovaná, až kým systém neskončí ďalšiu plánovaciu fázu. Agenti medzi sebou nekomunikujú a bezkolíznosť je zabezpečená centrálnym systémom.

[3] Nemajú spoločný vrchol

Decentralizovaný systém nechá každého agenta plánovať vlastné úlohy a cesty, čo poskytuje lepšiu škálovateľnosť. Algoritmy, ktoré riešia MAPD inštanciu sú napríklad *Token Passing* (TP), *Token Passing with Task Swaps* (TPTS) [17]. Tieto algoritmy nechajú agentov prideľovať si úlohy a plánovať si cesty na základe globálnej informácie, ktorú dostávajú prostredníctvom *tokenu*. Vďaka tomuto prístupu je jednoduché pridávať nových agentov do systému. Tento prístup zvykne byť taktiež robustnejší. Pri výpadku jedného agenta môže byť tento agent odstránený, zatiaľ čo ostatní môže pokračovať vo vykonávaní a plánovaní.

2.2.4 Robustnosť a perzistentnosť

Najmodernejšie algoritmy dokážu vypočítať riešenie pre stovky robotov za niekoľko minút. Avšak vykonávanie týchto plánov na reálnych robotoch ostáva naďalej problémové, pretože väčšina známych riešičov používa nerealistické zjednodušenia a abstrakcie. Na sledovanie úspešnosti algoritmov v reálnych prostrediach môžu byť zavedené metódy: *perzistentnosť* a *robustnosť*. Vykonávanie plánu je považované za perzistentné, ak agent plní svoje úlohy bez nadbytočných wait akcií. Tieto akcie sa vyskytujú keď agent nedokáže vykonávať žiadnu akciu, lebo čaká kým mu riešič dodá plán, ktorý má vykonať.

Ako robustné vykonávanie plánu sa považuje to, kde sa nedochádza ku kolíziám agentov aj pri zmene času vykonania jednotlivých akcií. Tieto zmeny môžu nastať vo viacerých prípadoch ako: dočasná porucha agenta, či neočakávaná prekážka. Na zvýšenie robustnosti je možné zaviesť prístup k-robustnosť [20]. K-robustnosť pridáva ku klasickému plánu obmedzenie, že potom ako agent opustí pozíciu, žiadny iný agent nesmie vstúpiť na túto polohu ďalších k časových krokov.

2.2.5 Offline a online prístup

Pri zadávaní MAPD problému je potrebné určiť aké informácie algoritmus bude vedieť dopredu a aké mu budú prichádzať počas behu. Podľa toho, či algoritmus pozná úlohy, ktoré ma vyriešiť dopredu alebo mu budú prichádzať počas riešenia rozdeľujeme algoritmy na offline a online. Pri offline variante pozná algoritmus všetky úlohy a ich čas kedy majú byť vykonané a všetky cesty sú naplánované predtým než ich agenti začnú vykonávať. Vďaka tomu je možné skonštruovať algoritmus, ktorý dokáže vyriešiť inštanciu s nižším makespan. Príkladom offline algoritmu je už spomínaný *TA-Hybrid* [19]. Online prístup počíta s tým, že mu nová úloha môže prísť v každom okamžiku až do skončenia behu. Ich využitie odpovedá automatizovaným skladom, kde dopredu niesu známe všetky úlohy. Príkladom algoritmu, ktorý dokáže riešiť online inštancie je Token passing (TP), ktorému sa práca bude venovať v sekcii 2.2.6.

2.2.6 Token passing (TP)

Token Passing [17] je online, decentralizovaný algoritmus vyvinutý pre riešenie MAPD inštancií.

Základ algoritmu tvorí *token*, ktorý obsahuje všetky aktuálne cesty agentov, úlohy a ich priradenie k jednotlivým agentom. Ukážku pseudoalgoritmu je možné vidieť v Algoritmus 4. Algoritmus si nazačiatku vytvorí token, ktorému priradí aktuálne polohy agentov (riadok 2). Zvyšok prebieha v cykle kým neskončí celá simulácia a postupuje po jednom časovom úseku. Pri každej iterácii cyklu si algoritmus najskôr do množiny úloh priradí nové úlohy, ktoré prišli do systému počas predošlého časového úseku (riadok 4). Následne prechádza cez všetkých voľných agentov a vytvorí množinu úloh T tak, že žiadna iná cesta v $Token.paths$ nekončí v p_j alebo d_j úlohy t. Množina T sa môže nachádzať v 3 stavoch:

- T obsahuje aspoň jednu takú úlohu. Agent si vyberá z T úlohu, ktorej poloha vyzdvihnutia je k jeho momentálnej polohe najbližšie. Priradenie úlohy sa zapíše do tokenu a úloha je odstránená z množiny $Token.tasks$. Nakoniec sa v tokene aktualizuje cesta pre agenta a_i

pomocou funkcie *Path1*. *Path1* vypočíta cestu z momentálne polohy agenta do polohy vyzdvihnutia a následne do polohy donášky. Táto cesta musí byť bez kolízií s cestami v $Token.paths$ a optimálna. (riadok 7-11)

- Agent a_i sa nenachádza na d_i žiadnej úlohy z $Token.tasks$. V takom prípade algoritmus aktualizuje cestu pre agenta je momentálnou polohou. (riadok 12-13)
- V ostatných prípadoch bude cesta agenta a_i aktualizovaná pomocou funkcie *Path2*. *Path2* vypočíta cestu s minimálnou cenou z aktuálnej lokácie agenta do koncového bodu. Zvolený koncový bod musí byť rozdielny od pozícii donášok v množine úloh a koncov ciest ostatných agentov v $Token.paths$. Táto cesta musí byť taktiež bezkolízna s ostatnými cestami agentov v $Token.paths$. (riadok 14-15).

Po prejdení všetkých voľných agentov sa agenti posunú o cestu rovnú jednému časovému úseku.

Teraz bude dokázaná úplnosť a korektnosť algoritmu. Pre tento dôkaz je potrebné uviesť pomocné tvrdenie. To bude uvedené bez dôkazu, ktorý je možné nájsť v pôvodnej práci [17].

▶ **Tvrdenie 2.6.** *Funkcie Path1 a Path2 vrátia cestu pre všetky riešiteľné inštancie MAPD.*

▶ **Tvrdenie 2.7.** *Token passing rieši všetky riešiteľné inštancie MAPD problému.*

Dôkaz. *"V dôkaze bude ukázané, že každej úlohe je nakoniec pridelená nejaký agent, ktorý ju vykoná. Každému agentovi bude priradený token po obmedzenom počte časových krokov a žiadny agent nezostáva v mieste doručenia úlohy obsiahnutej v množine úloh, kvôli riadku 15. Podmienka na riadku 8 je teda nakoniec splnená a niektorý agent je priradený k niektorej z úloh na riadku 9. Agent ju potom môže vykonať vďaka Tvrdeniu 2.6.„* (17, vlastný preklad) ◀

Algoritmus 4: Token Passing

```
1  TP (MAPD instance)
2      Token.paths ←inital location for each agent
3      while true do
4          Token.tasks ← Token.tasks ∪ new tasks if any
5          for agent a_i in FreeAgents do
6              T ← {t_i ∈ Token.tasks | other path in Token.paths does not end in p_i or d_i}
7              if T is not empty then
8                  t ← t_i in T with closest p_i to a_i
9                  Token.assignments ← Token.assignments ∪ (a_i, t)
10                 Token.tasks.Remove(t)
11                 Token.paths ← Token.paths ∪ Path(a_i, t, Token)
12             else if a_i position not in d_i of any t_i ∈ Token.tasks then
13                 Token.paths ← Token.paths ∪ current path of a_i
14             else
15                 Token.paths ← Token.paths ∪ Path2(a_i, Token)
16         Move all agents for one timestep
```

Kapitola 3
Ozobot

V tejto kapitole práca predstaví a analyzuje schopnosti Ozobota. Predstavené bude hardwardové vybavenie, jeho využitie a limity. V ďalšej sekcii práca rozoberie možnosti zmeny základného chovania definovaného výrobcom. Posledná sekcia bude venovaná predchádzajúcim výskumom na Ozobotoch v oblasti hľadania ciest. Momentálne sú k dispozícia dva typy robotov: Ozobot Evo a Ozobot Bit. V práci bol použitý model Evo, preto budú nasledujúce sekcie venované práve tomuto modelu.

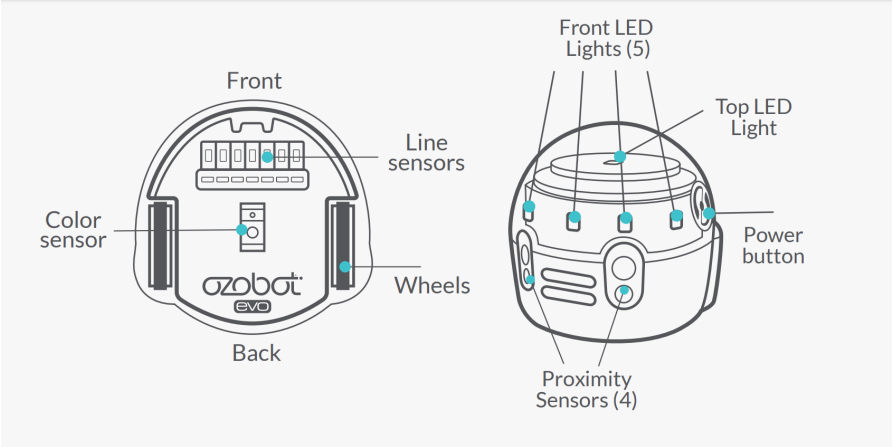
Ozobot [21] je malý interaktívny robot, ktorý používa RGB senzor na detekciu farebného povrchu na ktorom jazdí. Hlavnou schopnosťou robota je pohybovať sa po čiarach. Každý model má predom definované správanie, ktoré môže byť pozmenené dvoma spôsobmi farebné kódy (*Color Codes*) 3.3 a *OzoBlockly* 3.4.

Obr. 3.1 Ozobot Evo [22]

3.1 Hardwardové vybavenie

Základná stavba tela Ozobota je ukázaná na obrázku Obr. 3.2. Zo spodnej strany je možné vidieť niekoľko senzorov slúžiacich na sledovanie cesty[1] (*line sensors*). Ich účel je udržiavať orientáciu a smer pohybu robota. Každý zo senzorov rozpoznáva jas povrchu pod ním a podľa toho dokáže

[1] Cestu chápeme ako čiaru ktorú Ozobot sleduje



Obr. 3.2 Popis častí ozobota [22]

zistiť, kde sa nachádza cesta. Šírka senzoru dáva Ozobotovi možnosť rozoznať medzi otočkou o 90 stupňov a križovatkou. Za senzorom na sledovanie cesty sa nachádza snímač farieb (*color sensor*), ktorý rozlišuje farby povrchu, na ktorom sa Ozobot pohybuje. Senzor dokáže rozpoznať osem základných farieb: zelená, modrá, červená, čierna, biela, čierna, žltá, tyrkysová. Vďaka načítaným farám alebo farebným sekvenciám dokáže Ozobot meniť svoje správanie definované pomocou Color Codes 3.3. Senzor sa taktiež dá použiť ako jedna z variant načítania vlastného programu vytvoreného v OzoBlocky editore 3.4.

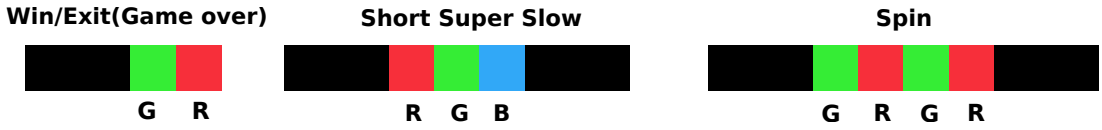
Pohyb Ozobota zabezpečuje motor a pár plastových kolies umiestnených po stranách. *Elektrický motor* vnútri dovoľuje robotovi pohyb dopredu aj dozadu. V základnom nastavení poskytuje motor rýchlosť 30 mm/s. Otáčanie je zabezpečené rotáciou kolies rôznou rýchlosťou. Robot zvláda pohyb po hladkých povrchoch ako displej či papier.

V tele Ozobota sú zabudované štyri *infračervené senzory* (dva vzadu a dva vpredu), ktoré rozpoznávajú objekty v okolí. Tie fungujú tak, že vyšlú infračervené svetlo, ktoré sa od prekážok. Ozobot vďaka svetlu, ktoré sa odrazilo pozná ako ďaleko sa prekážka nachádza. OzoBlockly editor môže byť využitý na zmenu citlivosti senzoru. Pri nastavení citlivosti na nulu Ozobot ignoruje prekážky, naopak pri maximálnom nastavení dokáže spozorovať prekážky až do 10 centimetrov. Manuál odporúča nastaviť citlivosť väčšiu ako 20.

Vo vybavení Ozobota sa nachádza *Bluetooth*, ktorý môže byť použitý na komunikáciu robota s aplikáciami preň vyvinutými. Bluetooth však neposkytuje možnosť prenosu informácií medzi skupinou Ozobotov. Zo zadnej strany má Ozobot micro-USB port, využívaný k napojeniu ku zariadeniam a napájaniu batérie. Pri potrebe nabiť Ozobota sa načerveno rozsvieti LED svetlo umiestnené zhora. Manuál uvádza, že nabitie batérie trvá približne 40 minút. Ak je batéria plne nabitá LED svetlo sa rozsvieti na zeleno. Takto plne nabitá batéria vydrží Ozobotovi na 60-90 minút.

3.2 Konfigurácia a pohyb

Pri zapnutí Ozobota alebo zmene povrchu, na ktorom sa pohybuje manuál odporúča robota *kalibrovať*. Kalibrácia pomáha Ozobotovi spresniť sledovanie cesty a s rozoznávaním farieb. Existujú 2 spôsoby kalibrácie. Ak sa robot bude pohybovať na bielom papieri, je potrebné naň nakresliť čierny kruh, ktorý bude o niečo väčší ako je Ozobot. Ozobota je potrebné položiť na kruh a držať tlačidlo Power Button 2-4 sekundy, až kým sa LED svetlo navrchu nerozsvieti na bielo. Ozobot začne s kalibráciou, otočí a dokončí kalibráciu tak, že vyjde z kruhu von. Následne, ak sa LED svetlo zasvieti nazeleno, znamená to, že kalibrácia bola úspešná. Naopak pri červenom



Obr. 3.3 Ukážka farebných kódov [24]

signále je nutné kalibráciu opakovať. Pri používaní displeja sa odporúča nastaviť maximálny jas pre zvýšenie presnosti. Ozobota je nutné položiť na biely povrch. Zvyšok kalibrácie pokračuje rovnako ako pri bielom papieri. [23]

V základnom nastavení sa Ozobot začne pohybovať hneď ako pod sebou nájde linku. Pri nájdení križovatky Ozobot vyberie náhodne cestu, ktorou sa bude pohybovať ďalej. Toto nastavenie môže byť upravené ako bude popísané v sekcii 3.4. Pohyb robot sa zastaví akonáhle senzor zaznamená koniec linky.

3.3 Farebné kódy (Color Codes)

Color Codes [25] sú unikátne sekvencie 2-4 farieb, ktoré dokážu pozmeniť chovanie Ozobota. Vďaka optickému senzoru Ozobot dokáže detektovať farbu cesty, po ktorej sa pohybuje. Farebné kódy sa zakladajú presne na tejto schopnosti. Ako Ozobot nasleduje linku, sníma sekvencie a ak spozná nejakú preddefinovanú, vykoná výrobcom definované správanie.

Na definovanie farebných kódov sú využívané štyri z ôsmich farieb, ktoré snímač dokáže rozpoznať: zelená, modrá, červená, čierna. Manuál popisuje 29 rôznych sekvencií farieb. Podľa funkčnosti sa dajú rozdeliť do piatich kategórii podľa ich funkčnosti na: rýchlosť, smer a špeciálne pohyby, časovače, výhra/koniec, počítadlá. Prvá kategória poskytuje zmenu rýchlosti od pomalej až po veľmi rýchlu. Smer určuje robotovi ako sa otočiť buď okamžite (napríklad príkazy: U-turn[2], zmeň linku doprava), alebo keď senzor načíta istú udalosť(napríklad príkaz: na priesečníku doprava). V tejto kategórii sa nachádza aj sada pohybov, ktoré sa skladajú z viacerých inštrukcií. Ide napríklad o príkaz Spin, kde sa Ozobot otočí dvakrát dookola a následne pokračuje rovno. Časovače hovoria Ozobotovi aby sa pozastavil alebo počítal sekundy. Príkladom je príkaz Pauze, ktorý zastaví Ozobota na 3 sekundy. Kódy výhra/koniec sa používajú na znázornenie, že Ozobot dokončil sériu príkazov. Počítadlá hovoria Ozobotovi aby počítal výskyty istých javov na linke (otočiek, zmien farieb, križovatiek). Následne po 5 výskytoch sa zastaví a LED svetlo zabliká načerveno. Ukážku niektorých farebný kódov je možné vidieť na obrázku Obr. 3.3. Farebné sekvencie v príklade sú čítané zľava doprava.

Pre správne fungovanie manuál uvádza odporúčania, ktoré musia byť dodržané pre správne načítanie. Každá farba v sekvencii by mala mať tvar štvorca s rozmermi $5x5mm$. Ak by boli farby pridlhé, robot by si mohol myslel, že ide o zmenu farby cesty. Šírka jednotlivých štvorcov by mala byť rovnaká ako šírka linky. Medzi jednotlivými sekvenciami ako aj pred a za križovatkami by mala byť medzera minimálne o veľkosti $2,5cm$. Ozobot musí pred a po farebnej sekvencii sledovať čiernu cestu. Ďalej sa neodporúča používať farebné kódy na rohoch cesty, kde sa Ozobot otáča. Sekvencie zložené z 2 farieb musia byť umiestnené na konci linky.

Farebné kódy predstavujú spôsob ako zmeniť Ozobotove správanie počas sledovania cesty. Ide však o obmedzený spôsob, keďže výrobca neponúka možnosť definovať si vlastné farebné kódy.

[2] Otočenie o 180 stupňov

3.4 OzoBlockly

OzoBlockly je vizuálny programovací jazyk, ktorý sa používa na programovanie Ozobota. Editor využívajúci tento jazyk je dostupný na stránke [26]. Poskytuje 5 rôznych úrovní, od začiatočníka až po experta, v ktorých je možné kód skladať. Každá úroveň uvádza nové zložitejšie koncepty, s ktorými je možné pracovať. Ide o jazyk, kde sa jednotlivé prvky postupne skladajú do seba až nakoniec vytvoria hotový program.

Existujú 2 spôsoby ako preniesť OzoBlockly kód priamo do Ozobota. Prvým z nich je načítať program cez *Flash Loading*. Metóda Flash Loadingu funguje vďaka optickému senzoru, pri ktorej sa celý program preloží do farebných sekvencií. Ozobot musí byť umiestnený na predom vyhradenom mieste, ktoré je vyznačené na obrazovke. Po začatí načítania sa spustí sekvencia farieb, do ktorých bol program zakódovaní. Počas načítania musí na hornej strane Ozobota blikať zelené LED svetlo, ktoré značí, že program sa načítava úspešne. Ak sa farba LED svetla zmení na červenú je nutné načítanie spustiť znova. Táto metóda môže v závislosti od dĺžky programu trvať niekoľko sekúnd až minút. Druhým spôsobom je načítať program cez Bluetooth pomocou aplikácie Evo. Ide o mobilnú aplikáciu, kde je dostupný OzoBlockly editor. Pred načítaním je potrebné Ozobota pripojiť k Evo aplikácii. Následne pomocou tlačidla Run spustiť načítanie.

Po načítaní je program dostupný v Ozobotovi, kým není prepísaný iným. Program sa pustí pri dvojitom stlačení tlačidla Power Button.

3.5 Predchádzajúce práce na Ozobotoch

Ozoboti tvoria vhodnú variantu agentov, na ktorých je možné *testovať* a *vizualizovať* MAPF plány. Väčšinou však ide o práce, ktoré používajú jednorázové na papieri vytlačené cesty, ako prostredie pre Ozobota. Príkladom takejto práce je [27], kde autori vytvorili mriežkovanú mapu so štartovacími a koncovými bodmi pre agentov. Túto inštanciu vyriešil MAPF algoritmus, následne bola preložená pomocou OzoBlockly editoru a zadaná do Ozobotov. Každý Ozobot si teda pamätal svoju cestu a orientoval sa pomocou križovatiek na mape. Nevýhodou práce bolo, že Ozobotom musel byť každú simuláciu preložený a nahratý nový plán.

Tento problém vyriešil framework *ESO-NAV* [28, 29], ktorý nazačiatku naplánuje plán pre agentov a následne vykresľuje cesty na obrazovke. Taktiež využíva Ozobot Evo agentov a MAPF inštancie rieši SMT-CBS 2.1.5 algoritmus. Nasledujúce sekcie budú venované práve tomuto frameworku.

3.5.1 Reprezentácia mapy

Mapa je skonštruovaná z $2d$ mriežkovaného poľa rozdeleného na dlaždice. Jednotlivé dlaždice sú následne zafarbené podľa ich účelu. Ak je dlaždica štartovacou pozíciou agenta, tak je zafarbená nazeleno. V prípade, že agent končí svoju cestu na danej dlaždici, tak je zafarbená na červeno. Ak dlaždica tvorí štart aj koniec vyplní sa oboma farbami šachovnicového vzoru. Medzi dlaždicami nemusí byť vždy cesta. V takom prípade je medzi nimi vykreslená stena. Farby však musia mať nízku sýtosť aby neboli zaznamenané senzorom Ozobota pri prejazde. Príklad mapy možné vidieť na ukážke Obr. 3.4.

Atribúty mapy je možné nastavovať pomocou konfiguračného súboru. Nastaviť je možné veľkosť dlaždice, hrúbku steny, ako dlho má trvať presun agenta po 1 dlaždici (dĺžka jedného časového kroku) a vlastnosti vykresľovanej cesty. Framework taktiež implementuje editor na vytváranie MAPF inštancií, kde je možné nastaviť začiatky a konce ciest pre agentov ako aj pridávať a odoberať steny.

3.5.2 Vykresľovanie ciest

Základ simulácie tvoria *farebné cesty*, ktoré sa zobrazujú na monitore a navigujú agenta. Cesty sú zostavené zo segmentov, ktoré sa postupne vykresľujú pred agentom a strácajú za ním. Je podstatné voliť dĺžku segmentu tak, aby sa neprekrývala s inými a nedochádzalo tak k chybnému načítaniu ciest medzi agentmi.

Cieľom farebnosti ciest je zabrániť *desynchronizácii*, ktorá by mohla nastať zrýchlením, poprípade spomalení Ozobota. Farby na ceste zastupujú rôzne rýchlosti, ktorými sa Ozobot pohybuje. Každá z farebných ciest sa skladá z 3 farieb: modrá, čierna a červená. Modrá farba sa vyskytuje na začiatku a má priradenú najnižšiu rýchlosť. Ak by teda agent neočakávane zrýchlil modrá farba ho spomalí a zníži riziko desynchronizácie. Čierna rýchlosť zastupuje strednú rýchlosť (pomalšie ako červená a rýchlejšie ako modrá). Touto rýchlosťou sa Ozobot pohybuje pri pohybe rovno. Nachádza sa v strede medzi modrým a červeným segmentom cesty. Posledná červená farba je najrýchlejšia a nachádza sa na konci segmentu. Jej úloha je presným opakom modrej a to zrýchliť Ozobota v prípade neočakávaného spomalenia. Toto správanie Ozobota môže byť zadefinované pomocou aplikácie OzoBlockly popísanej v sekcii 3.4. Správanie Ozobota je možné ovplyvniť iba pri akcii, ktorá ho vyvolá. Ozoblockly poskytuje poskytuje iba jedinú takúto akciu a to: nasleduj cestu do ďalšej križovatky alebo do konca čiary. Autori preto v práci zavádzajú *umelo vytvorené križovatky*, pri ktorých k čítaniu farieb dochádza.

Aby Ozobot dokázal správne snímať križovatky je potrebné dodržiavať nasledujúce odporúčania. Hlavnými faktormi je ich dĺžka aby senzor Ozobota zachytil križovatku a ako často sa majú križovatky vykresľovať. Vo frameworku sa autori rozhodli vykresľovať tri križovatky na jednu dlaždicu. Hrúbka je nastaviteľná podľa potrieb používateľa. Tieto križovatky sú vykresľované len ak agent prechádza dlaždicou rovno. Naopak ak dochádza k otočke alebo agent na dlaždici stojí križovatky nebudú vykreslené. Farba týchto križovatiek sa mení spoločne so zmenou farebných segmentov. Autori týmto zaručujú, že ak je agent spomalený/zrýchlený dôjde k úprave rýchlosti podľa jeho polohy. Výnimku tvoria posledné križovatky pred otočkou, ktorá je vždy modrá aby zabezpečila zbrzdila agenta pred otáčaním sa.

Počas cesty agent stále sníma Color Codes 3.3. Framework ich využíva pokiaľ agent stoji na dlaždici a jeho ďalšia akcia je smerom ktorým prišiel. Vtedy je na dlaždici vykreslený U-turn color code, ktorý otočí agenta správnym smerom.

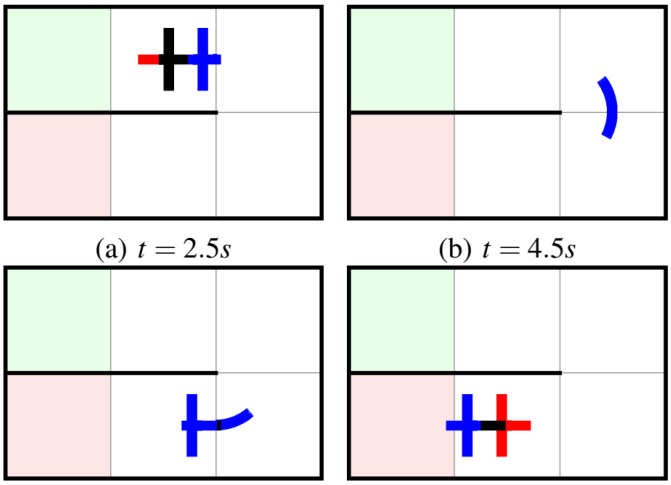
Ukážka cesty pre jedného agenta je na obrázku Obr. 3.4

3.5.3 Správanie agentov

Pre správne fungovanie frameworku muselo byť základné nastavenie Ozobota zmenené. Zmena bola vytvorená pomocou programu OzoBlockly a následne načítaná do Ozobotov. Algoritmus správania Ozobotov je možné vidieť v ukážke Algoritmus 5. Program pracuje v cykle od načítania až kým nebude Ozobot vypnutý alebo nebude načítaný iný program. Nazačiatku je načítaná rýchlosť podľa toho akej farby je linka pod Ozobotom. Pri načítaní križovatky je upravené správanie tak, aby Ozobot nevyberal cestu náhodne ale išiel stále rovno (riadok 3-4). V inom prípade, a to ak robot nezaznamená linku pod sebou, robot sa zastaví (riadok 5-6). Následne Ozobot pokračuje kým nenarazí na ďalšiu križovatku alebo koniec linky. Funkcia *GetSpeedFromLineColor* načítava farby pomocou senzoru a podľa nich vráti rýchlosť pohybu.

3.5.4 Limitácie

Práca [29] popisuje limitácie frameworku, ktoré boli odhalené pri experimentoch. Jedna z limitácii bola odhalená v plánovaní MAPF algoritmom. V plánoch sa môže objaviť cesta, keď agent vchádza do dlaždice v jednom časom okamžiku a následne hneď v ďalšom ju potrebuje opustiť



Obr. 3.4 Ukážka postupného vykresľovania cesty [28]

Algoritmus 5: Ozobot behaviour

```
1 while true do
2     set Ozobot speed (in mm/s): GetSpeedFromLineColor()
3     if straight line ahead then
4         Ozobot direction: straight
5     else
6         Ozobot stop
7     Follow line to next intersection or line end
```

cestou ktorou prišiel. Takúto situáciu nazývajú *oscilácia*. Pre Ozobotov je nemožné v takomto krátko okamžiku uskutočniť U-turn. Autori preto navrhli použiť iný MAPF algoritmus, ktorý by zabraňoval vzniku tohto manévru. V práci [29] boli manuálne odstránené všetky takéto výskyty.

Citlivosť senzorov Ozobota tvori ďalšiu limitáciu. Roboti majú nesprávne zaznamenávať farby a taktiež vychádzať zo svojich ciest. Z tohto dôvodu musia byť senzory často kalibrované aby zlepšilo ich čítanie. Aj napriek tomu sú Ozoboti citlivý na svetelné podmienky. Tiene spôsobené iným svetelným zdrojom môžu spôsobiť, že ich senzor nasníma ako cestu a preto sa vychýlia. Tiene taktiež môžu zmeniť zafarbenie segmentu cesty a Ozobot nemusí už reagovať ako to mal naprogramované.

Pre účeli práce je limitáciou taktiež, že framework musí dopredu poznať všetky cesty predtým ako ich začne vykonávať. Cesty musia prejsť predspracovaním, ktoré určí ako majú byť segmenty cesty vykreslené. Ide napríklad o segmenty pred otáčkou, kde musí byť vykreslená modrá križovatka na spomalenie agenta a tým nedochádzalo k desynchronizácii.

Kapitola 4
Návrh simulácie skladiska

Kapitola sa venuje návrhu modelu zmenšeného modelu skladiska, na ktorom budú vykonávane experimenty na Ozobotoch. Nazačiatku kapitoly práca špecifikuje grafické rozhranie a zobrazovanie prvkov MAPD inštancie. Následne popíše algoritmus, ktorý bude riešiť MAPD inštancie a úpravy, ktoré museli byť zavedené z ohľadom na limitácie reálneho prostredia a Ozobotov. Nakoniec budú popísané, zmeny už existujúceho riešenia pre vykresľovanie ciest a editovanie máp.

4.1 Grafické rozhranie

Práca priamo naväzuje na *ESO-Nav* framework, ktorý používa na zobrazenie ciest. Framework bol zvolený kvôli jeho vlastnosti vykresľovať a mazať cesty agentov. Keďže simulácia bude vykonávať niekoľko plánov v jednom behu, budú aj cesty agentov zložitejšie. Z tohto dôvodu je prístup je mazanie ciest podstatnou vlastnosťou, ktorú práca bude využívať.

Framework bol taktiež použitý na zobrazovanie jednotlivých prvkov mapy a MAPD inštancie, pričom musel byť rozšírený z MAPF na MAPD variantu. Táto sekcia ďalej popisuje ako práca reprezentuje mapu skladiska a pridanie výpisu úloh.

4.1.1 Reprezentácia skladiska pre ručné zadávanie úloh

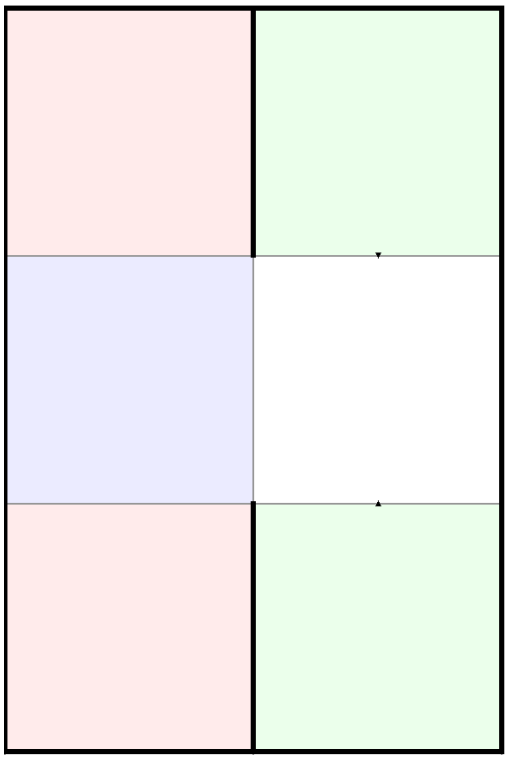
Skladisko je reprezentované vo formáte *YAML*. Každá dlaždica na mape má priradenú dvojicu súradníc $[x, y]$ podľa, ktorých je zadefinovaná ich pozícia. Agentom je priradené identifikačné meno (*name*) a ich štartovacie pozície (*start*). Aby bola MAPD inštancia valídna a pre správne fungovanie algoritmu, ktorý ju rieši musia byť oba tieto atribúty unikátne pre každého agenta. Mapu definujú 3 prvky: rozmer (*dimensions*), steny (*walls*) a ostatné koncové body (*non-task endpoints*). Rozmery predstavujú dvojicu šírka a výška mapy. Steny sú definované ako dvojica polôh dlaždíc určená dvojicou $[x, y]$. Ide o hranu medzi dvomi dlaždicami, ktorú agent nemôže použiť. Ostatné koncové body predstavujú parkovacie miesta z *definície MAPD* a sú určené polohou dlaždice. Práca bude za parkovacie miesto považovať ako ostatné koncové body, tak aj štartovacie miesta agentov. Poslednú časť tvoria úlohy (*tasks*). Každej úlohe je priradený počiatočný čas (*start_time*), kedy ju má systém priradiť do množiny nevykonaných úloh. Čas je zadaný v časových krokoch (*timestep*). Miesto vyzdvihnutia (*start*) a miesto donášky (*goal*) úlohy definuje poloha dlaždice. Pre správne fungovanie algoritmu musí mať každá úloha unikátny identifikátor (*task_name*).

Príklad skladiska reprezentácie modelu skladiska je ukázaný na Obr. 4.1. Obrázok napravo zobrazuje toto skladisko v grafickej podobe. Štartovacie polohy agentov ako aj ostatné koncové

```yaml
agents:
-   start: [0, 0]
    name: agent0
-   start: [2, 0]
    name: agent1

map:
    dimensions: [3, 2]
    walls:
        - locationA: [0, 0]
          locationB: [0, 1]
        - locationA: [ 2, 0 ]
          locationB: [ 2, 1 ]
    non_task_endpoints:
        - !!python/tuple [0, 0]
        - !!python/tuple [2, 0]

tasks:
-   start_time: 0
    start: [1, 1]
    goal: [0, 1]
    task_name: task0
-   start_time: 2
    start: [1, 1]
    goal: [2, 1]
    task_name: task1
```



Obr. 4.1 Príklad skladiska v yaml (naľavo) a v grafickej podobe (napravo)

body sú znázornené zelenou farbou. Miesta vyzdvihnutia majú modrú farbu a miesta donášky červenú. Ostatné grafické prvky ostávajú rovnaké ako v ESO-Nav. Výpis kódu naľavo v Obr. 4.1 zobrazuje definuje túto grafickú podobu vo formáte YAML.

4.1.2 Reprezentácia skladiska pre generovanie úloh

V prostredí skladu sa množstvo úloh, ktoré systém musí spracovať mení. Na simuláciu reálneho skladu teda práca bude náhodne generovať úlohy. Je však potrebné do MAPD inštancie, presnejšie do definície mapy pridať koncové miesta úloh. Generovanie úloh potom funguje nasledovne. Nazačiatku sú zostrojené množiny koncových bodov vyzdvihnutia a koncových bodov donášky. Každej úlohe bude náhodne pridelený 1 koncový bod z oboch množín, ktorý reprezentuje jej miesto vyzdvihnutia a donášky. Popis generovania počiatočného času sa nachádza v sekcii 5.1.

4.1.3 Zobrazovanie úloh

Pri plánoch, kde sa vyskytuje viacero úloh nie je jasné, ktoré úlohy sú *aktívne*, ktoré boli už *dokončené*, a ktoré sa ešte *nezačali vykonávať*. Na zlepšenie prehľadnosti pri vykonávaní bolo preto pridané zobrazovanie úloh.

Ide o výpis umiestnený na spodnej časti displeja, ktorý spočíva v 3 nasledujúcich atribútov:

- Neaktívne úlohy (*inactive tasks*) vypisuje úlohy, ktorých počiatočný čas je menší ako aktuálny čas behu systému.
- Aktívne úlohy (*active tasks*) vo výpise tvoria úlohy, ktorých počiatočný čas je väčší ako aktuálny čas behu systému ale agent, ktorý ich vykonáva ešte nedorazil na ich miesto donášky.
- Ukončené úlohy (*finished tasks*) sú úlohy, ktoré boli vykonané a teda doručené na miesto donášky.

Úloha môže byť preradená z jednej množiny do druhej jedine v momente, kedy sa agent nachádza v strede dlaždice. To je zabezpečené pomocou prepočtu časových krokov, v ktorých je určený plán agentov na reálny čas v milisekundách. Prepočet sa prevádza s nastaviteľnou konštantou umiestnenou v konfiguračnom súbore, ktorá určuje ako dlho trvá 1 časový krok.

Príklad behu systému s zobrazovaním úloh je možne vidieť na obrázku Obr. 4.2. Agent začína na svojej štartovacej pozície. Nazačiatku systém naplánuje cesty pre MAPD inštanciu. Následne agent začína vykonávať úlohu *task0*, tým sa úloha pridáva do množiny aktívnych úloh. Úloha *task1* ostáva neaktívna. Následne keď agent dosiahne pozíciu donášky úlohy *task0*, úloha sa stáva ukončenou.

4.2 Plánovanie ciest

Simulátor skladiska používa na riešenie MAPD inštancií existujúcu implementáciu *Token Passing* (ďalej už ako TP) algoritmu, napísaného v jazyku *Python*. Implementácia poskytuje ako základnú variantu TP, tak aj variantu rozšírenú o robustnosť k-TP 2.2.4. *Path1* a *Path2* funkcie z ukážky Algoritmus 4 sú realizované použitím implementácie *CBS*, ktorá dokáže vypočítať bezkolízne cesty pre agenta pri každej iterárii programu. Nízko-úrovňové prehľadávanie pre CBS vykonáva *A-star* algoritmus.

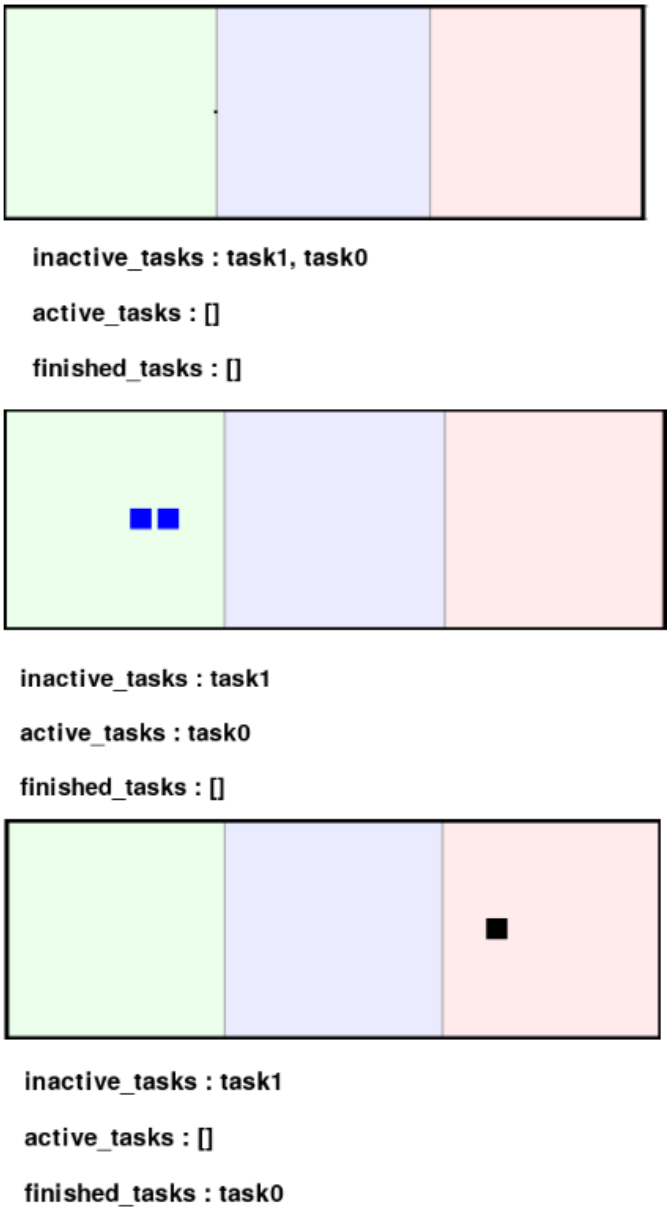
Pri výbere algoritmu boli zohľadnené limitácie ESO-Nav frameworku 3.5.4. Keďže framework potrebuje dopredu poznať plán pre agentov aby dokázal určiť počiatočný smer agenta a správne vykresľoval cesty, práca bude implementovať offline variantu MAPD problému 2.2.5. Systém preto nechá TP spočítať plán pre celý beh programu a následne ho vráti do frameworku. Výhodou je, že pokiaľ by bol v budúcnosti použitý iný framework na vykresľovanie ciest, ktorý by poskytoval možnosť dynamicky meniť cesty, algoritmus TP by nemusel byť už menený.

Nasledujúce sekcie popisujú úpravy, ktoré museli byť prevedené na algoritme TP vzhľadom na kompatibilitu s ESO-Nav a limitácie Ozobotov.

4.2.1 Úprava mapy

Framework EVO-Nav pracuje s mapou, kde medzi dlaždicami zavádza *hrany* alebo *steny* podľa toho, či tam existuje cesta, či nie. Token passing na druhej strane používa *prekážky*, ktoré nedovoľujú vstupu agenta ani úlohy na definovanú dlaždicu. Algoritmus preto bude prevádzať reprezentáciu prekážok na steny. Tento prístup umožňuje väčšiu variabilitu pri modelovaní skladiska. Dokáže ako zabrániť agentom vstúpiť na dlaždicu, tým že ju obklopí stenami, tak aj bežnú funkciu stien. Príklad zmeny je možné vidieť v Obr. 4.3. Naľavo je reprezentácia prekážky. Napravo sa nachádza prepísanie prekážky pomocou stien, kde sa zakáže vstup zo všetkých okolitých dlaždíc, čo zamedzí použitie dlaždice pre každého agenta.

So stenami sa potom pracuje nasledovne. Pri načítavaní mapy si program uloží všetky steny do množiny a predá ju prostrediu. S prostredím následne pracuje A-star algoritmus. Steny sú využité v momente generovania susedného stavu pri prehľadávaní priestoru, keď A-star overuje, či sa dá stav dosiahnuť (funkcia *stateValid*), a či je prechod doň možný (funkcia *transitionValid*).



Obr. 4.2 Beh automatizovaného skladiska s výpisom úloh

```
obstacles:
    - !!python/tuple [1, 1]
```

```
walls:
    - locationA: [ 1, 0 ]
      locationB: [ 1, 1 ]
    - locationA: [ 0, 1 ]
      locationB: [ 1, 1 ]
    - locationA: [ 2, 1 ]
      locationB: [ 1, 1 ]
    - locationA: [ 1, 2 ]
      locationB: [ 1, 1 ]
```

Obr. 4.3 Príklad zmeny reprezentácie prekážky v YAML (prekážka naľavo, steny napravo)

Algoritmus 6: Get Neighbours for A*

```
1  getNeighbours (state)
2      neighbours ← []
3      ...
4                                                    ▷ Rotate state
5      new_state ← State(state.time + 1, state.location, (state.rotation + 180) % 360)
6      if stateValid(n) then
7          neighbours.add(n)
8                                                    ▷ Down state
9      new_state ← State(state.time + 1, Location(state.location.x, state.location.y - 1), 0)
10     if stateValid(n) and transitionValid(n) and state.rotation != 180 then
11         neighbours.add(n)
12     ...
13     return neighbours
```

Časť algoritmu generovania susedného stavu, kde sú tieto funkcie využité je možné vidieť na ukážke Algoritmus 6. Pridaním stien sa obe tieto funkcie museli prepísať.

Pôvodne overovanie prechodu kontrolovalo, či nedochádza ku kolízii na hrane medzi agentmi. Keďže hrana môže obsahovať aj stenu, je potrebné validovať aj túto možnosť. Bolo teda pridané overovanie, či hrana z polohy A, do polohy B je obsiahnutá v množine stien uloženej v prostredí. Ak áno, tak tento prechod nie je možný a teda nový stav je neplatný.

Implementácie validácie stavu kontroluje, či nedochádza v novom stave ku konflikte na vrchole. Táto situácia sa mohla stať ak sa už iný agent nachádzal na dlaždici, ktorá je obsiahnutá v novom stave a v prípade, že dlaždica obsahovala prekážku. Z funkcie bolo teda odstránená kontrola prekážok, čo zabránilo implementačným problémom.

4.2.2 Pridanie rotácii

CBS implementované v TP, generuje *oscilácie* spomenuté v limitáciách ESO-Nav 3.5.4. Tieto oscilácie sú spôsobené tým, že A-star v CBS neberie v úvahu čas, ktorý Ozobot potrebuje na rotáciu o 180 stupňov (U-turn). Oscilácie niesu pri CBS častým javom, keďže A-star sa vďaka použití heuristiky snaží približovať cieľu. Oscilácia by však znamenala vrátenie sa späť o dlaždicu.

Z tohto dôvodu A-star preferuje všetky ostatné dlaždice, ak je ich možné použiť. V opačnom prípade nastáva oscilácia. Ich hlavný výskyt je v úzkych oblastiach s veľkým počtom agentov na dlaždice. Príkladom takejto situácie je keď agent musí uvoľniť cestu inému agentovi, pričom má cestu dopredu zablokovanú.

TP plánuje s CBS *dvakrát*. Najprv do miesta vyzdvihnutia a následne do miesta donášky. Tieto plány sa následne spoja do jedného a uložia sa do tokenu. CBS berie polohu vyzdvihnutia a polohu donášky ako počiatočnú polohu agenta, zatiaľ čo neberie do úvahy rotáciu agenta. Ak teda agent má priradenú úlohu predtým ako príde na miesto vyzdvihnutia/donášky, CBS môže vypočítať cestu, ktorá začína miestom z ktorého agent prišiel hneď po skončení predošlého plánu. Príklad oscilácie je možné vidieť na Obr. 4.4. V čase $t = 1$ agent prichádza do miesta vyzdvihnutia v čase $t = 2$ by mal nasledovať *farebný kód*, ktorý signalizuje U-turn. Keďže plán nemá *wait* akciu, kde by bol U-turn zobrazený je vykreslená iba cesta späť. Agent sa preto neotočí a v čase $t = 3$ stratí vykreslenú cestu.

Prvotné riešenie tohto problému bolo pridanie wait akcie po naplánovaní cesty pre agenta. Potom čo CBS vráti plán do TP sa skopíruje *posledný krok plánu* a zväčší sa čas tohto skopírovaného stavu o 1. To zabezpečí čas pre otočenie Ozobota na miestach vyzdvihnutia a donášky a zároveň umožní frameworku vykresliť farebný kód pre otočenie.

Predchádzajúce riešenie síce rieši oscilácie na začiatku a konci plánov no nerieši oscilácie medzi. Tie sa síce vyskytujú zriedka, no pri zložitejších plánoch sú pravdepodobnejšie. Z tohto dôvodu bolo pridané do celého TP implementácia rotácii. To znamenalo úpravu MAPF definície a to pridaním nového typu akcie *rotate*. Pre každý smer, ktorým sa môže Ozobot pohybovať bol definovaný stupeň rotácie následovne:

- Hore (kladný pohyb agenta po y súradnici): 0
- Dole (záporný pohyb agenta po y súradnici): 180
- Doprava (kladný pohyb agenta po x súradnici): 90
- Doľava (záporný pohyb agenta po y súradnici): 270

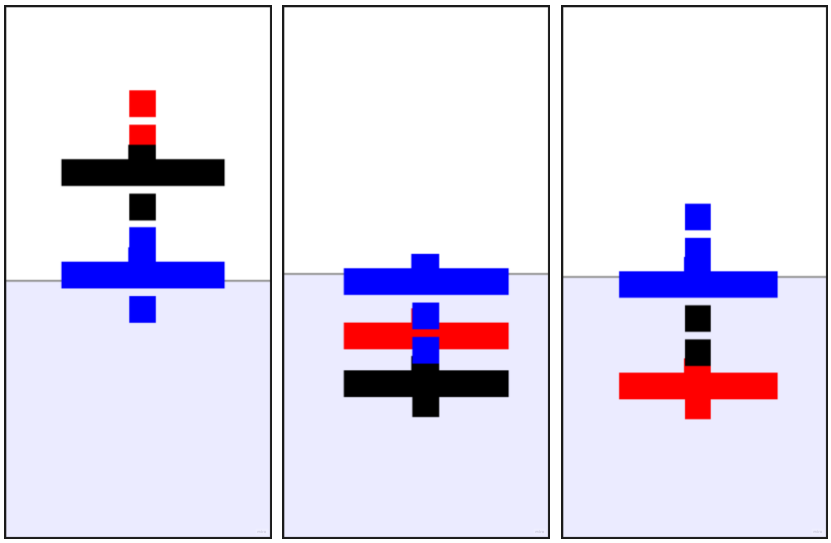
Nazačiatku si program inicializuje token pre každého agenta s predvoleným otočením *hore*. Pred plánovaním cesty pre agenta sa rotácia z tokenu predáva do CBS a následne do A-star, ktorý s ním následne pracuje. Nutné bolo pridanie rotácie do každého nového stavu. Do funkcie na generovanie susedných stavov (*getNeighbours*) boli pridané 2 funkcionality. Prvou je vytváranie stavu, kde agent rotuje o 180 stupňov. Tento stav simuluje U-turn pri vykonávaní plánu Ozobotom. Zabezpečí teda agentovi čas na rotovanie. Druhou je kontrola validity stavov a to, že nový stav nesmie byť v opačnom smere [1] než akým je otočený agent. Táto podmienka zabezpečí, že sa agent nevykoná osciláciu.

Časť pseudoalgoritmu, na generovanie susedných stavov, je možné vidieť v ukážke Algoritmus 6. Nazačiatku sa inicializuje prázdne pole susedných stavov *neighbours*. Prvú zmenu znázorňuje vytváranie stavu *rotate* (riadok 5-7), ktorý predstavuje rotáciu agenta o 180 stupňov a jeho následnú validáciu. Kontrola rotácie (riadok 9-11) je zabezpečená pridaním atribútu rotate do objektu *State* a jeho následnou validáciou. Na konci funkcia vráti pole *neighbours*. Ukážka zachycuje len vytváranie 2 z 6 možných stavov, netvorí plný pseudoalgoritmus reálnej funkcie.

4.3 Úpravy ESO-Nav

Okrem grafickej stránky bolo v ESO-Nav pozmenené aj spracovanie plánu. Ten bolo nutné upraviť aby dokázal spracovať cesty pre každý valídny MAPD plán. Taktiež bol vytvorený editor máp skladiska odvodený od pôvodného editora MAPF máp. Nasledujúce sekcie sa venujú týmto dvom zmenám. Ukážka výsledného behu automatizovaného skladiska je následne ukázaná na obrázku Obr. 4.5

[1] $|state.rotaton - new\_state.rotation| \; ! = 180$



(a) t=1 **(b)** t=2 **(c)** t=3

Obr. 4.4 Priklad oscialácie

4.3.1 Spracovanie plánu

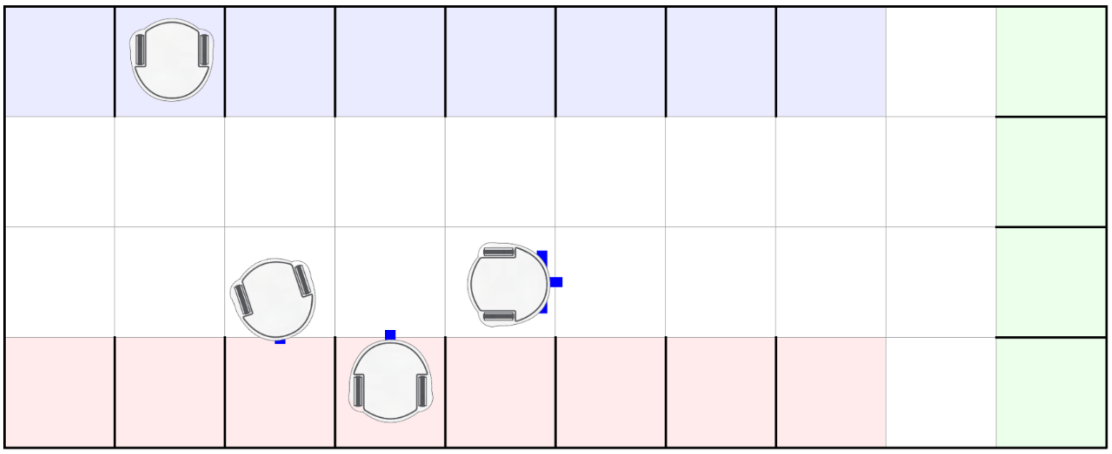
Problém so spracovávaním plánu sa vyskytoval v prípade, keď algoritmus Token Passing vrátil frameworku *prázdnu cestu* jednému alebo viacerým agentom. V prípade MAPD problému má prázdna cesta význam, keďže agent sa počas celého behu programu nemusí pohnúť zo svojho parkovacieho miesta. Naopak v prípade simulácie MAPF je prázdna cesta zbytočná. Preto autori túto variantu neimplementovali.

Presnejšie sa problém vyskytoval, keď framework dostal pole *steps* ako hodnoty *Null*. Pri parsovaní pozícii následne siahal mimo pole, čo vyhadzovalo výnimku. Riešením bolo vrátiť prázdne pole v prípade, že táto situácia nastala.

4.3.2 Editor

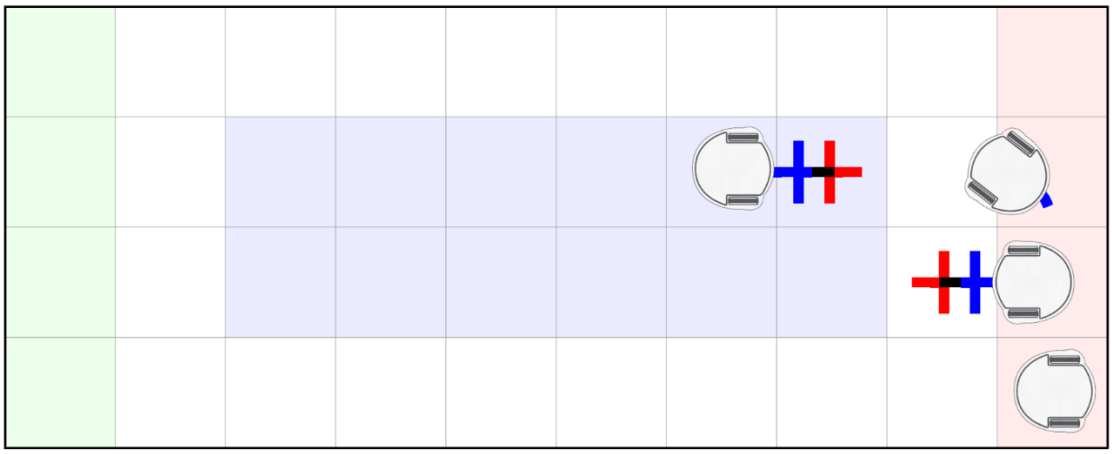
Na urýchlenie vytvárania máp automatizovaného skladiska bol vytvorený *editor*. Editor preberá časti implementácie z Eso-Nav. Po štarte sa editor dopytuje užívateľa na rozmery mapy, ktorú má vytvoriť. Následne je vytvorená *prázdna mapa* s požadovanými rozmermi a ohraničením. Klávesami na klávesnici si užívateľ vyberá z módou: *štart*, *vyzdvihnutie*, *donáška*, *ostatný koncový bod* a *stena*. Kliknutím na dlaždicu alebo medzi dlaždice (v móde stena) sa podľa zvoleného módu vytvorí alebo odstráni vybraný atribút mapy na zvolenom mieste. Po dokončení vytvárania inštancie editor túto mapu uloží vo formáte YAML k ostatným mapám v priečinku, pričom jej meno generuje buď automaticky alebo ho môže zadať užívateľ sám.

Editor je nastavený na prípravu máp pre náhodné generovanie úloh. Do súboru treba však pridať *funkciu hustoty pravdepodobnosti* 5.1, ktorú chce užívateľ použiť a jej parametre. Prednastavenou hodnotou je exponenciálna funkcia hustoty pravdepodobnosti s parametrom $\lambda = 5$.



inactive_tasks : task0, task21, task29, task13, task20
active_tasks : task12, task1, task6, task22, task24, task15, task2, task3, task17, task18, ...
finished_tasks : task8

(a)



inactive_tasks : task17, task16, task18, task7, task12, task13, task4, task11, task15
active_tasks : task14, task24, task28, task5, task8, task1, task0, task9, task29, task3, ...
finished_tasks : task25, task10, task6

(b)

Obr. 4.5 Ukážka behu modelu automatizovaného skladiska

Kapitola 5
Experimentálne výsledky

V tejto kapitole práca testuje a hodnotí ako upravený algoritmus Token Passing, tak aj vykonávanie plánov na Ozobotoch. Pred samotným testovaním je predstavený spôsob vytvárania počiatočného času, pre náhodné generovanie úloh. Potom nasleduje samotné testovanie rozdelené na 2 časti. V 1. časti bude testovaný vplyv počtu jednotlivých prvkov MAPD inštancie na výsledky algoritmu. V 2. časti práca otestuje úspešnosť vykonávania plánov Ozobotov na náhodne vygenerovaných úlohách. Štruktúra oboch častí je zhodná. Najskôr sú predstavené mapy, na ktorých budú testy vykonávané. Následne budú predstavené výsledky experimentov ako aj metriky, ktorými boli hodnotené. Nakoniec sú tieto výsledky hodnotené.

Experimenty boli vykonávané na 2 monitoroch. Monitor od notebooku *Dell Latitude 5531, 15.6ÏPS antireflexný 1920 × 1080*. Externý monitor *Philips 243V7QJABF, 24"LCD monitor Full HD 1920 × 1080*. Nastavené sú na 100% jas, kvôli zlepšeniu funkčnosti senzorov Ozobotov. Vykonávanie plánov ako aj kalibrácia boli vykonané v miestnosti s uspokojujúcim svetelným zdrojom, aby boli dosiahnuté čo najlepšie podmienky.

5.1 Generovanie počiatočných časov úloh

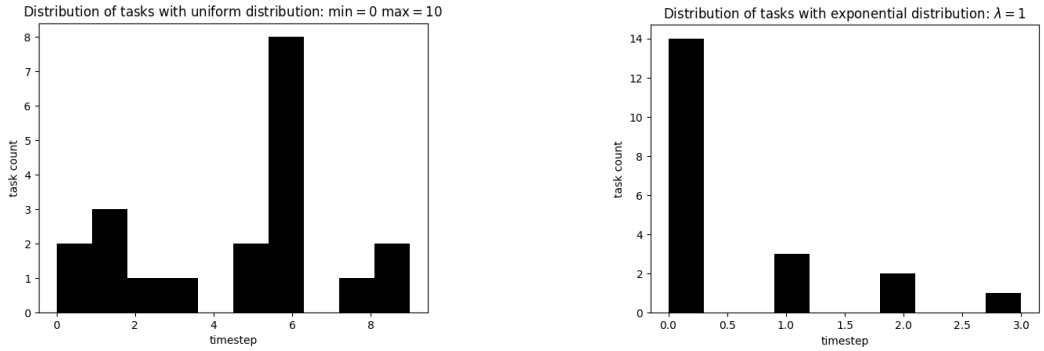
Na simulovanie rôznych situácií, pod ktorými sa sklad môže nachádzať boli v práci na generovanie počiatočného času úloh použité funkcie hustoty pravdepodobnosti pre nasledujúce pravdepodobnostné rozdelenia:

- *Exponenciálne rozdelenie* 5.1b modeluje nárazovú stresovú situáciu, kedy najväčšia záťaž je kladená na systém hneď od štartu programu a postupne klesá. Rozdelenie je možné upravovať pomocou parametru λ. Pri zmenšení parametra záťaž klesá, naopak pri zväčšení stúpa.
- *Uniformné rozdelenie* 5.1a znázorňuje príklad skladiska, kedy všetky úlohy prichádzajú rovnomerne, ide teda o situáciu stáleho chodu skladu. Upraviť je ho možné pomocou parametrov *min* a *max*, ktorá udávajú minimálny a maximálny časový okamžik, kedy môže prísť úloha do systému.

Obe rozdelenia boli získane pomocou knižnice *NumPy*, pričom sa výsledky zaokrúhľovali na celé čísla. Tieto čísla potom značia hodnotu *timestep*.

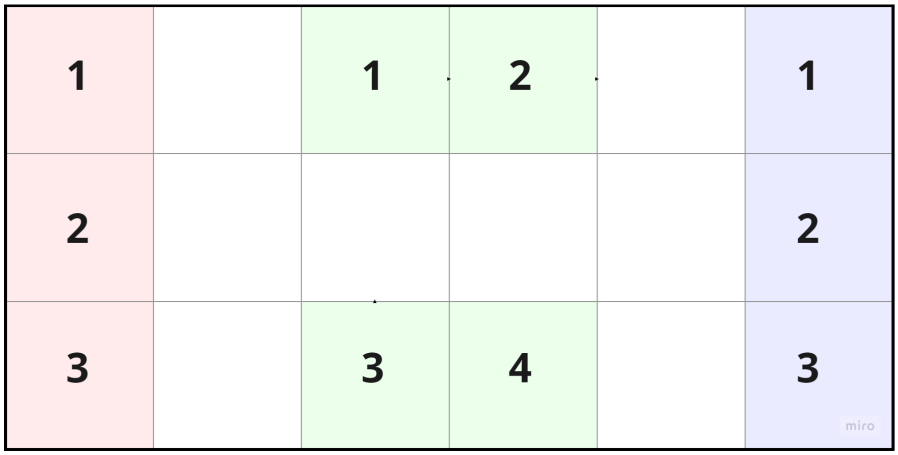
5.2 Vplyv atribútov MAPD na TP

Práca v tejto sekcii sleduje ako zmeny v inštancii MAPD ovplyvňujú výsledné plány a čas plánovania v TP. Tieto časy následne porovnáva a diskutuje, čo ich spôsobilo.



(a) Uniformné rozdelenie **(b)** Exponenciálne rozdelenie

Obr. 5.1 Príklad rozdelenia 20 úloh podľa pravdepodobnostných rozdelení



Obr. 5.2 MAPD mapa: Corners

5.2.1 Mapy

Testy boli uskutočnené na 5 variáciách mapy *Corners* 5.2, označených ako $pxdy$, kde x označuje počet miest vyzdvihnutia a y počet miest donášky. Testované boli nasledujú variácie: $p1d3$, $p2d3$, $p3d3$, $p3d2$, $p3d1$. Pôvodná mapa sa upravuje spôsobom, že sa vezmú hodnoty x, y a miestami vyzdvihnutia/donášky ostanú už len tie označené od 1 až x / 1 až y. Ak sa teda testuje $p2d3$ aktívne miesta donášky budú tie označené číslom 1 až 3 a miesta vyzdvihnutia 1 až 2.

Mapa Corners bola navrhnutá tak aby pridávanie a odoberanie miest vyzdvihnutí a donášok ovplyvňovalo dĺžky ciest čo najmenej a tým sa zamerať na výsledky algoritmu.

5.2.2 Výsledky

Na každej variácii mapy bolo uskutočnených 10 testov na počte úloh 10 a 50. Počet úloh bol zvolený tak, aby výsledky testov mohli byť použité v testoch na Ozobotoch pri navrhovaní máp. Množstvo agentov na mape bolo 4 pri všetkých testoch. Hodnotili sa nasledujúce vlastnosti:

- *Priemer aktívnych agentov*, tj. priemer agentov, ktorý vykonali aspoň 1 úlohu (AAA).
- *Počet úloh*, ktoré TP musel vyriešiť (TC).

- *Priemerný čas*, ktorý TP potreboval na vyriešenie MAPD inštancie v sekundách (*TTS*), zaokrúhlený na 3 desatinné čísla (*SoC*).
- Priemer hodnoty *sum of cost*[1], zaokrúhlený na celé čísla (*SoC*).
- Priemer hodnoty *makespan*, zaokrúhlený na celé čísla (*MS*).

Výsledky behov simulácií je možné vidieť v tabuľke 5.1.

5.2.3 Diskusia výsledkov

Prvým pozorovaním vyplývajúcim z výsledkov je priemerný *počet aktívnych agentom*. Ten sa v 100% prípadov zhodoval s počtom miest donášok. Tento faktor je spôsobený distribúciou úloh v algoritme TP. Presnejšie ide o riadok 6 v ukážke Algoritmus 4. Algoritmus nepriradí úlohu agentovi, kým je jej miesto donášky obsadené inou vykonávanou úlohou. Teda počet vykonávaných úloh obmedzuje počet aktívnych agentov. Pridanie algoritmu, ktorý by dokázal pracovať s agentmi nezávisle na počte miest donášky, by vyriešilo tento problém.

Druhým pozorovaním je vplyv počtu pozícii donášok a vyzdvihnutí na hodnoty *makespan* a *sum-of-cost*. Zvýšením počtu pozícii donášky z 1 na 2 pri 10 úlohách znížilo priemrný makespan aj sum-of-cost o 47%. Pri 50 úlohách bolo zníženie o 41% pre oba atribúty. Pridanie ďalšieho miesta donášky prinieslo priemerné zlepšenie MS a SoC o 20% pri 10 úlohách a 32% priemerné zlepšenie pre 50 úloh. Dôvod vyplýva hlavne z 1. pozorovania a to, že sa zväčšil počet aktívnych agentov.

Zvýšenie počtu miest vyzdvihnutia z 1 na 2 prinieslo zníženie o 28% pre MS aj SoC. Pri zvýšení z 2 na 3 prinieslo 5% zlepšenie pri 10 úlohách a 1% pri 50. Dáta naznačujú, že zlepšenia vyplýva zo zníženia akcií wait, ktoré agenti musia vykonávať aby uvoľnili cestu iným agentom pri miestach vyzdvihnutia. Mapy skladísk je teda výhodné vytvárať s počtom miest vyzdvihnutí aspoň rovnému počtu agentov.

Tieto pozorovania približujú spôsob, akým by mali byť modelované mapy skladiska, aby sa dosiahlo efektívneho vykonávania úloh. Je možné pozorovať, že pre efektívne vykonávanie plánov agentmi bolo podstatnejšie zvýšenie počtu miest donášok na počet agentov v skladisku ako zvýšenie miest vyzdvihnutí. Reálne skladisko však potrebuje veľké množstvo miest vyzdvihnutí na uskladnenie prenášaného tovaru. Experimenty zároveň ukázali, že TP bol schopný vyriešiť inštancie s 50 úlohami v radoch stovák milisekúnd.

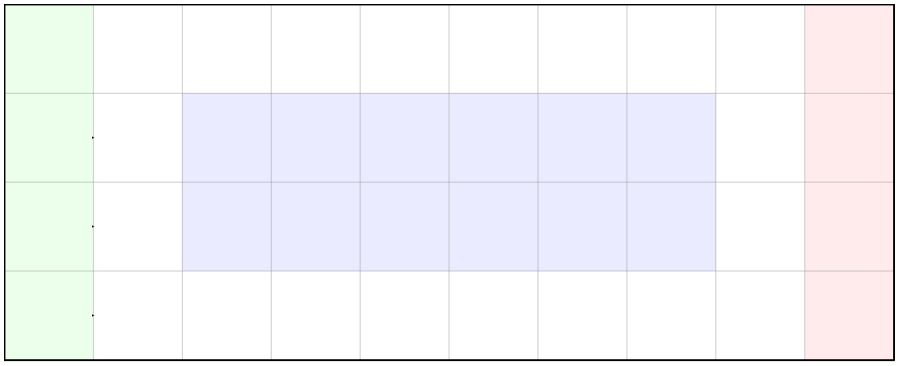
5.3 Vykonávanie na Ozobotoch

Sekcia je zameraná na vykonávanie plánov na Ozobotoch na modeli zmenšeného skladu. Najskôr budú definované mapy, na ktorých budú plány vykonávané. Následne bude predstavený spôsob vyhodnocovania experimentov, jednotlivé výsledky a ich diskusia.

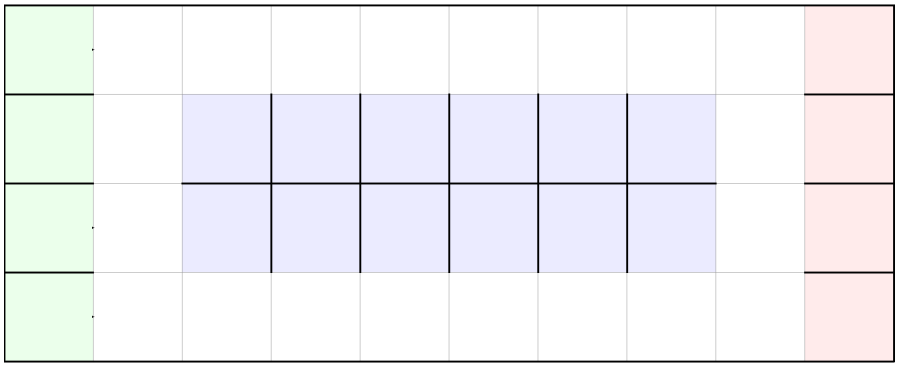
	TC = 10				TC = 50			
Názov mapy	AAA	TTS	SoC	MS	AAA	TTS	SoC	MS
p3d1	1	0.053	616	153	1	0.278	3118	779
p3d2	2	0.035	328	81	2	0.202	1826	456
p3d3	3	0.029	261	64	3	0.148	1233	307
p2d3	3	0.033	276	68	3	0.156	1249	311
p1d3	3	0.037	388	96	3	0.182	1651	412

Tabuľka 5.1 Výsledky testov vplyvu MAPD inštancie na plán

[1]Započítavá aj stavy wait a rotate



Obr. 5.3 MAPD mapa: Kiva



Obr. 5.4 MAPD mapa: Kiva so stenami

5.3.1 Mapy

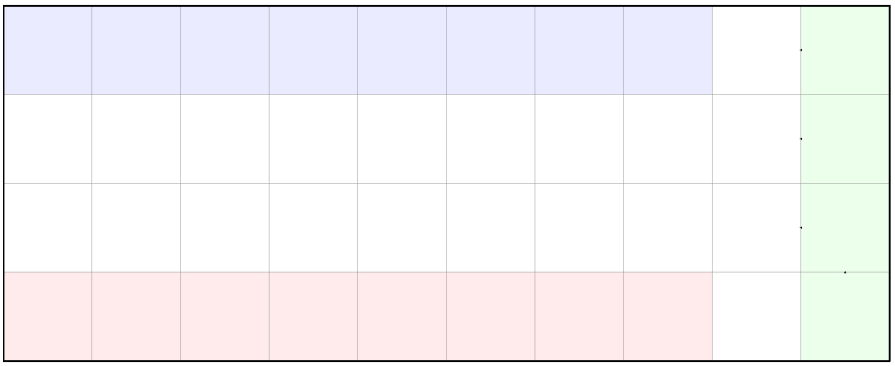
Pre účeli experimentov na Ozobotoch bolo vytvorených 5 máp. Všetky mapy boli vytvorené na základe pozorovaní z experimentov v predošlej sekcii. Mapa *Kiva* 5.3 modeluje zmenšený model rovnomenného automatizovaného skladu Kiva [30]. Mapa *Highway* 5.5 modeluje situáciu, kde sa miesta vyzdvihnutia nachádzajú na hornej strane skladiska a miesta donášky na spodnej strane. Hlavným miestom, kde sa očakávajú problémy s vykonávaním je stred mapy, ktorý tvoria voľné dlaždice.

Pre obe mapy boli vytvorené aj verzie so stenami medzi koncovými bodmi. Modeli sú na obrázkoch Obr. 5.4 a Obr. 5.6. Ich hlavným cieľom bolo obmedziť pohyb po koncových bodoch a tým sa vyhnúť kolíziám medzi *aktívnymi* a *stojacimi* Ozobotmi. Zároveň tieto úpravy komplikujú cesty pre agentov, preto sa očakáva zhoršenie sledovania ciest a zlepšenie problému kolízii. Všetky 4 mapy majú rozmer $10x4$ a boli navrhnuté pre externý monitor.

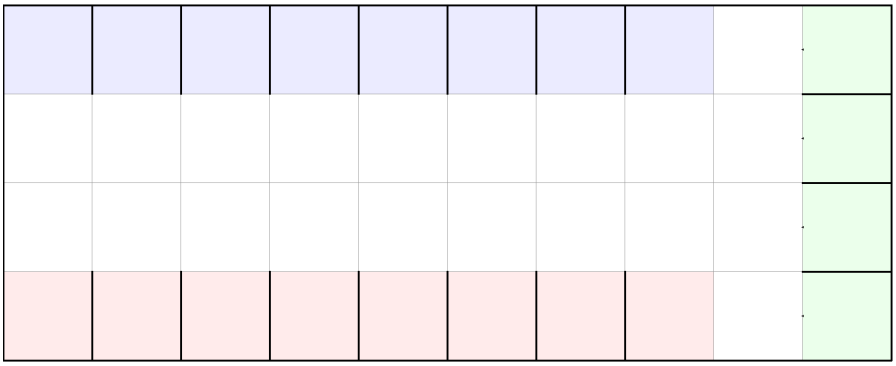
Posledným modelom mapy je *Kiva zmenšená*. Ide o mapu Kiva zmenšenú na rozmery $6x3$. Na mape bolo sledované správanie Ozobotov na monitore od notebooku Dell s externým monitorom Philips. Taktiež sa pozorovalo ako zmenšení priestor a skrátené vzdialenosti ovplyvní vykonávanie plánov.

5.3.2 Výsledky

Na každej mape bolo prevedených 10 testov s Ozobotmi pri náhodnom generovaní úloh. Pre generovanie bolo zvolené exponenciálne rozdelenie s $\lambda = 5$ a uniformné rozdelenie s $min = 0$ a $max = 10$ pre mapu Kiva zmenšená a $min = 0$ a $max = 30$ pre ostatné mapy. Generovaných



Obr. 5.5 MAPD mapa: Highway



Obr. 5.6 MAPD mapa: Highway so stenami

bolo 10 úloh, pre mapu s rozmermi $6x3$ a 10 pre ostatné mapy. Počet bol zvolený tak, aby boli generované dostatočne komplikované plány a zároveň plány boli dokončené v rozumnom časovom horizonte[2].

Pred každou sadou testov bola na Ozobotoch vykonaná kalibrácia, aby sa vyrovnali podmienky pre Ozobotov. Hodnotilo sa koľko percent agentov priemerne nedokončilo svoj plán (FA), a koľkokrát došlo ku kolízii počas vykonávania plánu (CA). Taktiež boli prebrané metriky SoC a MS z testov v predošlej sekcii.

Pri vykonávaní plánu sa mohlo stať, že nie všetci agenti sa pohli zo svojich počiatočných pozícii. V takomto prípade boli títo neaktívny agenti vylúčený z počítania metrík FA a CA. Títo agenti ale boli započítaný do SoC a MS, keďže akcie *wait* a *rotate* zväčšujú tieto hodnoty.

Výsledky sú zaznamenané v tabuľke 5.2.

5.3.3 Diskusia výsledkov

Pri behu všetkých testov sa ani raz nestalo, že všetci Ozoboti úspešne dokončili svoj plán. Z toho je možné usúdiť, že nakoľko je vykresľovanie ciest, ktoré poskytuje framework vhodné na vykonávanie MAPF plánov, komplikovanejšie a dlhšie plány pre Ozobota znamenajú skoro istú stratu cesty a teda neúspešné vykonávanie plánu. Zložitosť plánov stúpa množstvom po sebe idúcich otočení alebo zastavení a následnom pohnutí sa. Zlyhanie pri vykonávaní pozostávalo buď z *kolízií* alebo zo *straty cesty*.

Kolízie sa vyskytovali zväčša v momentoch, kedy agent opúšťajúci miesto vyzdvihnutia-/donášky narazil do agenta čakajúceho na uvoľnenie tohto miesta. Tieto kolízie často znamenali

[2] Cca 2 minúty

	Uniformné rozdelenie				Exponenciálne rozdelenie			
Názov mapy	SoC	MS	FA	CA	SoC	MS	FA	CA
Kiva	288	71	77	4	231	57	68	5
Kiva so stenami	291	72	74	2	301	82	93	3
Highway	222	55	68	3	175	55	77	7
Highway so stenami	262	65	73	1	232	57	75	5
Kiva zmenšená	95	31	76	4	103	33	77	3

Tabuľka 5.2 Výsledky vykonávania plánov na Ozobotoch

stratu cesty oboch agentov. Kolízie sa zriedka vyskytovali aj počas pohybu oboch agentov, no v týchto prípadoch dokázali Ozoboti z väčšiny prípadov pokračovať po svojej trase.

Predpoklad učinený pri návrhu máp so stenami, ktorý tvrdil, že steny znížia početnosť kolízii sa potvrdil. Pri vykonávaní sa na týchto mapách znížil počet kolízií pri miestach vyzdvihnutí a donášok. Výsledná úspešnosť agentov však klesla v 2 zo 4 prípadov. Spôsobilo to predĺženie, zobrazené aj na hodnotách MS a SoC, a v niektorých prípadoch aj sťažené manévrovanie agentov.

Z výsledkov je ďalej možné pozorovať ako zvýšenie SoC a MS vplývalo na úspešnosť vykonávania. Vo väčšine prípadov máp s rozmermi $10x4$ výsledky naznačujú, že čím dlhšie plány, tým bola aj pravdepodobnosť úspech agentov menšia. Mapa Kiva so stenami, ktorá mala najväčšiu metriku SoC medzi všetkými znamenala najnižšiu úspešnosť agentov a to 7%. Potvrdzuje to teda pozorovanie, že agenti sa dokážu udržať na svojej ceste len pri kratších a menej komplikovaných cestách.

Na druhej strane z pozorovaní na mape Kiva zmenšená je možné vidieť, že ani zníženie dimenzií mapy a z toho plynúce menšie SoC a MS nezlepšilo úspešnosť vykonania plánu. Teda je možné tvrdiť, že hlavným faktorom úspešnosti Ozobotov je *zložitosť plánov*. Experimenty na tejto mape boli zhotovené ako na externom monitore, tak aj na displeji notebooku. Výmena monitorov nemala na vykonávanie plánov značný vplyv.

Z predošlých pozorovaní vyplýva, že vykresľovanie ciest pomocou frameworku ESO-Nav síce tvorí dobrý základ pre agentov Ozobot, no v modeloch automatizovaných skladísk dáva neuspokojivé výsledky.

Kapitola 6
Záver

Prvým cieľom práce bolo definovať potrebné termíny a urobiť rešerš na existujúce riešenia plánovacích algoritmov v prostredím automatizovaného skladiska. Taktiež sa s týmto cieľom spájalo preskúmať možnosti robotov Ozobot a ich využitie už existujúcich riešenia. Tento cieľ bol splnený v kapitole 2 a kapitole 3. V kapitole 2 boli definované pojmi MAPF a následne naň nadväzujúci MAPD. Pri oboch problémoch boli uvedené algoritmy, ktorými sa dajú riešiť a taktiež ich rozšírenia o k-robustnosť. Kapitola 3 sa zaoberala analýzou modelu robota Ozobot Evo, za ktorou boli popísané predchádzajúce práce na tomto type robota. Predovšetkým bol popísaný framework ESO-Nav, ktorý práca používa.

Druhým cieľom práce bolo implementovať zmenšení model automatizovaného skladiska, kde budú následne prevedené experimenty na Ozobotoch. Tento cieľ bol splnený v kapitole 4. Táto kapitola popisuje rozšírenie frameworku ESO-Nav o MAPD variantu a spôsob, akým to bolo docielené. Taktiež kapitola vysvetľuje aký algoritmus bol použitý na riešenie MAPD inštancií a ako muselo byť upravené už existujúce riešenie. V neposlednom rade kapitola zoznamuje čítatela s grafickou a textovou reprezentáciou MAPD inštancie.

Posledným a teda tretím cieľom práce bolo otestovať vykonávanie plánov automatizovaného skladu na Ozobotoch. V 1. časti testov, bol najskôr testovaný upravený algoritmus Token Passing, aby sa zistilo ako vplývajú jednotlivé atribúty MAPD inštancii na výsledný plán. Zistením bolo, že najväčšie zvýšenie efektivity výkonu plánu potrebuje algoritmus dostatok miest donášky aby bol schopný v plánovaní použiť viac agentov. Výsledky testov potom boli použité pri návrhu máp pre 2. časť. V 2. časti sa testovalo vykonávanie plánov na Ozobotoch. Zistením bolo, že aj napriek úpravám algoritmu Token Passing a zmenám vo frameworku, ktorý práca používala na vykresľovanie ciest nepriniesli testy na Ozobotoch uspokojivé výsledky. Chybovosť agentov bola spôsobená stratov svojej trasy a kolíziami.

Ako jedno možné rozšírenie do budúcnosti by mohlo byť implementovanie nového alebo upraveného frameworku, ktorý by bol použitý na sledovanie ciest pre Ozobotov. Prípadne by tento framework mohol byť použitý pre online variantu MAPD problému. Ďalšou z možností je použitie iného algoritmu na riešenie MAPD inštancií a porovnanie jeho výsledkov s algoritmom Token Passing, ktorý bol použitý v tejto práci.

Dodatok A

Tabuľky všetkých behov experimentov

A.1 Tabuľky experimentov s algoritmom Token Pasing

TC = 10			TC = 50		
Trvanie výpočtu	sum-of-cost	makespan	Trvanie výpočtu	sum-of-cost	makespan
0,053	620	154	0,2759	3112	777
0,05	608	151	0,29	3112	777
0,051	624	155	0,284	3128	781
0,05	624	155	0,271	3112	777
0,065	624	155	0,268	3112	777
0,05	620	154	0,285	3128	781
0,051	608	151	0,285	3128	781
0,051	608	151	0,268	3112	777
0,05	608	151	0,271	3112	777
0,064	620	154	0,285	3124	780

Tabuľka A.1 Mapa p3d1

TC = 10			TC = 50		
Trvanie výpočtu	sum-of-cost	makespan	Trvanie výpočtu	sum-of-cost	makespan
0,032	308	76	0,184	1828	456
0,037	344	85	0,375	2236	558
0,037	340	84	0,179	1808	451
0,032	316	78	0,202	1804	450
0,032	324	80	0,178	1684	420
0,034	328	81	0,16	1684	420
0,033	320	79	0,18	1824	455
0,036	340	84	0,197	1792	447
0,048	324	80	0,187	1804	450
0,035	332	82	0,187	1800	449

Tabuľka A.2 Mapa p3d2

TC = 10			TC = 50		
Trvanie výpočtu	sum-of-cost	makespan	Trvanie výpočtu	sum-of-cost	makespan
0,032	264	65	0,134	1216	303
0,03	260	64	0,145	1248	311
0,026	260	64	0,1419	1236	308
0,027	248	61	0,16	1228	306
0,0301	260	64	0,146	1212	302
0,035	272	67	0,162	1228	306
0,029	268	66	0,147	1248	311
0,028	256	63	0,147	1220	304
0,028	264	65	0,155	1252	312
0,0281	256	63	0,146	1244	310

Tabuľka A.3 Mapa p3d3

TC = 10			TC = 50		
Trvanie výpočtu	sum-of-cost	makespan	Trvanie výpočtu	sum-of-cost	makespan
0,039	260	64	0,182	1632	407
0,035	272	67	0,151	1484	370
0,037	264	65	0,175	1656	413
0,028	240	59	0,157	1496	373
0,034	256	63	0,1742	1484	370
0,036	276	68	0,1666	1520	379
0,035	272	67	0,176	1632	407
0,034	272	67	0,178	1664	415
0,033	248	61	0,158	1492	372
0,033	268	66	0,1881	1644	410

Tabuľka A.4 Mapa p2d3

TC = 10			TC = 50		
Trvanie výpočtu	sum-of-cost	makespan	Trvanie výpočtu	sum-of-cost	makespan
0,0361	388	96	0,194	1652	412
0,0361	384	95	0,171	1656	413
0,038	388	96	0,177	1660	414
0,043	400	99	0,169	1604	400
0,033	380	94	0,165	1664	415
0,041	396	98	0,196	1656	413
0,035	388	96	0,176	1668	416
0,034	380	94	0,172	1616	403
0,036	384	95	0,21	1688	421
0,0381	388	96	0,19	1652	412

Tabuľka A.5 Mapa p1d3

A.2 Tabuľky experimentov na Ozobotoch

Uniformné rozdelenie				
Trvanie výpočtu	sum-of-cost	makespan	Počet neúspešných agentov	Výskyt kolízie
192	47	0.026	3	NIE
184	45	0.025	2	NIE
212	52	0.029	2	NIE
224	55	0.029	4	ANO
256	63	0.033	3	NIE
196	48	0.300	3	NIE
300	74	0.035	4	ANO
248	61	0.032	2	ANO
228	56	0.049	2	NIE
268	66	0.034	2	ANO

Tabuľka A.6 Kiva

Exponenciálne rozdelenie				
Trvanie výpočtu	sum-of-cost	makespan	Počet neúspešných agentov	Výskyt kolízie
192	47	0.026	3	ANO
184	45	0.025	2	NIE
212	52	0.029	2	NIE
224	55	0.029	4	ANO
256	63	0.033	3	ANO
196	48	0.300	3	NIE
300	74	0.035	4	ANO
248	61	0.032	2	NIE
228	56	0.049	2	NIE
268	66	0.034	2	ANO

Tabuľka A.7 Kiva

Uniformné rozdelenie				
Trvanie výpočtu	sum-of-cost	makespan	Počet neúspešných agentov	Výskyt kolízie
272	67	0.037	4	NIE
276	68	0.031	4	ANO
308	76	0.061	3	NIE
304	75	0.079	3	NIE
276	68	0.077	2	NIE
260	64	0.051	1	NIE
288	71	0.076	2	NIE
340	84	0.063	4	ANO
332	82	0.123	4	NIE
252	62	0.068	2	NIE

Tabuľka A.8 Kiva so stenami

Exponenciálne rozdelenie				
Trvanie výpočtu	sum-of-cost	makespan	Počet neúspešných agentov	Výskyt kolízie
272	67	0.026	4	NIE
324	80	0.1303	3	ANO
301	75	0.105	4	NIE
316	78	0.108	4	NIE
152	112	0.217	4	NIE
440	109	0.148	3	NIE
268	66	0.129	3	NIE
356	88	0.108	4	ANO
252	62	0.142	3	NIE
336	83	0.121	3	ANO

Tabuľka A.9 Kiva so stenami

Uniformné rozdelenie				
Trvanie výpočtu	sum-of-cost	makespan	Počet neúspešných agentov	Výskyt kolízie
228	56	0,058	2	NIE
232	57	0,077	2	ANO
224	55	0,062	2	NIE
232	57	0,052	3	NIE
232	57	0,057	3	NIE
208	51	0,06	3	NIE
228	56	0,047	3	ANO
232	57	0,054	3	NIE
212	52	0,049	3	ANO
196	48	0,044	3	NIE

Tabuľka A.10 Highway

Exponenciálne rozdelenie				
Trvanie výpočtu	sum-of-cost	makespan	Počet neúspešných agentov	Výskyt kolízie
180	44	0,048	2	ANO
164	40	0,038	3	ANO
180	44	0,0399	3	ANO
172	42	0,042	3	ANO
164	40	0,039	3	ANO
184	45	0,0449	3	ANO
168	41	0,061	4	NIE
184	45	0,043	4	NIE
176	43	0,0411	3	NIE
176	43	0,041	3	ANO

Tabuľka A.11 Highway

Uniformné rozdelenie				
Trvanie výpočtu	sum-of-cost	makespan	Počet neúspešných agentov	Výskyt kolízie
216	53	0.076	3	NIE
180	44	0.06	1	NIE
252	62	0.064	3	NIE
308	76	0.073	3	NIE
236	58	0.07	3	NIE
292	72	0.078	3	NIE
316	78	0.081	4	ANO
256	63	0.073	4	NIE
260	64	0.069	3	NIE
308	76	0.072	2	NIE

Tabuľka A.12 Highway so stenami

Exponenciálne rozdelenie				
Trvanie výpočtu	sum-of-cost	makespan	Počet neúspešných agentov	Výskyt kolízie
216	53	0.09	3	NIE
256	63	0.09	3	NIE
196	48	0.163	3	ANO
192	47	0.081	4	ANO
220	54	0.07	3	ANO
272	67	0.068	3	ANO
280	69	0.083	3	NIE
212	52	0.082	3	NIE
200	49	0.091	2	ANO
276	68	0.09	3	NIE

Tabuľka A.13 Highway so stenami

Uniformné rozdelenie				
Trvanie výpočtu	sum-of-cost	makespan	Počet neúspešných agentov	Výskyt kolízie
108	35	0,011	2	ANO
81	26	0,011	2	ANO
96	31	0,011	2	NIE
81	26	0,011	2	NIE
111	36	0,101	2	ANO
75	24	0,105	2	NIE
108	35	0,108	3	NIE
93	30	0,106	2	NIE
105	34	0,111	3	ANO
87	28	0,115	2	NIE

Tabuľka A.14 Kiva zmenšená

Exponenciálne rozdelenie				
Trvanie výpočtu	sum-of-cost	makespan	Počet neúspešných agentov	Výskyt kolízie
138	45	0,0261 2	NIE	
114	37	0,011	2	NIE
102	33	0,021	2	NIE
90	29	0,024	2	NIE
117	38	0,0208	3	ANO
108	35	0,02	2	NIE
72	23	0,01	1	NIE
75	24	0,02	2	NIE
93	30	0,032	3	ANO
120	39	0,02	2	ANO

Tabuľka A.15 Kiva zmenšená

Bibliografia

1. BOTEA, Adi; BOUZY, Bruno; BURO, Michael; BAUCKHAGE, Christian; NAU, Dana. Pathfinding in Games. In: LUCAS, Simon M.; MATEAS, Michael; PREUSS, Mike; SPRONCK, Pieter; TOGELIUS, Julian (ed.). *Artificial and Computational Intelligence in Games*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, zv. 6, s. 21–31. Dagstuhl Follow-Ups. ISBN 978-3-939897-62-0. ISSN 1868-8977. Dostupné z DOI: 10.4230/DFU.Vol6.12191.21.
2. VELOSO, Manuela; BISWAS, Joydeep; COLTIN, Brian; ROSENTHAL, Stephanie. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In: *Proceedings of the 24th International Conference on Artificial Intelligence*. Buenos Aires, Argentina: AAAI Press, 2015, s. 4423–4429. IJCAI'15. ISBN 9781577357384.
3. RYAN, Malcolm. 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. SURYNEK, Pavel. Conflict Handling Framework in Generalized Multi-agent Path finding: Advantages and Shortcomings of Satisfiability Modulo Approach. In: ROCHA, Ana Paula; STEELS, Luc; HERIK, Jaap van den (ed.). *Proceedings of the 11th International Conference on Agents and Artificial Intelligence, ICAART 2019, Volume 2, Prague, Czech Republic, February 19-21, 2019*. SciTePress, 2019, s. 192–203. ISBN 978-989-758-350-6. Dostupné z DOI: 10.5220/0007374201920203.
5. ROCHA, Ana Paula; STEELS, Luc; HERIK, Jaap van den (ed.). *Proceedings of the 11th International Conference on Agents and Artificial Intelligence, ICAART 2019, Volume 2, Prague, Czech Republic, February 19-21, 2019*. SciTePress, 2019. ISBN 978-989-758-350-6.
6. SURYNEK, Pavel. On Propositional Encodings of Cooperative Path-Finding. In: *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*. 2012, zv. 1, s. 524–531. Dostupné z DOI: 10.1109/ICTAI.2012.77.
7. STANDLEY, Trevor. Finding Optimal Solutions to Cooperative Pathfinding Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2010, roč. 24, č. 1, s. 173–178. Dostupné z DOI: 10.1609/aaai.v24i1.7564.
8. YU, Jingjin; LAVALLE, Steven. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2013, roč. 27, č. 1, s. 1443–1449. Dostupné tiež z: https://ojs.aaai.org/index.php/AAAI/article/view/8541.

9. BARER, M.; SHARON, Guni; STERN, Roni; FELNER, A. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. *Frontiers in Artificial Intelligence and Applications*. 2014, roč. 263, s. 961–962. Dostupné z DOI: 10.3233/978-1-61499-419-0-961.
10. SILVER, David. Cooperative Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2021, roč. 1, č. 1, s. 117–122. Dostupné z DOI: 10.1609/aiide.v1i1.18726.
11. DE WILDE, Boris; TER MORS, Adriaan W.; WITTEVEEN, Cees. Push and Rotate: A Complete Multi-Agent Pathfinding Algorithm. *J. Artif. Int. Res.* 2014, roč. 51, č. 1, s. 443–492. ISSN 1076-9757.
12. SHARON, Guni; STERN, Roni; GOLDENBERG, Meir; FELNER, Ariel. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. In: 2011, zv. 195, s. 662–667. Dostupné z DOI: 10.1016/j.artint.2012.11.006.
13. SHARON, Guni; STERN, Roni; FELNER, Ariel; STURTEVANT, Nathan R. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*. 2015, roč. 219, s. 40–66. ISSN 0004-3702. Dostupné z DOI: https://doi.org/10.1016/j.artint.2014.11.006.
14. HART, Peter E.; NILSSON, Nils J.; RAPHAEL, Bertram. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*. 1968, roč. 4, č. 2, s. 100–107. Dostupné z DOI: 10.1109/TSSC.1968.300136.
15. SURYNEK, Pavel; FELNER, Ariel; STERN, Roni; BOYARSKI, Eli. Efficient SAT Approach to Multi-Agent Path Finding Under the Sum of Costs Objective. In: KAMINKA, Gal A.; FOX, Maria; BOUQUET, Paolo; HÜLLERMEIER, Eyke; DIGNUM, Virginia; DIGNUM, Frank; HARMELEN, Frank van (ed.). *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*. IOS Press, 2016, zv. 285, s. 810–818. Frontiers in Artificial Intelligence and Applications. ISBN 978-1-61499-671-2. Dostupné z DOI: 10.3233/978-1-61499-672-9-810.
16. SURYNEK, Pavel. Unifying Search-based and Compilation-based Approaches to Multi-agent Path Finding through Satisfiability Modulo Theories. In: KRAUS, Sarit (ed.). *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. ijcai.org, 2019, s. 1177–1183. Dostupné z DOI: 10.24963/ijcai.2019/164.
17. MA, Hang; LI, Jiaoyang; KUMAR, T.K. Satish; KOENIG, Sven. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. São Paulo, Brazil: International Foundation for Autonomous Agents a Multiagent Systems, 2017, s. 837–845. AAMAS '17.
18. ČÁP, Michal; VOKŘÍNEK, Jiří; KLEINER, Alexander. Complete Decentralized Method for On-Line Multi-Robot Trajectory Planning in Valid Infrastructures. *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*. 2015, roč. 2015. Dostupné z DOI: 10.1609/icaps.v25i1.13696.
19. LIU, Minghua; MA, Hang; LI, Jiaoyang; KOENIG, Sven. Task and Path Planning for Multi-Agent Pickup and Delivery. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. Montreal QC, Canada: International Foundation for Autonomous Agents a Multiagent Systems, 2019, s. 1152–1160. AAMAS '19. ISBN 9781450363099.

20. ATZMON, Dor; STERN, Roni; FELNER, Ariel; BARTÁK, Roman; WAGNER, Glenn; ZHOU, Neng Fa. Robust multi-agent path finding. In: *17th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2018*. International Foundation for Autonomous Agents a Multiagent Systems (IFAAMAS), 2018, s. 1862–1864. Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS. ISBN 9781510868083. 17th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2018 ; Conference date: 10-07-2018 Through 15-07-2018.

21. EVOLLVE, Inc. *Ozobot* [online]. 2022 [cit. 2022-03-27]. Dostupné z : `https://ozobot.com/`.

22. EVOLLVE, Inc. *Ozobot Educator's Guide* [online]. 2022 [cit. 2022-03-27]. Dostupné z : `https://files.ozobot.com/stem-education/ozobot-educators-guide.pdf`.

23. EVOLLVE, Inc. *Ozobot Calibration Tips* [online]. 2022 [cit. 2022-03-28]. Dostupné z : `https://files.ozobot.com/stem-education/ozobot-calibration-tips.pdf`.

24. EVOLLVE, Inc. *Ozobot Color Codes Chart* [online]. 2022 [cit. 2022-03-28]. Dostupné z : `https://files.ozobot.com/stem-education/ozobot-color-codes.pdf`.

25. EVOLLVE, Inc. *Ozobot Color Codes Tips* [online]. 2022 [cit. 2022-03-28]. Dostupné z : `https://files.ozobot.com/stem-education/color-codes-tips.pdf`.

26. EVOLLVE, Inc. *OzoBlockly* [online]. 2022 [cit. 2022-03-28]. Dostupné z : `https://ozoblockly.com/`.

27. BARTÁK, Roman; ŠVANCARA, Jiří; ŠKOPKOVÁ, Věra; NOHEJL, David. Multi-agent Path Finding on Real Robots: First Experience with Ozobots. In: SIMARI, Guillermo R.; FERMÉ, Eduardo; GUTIÉRREZ SEGURA, Flabio; RODRÍGUEZ MELQUIADES, José Antonio (ed.). *Advances in Artificial Intelligence - IBERAMIA 2018*. Cham: Springer International Publishing, 2018, s. 290–301. ISBN 978-3-030-03928-8.

28. CHUDÝ, Ján; POPOV, Nestor; SURYNEK, Pavel. Emulating Centralized Control in Multi-Agent Pathfinding Using Decentralized Swarm of Reflex-Based Robots. In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2020, s. 3998–4005. Dostupné z DOI: `10.1109/SMC42975.2020.9283368`.

29. CHUDÝ, Ján. *Simulation of Centralized Algorithms for Multi-Agent Path Finding on Real Robots* [online]. 2020 [cit. 2022-12-02]. Dostupné z : `https://dspace.cvut.cz/bitstream/handle/10467/88327/F8-BP-2020-Chudy-Jan-thesis.pdf`.

30. KAMAGAEW, Andreas; STENZEL, Jonas; NETTSTRÄTER, Andreas; HOMPEL, Michael. Concept of Cellular Transport Systems in facility logistics. In: 2011, s. 40–45. Dostupné z DOI: `10.1109/ICARA.2011.6144853`.

Obsah přiloženého média