FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Řešení kolizí mezi geometrickými roboty ve spojitém prostoru
Student:	Yana Zabrodskaya
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Informační systémy a management
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Téma cílí na důležitý problém vyhýbání se srážkám mezi roboty při plánování pohybu. Speciálně se chceme zaměřit na případ, kdy se roboti pohybují spojitě ve spojitém prostoru. Předpokládáme, že roboti mohou mít libovolný geometrický tvar popsaný exaktně. V průběhu času se tvar robotů nebude měnit. Cílem je integrovat vyhýbání se srážkám do vybraného existujícího algoritmu pro plánování pohybu alespoň dvou robotů případně algoritmus upravit. Otázkou je rovněž ekonomický dopad případné úspěšné koordinace robotů ve vybrané doméně např. dopravě. Předpokládáme, že postup podle následujících kroků:

1. Provede rešerši algoritmů pro plánování pohybu více robotů.
2. Navrhne koncept pro vyhýbání se srážkám pro geometrické roboty.
3. Integruje vyhýbání se srážkám do algoritmu pro plánování pohybu a implementuje softwarový prototyp.
4. Návrh teoreticky a experimentálně vyhodnotí.
5. Zhodnotí ekonomicko-manažerské dopady úspěšné geometrické koordinace reálných robotů ve vybrané aplikační oblasti.

Seznam odborné literatury

[1] Anton Andreychuk, Konstantin S. Yakovlev, Dor Atzmon, Roni Stern: Multi-Agent Pathfinding with Continuous Time. IJCAI 2019: 39-45.

[2] Anton Andreychuk, Konstantin S. Yakovlev, Dor Atzmon, Roni Stern: Multi-Agent Pathfinding (MAPF) with Continuous Time. CoRR abs/1901.05506 (2019).

[3] Pavel Surynek: Multi-Agent Path Finding with Continuous Time and Geometric Agents Viewed through Satisfiability Modulo Theories. IJCAI Workshop WoMAPF 2019.

[4] Steven M. LaValle: Planning algorithms. Cambridge University Press 2006.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 22. prosince 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Řešení kolizí mezi geometrickými roboty ve spojitém prostoru

Yana Zabrodskaya

Katedra softwarového inženýrství
Vedoucí práce: doc. RNDr. Pavel Surynek, Ph.D.

28. května 2020

Poděkování

Ráda bych tímto poděkovala svému vedoucímu doc. RNDr. Pavlu Surynkovi, Ph.D., za veškerou pomoc při tvorbě bakalářské práce. Také bych poděkovala mamince a babičce za podporu, kamarádce Polině za pomoc s češtinou a kamarádce Nastě za pomoc s goniometrií.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu 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 předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 28. května 2020 . . . . . .

Odkaz na tuto práci

Zabrodskaya, Yana. *Řešení kolizí mezi geometrickými roboty ve spojitém prostoru.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce se zabývá problémem vyhýbání se srážkám mezi roboty při hledání cesty. Zde je navržen algoritmus pro nalezení nejkratší cesty bez srážek pro dva roboty libovolného geometrického tvaru. Algoritmus je teoreticky a experimentálně vyhodnocen a je provedena analýza ekonomického dopadu využití autonomních robotů ve skladech a maloobchodech.

Klíčová slova multi-agentní hledání cest, MAPF, konfliktní prohledávání, CBS, geometrický robot, hledání cest

Abstract

This thesis describes the problem of avoiding collisions among robots in pathfinding. First, the algorithm for finding a shortest collision free path for two robots of any geometric shape is proposed. Then the algorithm is theoretically and experimentally evaluated and an analysis of the economic impact of using autonomous robots in warehouses and retail stores is performed.

Keywords multi-agent pathfinding, MAPF, conflit-based search, CBS, geometric robot, pathfinding

Obsah

Seznam obrázků

Seznam tabulek

Seznam algoritmů

Úvod

Multi-agentní hledání cest ve spojitém prostoru[1] je důležitou problematikou v oboru umělé inteligence. Zabývá se problémem hledání nejkratší cesty pro několik robotů pohybujících se současně ve spojitém prostoru, přičemž ty roboty se nesmějí srazit. Klasický MAPF_R počítá jen s kruhovými roboty bez otáčení [1], proto se v této práci soustředíme na případ hledání cest pro dva roboty libovolného geometrického tvaru (např. kružnice, trojúhelník, čtverec, obdélník a další). Předpokládáme, že roboty se můžou pohybovat jenom jedním bokem dopředu, proto se během své cesty budou potřebovat otáčet. Řešení tohoto problému se může uplatnit při využití geometrických robotů ve skladech a obchodech pro převážení zboží mezi regály a doručování zboží pracovníkovi nebo zákazníkovi.

Cíle práce

Budeme se zabývat speciálním případem MAPF_R, kde máme pouze dva roboty. Hlavním cílem práce je návrh rozšíření MAPF_R pro roboty, které budou mít nějaký komplexní geometrický tvar. Další cíle práce lze shrnout následujícím způsobem:

- Analýza algoritmů pro plánování pohybu více robotů
- Analýza konfliktů při srážkách a jejich řešení
- Rozšíření algoritmu o vyhýbání se srážkám
- Teoretické a experimentální vyhodnocení algoritmu
- Analýza ekonomicko-manažerského dopadu geometrické koordinace reálných robotů

[1] Multi-agent pathfinding with continuous time (MAPF_R) [1]

Od navrženého algoritmu očekáváme, že

- *Hypotéza 1:* Pro dva roboty libovolného geometrického tvaru bude nalezena cesta tak, aby se nesrazily, pokud tato cesta existuje.
- *Hypotéza 2:* Pro různé případy srážek bude zvoleno optimálnější řešení, aby výsledná cesta byla co nejkratší.
- *Hypotéza 3:* Hledání cesty s vyhýbáním bude efektivnější než bez vyhýbání (separátní cesta pro každého robota).

Struktura práce

Práce je rozdělena do několika kapitol. V 1. kapitole je popsána teoretická část použitých metod. V 2. kapitole je poté uvedeno rozšíření základního algoritmu o dodatečné potřeby geometrických robotů, např. jak vypadá popis tvaru robota a k čemu slouží otáčení. Další kapitola popisuje návrh, implementaci a vyhodnocení algoritmu. Ve 4. kapitole je ukázka práce navrženého algoritmu a analýza výsledků. Konečně poslední kapitola uvádí ekonomickou analýzu využití reálných robotů.

Kapitola 1

Východiska

Tato kapitola definuje multi-agentní hledání cest ve spojitém prostoru, vysvětluje, co je řešením MAPF$_R$, jaké jsou metriky efektivity nalezeného řešení a jaké metody se používají pro řešení tohoto problému. Dále jsou popsány algoritmy pro konfliktní prohledávání v diskrétním a spojitém prostoru, jejich rozdíly a pseudokódy.

1.1 Multi-agentní hledání cest ve spojitém prostoru

Definice 1: V multi-agentním hledání cest ve spojitém prostoru[2] máme graf, *G(V, E)*, kde V je množina vrcholů a E je množina hran tohoto grafu, a k agentů[3], $a_1 \dots a_k$. Každý agent má úkol se dostat z počátečního vrcholu $s_i \in V$ do vrcholu svého cíle $g_i \in V$ a nesrazit se s ostatními [2].

Graf je vnořený do R^2 prostoru, kde vrcholům jsou přiřazeny body v prostoru a hranám jsou přiřazeny křivky v tomto prostoru, spojující tyhle body.

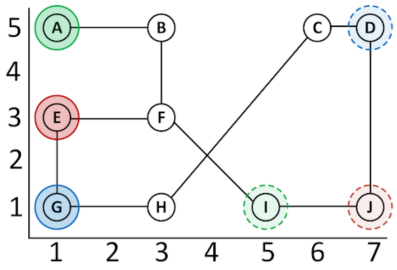
Agenty se pohybují spojitě po křivkách, nesmějí se současně nacházet ve stejném bodu a procházet se stejnou křivkou. Aby se agent dostal do svého cíle, musí provést posloupnost nějakých akcí. Akce může být buď pohyb, nebo čekání. Pohyb znamená, že se agent přesune na sousední vrchol. Čekání znamená, že agent zůstane v tomto vrcholu nějaký čas. Tato posloupnost definuje cestu agenta.

Multi-agentní hledání cest je NP-úplný problém [3] a existují různé metody a algoritmy pro jeho řešení [4]. Existují dva základní způsoby měření efektivity řešení MAPF$_R$ problému:

[2] Multi-agent pathfinding with continuous time (MAPF$_R$)
[3] agent je robot

- *Makespan*[4] - je rozdíl v čase mezi počátečním odjezdem prvního agenta a cílovým příjezdem posledního.
- *Sum of costs (SOC)* je součet délek všech cest.

Řešením MAPF$_R$ problému je nalezení cesty pro každého agenta, aby se nesrazil s ostatními agenty (obrázek 1.1), a minimalizace makespan nebo SOC [2].



Obrázek 1.1: MAPF$_R$ problém [1]

1.2 Konfliktní prohledávání

Konfliktní prohledávání[5] patří mezi nejčastěji používané algoritmy pro řešení multi-agentního vyhledávání.

CBS předpokládá, že akce trvají stejný čas a hledání cesty agentů se řídí omezeními (*constraints*). Omezení pro jednoho agenta je trojice (a_i, v, t), což znamená, že agent a_i nemůže navštívit vrchol v v čase t.

CBS se skládá ze dvou vrstev [5, 4, 6].

V horní vrstvě algoritmus prohledává *constraint tree* (CT) a řeší konflikty. CT je binární strom, ve kterém každý uzel N obsahuje

- množinu omezení pro agenty ($N.constraints$)
- řešení ($N.solution$)
- makespan nebo SOC řešení ($N.\mu$)

Kořen CT má množinu omezení prázdnou. Každý další následník zdědí omezení svého předka a přidá nové omezení pro jednoho agenta. Řešením je taková množina s k cesty, kde každá cesta agenta splňuje jeho omezení.

Ve spodní vrstvě algoritmus pro každého agenta najde nejkratší cestu v závislosti na jeho omezeních.

[4] doba trvání plánu, dále se bude používat anglický název
[5] Conflict-based search (CBS)

Pokud řešení v uzlu N není finální, pak obsahuje konflikt. Konflikt C je definován jako (a_i, a_j, v, t). To znamená, že agenty a_i a a_j navštíví vrchol v současně v čase t. Pro vyřešení tohoto konfliktu se uzel N rozdělí na dva uzly. Levý následník zdědí omezení (a_i, v, t) a pravý zdědí (a_j, v, t). Pro agenta s novým omezením bude nalezena jiná cesta.

Horní vrstva CBS je představena v algoritmu 1.1.

Algoritmus 1.1 CBS: konfliktní prohledávání (horní vrstva)

```
R.constraints ← ∅
R.solution ← najdi cestu pro každého agenta
R.μ ← max_μ(R.solution(a_i))
OPEN ← R
while OPEN ≠ ∅ do
    N ← min_μ(OPEN)
    if N.solution nemá konflikt then
        return N.solution
    end if
    C ← první konflikt (a_i, a_j, s, t) z N.solution
    for x ∈ {i, j} do
        P ← nový uzel
        P.constraints ← N.constraints ∪ (a_x, s, t)
        P.solution ← N.solution
        P.solution(x) ← cesta pro a_x
        P.μ ← Σ_{i=1}^{k} μ(P.solution(a_i))
        OPEN ← OPEN ∪ P
    end for
    OPEN.pop()
end while
```

1.3 Konfliktní prohledávání ve spojitém prostoru

Konfliktní prohledávání ve spojitém prostoru[6] je založeno na hledání potenciálních srážek a vypočítání nebezpečných intervalů. Nebezpečný interval (*unsafe interval*) je maximální čas, během kterého agent nesmí vykonávat žádnou akci, to znamená, agent čeká, jinak se srazí s jiným agentem [1]. Při nalezení potenciálního konfliktu je pro každého agenta vypočítán nebezpečný interval, pak se na základě těchto intervalů vytváří cesta agenta.

V CCBS omezení je ve tvaru $(a_i, \{u, v\}, [t_0, t_+))$. To znamená, že agent a_i se nemůže pohybovat hranou $\{u, v\}$ nebo se nacházet ve vrcholu u, jestli $u = v$, v čase $[t_0, t_+)$ [1].

V horní vrstvě CCBS stejně jako CBS prohledává CT a řeší konflikty.

[6] Conflict-based search with continuous time (CCBS) [1]

Spodní vrstva hledá pro každého agenta a_i nejkratší cestu od s_i do d_i s ohledem na omezení pro tohoto agenta.

Pokud $N.solution$ obsahuje konflikt, ten nastává, když agent a_i se pohybuje hranou $\{u, v\}$ v čase $[t_0, t_+)$ a agent a_j se pohybuje hranou $\{u', v'\}$ v čase $[t'_0, t'_+)$. Konfliktu se lze zbavit, jestli pro každého agenta bude přidáno omezení s jeho hranou v čase $[\tau_0, \tau_+)$. Čas $[\tau_0, \tau_+)$ je nebezpečný interval. Uzel N se rozdělí na dva uzly. Levý následník dostane omezení $(a_i, \{u, v\}, [\tau_0, \tau_+))$ a pravý – $(a_j, \{u', v'\}, [\tau_0, \tau_+))$ [7].

Algoritmus 1.2 předvádí pseudokód horní vrstvy CCBS.

Algoritmus 1.2 CCBS: konfliktní prohledávání ve spojitém prostoru (horní vrstva)

```
R.constraints ← ∅
R.solution ← najdi cestu pro každého agenta
R.μ ← max_μ(N.solution(a_i))
OPEN ← R
while OPEN ≠ ∅ do
    N ← min_μ(OPEN)
    collisions ← konflikty v N.solution
    if collisions = ∅ then
        return N.solution
    end if
    let (a_i, {u, v}, [t_0, t_+)) × (a_j, {u', v'}, [t'_0, t'_+)) ∈ collisions
    [τ_0, τ_+) ← computeUnsafeInterval()
    for each (a, {w, z} ∈ {(a_i, {u, v}), (a_j, {u', v'})}) do
        P ← nový uzel
        P.constraints ← N.constraints ∪ {(a, {w, z}, [τ_0, τ_+))}
        P.solution ← N.solution
        P.solution(a) ← cesta pro a
        P.μ ← Σ_{i=1}^{k} μ(P.solution(a_i))
        OPEN ← OPEN ∪ P
    end for
    OPEN.pop()
end while
```

Kapitola 2

Zobecnění vlastností agentu

Tato kapitola představuje vlastní rozšíření vlastností agentu. Současná verze CCBS používá jenom kruhové agenty bez otáčení. Toto není moc praktické pro aplikace, protože tam se agenty dost často otáčejí a mají jiný tvar než kulatý, například roboty ve skladu Amazon [8], KUKA [9] a Fanuc [10]. Z těchto důvodů je navržen algoritmus, ve kterém mají agenty obecný geometrický tvar a můžou se otáčet.

2.1 Geometrický tvar agentu

Algoritmus lze přizpůsobit pro libovolný polygon. Jelikož většina robotů má půdorys kruhový (obrázek 2.1) nebo obdélníkový, bylo testováno na kruhu, obdélníku a trojúhelníku. Testování na trojúhelníku ukazuje, že program se správně vypořádá s libovolným polygonem.

Obrázek 2.1: Příklad geometrického robotu

Geometrický tvar je popsán exaktně. Uvnitř agentu je zvolen bod (centrum agentu), jenž bude odpovídat souřadnicím vrcholu v mapě. Na základě tohoto

bodu jsou poté vypočítané vrcholy geometrického tvaru. Tvar agentu se zadává množinou dvojic. Každá dvojice se skládá z d a a kde:

- d je vzdálenost vrcholu od centra
- a je úhel mezi centrem a vrcholem

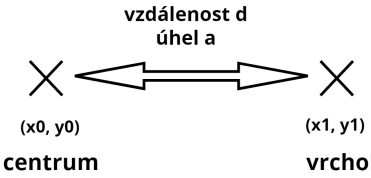
Souřadnice vrcholu lze vypočítat následujícím způsobem:

$$x = x_{centrum} + d \cdot \cos a$$
$$y = y_{centrum} + d \cdot \sin a$$

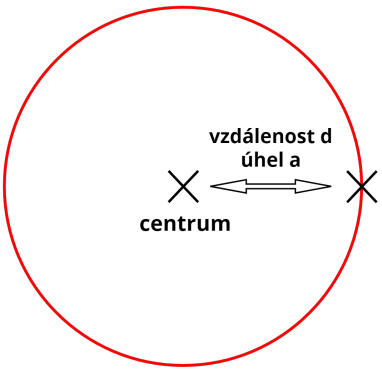
2.1.1 Popis kružnice

Pokud má množina dvojic jenom jeden prvek, pak má agent tvar kruhu, kde vzdálenost d je jeho poloměr.

Obrázek 2.2 ukazuje přidání právě jednoho vrcholu se vzdáleností d od centra a úhlem a. Jelikož má každý bod na kružnici stejnou vzdálenost od centra, úhel může být libovolný. Obrázek 2.3 poté zobrazuje výsledný tvar agentu.



Obrázek 2.2: Kruh: přidání vrcholu



Obrázek 2.3: Kruh: konečný tvar agentu

2.1.2 Popis polygonu

Ve výchozím stavu je agent otočen doprava. Vrcholy musejí být zadané od nejmenšího úhlu k největšímu (proti hodinové ručičce). Poté jsou vrcholy spojené v posloupnosti jejich zadávání a poslední vrchol je spojen s prvním.

Obrázek 2.4 znázorňuje množinu vrcholů pro obdélník. Každý vrchol má stejnou vzdálenost od centra a požadovaný úhel. Obrázek 2.5 předvádí konečný tvar obdélníku. Poté:

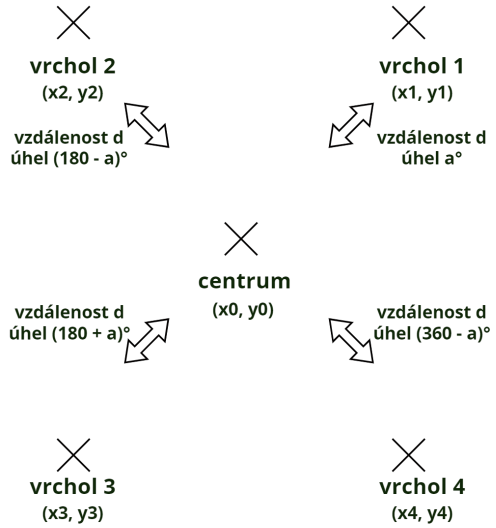
$$\text{délka} = 2 \cdot d \cdot \sin a$$

$$\text{šířka} = 2 \cdot d \cdot \cos a$$

Pokud známe šířku a délku a potřebujeme vypočítat d a a:

$$d = \frac{\sqrt{\text{šířka}^2 + \text{délka}^2}}{2}$$

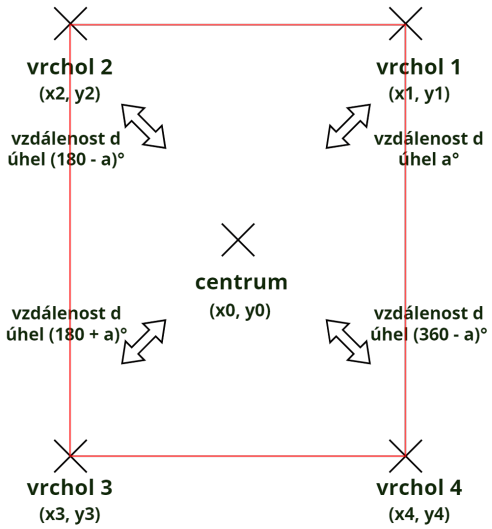
$$a = \arcsin \frac{\frac{\text{délka}}{2}}{d}$$



Obrázek 2.4: Obdélník: přidání vrcholů

2.2 Otáčení agentu

Agent má navíc novou vlastnost – směr, v jakém je natočen. Na začátku je agent otočen směrem k sousednímu vrcholu s nejmenším indexem. Během svého pohybu do cíle bude docházet ke změně směru, proto se agent musí otáčet.



Obrázek 2.5: Obdélník: konečný tvar agentu

Cesta agentu vypadá jako rozvrh, kde každá položka má čas a úhel (*arriveAngle*) příjezdu do vrcholu a čas a úhel (*departAngle*) odjezdu z vrcholu. Pokud se úhel příjezdu liší od úhlu odjezdu, agent se musí otočit o úhel β, kde

$$\beta = || \, departAngle - arriveAngle \, | - \left\lfloor \frac{| \, departAngle - arriveAngle \, |}{180} \right\rfloor \cdot 360 \, |$$

Směr otáčení se řídí následujícím kódem (algoritmus 2.1).

Algoritmus 2.1 rotateDirection: výběr směru otáčení

```
if (arriveAngle ≤ departAngle & departAngle − arriveAngle ≤ 180)
    || (arriveAngle > departAngle & arriveAngle − departAngle > 180)
then
    return right
else
    return left
end if
```

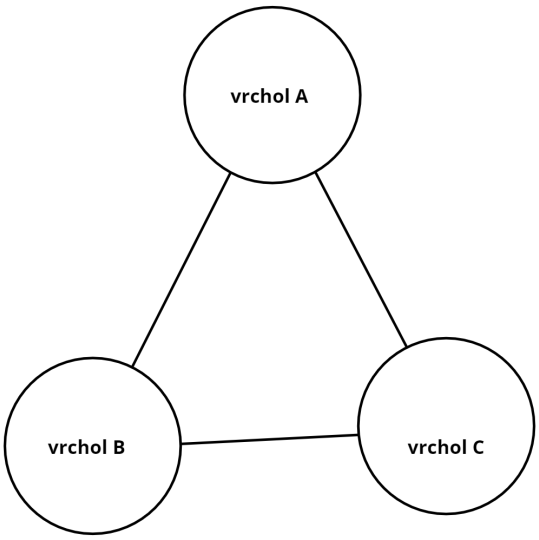
Kapitola 3

Integrace rozšíření do algoritmu

Tato kapitola popisuje rozšíření oproti původnímu CCBS. Dále je definována mapa, co je a k čemu slouží vnitřní vrchol a také jaké vlastnosti má agent. Detailně je vysvětlen překryvný konflikt, jeho různé případy a řešení každého. Je navržen algoritmus k vyřešení konfliktu pomocí čekání a otáčení. Výsledný algoritmus je teoreticky vyhodnocen.

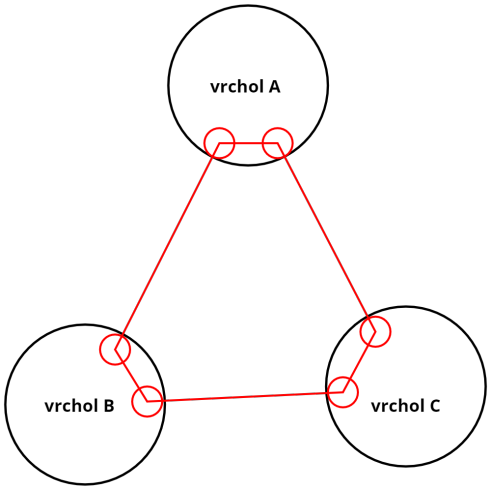
3.1 Reprezentace mapy

Definice 2: Mapa je struktura, která se skládá z n vrcholů (obrázek 3.1). Každý vrchol se skládá ze souřadnic x a y, indexu od 0 do $n - 1$ a množiny vnitřních vrcholů.



Obrázek 3.1: Graf se třemi vrcholy

Vrchol je spojen se sousedním vrcholem pomocí vnitřního vrcholu a současně ho agent využívá pro možnost otáčení. Rotace agentu znamená přechod mezi vnitřními vrcholy stejného rodičovského vrcholu. Vrchol má tolik vnitřních vrcholů, kolik má sousedních vrcholů. Vnitřní vrchol je spojen se všemi vnitřními vrcholy svého rodičovského vrcholu a právě s jedním vnitřním vrcholem sousedního vrcholu (obrázek 3.2).



Obrázek 3.2: Ukázka vnitřních vrcholů (červené)

Každý vnitřní vrchol se skládá z indexu rodičovského vrcholu, indexu sousedního vrcholu a úhlu mezi rodičovským vrcholem a sousedním.

3.2 Vlastnosti agentu

Mapou se pohybují agenty za účelem dostat se z počátečního vrcholu do cílového. Všechny agenty mají různé počáteční vrcholy a různé cílové vrcholy. Předpokládáme, že agenty mají stejný druh tvaru (stejný počet vrcholů).

Agent dostává následující informace jako vstupní:

- *start* – vrchol, kde začíná svou cestu
- *goal* – vrchol, kde končí svou cestu
- *speed* – rychlost, se kterou se agent pohybuje mezi vrcholy
- *rotation_speed* – rychlost otáčení, se kterou se agent otáčí, když se nachází ve vrcholu
- *vertices* – množina vrcholů, která definuje tvar agentu

3.3 Hledání cesty s vyhýbáním

Hledání cesty s vyhýbáním vychází z cesty nalezené pomocí algoritmu BFS[7] úpravou míst, kde došlo ke konfliktu.

1. Nejdříve je potřeba vybrat ze dvou delší cestu čili cestu agentu, který dorazí do svého cílového vrcholu poslední.
2. Poté musíme kontrolovat souřadnice agentů jednou za zvolený interval času (*step*).
3. Pokud v tento čas došlo ke konfliktu, je potřeba upravit cesty agentů podle typu konfliktu.
4. Pokud vnitřní cyklus skončil bez detekce konfliktu, je tato cesta konečná a hledání cesty je ukončeno.

V algoritmu 3.1 je představen algoritmus pro hledání cesty s vyhýbáním.

Algoritmus 3.1 AvoidancePath: hledání cesty s vyhýbáním

```
path_1 ← shortestPath(agent_1)
path_2 ← shortestPath(agent_2)
while true do
    conflict ← false
    path ← max(path_1, path_2)
    t ← 0
    while t ≤ maxTime(path) do
        if konflikt v čase t then
            conflict ← true
            break
        end if
        t ← t + step
    end while
    if conflict then
        if konflikt lze vyřešit then
            path_1, path_2 ← resolveConflict(t, path_1, path_2)
        else
            return ∅
        end if
    else
        return path_1, path_2
    end if
end while
```

[7] Breadth-first search – prohledávání do šířky

3.4 Překryvný konflikt

Překryvný konflikt nastává, když se agenty nachází tak blízko jeden druhého, že se srážejí. Detekce konfliktu je stejná pro všechny tvary agentů.

Distance mezi dvěma body $A\ (x_a,\ y_a)$ a $B\ (x_b,\ y_b)$ se počítá následujícím způsobem:

$$\mathrm{dist}(A, B) = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$$

Pro určení překryvného konfliktu je potřeba znát 4 souřadnice:

- centrum jednoho agentu,
- bod na jeho hraně, který se nachází nejblíž k druhému agentu,
- centrum druhého agentu,
- bod na jeho hraně, který se nachází nejblíž k prvnímu agentu.

Ke konfliktu dochází, jestliže hranový bod jednoho agentu leží mezi centrem a hranovým bodem druhého agentu.

Bod $A\ (x_a,\ y_a)$ leží mezi body $B\ (x_b,\ y_b)$ a $C\ (x_c,\ y_c)$, jestli

$$\begin{aligned} (x_a - x_b) \cdot (y_c - y_b) &= (y_a - y_b) \cdot (x_c - x_b) \\ &\cap \\ \mathrm{dist}(A, B) &\leq \mathrm{dist}(B, C) \\ &\cap \\ \mathrm{dist}(A, C) &\leq \mathrm{dist}(B, C) \end{aligned} \tag{3.1}$$

3.4.1 Vypočítání času čekání

Při srážce agentů je potřeba vypočítat čas, jak dlouho jeden agent počká, aby mohly uniknout srážce. Čas je co možná nejkratší, protože kdyby agent vyrazil o něco dřív, zase by došlo ke konfliktu. To zajistí nejrychlejší cestu.

Pro dva *kulaté* agenty lze hledání času čekání převést na hledání průsečíku dvou kružnic:

- $x_1,\ y_1,\ R_1$ – souřadnice centra a poloměr agenta, který čeká
- $x_2,\ y_2,\ R_2$ – souřadnice centra a poloměr agenta, který se pohybuje dál
- α – úhel, se kterým se agent právě pohybuje
- *speed* – rychlost pohybujícího se agenta
- $x,\ y$ – průsečík dvou kružnic, budoucí centrum pohybujícího se agenta
- t – čas čekání

1. Distance mezi $(x_1,\ y_1)$ a $(x,\ y)$ musí být rovná $R_1 + R_2$

$$\sqrt{(x_1 - x)^2 + (y_1 - y)^2} = R_1 + R_2$$

2. Průsečík se počítá tak:

$$x = x_2 + t \cdot speed \cdot \cos\alpha$$

$$y = y_2 + t \cdot speed \cdot \sin\alpha$$

3. Po dosazeni $x,\ y$ do první rovnice

$$(x_1 - x_2 - t \cdot \cos\alpha)^2 + (y_1 - y_2 - t \cdot \sin\alpha)^2 = (R_1 + R_2)^2$$

4. Potom

$$t = \frac{(-b + \sqrt{b^2 - 4c})}{2 \cdot speed},$$

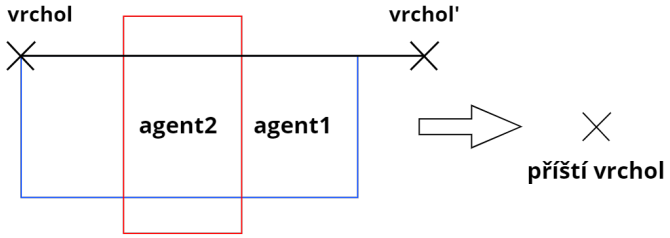
kde

$$b = -2\cos\alpha(x_1 - x_2) - 2\sin\alpha(y_1 - y_2)$$
$$c = (x_1 - x_2)^2 + (y_1 - y_2)^2 - (R_1 + R_2)^2$$

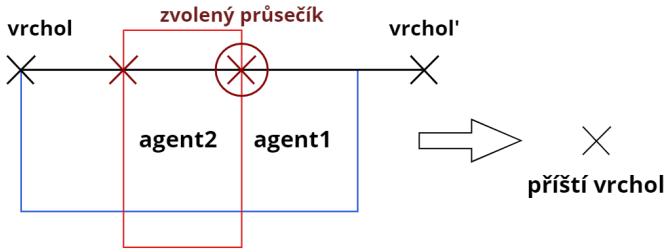
Pro dva *polygony* je potřeba hledat průsečík pro každou hranu čekajícího agenta a každý vrchol pohybujícího se agenta:

- t – čas, kdy došlo ke konfliktu
- $agent_1$ – pohybující se agent
- $agent_2$ – čekající agent
- $C_1\ (x_1,\ y_1)$ – souřadnice centra pohybujícího se agenta v čase t
- $C_2\ (x_2,\ y_2)$ – souřadnice centra čekajícího agentu v čase t

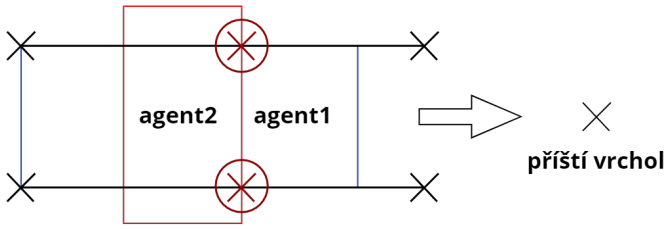
1. Nejdříve vytvoříme úsek od vrcholu do jeho budoucí polohy na příštím vrcholu (obrázek 3.3).
2. Poté pro každou hranu čekajícího agenta najdeme průsečík s tímto úsekem.
3. Z těchto průsečíků vybereme jeden, který je nejbližší k budoucí poloze (obrázek 3.4).



Obrázek 3.3: Úsek od vrcholu do jeho budoucí polohy



Obrázek 3.4: Výběr průsečíku pro jeden vrchol



Obrázek 3.5: Výběr finálního průsečíku

4. Po nalezení průsečíku pro každý vrchol, musíme vybrat ten, který se nachází nejdál od svého vrcholu (obrázek 3.5).

5. Na základě vzdálenosti průsečíku od původní pozice najdeme čas čekání. Poté bude konflikt vyřešen (obrázek 3.6).

Popsané vypočítání je předvedeno v algoritmu *calcWaitingTime* (algoritmus 3.2).

Obrázek 3.6: Nová pozice agentů v čase t

Algoritmus 3.2 calcWaitingTime: spočítání času čekání pro polygony

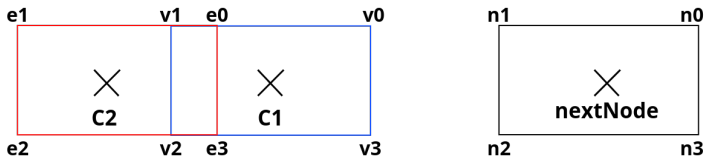
$nextNode \leftarrow$ další vrchol po času t
$finalIntersections \leftarrow \varnothing$
for all v **in** $agent_1$ $vertices$ **do**
  $vrt \leftarrow$ souřadnice v
  $nextVrt \leftarrow$ souřadnice v na $nextNode$
  $intersections_v \leftarrow \varnothing$
  for all e **in** $agent_2$ $vertices$ **do**
    $edgeStart \leftarrow$ souřadnice e
    $edgeEnd \leftarrow$ souřadnice dalšího vrcholu od e
    $line_1 \leftarrow$ od vrt do $nextVrt$
    $line_2 \leftarrow$ od $edgeStart$ do $edgeEnd$
    $i \leftarrow$ průsečík $line_1$ a $line_2$
    $intersections_v \leftarrow intersections_v \cup i$
  end for
  $closest \leftarrow$ nejbližší průsečík k $nextVrt$
  $finalIntesections \leftarrow finalIntesections \cup (closest, vrt)$
end for
$distance \leftarrow$ nejvzdálenější od svého vrcholu průsečík
return $\frac{distance}{agent_1\ speed}$

Příklad

Ukázka práce algoritmu 3.2 **calcWaitingTime** na konkrétních hodnotách.

Agenty mají stejný obdélníkový tvar ($d = 1, a = 30$) a stejnou rychlost $speed = 1$. Ať konflikt nastane v čase t, kdy centrum pohybujícího se agentu C_1 má souřadnice $(0; 0)$ a centrum čekajícího agentu C_2 se nachází na $(-1; 0)$. Souřadnice dalšího vrcholu ($nextNode$) je $(5; 0)$ (obrázek 3.7). Oba agenty se pohybují doprava.

Nalezneme souřadnice všech vrcholů obou agentů a budoucí polohy pohybujícího se agenta: tabulka 3.1.



Obrázek 3.7: Příklad překryvného konfliktu: konflikt

Tabulka 3.1: Příklad překryvného konfliktu: souřadnice vrcholů

vrchol	x	y
v_0	0,86	0,5
v_1	$-0,86$	0,5
v_2	$-0,86$	$-0,5$
v_3	0,86	$-0,5$
e_0	$-0,14$	0,5
e_1	$-1,86$	0,5
e_2	$-1,86$	$-0,5$
e_3	$-0,14$	$-0,5$
n_0	5,86	0,5
n_1	4,14	0,5
n_2	4,14	$-0,5$
n_3	5,86	$-0,5$

Potom nalezneme průsečíky mezi úsekem $(v_j, n_j),\ j \in \{0, 1, 2, 3\}$ a každou hranou čekajícího agenta, pokud tyhle průsečíky existují: tabulka 3.2.

- U vrcholu v_1 zvolíme průsečík i_{11}, protože

$$\text{dist}(i_{11}, n_1) < \text{dist}(i_{10}, n_1)$$

- U vrcholu v_2 zvolíme průsečík i_{21}, protože

$$\text{dist}(i_{21}, n_2) < \text{dist}(i_{20}, n_2)$$

Mezi těmito průsečíky musíme vybrat ten, který je vzdálenější od svého vrcholu. Protože

$$\text{dist}(i_{11}, v_1) = \text{dist}(i_{21}, v_2),$$

můžeme zvolit libovolný z nich, ať to bude i_{11}.

Ve výsledku najdeme vzdálenost tohoto průsečíku od jeho vrcholu:

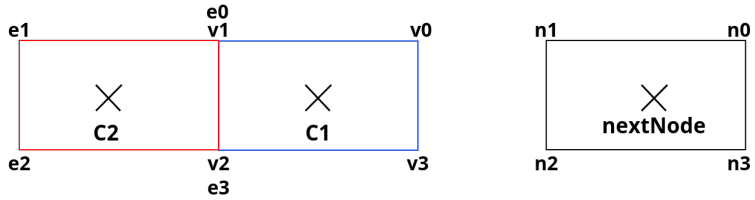
$$distance = \text{dist}(i_{11}, v_1) = \sqrt{(-0,86 - (-0,14))^2 + (0,5 - 0,5)^2} = 0,72.$$

Tabulka 3.2: Příklad překryvného konfliktu: průsečíky

$line_1$	$line_2$	i
(v_0, n_0)	(e_0, e_1)	-
(v_0, n_0)	(e_1, e_2)	-
(v_0, n_0)	(e_2, e_3)	-
(v_0, n_0)	(e_3, e_0)	-
(v_1, n_1)	(e_0, e_1)	$(-0,86;\ 0{,}5) = v_1 = i_{10}$
(v_1, n_1)	(e_1, e_2)	-
(v_1, n_1)	(e_2, e_3)	-
(v_1, n_1)	(e_3, e_0)	$(-0,14;\ 0{,}5) = e_0 = i_{11}$
(v_2, n_2)	(e_0, e_1)	-
(v_2, n_2)	(e_1, e_2)	-
(v_2, n_2)	(e_2, e_3)	$(-0,86;\ -0,5) = v_2 = i_{20}$
(v_2, n_2)	(e_3, e_0)	$(-0,14;\ -0,5) = e_3 = i_{21}$
(v_3, n_3)	(e_0, e_1)	-
(v_3, n_3)	(e_1, e_2)	-
(v_3, n_3)	(e_2, e_3)	-
(v_3, n_3)	(e_3, e_0)	-

Poté čas čekání bude $\frac{distance}{speed} = \frac{0{,}72}{1} = \mathbf{0{,}72}$.

Na obrázku 3.8 je výsledná pozice agentů v čase t, jestli čekající agent zůstane na 0,72 v předchozím vrcholu.



Obrázek 3.8: Příklad překryvného konfliktu: řešení

3.4.2 Vypočítání času pro otáčení

Při vzniku konfliktu, když jeden agent je ve vrcholu a svou polohou přehradil cestu jinému agentu, musíme pro něj najít takovou polohu, po otáčení do které ke srážce už nedojde. Pro nalezení úhlu otáčení musíme znát:

- t – čas, kdy došlo ke konfliktu
- $agent_1$ – otáčející se agent

- $agent_2$ – pohybující se agent
- $C_1\ (x_1,\ y_1)$ – souřadnice centra otáčejícího se agenta v čase t
- $C_2\ (x_2,\ y_2)$ – souřadnice centra pohybujícího se agenta v čase t
- $edgeCoord_2$ – bod na jeho hraně, který se nachází nejblíž k prvnímu agentu

1. Zkusíme během plného obratu o 360° najít nejmenší úhel, kdy konflikt nenastane. $Agent_1$ se otočí o tolik stupňů.
2. $Agent_2$ počká v předchozím vrcholu čas, který se rovná *(úhel · rychlost otáčení)*.
3. Jestli se otočený agent bude pohybovat dál, vypočítáme pro něj čas pomocí algoritmu *calcWaitingTime* (algoritmus 3.2), jak dlouho musí zůstat v této poloze.
4. Potom se tento agent otočí do potřebné polohy pro další pohyb.

Popsané vypočítání znázorňuje algoritmus 3.3 *calcRotation*.

Algoritmus 3.3 calcRotation: vypočítání času pro otáčení

```
currentAngle ← úhel v čase t
for i od 0 do 360 do
    newAngle ← (currentAngle + i) mod 360
    edgeCoord_1 ← bod na hraně pro novou polohu
    if not overlapConflict(C_1, C_2, edgeCoord_1, edgeCoord_2) then
        return i · agent_1 rotation_speed
    end if
end for
```

3.5 Rozbor případů možných konfliktů

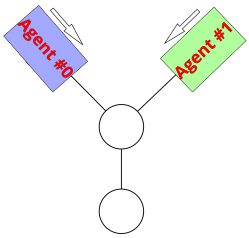
Tato část popisuje případy, které mohou nastat při řešení konfliktů vzhledem k tomu, jak se agenty pohybují a kde ten konflikt může nastat.

3.5.1 Agenty jsou ve vrcholech

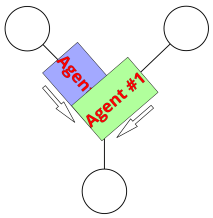
Charakteristika: Oba agenty se pohybují směrem ke stejnému vrcholu (obrázek 3.9). V určitý čas se budou oba agenty současně nacházet v tomto vrcholu (obrázek 3.10).

Řešení: Jeden z agentů musí počkat na předchozím vrcholu nejkratší možný čas (obrázek 3.11, algoritmus 3.2). Pokud agenty mají různou rychlost,

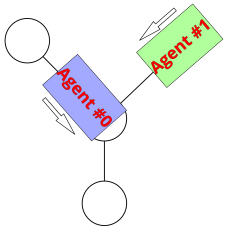
čekat bude ten, který je pomalejší. Díky tomu zvětší rychlejší agent odstup mezi agenty a sníží šanci na to, že ho pomalejší agent dožene (obrázek 3.12). Jestli mají agenty stejnou rychlost, výběr čekajícího agentu bude záležet na času čekání. Agent s kratším časem čekání počká pro zajištění nejrychlejší cesty.



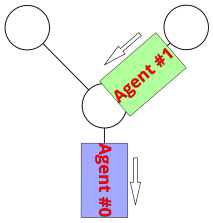
Obrázek 3.9: Počáteční pozice agentů



Obrázek 3.10: Vrcholový konflikt



Obrázek 3.11: Agent #1 čeká



Obrázek 3.12: Vyřešení konfliktu

3.5.2 Jeden agent je ve vrcholu, druhý se pohybuje

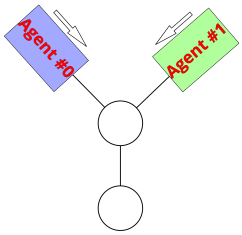
V nějaký okamžik se jeden agent nachází ve vrcholu, druhý se pohybuje a došlo ke srážce. Tento konflikt má dvě řešení.

Charakteristika: vrchol je společný pro obě cesty (obrázek 3.13).

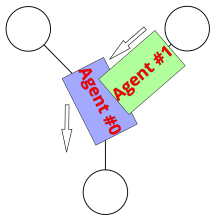
Řešení: problém lze převést na řešení vrcholového konfliktu s výjimkou, že není potřeba vybírat, který agent čeká. Agent, který dorazí na vrchol později (obrázek 3.14), musí čekat na předchozím vrcholu (obrázek 3.15) tak dlouho, aby jiný agent opustil konfliktní vrchol a nedošlo ke srážce (algoritmus 3.2). Pak oba agenty pokračuji ve svých cestách (obrázek 3.16).

Charakteristika: pohybující se agent nepotřebuje navštěvovat konfliktní vrchol. Agent, který se nachází ve vrcholu, překáží svou polohou v pohybu druhému agentu (obrázek 3.17).

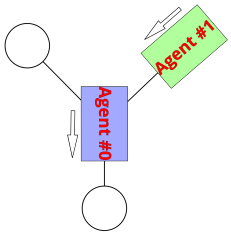
Řešení: pro tohoto agenta bude vypočítána pozice, do níž se musí otočit (algoritmus 3.3), aby přestal bránit pohybu (obrázek 3.18). Na základě nové polohy pohybující se agent počká na předchozím vrcholu čas, který je roven *(úhel otáčení · rychlost otáčení agentu)*. Poté otočený agent zůstane



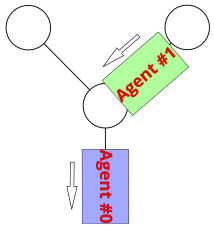
Obrázek 3.13: Počáteční pozice agentů



Obrázek 3.14: Překryvný konflikt

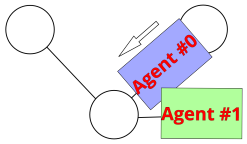


Obrázek 3.15: Agent #1 čeká

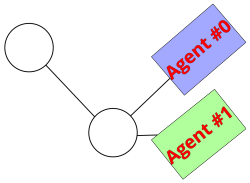


Obrázek 3.16: Vyřešení konfliktu

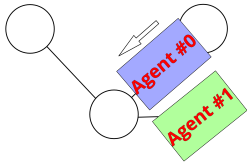
v této poloze tak dlouho, než se druhý agent posune na bezpečnou vzdálenost (algoritmus 3.2, obrázek 3.19). Následně se tento agent otočí do úhlu, který potřebuje pro dálnější pohyb (obrázek 3.20).



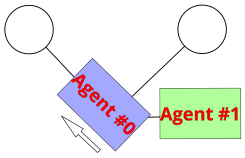
Obrázek 3.17: Překryvný konflikt



Obrázek 3.18: Otáčení agentu



Obrázek 3.19: Vyřešení konfliktu



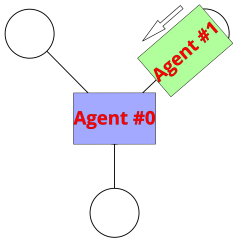
Obrázek 3.20: Zpáteční otáčení

3.5.3 Jeden agent dosáhl cílového vrcholu, druhý se pohybuje

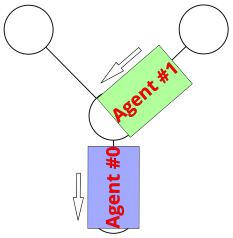
Agent ukončil svou cestu v nějakém vrcholu, ale jiný agent později potřebuje projet přes tento vrchol (obrázek 3.21). Tento problém lze řešit třemi způsoby.

Charakteristika: cílový vrchol má alespoň 3 vnitřní vrcholy.

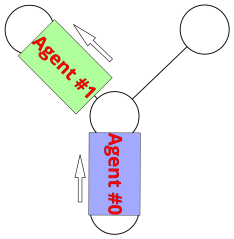
Řešení: agent, který již skončil, musí uvolnit cestu. Tento agent vybere sousední vrchol, který zároveň není ten vrchol, odkud přijíždí druhý agent, a není ten vrchol, kam směřuje. Odjede na tento vrchol (obrázek 3.22) a následně se vrátí (obrázek 3.23). Jiný agent mezitím mine jeho cílový vrchol a pokračuje dál (obrázek 3.24).



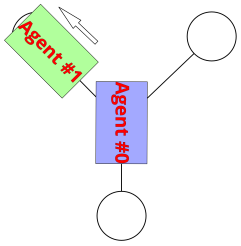
Obrázek 3.21: Počáteční pozice



Obrázek 3.22: Uvolnění cesty



Obrázek 3.23: Vracení se



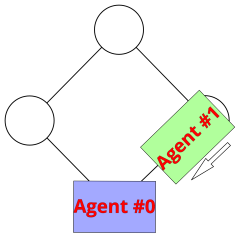
Obrázek 3.24: Konflikt je vyřešen

Charakteristika: existuje jiná cesta.

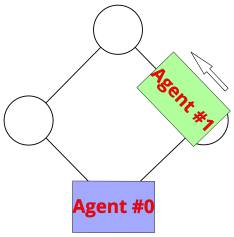
Řešení: pokud cílový vrchol má nejvýše dva sousední vrcholy (obrázek 3.25), pohybující se agent zkusí najít alternativní cestu s vyhýbáním se konfliktnímu vrcholu (obrázek 3.26).

Charakteristika: agenty mají společnou část cesty.

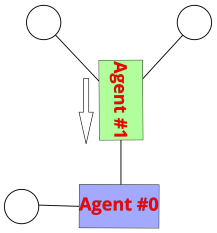
Řešení: agenty mají stejnou část cesty před tím, než agent ukončil svou cestu (obrázek 3.27). Je potřeba najít vrchol před tou společnou částí (obrázek 3.28). Agent s ukončenou cestou počká na tomto vrcholu tak dlouho, aby poté dorazil na svůj cílový vrchol právě tehdy, kdy druhý agent tento vrchol opustí (obrázek 3.29).



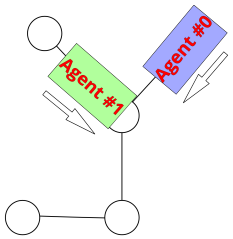
Obrázek 3.25: Vrcholový konflikt



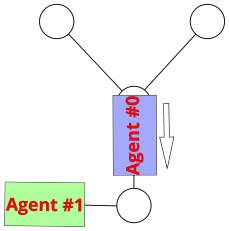
Obrázek 3.26: Výběr jiné cesty



Obrázek 3.27: Počáteční pozice



Obrázek 3.28: Uvolnění cesty



Obrázek 3.29: Konflikt je vyřešen

3.5.4 Agenty se pohybují

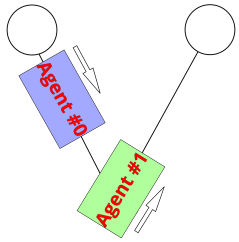
Charakteristika: oba agenty se pohybují (obrázek 3.30) a v nějaký okamžik dojde ke srážce (obrázek 3.31).

Řešení: proto ten agent, který je pozadu, musí počkat nějaký čas na předchozím vrcholu (obrázek 3.32), až se druhý agent přesune tak daleko, že už se nesrazí (obrázek 3.33, algoritmus 3.2).

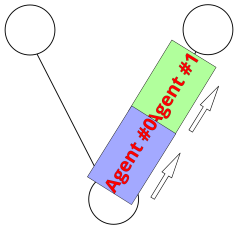
3.6 Teoretické vyhodnocení návrhu

Byly prozkoumány vlastnosti algoritmu 3.1 **AvoidancePath** a následně vyhodnoceny jeho časová složitost, korektnost a úplnost.

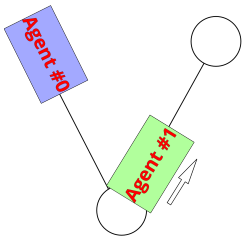
Tvrzení 1: algoritmus **AvoidancePath** má složitost $\mathcal{O}(|V|+|E|)$, kde V je počet vrcholů v mapě a E je počet hran.



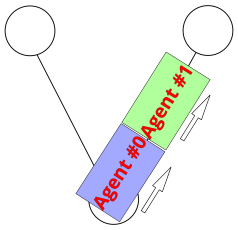
Obrázek 3.30: Agenty se pohybují



Obrázek 3.31: Překryvný konflikt



Obrázek 3.32: Agent #0 čeká



Obrázek 3.33: Vyřešení konfliktu

Důkaz: složitost **AvoidancePath** vychází ze složitosti algoritmu pro hledání cesty $\mathcal{O}(|V| + |E|)$ [11] (BFS) a složitosti algoritmů pro řešení konfliktů. Ty jsou buď konstantní $\mathcal{O}(1)$ (čekání, otáčení) nebo $\mathcal{O}(|V| + |E|)$ (hledání alternativní cesty, BFS). Výsledná složitost algoritmu **AvoidancePath** je $\mathcal{O}(|V| + |E|)$.

Tvrzení 2: algoritmus **AvoidancePath** je korektní, tj. jestli dá nějaké řešení, to řešení skutečně fungeje.

Důkaz: algoritmus BFS je korektní [11]. **AvoidancePath** upravuje cestu nalezenou BFS čekáním nebo otáčením ve vrcholu (výsledná posloupnost vrcholů v cestě se nemění) a hledáním alternativní cesty z vrcholu, kde došlo ke konfliktu, do cílového vrcholu. Alternativní cesta se hledá také BFS, a pokud existuje, je správná. Proto je algoritmus **AvoidancePath** korektní.

Tvrzení 3: algoritmus **AvoidancePath** je *úplný* a vždy skončí.

Důkaz: jestli dříve nalezená cesta obsahuje konflikt, který nelze vyřešit výše popsanými metodami, algoritmus to detekuje a vrátí výsledek, že cesta neexistuje. Jinak bude vrácena cesta, která nemá konflikty.

Kapitola 4

Experimentální vyhodnocení

Teoretický návrh byl implementován formou softwarového prototypu v programovacím jazyku C++. Program se spouští přes příkazový řádek a potřebuje dva vstupní soubory: popis mapy a popis agentů.

Formát mapy

- na prvním řádku je číslo N, $N \geq 2$, které udává počet vrcholů v mapě
- poté nasleduje N řádků, kde každý obsahuje dvě čísla X a Y, která udávají souřadnice vrcholů
- na dalším řádku je číslo M udávající počet vazeb mezi vrcholy
- dále nasleduje M řádků. Na každém řádku jsou dvě čísla A a B, $A \neq B$, $0 \leq A,\ B < N$, udávající, že vrchol A je spojen s vrcholem B

Formát agentů

- na prvním řádku je číslo P, $P \geq 1$, které udává počet vrcholů ve tvaru agenta (kulatý agent má $P = 1$)
- poté nasleduje P řádku. Na každém řádku jsou dvě čísla d a a, $0 \leq d$, $0 \leq a < 360$, kde d je vzdálenost vrcholu od centra a a je úhel vrcholu
- dále nasleduje pět čísel *s, g, v, rs, i*:

  s a g jsou počáteční a cílový vrcholy, $s \neq g,\ 0 \leq s,\ g < N$

  v je rychlost a rs je rychlost otáčení, $0 \leq v,\ rs$

  i je index agentu, každý agent má vlastní index, $i \in \{0, 1\}$
- stejným způsobem je popsán druhý agent

Celkem bylo vytvořeno 49 testovacích scénářů s agenty různých tvarů na 5 mapách. V následujících sekcích jsou detailně popsány čtyři scénáře.

Scénáře jsou motivovány nákupním [12] a skladovým [13] robotem. Nákupní robot se může pohybovat po obchodnímu centru a vozit zboží zákazníkovi. Skladový robot může převážet zboží od jednoho regálu do jiného.

4.1 Hledání separátních cest

Při plánování separátních cest (bez vyhýbání) ke konfliktu dojde málokdy. Proto bylo porovnáno hledání cest bez vyhýbání a s vyhýbáním.

Hledání separátních cest využívá BFS s ignorováním vrcholů, které navštíví druhý agent. První agent se vyhýbá jen počátečnímu a cílovému vrcholu druhého agenta a nalezne svou cestu, pokud je to možné. Druhý agent se vyhýbá všem vrcholům, které jsou v cestě prvního agenta, nebo, pokud první agent nenašel cestu, jeho počátečnímu a cílovému vrcholu (algoritmus 4.1).

Tvrzení 1: časová složitost algoritmu *SeparatePath* je dvakrát složitost BFS, což je $2 \cdot \mathcal{O}(|V| + |E|)$, kde $| V |$ je počet vrcholů v mapě a $| E |$ je počet hran v mapě.

Tvrzení 2: algoritmus je *neúplný*, protože cesta druhého agentu ovlivňuje nalezení řešení. Špatnou volbou první cesty bude zablokována druhá cesta, i když nějaké dvě alternativní cesty mohly existovat.

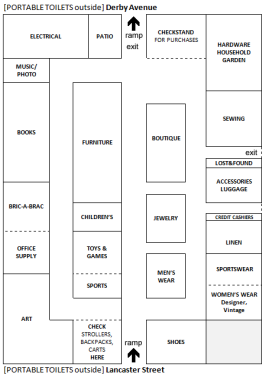
Tvrzení 3: algoritmus je *korektní*, protože vychází z BFS a ten je korektní.

Algoritmus 4.1 SeparatePath ($agent_x, agent_y$): vypočítání separátních cest

```
avoidVertices ← agent_y path ∪ agent_y start ∪ agent_y goal
closed, path ← ∅
opened ← agent_x start
while opened ≠ ∅ do
    current ← opened.front()
    if current is agent_x goal then
        agent_x path ← path
        break
    end if
    for all v ∈ current neighbors do
        if v ∉ opened & v ∉ closed & v ∉ avoidVertices then
            path ← (current, v)
        end if
    end for
    closed ← closed ∪ current
end while
if agent_x je první agent then
    SeparatePath(agent_y, agent_x)
end if
```

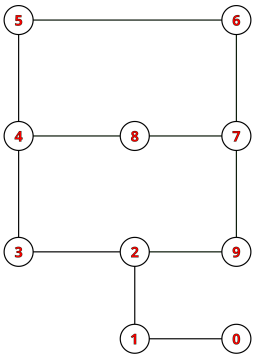
4.2 Scénář č. 1

Mapa č. 1 je představena na obrázku 4.2. Mapa vychází z mapy obchodního centra White Elefant Sale v Oaklandu, USA (obrázek 4.1) [14].

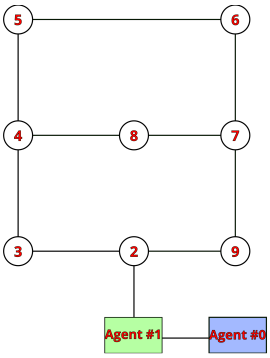


Obrázek 4.1: Motivační mapa pro mapu č. 1

Agenty mají obdélníkový tvar. Agent #0 se pohybuje z 0. vrcholu k 8. vrcholu. Agent #1 se pohybuje z 1. vrcholu k 2. vrcholu a je dvakrát rychlejší (obrázek 4.3, tabulka 4.1).



Obrázek 4.2: Testovací mapa č. 1



Obrázek 4.3: Počáteční pozice

V čase 3,9 agent #1 dosáhne svého cíle a zůstane tam, ale agent #0 potřebuje v čase 4,9 – 5,8 navštívit tento vrchol (obrázek 4.4). V tento okamžik dojde ke konfliktu, kde jeden agent už skončil svou cestu, ale druhý agent potřebuje využít jeho cílový vrchol. Agent #1 má jedinou možnost – posunout se na sousední vrchol. Agent #0 počká nejdřív 0,2 ve vrcholu 0 a poté 1,7 ve vrcholu 1, aby se nesrazily (tabulka 4.2).

Jelikož vnitřní vrcholy jsou seřazeny vzestupně, agent #1 se posune na vrchol 3. V čase 7,7 – 10,2 agent #0 se pohybuje z vrcholu 2 na vrchol 3.

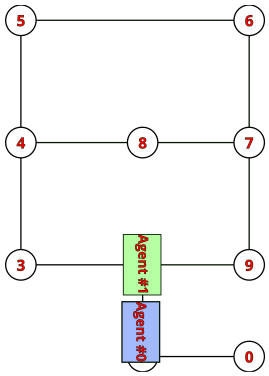
Tabulka 4.1: Původní cesta nalezena algoritmem **BFS**

Agent #0			Agent #1		
čas příjezdu	čas odjezdu	vrchol	čas příjezdu	čas odjezdu	vrchol
0	0	0	0	0,9	1
2,5	3,4	1	3,9	3,9	2
4,9	5,8	2			
8,3	9,2	3			
10,7	11,6	4			
14,1	14,1	8			

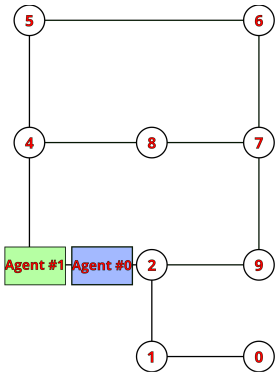
Současně se agent #1 pohybuje opačným směrem (obrázek 4.5). Dojde k hranovému konfliktu. Agent #0 najde jinou cestu přes vrchol 9 (obrázek 4.6).

Tabulka 4.2: Cesta agentů nalezena algoritmem **AvoidancePath**

Agent #0			Agent #1		
čas příjezdu	čas odjezdu	vrchol	čas příjezdu	čas odjezdu	vrchol
0	0,2	0	0	0,9	1
2,7	5,3	1	3,9	4,8	2
6,8	7,7	2	9,8	11,6	3
10,2	11,1	3	16,6	16,6	2
12,6	13,5	4			
16	16	8			



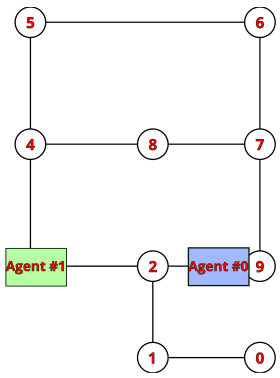
Obrázek 4.4: Překryvný konflikt



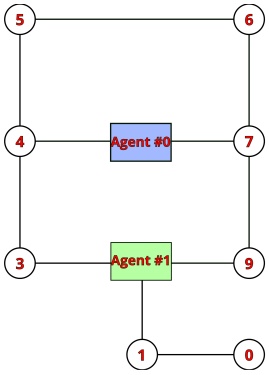
Obrázek 4.5: Hranový konflikt

Poté agenty dosáhnou své cílové vrcholy bez konfliktů (obrázek 4.7, tabulka 4.3). Výsledný čas pro agenta #0 je 16 a vzdálenost je 21, pro agenta #1 – 16,6 a 13.

Algoritmus pro hledání separátních cest nalezl cestu jen pro agenta #1. Cesta je přímá z vrcholu 1 do 2, trvá 3,9 a vzdálenost je 3. Cesta pro agenta #0



Obrázek 4.6: Alternativní cesta



Obrázek 4.7: Konečná pozice

Tabulka 4.3: Konečná cesta nalezena algoritmem **AvoidancePath**

Agent #0			Agent #1		
čas příjezdu	čas odjezdu	vrchol	čas příjezdu	čas odjezdu	vrchol
0	0,2	0	0	0,9	1
2,7	5,3	1	3,9	4,8	2
6,8	7,7	2	9,8	11,6	3
10,2	11,1	9	16,6	16,6	2
12,4	13,5	7			
16	16	8			

neexistuje, protože agent musí navštívit vrchol 1 a 2, které ale jsou počátečním a cílovým vrcholem agenta #1, a alternativní cesta z vrcholu 0 do vrcholu 3 nebo 9 neexistuje.

Ve výsledku algoritmus **AvoidancePath** je úspěšnější, protože nalezl obě cesty, když algoritmus **SeparatePath** si neporadil s cestou pro agenta #0 bez existence alternativní cesty.

4.3 Scénář č. 2

Druhý scénář byl také proveden na mapě č. 1.

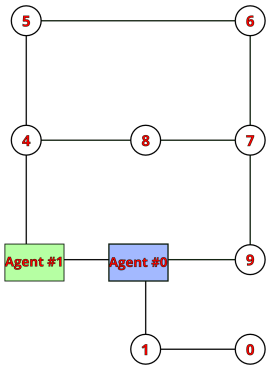
Agenty mají obdélníkový tvar. Agent #0 začíná cestu ve vrcholu 2, končí ve vrcholu 5 a má trojnásobnou rychlost. Agent #1 se pohybuje z vrcholu 3 na vrchol 8 (obrázek 4.8, tabulka 4.4).

V čase 4 dojde ke konfliktu, když agent #1 se bude otáčet ve vrcholu 4 a agent #0 dorazí na tento vrchol (obrázek 4.9). Agent #0 počká 2 ve vrcholu 3, než agent #1 se otočí a odjede na bezpečnou vzdálenost (obrázek 4.10).

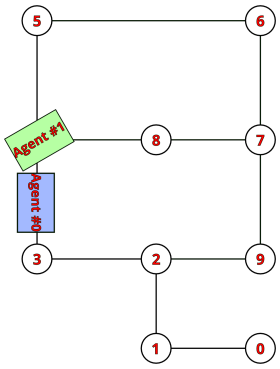
Výsledná cesta neobsahuje konflikty (obrázek 4.11, tabulka 4.5). Agent #0 ujel 12 za 7,9 a agent #1 ujel 8 za 9,8.

Tabulka 4.4: Původní cesta nalezena algoritmem **BFS**

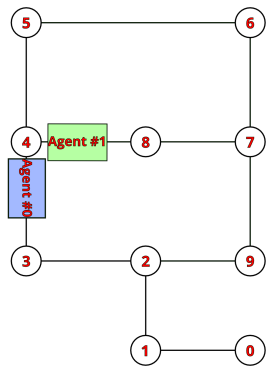
Agent #0			Agent #1		
čas příjezdu	čas odjezdu	vrchol	čas příjezdu	čas odjezdu	vrchol
0	0,9	2	0	0,9	3
2,6	3,5	3	3,9	4,8	4
4,5	4,5	4	9,8	9,8	8
5,8	5,8	5			



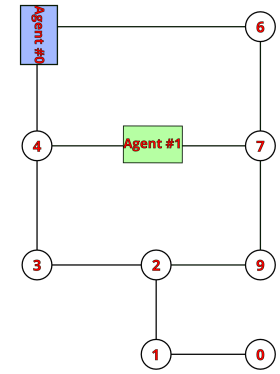
Obrázek 4.8: Počáteční pozice



Obrázek 4.9: Vrcholový konflikt



Obrázek 4.10: Agent #0 čeká



Obrázek 4.11: Konflikt je vyřešen

Separátní cesty existují (tabulka 4.6) pro oba agenty. Agent #1 má stejnou cestu. Agent #0 se měl vyhýbat vrcholům 3 a 4, a proto má delší cestu oproti cestě s vyhýbáním:

- čas je 10 oproti 7,9, což je o 27 % větší
- vzdálenost je 22 oproti 12, což je o 83 % větší

Tabulka 4.5: Konečná cesta nalezena algoritmem **AvoidancePath**

Agent #0			Agent #1		
čas příjezdu	čas odjezdu	vrchol	čas příjezdu	čas odjezdu	vrchol
0	0,9	2	0	0,9	3
2,6	5,6	3	3,9	4,8	4
6,6	6,6	4	9,8	9,8	8
7,9	7,9	5			

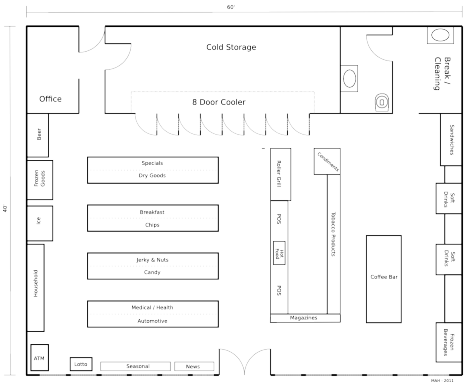
Z výsledných tabulek je vidět, že cesta nalezená algoritmem **AvoidancePath** je kratší a rychlejší, jelikož agent #0 se pohyboval kratší cestou.

Tabulka 4.6: Separátní cesta nalezená algoritmem **SeparatePath**

Agent #0			Agent #1		
čas příjezdu	čas odjezdu	vrchol	čas příjezdu	čas odjezdu	vrchol
0	0,9	2	0	0,9	3
2,6	3,5	9	3,9	4,8	4
4,5	4,5	7	9,8	9,8	8
5,8	6,7	6			
10	10	5			

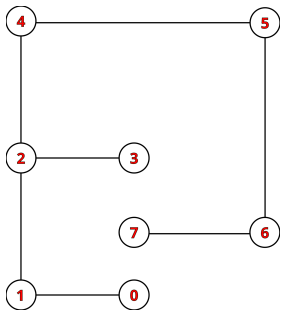
4.4 Scénář č. 3

Mapa č. 2 je představena na obrázku 4.13 a je motivována mapou skladu (obrázek 4.12) [15].



Obrázek 4.12: Motivační mapa pro mapu č. 2

Agenty mají obdélníkový tvar. Agent #0 se pohybuje z 0. vrcholu k 6. vrcholu a je dvakrát rychlejší. Agent #1 se pohybuje z 1. vrcholu k 7. vrcholu (obrázek 4.14, tabulka 4.7).

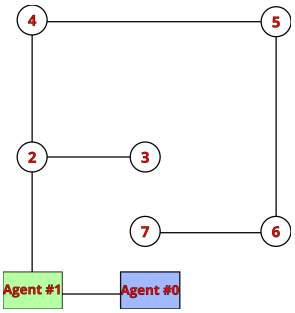


Obrázek 4.13: Testovací mapa č. 2

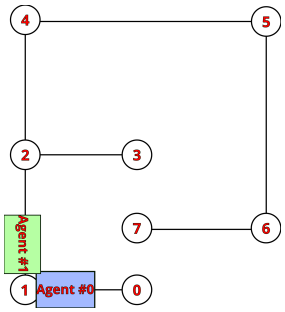
Tabulka 4.7: Původní cesta nalezena algoritmem **BFS**

Agent #0			Agent #1		
čas příjezdu	čas odjezdu	vrchol	čas příjezdu	čas odjezdu	vrchol
0	0	0	0	0,9	1
1,5	2,4	1	6,9	6,9	2
5,4	5,4	2	12,9	13,8	4
8,4	9,3	4	19,8	20,7	5
12,3	13,2	5	29,7	30,6	6
17,7	17,7	6	33,6	33,6	7

V čase 0,9 dojde ke konfliktu vedle vrcholu 1 (obrázek 4.15). Ke konfliktům bude docházet během společné části cesty, protože agent #0 je dvakrát rychlejší, ukončuje cestu dřív a agent #1 nemůže ho pustit před sebe. Proto agent #0 musí počkat 13,9, až agent #1 opustí společnou část cesty.



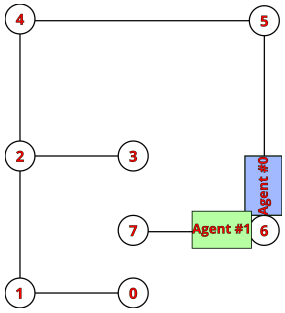
Obrázek 4.14: Počáteční pozice



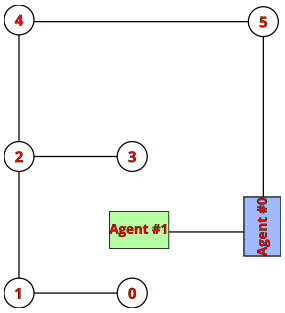
Obrázek 4.15: Překryvný konflikt

V čase 31 agenty se srazí vedle vrcholu 6 (obrázek 4.16), proto agent #0 musí počkat 1 ve vrcholu 5.

Cesty v tabulce 4.8 jsou finální a agenty dorazí do svých cílů (obrázek 4.17). Výsledný čas pro agenta #0 je 32,6 a vzdálenost je 30, pro agenta #1 – 33,6 a 30.



Obrázek 4.16: Překryvný konflikt



Obrázek 4.17: Konečná pozice

Tabulka 4.8: Konečná cesta nalezena algoritmem **AvoidancePath**

Agent #0			Agent #1		
čas příjezdu	čas odjezdu	vrchol	čas příjezdu	čas odjezdu	vrchol
0	13,9	0	0	0,9	1
15,4	16,3	1	6,9	6,9	2
19,3	19,3	2	12,9	13,8	4
22,3	23,2	4	19,8	20,7	5
26,2	28,1	5	29,7	30,6	6
32,6	32,6	6	33,6	33,6	7

Separátní cesty neexistují, protože agent #0 nesmí projíždět počátečním vrcholem agenta #1 a agent #1 nesmí projíždět cílovým vrcholem agenta #0.

Tato mapa má malou konektivitu a pro každou dvojici počátečního a cílového vrcholu existuje jenom jedna cesta. S tímto problémem se správně vypořádá algoritmus **AvoidancePath**.

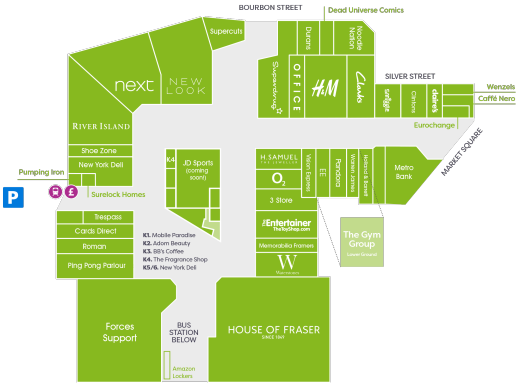
4.5 Scénář č. 4

Mapa č. 3 (obrázek 4.19) je motivována mapou obchodního centra Friars Square Shopping Centre v Aylesbury, Velká Británie (obrázek 4.18) [16].

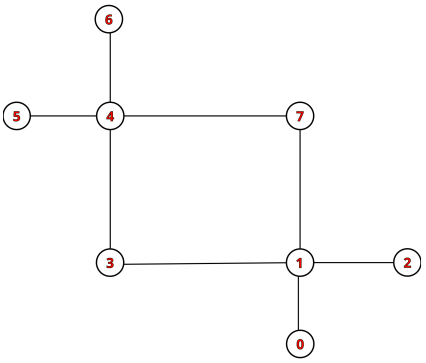
Agenty mají obdélníkový tvar. Agent #0 se pohybuje z 0. vrcholu k 6. vrcholu a je dvakrát rychlejší. Agent #1 se pohybuje z 2. vrcholu k 5. vrcholu (obrázek 4.20, tabulka 4.9).

V čase 1 se agenty srazí u vrcholu 1 (obrázek 4.21), proto agent #1 počká 1,3 v předchozím vrcholu. Ale to úplně nevyřeší problém. Příště se agenty potkají zase ve vrcholu 1, když agent #0 se bude otáčet (obrázek 4.22). Agent #1 počká 0,3 ve vrcholu 2, až agent #0 dokončí otáčení.

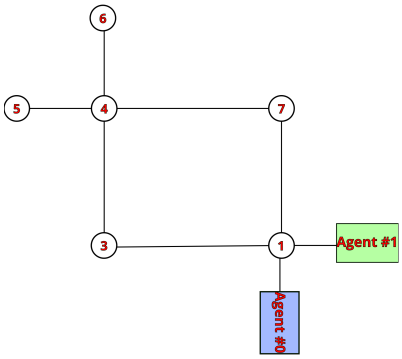
Díky tomu, že agent #0 je rychlejší, dorazí ke svému cíli první v čase 9,8 a nebude způsobovat dalším konfliktům (obrázek 4.23). V čase 19,4 dorazí i agent #1 (obrázek 4.24, tabulka 4.10). Agenty ujedou stejnou vzdálenost – 16.



Obrázek 4.18: Motivační mapa pro mapu č. 3



Obrázek 4.19: Testovací mapa č. 3



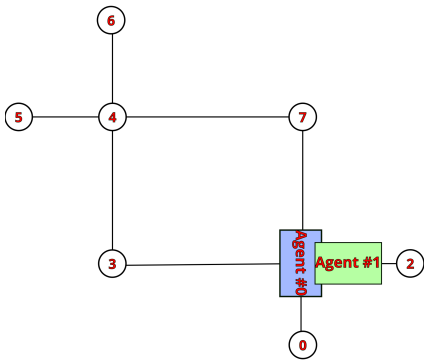
Obrázek 4.20: Počáteční pozice

Tabulka 4.9: Původní cesta nalezena algoritmem **BFS**

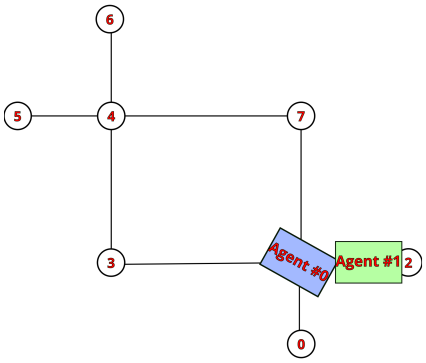
Agent #0			Agent #1		
čas příjezdu	čas odjezdu	vrchol	čas příjezdu	čas odjezdu	vrchol
0	0	0	0	0	2
1	1,9	1	2	2	1
4,9	5,8	3	8	8,9	3
8,8	8,8	4	14,9	15,8	4
9,8	9,8	6	17,8	17,8	5

Separátní cesta byla nalezena jen pro agenta #0 a odpovídá jeho finální cestě při prohledávání s vyhýbáním. Cesta pro agenta #1 nebyla nalezena, protože agent se musí vyhýbat vrcholům z cesty agenta #0. Ty vrcholy jsou 1, 3, 4.

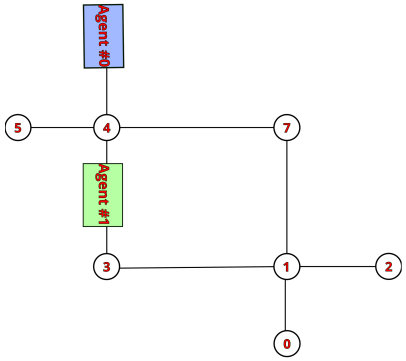
Ve výsledku oba algoritmy nalezly stejnou cestu pro agenta #0, ale algoritmus **AvoidancePath** navíc našel cestu pro agenta #1.



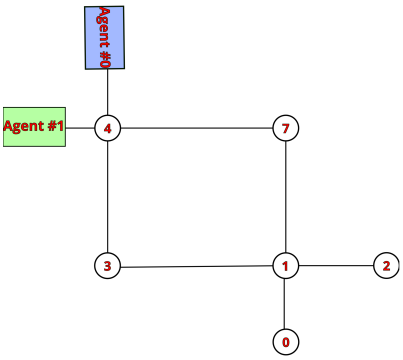
Obrázek 4.21: Překryvný konflikt



Obrázek 4.22: Konflikt při otáčení



Obrázek 4.23: Pohyb k cílím



Obrázek 4.24: Konečná pozice

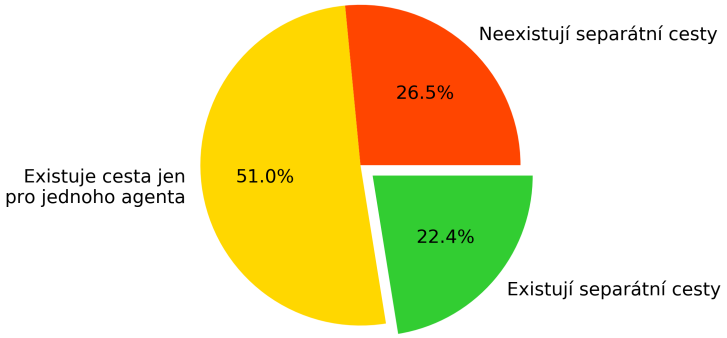
Tabulka 4.10: Konečná cesta nalezena algoritmem **AvoidancePath**

Agent #0			Agent #1		
čas příjezdu	čas odjezdu	vrchol	čas příjezdu	čas odjezdu	vrchol
0	0	0	0	1,6	2
1	1,9	1	3,6	3,6	1
4,9	5,8	3	9,6	10,5	3
8,8	8,8	4	16,5	17,4	4
9,8	9,8	6	19,4	19,4	5

4.6 Shrnutí

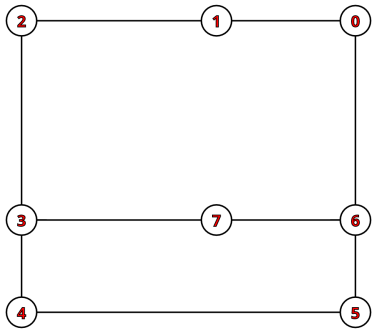
Po analýze všech 49 scénářů se ve výsledku prokázalo, že lze najít obě separátní cesty jen v 22,4 % případů (obrázek 4.25).

V grafech na obrázcích 4.31, 4.32 jsou srovnány scénáře, ve kterých algoritmus **SeparatePath** nalezl obě cesty, se stejnými scénáři s využitím algoritmu **AvoidancePath**.



Obrázek 4.25: Výsledek separátního prohledávání algoritmem **SeparatePath**

Z grafů je vidět, že několik z posledních scénářů pro mapu č. 4 jsou stejné. To je kvůli tomu, že tato mapa má velkou konektivitu[8] (obrázek 4.26) na rozdíl od ostatních map a několik alternativních cest pro každého agenta. V těchto scénářích se agent #0 pohybuje od 0. vrcholu do 3. vrcholu, agent #1 se pohybuje od 1. vrcholu do 2. vrcholu.



Obrázek 4.26: Testovací mapa č. 4

V ostatních scénářích pro mapu č. 4 byly zvoleny alternativní cesty, které jsou delší.

Ve scénáři pro mapu č. 3 (obrázek 4.19) algoritmy nalezly cesty stejné vzdálenosti, separátní cesty trvají o něco déle. V tomto scénáři se agent #0 pohybuje z vrcholu 3 do vrcholu 7 přes vrchol 1 (**AvoidancePath**) nebo přes

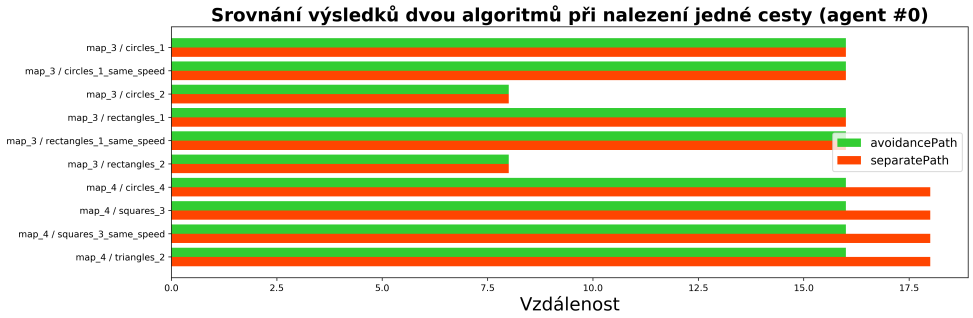
[8] každé dva vrcholy jsou spojeny více cestami, v tomto případě dvěma

vrchol 4 (**SeparatePath**). Agent #1 se pohybuje z vrcholu 1 do vrcholu 2 a je tak velký, aby v původní poloze na cílovém vrcholu zabraňoval pohybu agentu #0 a musel se trochu otočit. Ve výsledku algoritmus **AvoidancePath** ztrácí čas 0,4 na otáčení agentu #1 v cílovém vrcholu a algoritmus **SeparatePath** ztrácí čas 1,8 na dvojnásobném otáčení agentu #0 (jeho počáteční směr je doprava).

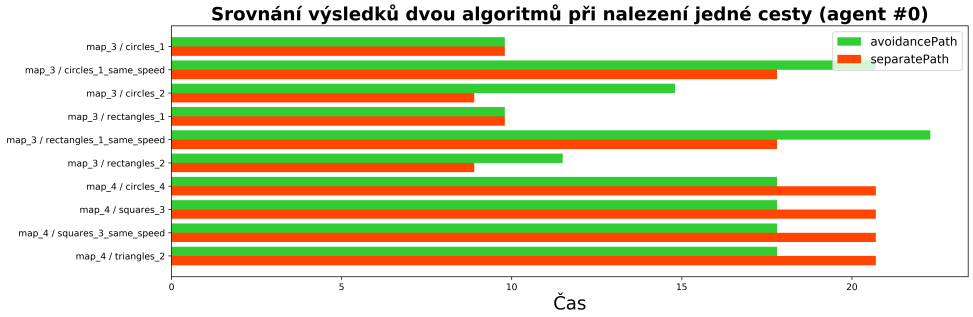
Scénáře pro mapu č. 1 se liší jak ve vzdálenosti, tak i v čase. Jeden z nich je detailněji popsán výše v části *Scénář č. 2*.

V 25 scénářích (51 %) byla nalezena cesta jenom pro jednoho agenta. V grafech na obrázcích 4.27, 4.28, 4.29, 4.30 jsou srovnány výsledky jednotlivých agentů nalezené pomocí algoritmu **SeparatePath** a **AvoidancePath**.

Jak je vidět z grafů pro agenta #0 (obrázky 4.27, 4.28), algoritmus **AvoidancePath** nalezl cestu buď stejné vzdálenosti, nebo kratší. Pokud je cesta kratší, pak i trvá kratší dobu. Naopak u některých stejných cest dochází ke srážkám, a proto tyto cesty trvají déle.

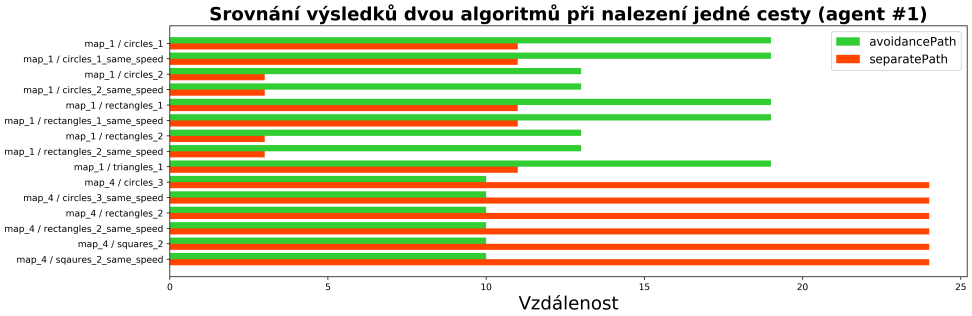


Obrázek 4.27: Srovnání výsledků dvou algoritmů při nalezení jedné cesty (vzdálenost)

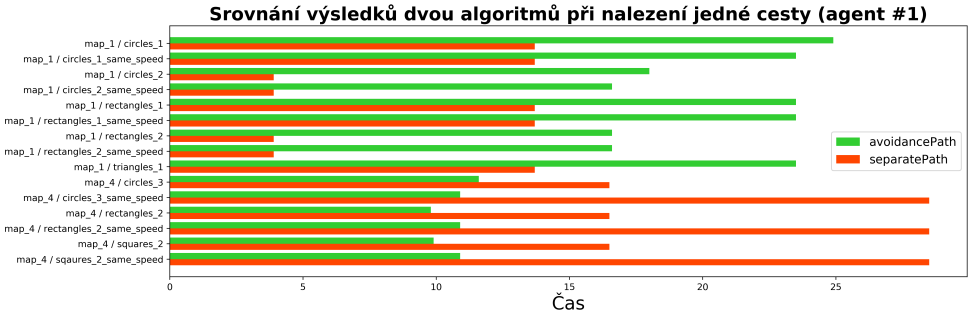


Obrázek 4.28: Srovnání výsledků dvou algoritmů při nalezení jedné cesty (čas)

Ve výsledcích byly pro agenta #1 (obrázky 4.29, 4.30) nalezeny různé cesty. Algoritmus **SeparatePath** nalezl delší cesty, protože se měl vyhýbat vrcholům agenta #0, když algoritmus **AvoidancePath** umožnil využití stejné cesty pro oba agenta. Algoritmus **AvoidancePath** nalezl delší cesty, protože někde došlo ke konfliktu, když agent #1 měl odbočit a uvolnit cestu.



Obrázek 4.29: Srovnání výsledků dvou algoritmů při nalezení jedné cesty (vzdálenost)

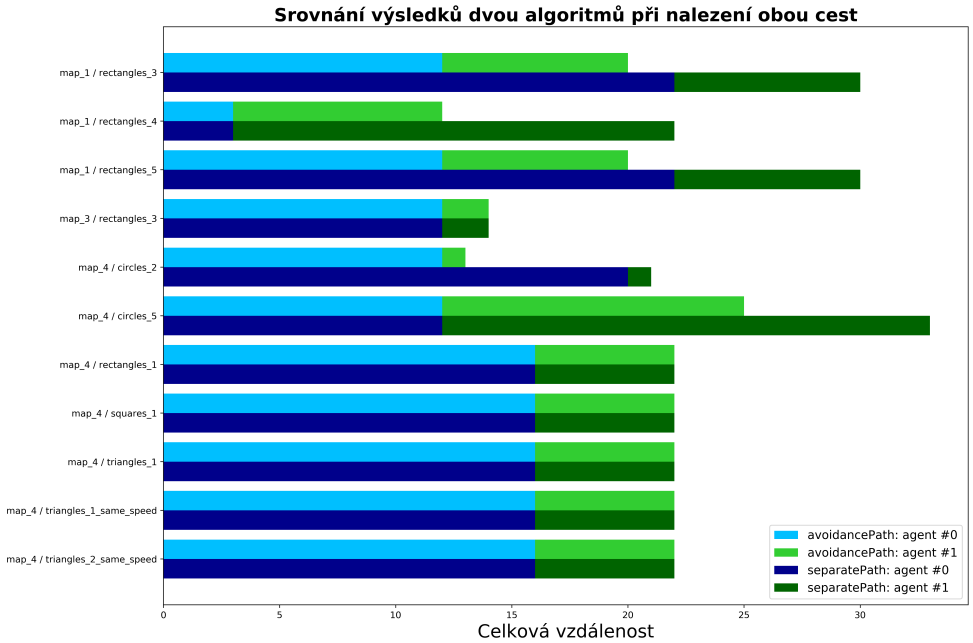


Obrázek 4.30: Srovnání výsledků dvou algoritmů při nalezení jedné cesty (čas)

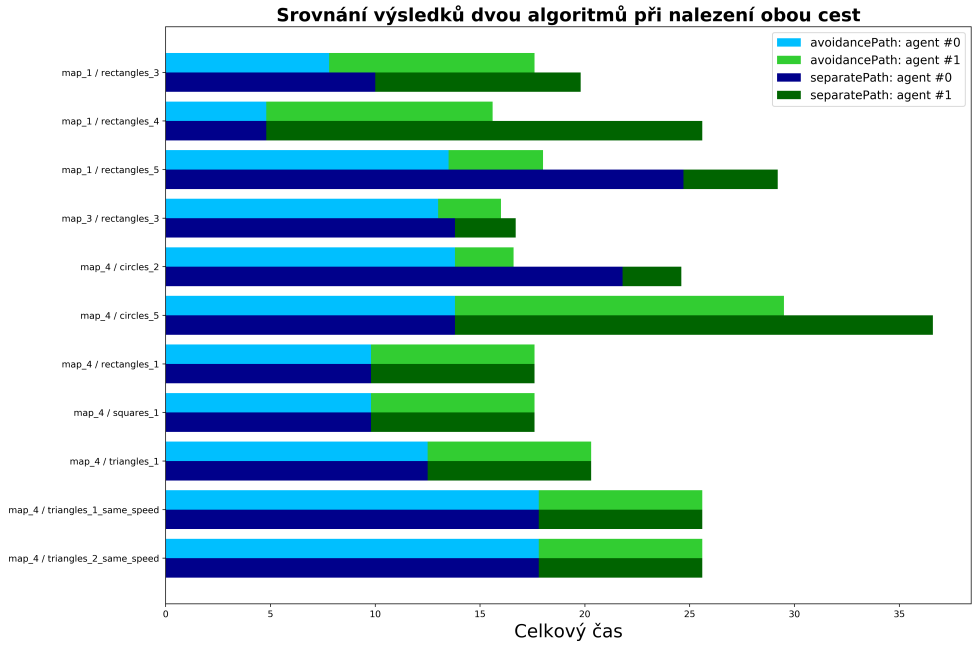
V 13 scénářích (26,5 %) nebyla nalezena žádná cesta.

Celkový závěr

Z těchto výsledků je vidět, že algoritmus s vyhýbáním je úspěšnější. Roli úspěšného nalezení cesty hraje konektivita mapy. Čím vyšší je konektivita mapy, tím častěji vyhýbací algoritmus ztrácí na významu, protože existuje více alternativních cest. Čím nižší je konektivita, tím významnější je využívání vyhýbacího algoritmu a tím nižší je úspěšnost nalezení separátních cest.



Obrázek 4.31: Srovnání výsledků dvou algoritmů při nalezení obou cest (vzdálenost)



Obrázek 4.32: Srovnání výsledků dvou algoritmů při nalezení obou cest (čas)

Kapitola 5

Analýza trhu

Robotika hraje důležitou roli v automatizaci skladů. První industriální robot byl vyroben v roce 1956 a byl schopen posouvat materiál jen o několik metrů [17]. Od té doby inženýři hodně pracují, aby propojili umělou inteligenci s automatickými roboty a tím pádem zvýšili efektivitu práce [18].

5.1 Současné využití

Zpočátku roboty vykonávaly jenom nebezpečnou a škodlivou práci jako třeba svařování a zvedání hmotných předmětů [18]. V průběhu času se manufakturní roboty výrazně vyvinuly a byly rozšířeny jejich pracovní možnosti. Současné skladové roboty se dělí na mnoho druhů, dál budou popsány tři nejpoužívanější.

Automaticky řízené roboty[9] se pohybují podle značek, tratě, senzorů, vodičů nebo jiných fyzických pomůcek na podlaze. Využívány jsou pro převoz materiálu a zásob ve skladu.

Autonomní mobilní roboty[10] jsou podobné AGVs tím, že pro navigaci ve skladu potřebují senzory. Na rozdíl od AGVs se neřídí značkami na podlaze, ale programem, vnějšími senzory a mapami. Využívány jsou pro třídění balíčků a jsou schopny měnit svou cestu na základě nových požadavků a překážek.

Automatizované skladové systémy[11] je technologie, která přináší a vyndává předměty ze skladu. Obvykle působí jako jeřáb s pevnou tratí a může snadně prohledávat celý regál. Využití AS/RS důrazně zrychluje kompletování objednávek a přemístění materiálů ve skladu.

Skladové roboty jsou využívány pro následující činnosti [18]:

[9] Automated guided vehicles (AGVs) [18, 19]
[10] Autonomous mobile robots (AMRs) [19]
[11] Automated storage and retrieval systems (AS/RS) [18, 19]

- *Sběr:* velká část kompletování objednávek je přesouvání se mezi regály a hledání jednotlivých položek, proto sbírání je nejběžnější operace pro skladové roboty

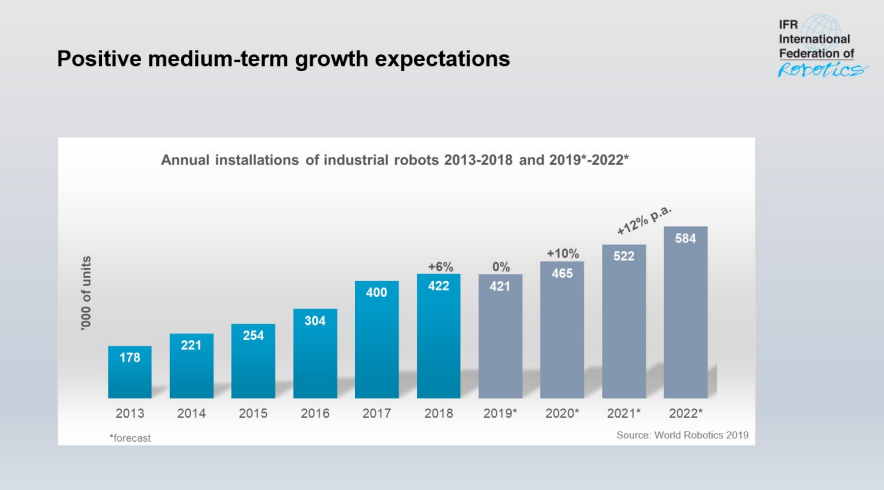
- *Třídění:* není tak jednoduchá úloha pro roboty, proto mají senzory, kamery a algoritmy, které pomáhají identifikovat předmět a poslat ho do správného regálu

- *Balení:* na základě váhy a rozměru objednávky roboty vypočítávají ideální velikost obalu

- *Doprava:* přeprava předmětů z jednoho konce skladu do jiného snižuje čas, který pracovník stráví kompletací objednávky

5.2 Výhody a nevýhody robotizace skladů

Počet industriálních robotů se zvyšuje každý rok a v roce 2020 přesáhl 3 miliony [20, 21] v celém světě (grafy na obrázcích 5.1, 5.2). Robotizace skladů má určité výhody, nicméně přináší i některé nedostatky.



Obrázek 5.1: Celková roční instalace industriálních robotů [21]

Výhody

Snížení zbytečného chození

Využití doručovacích robotů šetří čas pracovníků tím, že se nemusí tolik pohybovat skladem nebo ho vůbec navštěvovat. Pracovník by nevozil zboží ze

Obrázek 5.2: Celosvětový počet industriálních robotů [20]

skladu do obchodu, to by udělal robot a pracovník by jen vyndal zboží z robotu. Proto by sklad mohl být vzdálenější, protože robotu je jedno, jak daleko jede [22].

Zrychlení práce

Roboty můžou zvládat celý postup kompletování objednávky (sběr, třídění, doprava, balení) a dělat to paralelně. Pracovníkovi by jen zbývalo kontrolování kvality objednávky. Kromě toho pracovníci potřebují přestávky na odpočinek a jídlo, roboty tyhle přestávky nepotřebují, a proto mohou pracovat delší směny [22].

Zvýšení produktivity práce

Monotónní práce vyžaduje určitý čas, který pracovník může strávit víc intelektuálnějšími úkoly. Roboty můžou zrychlit rutinní procesy a řešit běžné problémy samostatně [22, 23, 24].

Snížení počtu chyb

Pracovníci občas dělají chyby z různých důvodů, roboty přece mají pevně definovány postup akce [23, 24].

Nevýhody

Žádná přizpůsobivost

Pracovníci jsou schopní analyzovat situaci a vynalézt nové řešení nějakého neočekávaného problému. Roboty se budou řídit jen programem [24].

Velké počáteční náklady

To je největší překážka nakupování robotů do skladu. Společnost musí investovat hodně peněz v implementaci a další údržbu, což může být podstatný problém pro malou firmu [23, 24].

Zvýšení nezaměstnanosti

Automatizace je přínosem pro zaměstnavatele, ale velkou hrozbou pro zaměstnance. S vývojem robotiky bude většina pracovníků nahrazena roboty [23, 24]. Tomuto problému se věnuje další podkapitola.

5.3 Vliv robotizace na ekonomiku a zaměstnanost

Automatizace a robotizace výroby přímo ovlivňují ekonomiku firmy. Například to přináší větší objem a kvalitu výrobků, snížení chybovosti produkce a počtu zaměstnanců [25, 26].

Nezaměstnanost

Každý rok víc robotů pracuje společně s lidmi, nebo úplně je nahrazuje. Už teď jsou roboty běžné ve skladech, například v Amazon, Tesla [27], KUKA [9] a Fanuc [10]. Ale brzy také nahradí i učitele, taxikáře, pokladníky a pracovníky různých dalších oborů [25, 28].

Roboty určitě představují velkou hrozbu pro nízko-kvalifikované pracovníky a pro některé středně-kvalifikované. Vědci předpovídají [29, 30], že k roku 2030 1,5 miliony Američanů ztratí práci kvůli robotizaci. V Číně bude toto číslo kolem 11 milionů. V Evropské unii zůstanou skoro 2 miliony lidí bez práce kvůli robotům.

Naopak zvýšení robotizace vytvoří více pracovních nabídek pro vysoko-kvalifikované pracovníky. Například roboty můžou vykonávat rutinní práci ve skladu, ale je potřeba, aby někdo kontroloval výsledek. S tímto se lépe vypořádá člověk s potřebnými dovednostmi [27].

Samozřejmě ne každý bude mít možnost změnit svou kvalifikaci. Lidé, kteří zůstali bez práci, budou mít peníze akorát na předměty denní potřeby, což přivede k tomu, že nabídka produkce bude vyšší než poptávka. Jedním z řešení tohoto problému je nepodmíněný příjem. Ten pomůže udržet úroveň spotřeby, která umožní společnostem prodávat svou výrobu.

Růst produktivity

Vyšší mzda, nižší ceny a vyšší kvalita výrobků povedou k vyšší životní úrovni. A to vede k vyšší produktivitě práce. Růst produktivity vyplývá ze zvýšení kvality práce, ke které dochází díky lepšímu vzdělávání a odborné přípravě zaměstnanců [27].

Růst HDP

Se zvýšením produktivity přichází růst HDP[12]. V článku [31] byl analyzován vliv robotů na ekonomiku 17 států. Ve výsledku se prokázalo, že zvýšení využití robotů vyvolalo růst HDP průměrně o 0,37 % ročně.

5.4 Přínos práce v kontextu robotizace skladu

Technologie se neustále zlepšují a roboty nabývají nové schopnosti. Výše byly popsány různé případy využití robotů v praxi a teď je propojíme s tématem práce.

Automatizace skladu umožňuje efektivnější organizaci práci a zkrácení skladových prostorů. Postupně se bude zvyšovat výška místností, protože roboty na rozdíl od lidí mohou bez problémů pracovat s víceúrovňovými regály.

V důsledku nárůstu výšek budov a snížení počtu pracovníků a počtu parkovacích míst bude možné umístění skladů s velkým obratem na menších pozemcích. Obdobně se snížením počtu zaměstnanců se zmenší i plocha sociálních prostorů, například místnosti pro občerstvení a odpočinek.

Proto tato práce směřuje k využití skladových robotů na malém prostoru. Čím více robotů se tam nachází, tím složitěji musejí manévrovat, aby nedošlo ke srážkám. Z toho důvodu je výhodné umět zajistit, aby se mohly otáčet a vyhýbat se konfliktům.

Domníváme se, že pro případy složitějšího manévrování, které je potřeba na menším prostranství, má tato práce manévrovací přínos.

[12] Hrubý domácí produkt

Závěr

Shrnutí práce

V první kapitole byla popsána teoretická východiska, bylo definováno MAPF$_R$ a metody pro jeho řešení. Ve druhé kapitole byla zavedena rozšíření o geometrický tvar a otáčení oproti původnímu algoritmu CCBS, také byl vysvětlen postup popisu tvaru agentu a potřebnost otáčení. Ve třetí kapitole je definován překryvný konflikt, jeho druhy a řešení každého případu. V další kapitole je předvedena ukázka práce algoritmu na konkrétních scénářích a jejich výsledné vyhodnocení. V poslední kapitole bylo analyzováno současné využití manufakturních robotů, výhody a nevýhody robotizace skladů pro zaměstnavatele a vliv na ekonomiku.

Rekapitulace cílů

Cíle práce byly:

1. Návrh rozšíření MAPF$_R$ pro dva geometrické roboty
2. Analýza algoritmů pro plánování pohybu více robotů
3. Analýza konfliktů při srážkách a jejich řešení
4. Rozšíření algoritmu o vyhýbání se srážkám
5. Teoretické a experimentální vyhodnocení algoritmu
6. Analýza ekonomicko-manažerského dopadu geometrické koordinace reálných robotů

Cíli 1, 3 a 4 jsme se zabývali v kapitolách 2 a 3. Cíl číslo 2 byl splněn v kapitole 1. Teoretické vyhodnocení bylo v kapitole 3 a experimentální – v kapitole 4. Poslednímu cíli se věnovala kapitola 5.

Na začátku jsme měli tři hypotézy o navrženém algoritmu:

- *Hypotéza 1:* Pro dva roboty libovolného geometrického tvaru bude nalezena cesta tak, aby se nesrazily, pokud tato cesta existuje.
- *Hypotéza 2:* Pro různé případy srážek bude zvoleno optimálnější řešení, aby výsledná cesta byla co nejkratší.
- *Hypotéza 3:* Hledání cesty s vyhýbáním bude efektivnější než bez vyhýbání (separátní cesta pro každého robota).

Ve výsledku všechny hypotézy platí:

- *Výsledek 1:* Došlo-li ke konfliktu, ten bude vyřešen, pokud je to možné, a algoritmus vrátí cestu bez srážek.
- *Výsledek 2:* Při srážce bude vybráno optimálnější řešení pro každý druh konfliktu, co zaručí nejkratší cestu.
- *Výsledek 3:* Z testovacích scénářů je vidět, že separátní cesta pro oba agenty byla nalezena málokdy a většinou v mapě s velkou konektivitou, když hledání s vyhýbáním dopadlo úspěšně v každé mapě.

Algoritmus dokáže najít nejkratší cestu pro dva agenty komplexního geometrického tvaru tak, aby se vyhnuly srážkám a nedošlo ke konfliktu.

Další rozšíření

Stále existuje mnoho možností pro rozšíření a zlepšení práce, například:

- zvýšení počtu agentů pohybujících se mapou
- kombinace agentů různých tvarů
- přizpůsobenost pracovníkovi: aby pracovník mohl interagovat s robotem
- hledání cesty pomocí jiných metod

Literatura

[1] Andreychuk, A.; Yakovlev, K.; Atzmon, D.; aj.: Multi-agent pathfinding with continuous time. *arXiv preprint arXiv:1901.05506*, 2019.

[2] Stern, R.; Sturtevant, N. R.; Felner, A.; aj.: Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Symposium on Combinatorial Search (SoCS)*, 2019: s. 151–158.

[3] Surynek, P.: An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, ročník 2, 01 2010, str. 1271–1273.

[4] Felner, A.; Stern, R.; Shimony, S. E.; aj.: Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Tenth Annual Symposium on Combinatorial Search*, 2017, s. 29—37.

[5] Wan, Q.; Gu, C.; Sun, S.; aj.: Lifelong Multi-Agent Path Finding in A Dynamic Environment. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, IEEE, 11 2018, s. 875–882, doi:10.1109/ICARCV.2018.8581181.

[6] Sharon, G.; Stern, R.; Felner, A.; aj.: Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, ročník 219, 2015: s. 40–66.

[7] Surynek, P.: Multi-agent Path Finding with Continuous Time Viewed Through Satisfiability Modulo Theories (SMT). *CoRR*, ročník abs/1903.09820, 2019, 1903.09820. Dostupné z: `http://arxiv.org/abs/1903.09820`

[8] Simon, M.: Your First Look Inside Amazon's Robot Warehouse of Tomorrow [online]. [Cit. 04.05.2020]. Dostupné z: `https://www.wired.com/story/amazon-warehouse-robots/`

[9] KUKA: Robotické systémy [online]. [Cit. 14.05.2020]. Dostupné z: https://www.kuka.com/cs-cz/produkty%2C-slu%C5%BEby/robotick%C3%A9-syst%C3%A9my

[10] Fanuc: Průmyslové roboty FANUC pro chytřejší automatizaci [online]. [Cit. 14.05.2020]. Dostupné z: https://www.fanuc.eu/cz/cs/roboty

[11] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; aj.: *Introduction to algorithms*. MIT press, 2009.

[12] Matthews, K.: 5 robots now in grocery stores provide a prevew of retail automation [online]. 04 2020, [Cit. 07.05.2020]. Dostupné z: https://www.roboticsbusinessreview.com/retail-hospitality/5-robots-grocery-stores-now/

[13] Time: Amazon's New Robots Are Shipping Your Order This Holiday [online]. 12 2014, [Cit. 07.05.2020]. Dostupné z: https://time.com/3605924/amazon-robots/

[14] Warehouse Map [online]. [Cit. 04.05.2020]. Dostupné z: http://www.whiteelephantsale.org/preview-sale/warehouse-map

[15] RETAIL DESIGN and CONSULTANCY - Retail Store Layout Drawing Service Provider from Gurgaon [online]. [Cit. 04.05.2020]. Dostupné z: http://www.indiamart.com/ab-cad-consultancy/retail-design-consultancy.html

[16] Centre plan: Friars Square Shopping Centre [online]. [Cit. 04.05.2020]. Dostupné z: https://friarssquareshopping.com/centre-plan/

[17] Robotic Industries Association: Unimate - The First Industrial Robot [online]. [Cit. 09.05.2020]. Dostupné z: https://www.robotics.org/joseph-engelberger/unimate.cfm

[18] Monroy, C.: What is warehouse robotics? [online]. 02 2020, [Cit. 09.05.2020]. Dostupné z: https://6river.com/what-is-warehouse-robotics/

[19] Bowles, R.: Warehouse Robotics: Everything You Need to Know in 2019 [online]. 03 2020, [Cit. 09.05.2020]. Dostupné z: https://www.logiwa.com/blog/warehouse-robotics

[20] IFR: Robots double worldwide by 2020 [online]. 05 2018, [Cit. 09.05.2020]. Dostupné z: https://ifr.org/ifr-press-releases/news/robots-double-worldwide-by-2020

[21] IFR: Industrial Robots: Robot Investment Reaches Record 16.5 billion USD [online]. 2019, [Cit. 14.05.2020]. Dostupné z: https://ifr.org/ifr-press-releases/news/robot-investment-reaches-record-16.5-billion-usd

[22] Glynn, F.: 3 key advantages of robotic warehouse systems [online]. 01 2020, [Cit. 10.05.2020]. Dostupné z: https://6river.com/3-key-advantages-of-robotic-warehouse-systems/

[23] Granta: Advantages and Disadvantages of Robotic Automation [online]. 2017, [Cit. 10.05.2020]. Dostupné z: https://www.granta-automation.co.uk/news/advantages-and-disadvantages-of-robotic-automation/

[24] Editorial: Advantages and disadvantages of warehouse robots [online]. 09 2019, [Cit. 10.05.2020]. Dostupné z: https://roboticsbiz.com/advantages-and-disadvantages-of-warehouse-robots/

[25] Javelosa, J.: Production Soared After This Factory Replaced 90% of Its Employees With Robots [online]. 02 2017, [Cit. 10.05.2020]. Dostupné z: https://futurism.com/2-production-soars-for-chinese-factory-who-replaced-90-of-employees-with-robots

[26] Kruglov, D.; Vorotynskaya, A.; Pozdeeva, E.: The impact of robotics on the labor market. *News from St. Petersburg State University of Economics*, 2017.

[27] Craig, A.: 3 Ways Robots Affect the Economy [online]. 01 2020, [Cit. 10.05.2020]. Dostupné z: https://www.investopedia.com/articles/markets-economy/091316/3-ways-robots-affect-economy.asp

[28] Job Tradition: 10 Common Jobs to Be Replaced by Robots Within 5 Years [online]. 08 2018, [Cit. 10.05.2020]. Dostupné z: https://www.jobtradition.com/10-common-jobs-to-be-replaced-by-robots-within-5-years/

[29] Economics, O.: How robots change the world: What automation really means for jobs and productivity [online]. 2019, [Cit. 10.05.2020]. Dostupné z: https://cdn2.hubspot.net/hubfs/2240363/Report%20-%20How%20Robots%20Change%20the%20World.pdf?utm_medium=email&_hsenc=p2ANqtz--K7kgPhJ7k-o3CX7f029ZmeMO_oDTNrwYYxrrVYFjKjh_0Oa3Wnz-U42mRNLGTqPLPd7TCgmS6n-ype13-3wEh-thBQw&_hsmi=74013545&utm_content=74013545&utm_source=hs_automation&hsCtaTracking=07b1855a-24f4-4b99-bcb8-b0d2a13b715e%7C53b7a48e-9591-4179-8eab-694443190b4f

[30] Taylor, C.: Robots could take over 20 million jobs by 2030, study claims [online]. 06 2019, [Cit. 10.05.2020]. Dostupné z: https://www.cnbc.com/2019/06/26/robots-could-take-over-20-million-jobs-by-2030-study-claims.html

[31] Graetz, G.; Michaels, G.: Robots at Work. *The Review of Economics and Statistics*, ročník 100, č. 5, 2018: s. 753–768, doi:10.1162/rest_a_00754.

Příloha **A**

Seznam použitých zkratek

AMRs Autonomní mobilní roboty

AS/RS Automatizované skladové systémy

AVGs Automaticky řízené roboty

BFS Prohledávání do šířky

CCSB Konfliktní prohledávání ve spojitém prostoru

CSB Konfliktní prohledávání

CT Constraint tree

HDP Hrubý domácí produkt

MAPF$_R$ Multi-agentní hledání cest ve spojitém prostoru

SOC Součet délek cest

Příloha B

Obsah přiložené SD-karty

- `readme.txt` ..... stručný popis obsahu SD-karty
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF

Příloha C

Příloha

Popis důležitých částí kódu.

C.1 Algoritmus AvoidancePath

Algoritmus slouží k nalezení nejkratší cesty pro dva agenty. Výsledná cesta nesmí obsahovat srážky. Algoritmus je implementován v rámci funkce *findPath*, která vrací *true*, pokud požadovaná cesta byla nalezena, a *false*, pokud se nevyšlo vyhnout konfliktu a výsledná cesta je prázdná. Pseudokód je popsán v algoritmu 3.1.

C.2 Algoritmy pro detekci konfliktu

V programu je pět algoritmů, které detekují konflikty. A jsou implementovány v funkcích:

- *checkVertexConflict1*: kontroluje, aby se agenty nenacházely současně ve stejném vrcholu
- *checkVertexConflict2*: kontroluje, jestli agent nevyužívá cílový vrchol druhého agenta, když ten už ukončil pohyb
- *checkEdgeConflict1*: kontroluje, aby se agenty nepohybovaly stejnou hranou, když jedou vstříc jeden druhému
- *checkEdgeConflict2*: kontroluje, jestli jeden agent nepředhání druhého, když se pohybují stejným směrem
- *checkDistance*: kontroluje, aby nedocházelo k překryvnému konfliktu

Nejdřív jsou zkontrolovány a vyřešeny vrcholové a hranové konflikty, poté jsou vyřešeny případy, když se oba agenty pohybují, nebo jeden agent se pohybuje a druhý je ve vrcholu.

C.3 Algoritmy pro řešení konfliktu

Konflikty se řeší různými způsoby: čekáním, otáčením nebo hledáním jiné cesty.

Algoritmus 3.2 **calcWaitingTime** je implementován ve funkci *calcWaitingTime*, která následně vybere potřebný výpočet času pro čekání na základě geometrického tvaru agenta.

Algoritmus 3.3 **calcRotation** pro počítání úhlu otáčení implementuje funkce *calcRotation*.

Hledá alternativní cestu funkce *edgeConflict*.

Pokud došlo k vrcholovému konfliktu prvního typu, to vyřeší funkce *vertexConflict*.

Druhý typ s cílovým vrcholem řeší funkce *atGoal*, která vybere mezi čekáním, posunutím na sousední vrchol nebo nalezením alternativní cesty.

C.4 Algoritmus SeparatePath

Algoritmus hledá dvě separátní cesty. Pseudokód je popsán v algoritmu 4.1 a implementován v samostatném programu.

C.5 Spouštění programu

Program se spouští přes příkazový řádek a potřebuje dva vstupní soubory (mapa a agenty) a algoritmus (**AvoidancePath** nebo **SeparatePath**).

- Mapa se nachází ve složce `/inputs/map$i/map$i`, kde `$i` je číslo mapy, `$i`$\in \{1, 2, 3, 4\}$.
- Popis agentů se nachází ve složce `/inputs/map$i/agents/$agent`, kde `$i` je již zvolené číslo mapy a `$agent` je soubor ve tvaru `$shape_$n`.

  `$shape` je geometrický tvar agenta a `$n` je číslo agenta tohoto geometrického tvaru pro tuto mapu.

  $$\texttt{\$shape} \in \{circles,\ rectangles,\ squares,\ triangles\}$$

  Poté program spustí příkaz `cat mapa agent | ./algoritmus`
  Příklad:
  `cat inputs/map1/map1 inputs/map1/agents/circles_1 | ./AvoidancePath`