VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

PLÁNOVÁNÍ TRAS MULTIROBOTICKÉHO SYSTÉMU V DYNAMICKÉM PROSTŘEDÍ

MULTIROBOT PATH PLANNING IN A DYNAMIC SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DOMINIK PLACHÝ**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Plachý Dominik**
Program: Informační technologie
Název: **Plánování tras multirobotického systému v dynamickém prostředí**
Multirobot Path Planning in a Dynamic System
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se se současnými metodami skupinového plánování cest.
2. Pro systém s více roboty-agenty pracující v dynamicky se měnícím prostředí zvolte nebo navrhněte takové přístupy plánování tras, které povedou k plošnému pokrytí oblasti. Uvažujte možné konflikty mezi roboty a možnost nutnosti přehodnocování naplánovaných tras při změnách prostředí.
3. Implementujte zvolené metody a ověřte jejich fungování ve vhodně zvoleném systému, nejlépe v systému soutěže MAPC z let 2019 a 2020.
4. Porovnejte fungování zvoleným metod pro různé parametry dynamiky prostředí, případně oproti již existujícím řešením, pokud existují. Zhodnoťte dosažené výsledky a diskutujte možná vylepšení do budoucna.

Literatura:

- Bude upřesněna, pravidla soutěže MAPC, plánovací algoritmy

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Tato práce se zabývá návrhem algoritmu pro plánování tras multirobotického systému za účelem pokrytí dynamického prostředí.

Problém pokrytí je vyřešen rozdělením prostoru na stejně velké oblasti, kterých je minimálně stejné množství jako agentů pokrývajících tento prostor. Agenti si mezi sebou oblasti rozdělují a řeší nastalé kolize. Také jsou schopni reagovat změnou plánu při nárazu na překážku.

Práce obsahuje naměřená a okomentovaná data z běhu algoritmu pro různé parametry dynamiky prostředí. Výsledkem práce je algoritmus, jehož efektivita se přímo úměrně zvyšuje s počtem agentů.

Abstract

This thesis deals with the design of a path planning algorithm for a multi-robot system in order to cover a dynamic environment.

The coverage problem is solved by dividing the space into equal sized areas, of which there are at least as many as the number of agents covering the space. The agents divide the areas among themselves and resolve any collisions that occur. They are also able to react by changing their plan when they encounter an obstacle.

The thesis contains measured and commented data from running the algorithm for different parameters of the environment dynamics. The result of the work is an algorithm whose efficiency increases directly proportional to the number of agents.

Klíčová slova

Multiagentní systém, agent, dynamické prostředí, neznámé prostředí, plánování tras, pokrytí prostoru.

Keywords

Multi-agent system, agent, dynamic system, unknown environment, path planning, terrain coverage.

Citace

PLACHÝ, Dominik. *Plánování tras multirobotického systému v dynamickém prostředí.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. František Zbořil, Ph.D.

Plánování tras multirobotického systému v dynamickém prostředí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Doc. Ing. Františka Zbořila, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Dominik Plachý
11. května 2021

Poděkování

Chtěl bych zde poděkovat vedoucímu práce Doc. Ing. Františku Zbořilovi, Ph.D. za výborné vedení, rady a doporučení.

Obsah

Kapitola 1

Úvod

Multirobotické systémy se staly velice populární kategorií umělé inteligence. S multirobotickými systémy se lze setkat ve spoustě odvětví, například: akademický výzkum, videohry, filmová produkce, počítačové sítě, doprava, logistika, různé simulace vývoje epidemií, nebo klimatu.

Inspirací pro tuto práci je mezinárodní soutěž multiagentních systémů „Multi-Agent Programming Contest“[1] (dále pouze MAPC). Soutěž MAPC je pořádána téměř každý rok a účastní se jí univerzitní týmy z celého světa včetně týmu z VUT FIT. Každý ročník soutěže má svůj scénář a týmy podle něj navrhují svoje multiagentní systémy. V rámci soutěže spolu týmy svádí zápasy svých multiagentních systémů, jejichž cílem je plnit úkoly, za které získávají body. V posledním ročníku těmito úkoly bylo přemísťování a seskupování bloků do zadaného tvaru[2]. Do práce agentů libovolně zasahuje prostředí změnou pozic překážek nebo změnou cíle a agenti musí na tyto změny reagovat.

Tato bakalářská práce se zabývá jednou částí toho, co musí multiagentní systémy v soutěži MAPC umět – efektivní prohledávání prostoru a s tím spojená komunikace mezi agenty. Cílem této práce je tedy pro multiagentní systém navrhnout algoritmus, podle kterého si budou jednotliví agenti plánovat cesty tak, aby jako celek co nejefektivněji opakovaně prohledávali oblast a zapisovali pozice překážek. Prostředí, ve kterém toto prohledávání probíhá, je dynamické, tudíž se v něm mohou různě objevovat a mizet překážky a agenti musí tyto změny co nejdříve detekovat a reagovat na ně. Klíčem pro efektivní prohledávání prostoru multiagentním systémem je rozprostření agentů po celé ploše a jejich vzájemná komunikace.

V kapitole 2 je nejdříve věnován prostor stručné teorii k multiagentním systémům a agentům. Následně je pojednáno o aktuálních řešeních obdobných problémů a nakonec je blíže specifikováno prostředí, ve kterém se bude algoritmus popisovaný touto prací testovat. V kapitole 3 je podrobně popsán návrh celého algoritmu, na což navazuje jeho implementace v kapitole 4. Nakonec je v kapitole 5 fungování algoritmu změřeno a okomentováno pro různé parametry prostředí. V závěru je pak celý algoritmus zhodnocen a je navrženo možné budoucí vylepšení algoritmu.

[1] https://multiagentcontest.org
[2] https://github.com/agentcontest/massim_2020/blob/master/docs/scenario.md

Kapitola 2

Multiagentní systémy

V následujících odstavcích se vyskytují informace převzaté z [7] a [6].

Multiagentní systémy spadají do kategorie distribuované umělé inteligence. První takové systémy se začaly objevovat v 80. letech 20. století. Od té doby se tato softwarová technologie stále vyvíjí. Inteligentní agenti, kteří společně tvoří multiagentní systém, nabízejí efektivní způsob budování inteligentních systémů. Principem multiagentních systémů je rozdělení daného komplexního problému na části, ty pak na menší a menší části, až vzniknou jednoduché podproblémy dohromady tvořící původní problém. Tyto jednoduché problémy pak mohou řešit jednotliví agenti multiagentního systému. Výhodou řešení komplexních problémů multiagentními systémy je rychlost vyřešení problému díky paralelně pracujícím agentům. Aby ale byl multiagentní systém opravdu rychlejší než jeden samotný agent, musí se vypořádat s několika problémy:

- **Prostředí** – Agenti často pracují na řešení svých problémů v nějakém prostředí. Prostředím může být prostá mřížka nebo například videoherní svět. Toto prostředí také často nejen že prozkoumávají, ale i modifikují. Zde nastává situace, že agent nemění jenom svoje okolí ale i okolí ostatních agentů a může se tak stát, že jeden agent zmodifikuje prostředí tak, že jinému agentovi překazí plány. Agenti by se tak měli snažit pokud možno předvídat akce ostatních agentů, předem s nimi počítat a plánovat podle nich svoje akce. Předvídání akcí musí být ale dobře navrženo, aby v multiagentním systému nedocházelo k možným nestabilitám a chaosu. Tento problém však v této bakalářské práci nemusí být řešen, protože agenti prostředí nijak nemodifikují.

- **Průzkum prostoru** – V systému jednoho agenta si agent celý pracovní prostor projde a uloží sám pro sebe. Pokud by si měl každý agent v multiagentním systému sám projít celé prostředí a uložit, nebylo by to o nic rychlejší než jednoagentní systém. Pro lepší efektivitu by si proto agenti měli navzájem poskytovat informace o prozkoumaných oblastech. Toho lze v případě spolupracujících agentů relativně snadno dosáhnout jejich komunikací.

  Na průzkum prostoru se váže ještě jeden problém, který je u multiagentního systému pro co nejlepší efektivitu potřeba vyřešit. Tím problémem je správné rozdělení prostoru mezi agenty, aby prozkoumávání přidělených oblastí trvalo všem agentům přibližně stejně dlouhou dobu a nedocházelo tak k situacím, že jeden agent má hotovo a stojí a druhý agent má prozkoumanou teprve polovinu své oblasti, protože se v ní například nacházelo mnoho překážek, které musel zdlouhavě obcházet. Brzy skončený agent by tak měl pro co nejvyšší efektivitu průzkumu pomoci stále prozkou-

mávajícímu agentovi, a to tak, že prozkoumávající agent se se skončeným agentem o svoji oblast podělí.

- **Konflikty** – Konflikty už byly lehce popsány v bodě „Prostředí“. Protože všichni agenti vykonávají naplánované kroky současně, může nastat jejich konflikt. Konflikt agentů je naplánování dvěma a více agenty takových akcí, které se navzájem vylučují, například krok na stejnou pozici. Všichni zúčastnění agenti si v takovém případě musí vykomunikovat řešení, které bude ze všech dostupných řešení nejlépe směřovat k cíli (často musí přistoupit na nějaký kompromis).

- **Komunikace** – Komunikace je v multiagentních systémech klíčová. Jakmile se má více agentů podílet na plnění nějakého zadaného úkolu, bez komunikace to nezvládnou. Komunikace je nejvíce podstatná na začátku plnění úkolu, kdy si agenti rozdělí podúkoly, a pak se už pouze případně koordinují. Ke komunikaci jsou nejčastěji využity agentní komunikační jazyky, jako KQML nebo FIPA-ACL.

2.1 Agent

Informace v této sekci jsou převzaty z [7], [6] a [9].

Označení agent o jeho nositeli naznačuje, že je podřízený nějakému vedení a že pro někoho pracuje. Agenti v multiagentních systémech jsou toho příkladem. Jsou podřízení celkovému systému a pracují pro něj. Pojem agent však nemá v informačních technologiích jednotnou definici, protože se inteligentní agenti mohou vyskytovat v mnoha podoborech od robotů po počítačové sítě. Autory nejuznávanější definice agenta jsou S. Russell a P. Norvig. Agent je podle nich definován jako flexibilní autonomní entita schopná pomocí senzorů přijímat informace z prostředí a plnit svoje cíle. Agentem je v podstatě kontrolér, který kontroluje svoje vlastní chování. Od běžných prostých kontrolérů jej ale liší následující vlastnosti:

- **Fyzická přítomnost v prostředí** – Agent se obvykle fyzicky nachází v prostředí (Agent také může ovládat entitu, která se v prostředí fyzicky nachází místo něj. Tuto entitu lze ovšem s agentem pro představu spojit.) a má omezené vidění svého okolí. To znamená, že se nachází na své pozici, vidí kolem sebe do určité vzdálenosti v určitém úhlu a aby se například dostal na druhou stranu prostředí, musí tam postupně dokráčet. Pomocí senzorů přijímá z prostředí podněty a reaguje na ně svými akcemi. V prostředí působí přímo a ne pouze jako prostředník.

- **Autonomie** – Agent se o svých akcích rozhoduje svobodně na základě svých cílů. Díky autonomii jsou agentovi znalosti a stavy izolovány od externích vlivů a ostatním agentům je poskytne pouze, když to sám uzná za potřebné.

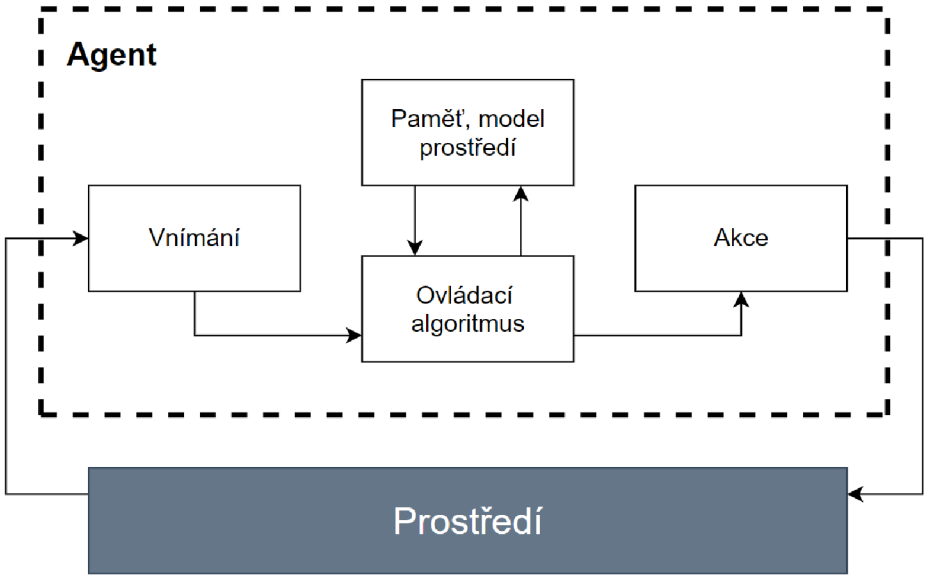
- **Proaktivita** – Agent se snaží zjednodušovat si plnění svého cíle ovlivňováním okolního prostředí, aby se dostalo do stavu co nejvýhodnějšího pro agenta a jeho cíl.

- **Schopnost správně reagovat** – Agent má často sestavený plán svých akcí tak, aby co nejrychleji nebo za co nejnižší cenu dosáhl svého cíle. Musí mít ale schopnost tento plán v případě jakékoliv překážky v provedení původního plánu přeplánovat, a to tak, aby byl nový plán v rámci nových možností znovu nejlepší.

- **Interakce s ostatními agenty** – Agent sice musí být svobodný ve svých úkonech, ale přesto často potřebuje interagovat s externími zdroji, například když je ke splnění cíle potřeba více spolupracujících agentů. Agent také musí být schopen sdílet svoje znalosti s ostatními agenty. S tím také souvisí, že musí být schopen znalosti od jiných agentů přijímat a například se z nich do budoucna učit. Agent však nemusí interagovat nutně pouze s jinými agenty, ale může také interagovat například s člověkem nebo prostými kontroléry.

Architektura

Základní architektura agenta je znázorněna na obrázku 2.1.



Obrázek 2.1: Základní architektura agenta převzatá z [6].

Architektury agentů mohou být různě modifikovány podle typů problémů, na které jsou agenti nasazeni. Hlavními architekturami jsou: (následující body jsou převzaty z [8])

- **Inteligentní architektura** – Agent pracující na této architektuře se snaží dosáhnout svých cílů pomocí svého vlastního rozhodování (inteligence).
- **Reaktivní architektura** – Agent neobsahuje žádnou interpretaci prostředí, ve kterém se nachází. Své kroky provádí spontánně jako reakce na změny v prostředí. Podle R. Brooka inteligentní jednání agenta přichází jako výsledek agentovi interakce s prostředím [3].
- **Deliberativní architektura** – Narozdíl od rekativní architektury agent pracující na deliberativní architektuře v sobě obsahuje interpretaci prostředí a tuto interpretaci společně s informacemi ze svých senzorů využívá k pečlivému naplánování svých budoucích kroků.
- **Kognitivní architektura** – Agent této architektury je hodně založen na učení se z prostředí kolem něj. Ze znalostí, které učením získá, pak dedukuje logické závěry, pomocí kterých se rozhoduje o svých akcích.

- **Racionální architektura** – Racionální architektura agenta je spojením všech výše uvedených architektur. Agent této architektury je tedy nejpokročilejší a využívá jak učení, tak plánování podle rychlosti a kvality dosažení svých cílů.

2.2 Aktuální řešení problému

Problém skupinového plánování cest za účelem prozkoumání celé oblasti se v multiagentních systémech často řeší ve statickém prostředí. V této sekci jsou proto popsány 2 algoritmy pro pokrytí neznámého statického prostředí multiagentním systémem, které byly největší inspirací pro výsledný algoritmus popisovaný touto prací.

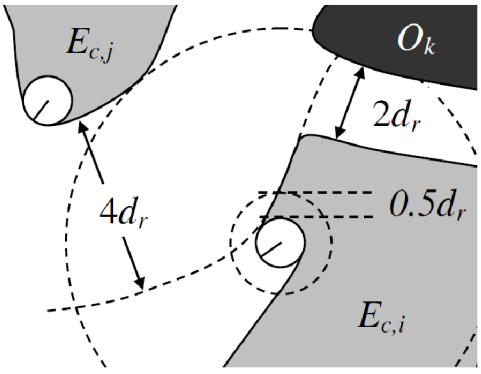
Pokrytí neznámého prostředí multirobotickým systémem s využitím falešných překážek

Algoritmus, který bude v následujících odstavcích popsán, je uveden v [4]. Jeho hlavním cílem je prozkoumat celou plochu s minimálním procházením již prozkoumaných oblastí. Prostředí, pro které je algoritmus navrhnut, je statické.

Pro fungování algoritmu musí agenti splňovat následující dva předpoklady:

- Všichni agenti mohou pomocí svých senzorů detekovat překážku na vzdálenost dvojnásobnou jejich vlastní velikosti – dohled agentů musí být vyšší nebo roven dvojnásobku jejich průměru.
- Všichni agenti znají hranice prostředí nebo je při nalezení dokáží identifikovat.

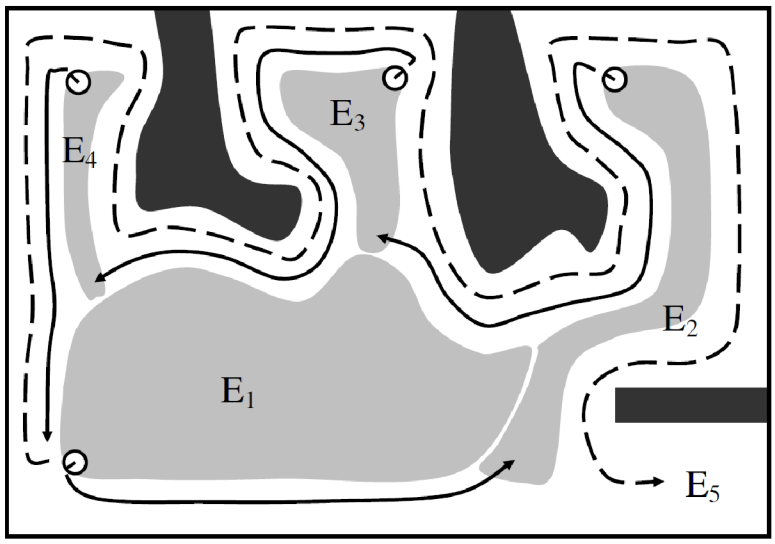
Důležitou vlastností algoritmu je snaha o minimální opakování průzkumu. Toho je dosaženo tím, že každý agent prostor, který prozkoumá, uloží jako takzvanou falešnou překážku, kterou ostatní agenti sice od normálních překážek odlišují, ale chovají se k ní jako k normální překážce. Dalším důležitým rysem algoritmu je zanechávání volných prostorů mezi jednotlivými překážkami (reálnými i falešnými) a hranicemi celé plochy, díky čemuž zůstávají všechny nepokryté části po celou dobu propojeny. Vynechávání volných prostorů je zobrazeno na obrázku 2.2.



Obrázek 2.2: Znázornění vynechávaných vzdáleností mezi překážkami převzaté z [4]. d_r – průměr agenta, O_k (černá plocha) – reálná překážka, $E_{c,x}$ (šedé plochy) – falešná překážka vytvořená agentem x.

Velikost těchto prostorů závisí na typech překážek, mezi kterými se prostor nachází. Mezi falešnou překážkou a reálnou překážkou nebo hranicí celé plochy agenti nechají volný prostor o šířce rovné dvojnásobku průměru agenta a mezi dvěma falešnými překážkami nechají agenti volný prostor o šířce rovné čtyřnásobku průměru agenta (to znamená, že vzniknou cesty, na kterých se mohou bez problému potkat 2 (4) agenti). Tyto volné prostory později slouží k cestování agentů, kteří hledají nové oblasti k prozkoumání. Z toho tedy vyplývá, že každý agent za sebou vytváří a „vlastní“ svoji falešnou překážku.

Agenti mohou nabývat dvou módů: normální a uzavírací. Během normálního módu agent vždy dodržuje zmíněné vzdálenosti od reálných i falešných překážek a hranice plochy. Pohybuje se kolem své falešné překážky a stále ji svým pohybem rozšiřuje do všech stran, dokud není limitován vzdáleností od ostatních překážek. Jakmile se ale agent dostane do bodu, odkud nemůže bez vstoupení do své falešné překážky pokračovat v pohybu, přepne se do uzavíracího módu. V tomto módu agent změní směr a postupuje zpět podél své falešné překážky po vynechaných prostorech. Neustále kontroluje vzdálenost své falešné překážky od ostatních překážek nebo okraje plochy a pokud je tato vzdálenost větší než minimální možná, tak se přepne zpět do normálního módu a začne tímto směrem rozšiřovat svoji falešnou překážku. Proces přepínání mezi módy je vidět na obrázku 2.3. Při uzavíracím módu může také agent detekovat úzký průchod do neprozkoumané oblasti. Informace o průchodu si uloží a až nebude mít kam rozšiřovat svoji falešnou překážku, tak se průchodem vydá. Tam buď v případě většího prostoru založí novou falešnou překážku, nebo pokud je prostor malý, tak jej projde a vrátí se zpět.



Obrázek 2.3: Ukázka postupu jednoho agenta v prozkoumávání převzatá z [4]. Černé plochy – reálné překážky, šedé plochy – falešné překážky. Agent vytvořil plochy v pořadí E_1-E_4 a má před sebou přesun do neprozkoumané oblasti E_5. Agent v obrázku vždy rozšiřuje svoji prozkoumanou oblast a jakmile se dostane do bodu, ze kterého nemá oblast kam rozšířit, tak aktivuje uzavírací mód a přesouvá se proti směru hodinových ručiček podél prozkoumané oblasti dokud nenarazí na neprozkoumanou plochu, kde zahájí další objevování.

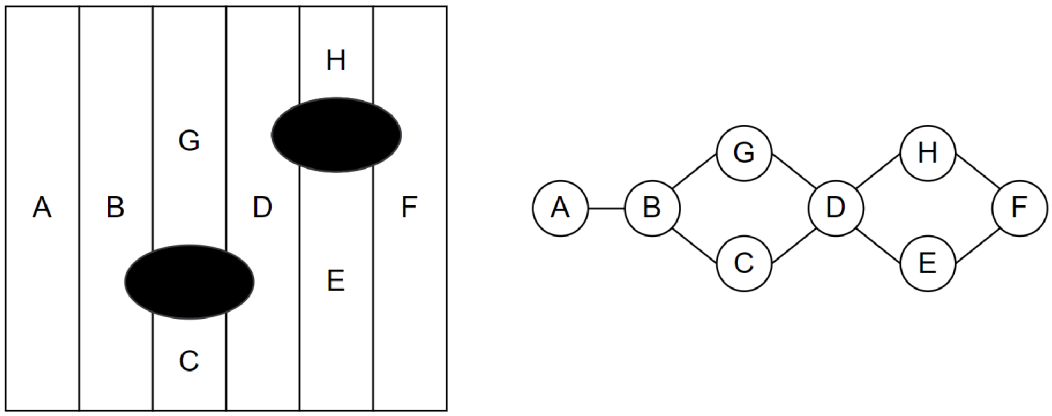
Jelikož pokrytý prostor není vyznačen pouze falešnými překážkami, ale jsou jím i cesty mezi překážkami, agenti dokončí průzkum ve chvíli, kdy žádný z nich už nemůže do žádného směru rozšířit svoji falešnou překážku. To totiž znamená, že jsou všechny falešné překážky

v minimální vzdálenosti od ostatních překážek nebo hranice plochy a agenti už tedy vše prozkoumali.

Tento algoritmus je ovšem výhradně určený pro statické prostředí a v algoritmu, který tato práce navrhuje, proto není použit.

Distribuované pokrytí neznámého prostředí multirobotickým systémem pomocí rozdělení prostředí na sloupce

Tento algoritmus popsaný v [5] se stejně jako předchozí algoritmus zaměřuje na kompletní pokrytí (prozkoumání) statického prostoru. Principem algoritmu je rozdělení prostoru na sloupce o šířce rovné dvojnásobku velikosti průměru dohledu agenta. Tyto sloupce jsou postupně prozkoumávány jeden po druhém a nikdy není žádný sloupec sdílen více agenty. Agenti mezi sebou sdílí takzvaný Reebův graf, jehož uzly představují jednotlivé sloupce a jsou mezi sebou spojeny hranami přesně tak, jak vedle sebe sloupce v prostředí leží. Příklad grafu je na obrázku 2.4.



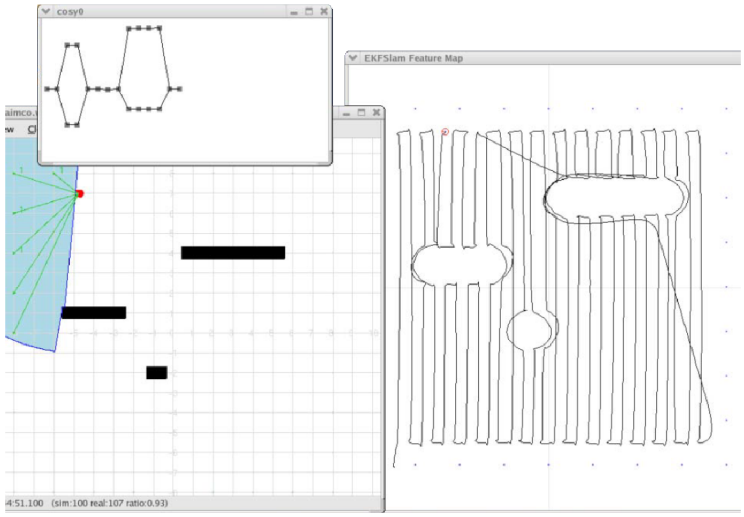
Obrázek 2.4: Příklad Reebova grafu převzatý z [5]. Vlevo: reálná podoba prostředí, vpravo: Reebův graf k tomuto prostředí. Černé ovály – překážky, A-H – jednotlivé sloupce (části sloupců), které jsou samostatně prozkoumávány.

Prozkoumávání jednoho sloupce probíhá dvěma cestami (nahoru a dolů) a poté následuje přesun do dalšího sloupce. Při prozkoumávání sloupce mohou nastat 3 případy blokace překážkou:

- Do sloupce žádná překážka nezasahuje – Agent jej projde klasicky cestou nahoru a dolů (nebo naopak).
- Do sloupce zasahuje okraj překážky – Agent překážku obejde aniž by opustil svůj sloupec.
- Sloupec je přerušen překážkou – Agent narazí na překážku, zjistí, že se dál ve sloupci nedostane a zahájí zpáteční cestu nebo cestu do nového sloupce podle jeho aktuálního stavu. Zároveň v Reebově grafu uzel reprezentující aktuální sloupec rozdělí na 2 a příslušně je v grafu propojí hranami s vedlejšími uzly. Ostatní agenti se tak z grafu o druhé neprozkoumané části sloupce dozví a mohou se ji vydat prozkoumat.

Jakmile jsou všechny sloupce prozkoumané a tím pádem i všechny uzly grafu označené za prozkoumané, je úkol splněn.

Tato práce vychází zejména z tohoto algoritmu. Konkrétně je v práci převzato řešení problematiky dělení prostředí na sloupce. Kvůli dynamickému prostředí v této práci je ale algoritmus v mnohých věcech modifikován.



Obrázek 2.5: Reálná ukázka fungování algoritmu převzatá z [5]. Prostor je zde prozkoumáván jedním robotem a v prostoru se nacházejí 3 překážky. Vlevo nahoře na obrázku je vidět Reebův graf, který si agent během prozkoumávání vytvořil, vpravo je vidět reálná trajektorie pohybu agenta.

2.3 Prostředí multiagentního systému pro tento projekt

Původním plánem této práce bylo implementovat algoritmus v prostředí soutěže MAPC. Prostředím soutěže MAPC je mřížka, jejíž délky stran jsou dány náhodně v intervalu asi 70-100 buněk. Obsahuje náhodné shluky překážek, které se mohou náhodně měnit, což z prostředí dělá dynamické prostředí. Specifickou vlastností prostředí soutěže MAPC je horizontální a vertikální nekonečnost. To znamená, že mřížka nemá hranice a agent, který z mřížky vyjde vpravo, se objeví vlevo. Další vlastností tohoto prostředí je, že agenti neznají svoje absolutní souřadnice a mohou si pouze podle svého pohybu odvozovat relativní souřadnice vůči startovnímu bodu. Aby mohlo více takových agentů spolupracovat na stejných úkolech (v této práci prozkoumávat oblast), musí se nejdříve navzájem nalézt a synchronizovat znalosti o prostředí a souřadnice. Teprve po synchronizaci jsou schopni nějaké spolupráce.

Protože tématem této bakalářské práce je pouze plánování tras za účelem prozkoumávání prostředí a ne synchronizace agentů, prostředím simulovaného multiagentního systému je zjednodušená verze prostředí soutěže MAPC. Zjednodušení se týká především nekonečnosti prostředí, které zde má hrany, a znalosti absolutních pozic agenty. Překážky jsou generovány vlastními algoritmy, ale jejich fungování zhruba odpovídá překážkám v prostředí MAPC.

Kapitola 3

Návrh algoritmu

Při navrhování algoritmu řízení multiagentního systému je potřeba definovat cíle, jak by se agenti měli v ideálním případě chovat a algoritmus těmto cílům co nejvíce uzpůsobit a přiblížit. Hlavním cílem je držet průměrnou dobu mezi objeveními jednotlivých pozic v prostředí na co nejnižší hodnotě. Aby toho bylo dosaženo, musí agenti spolupracovat a rozdělovat si prostor na přibližně stejně velké oblasti. Možným problémem při rozdělování prostoru na stejně velké oblasti může být nerovnost hustoty překážek v různých oblastech, což způsobí, že někteří agenti dokončí prozkoumávání své oblasti výrazně dříve než jiní. Tento problém ovšem při dynamickém prostředí není tak závažný, protože na rozdíl od statického prostředí musí agenti navštěvovat místa opakovaně, a proto agent, který skončí s prozkoumáváním své přidělené oblasti, nemusí svoje další kroky řídit podle fáze průzkumu jiných agentů. Dokonce v případě, že agent nenalezne žádnou jinou volnou oblast, může začít znovu prozkoumávat oblast, jejíž průzkum právě dokončil.

Při prozkoumávání statického prostředí multiagentním systémem si agenti rozdělí prostor na oblasti, tyto oblasti prozkoumají a případně pomůžou s průzkumem oblastí, které jsou kvůli překážkám na prozkoumání složitější. V dynamickém prostředí se, v rámci algoritmu popisovaného touto prací, agenti o oblasti nedělí. Každý si zabere jednu oblast pro sebe a jiného agenta do ní nepustí, dokud ji sám celou neprozkoumá. Po prozkoumání své oblasti si agent spočítá, která z aktuálně volných oblastí pro něj bude nejvýhodnější na cestování a průzkum. Pokud tato oblast není (a ani se časem nestane) výhodnější pro jiného agenta, tak se do ní vydá. O rozdělování oblastí mezi agenty je více psáno v sekci 3.2.

Určování výchozích pozic agentů probíhá před každou simulací zcela náhodně. Pokud by pak byl algoritmus modifikován a agenti by si mohli výchozí pozice sami vybírat podle vlastních výpočtů, vybrali by si takové pozice, do kterých v aktuálním algoritmu musí nejdříve docestovat, a proces průzkumu prostředí by byl zkrácen o cesty do startovních pozic průzkumu. Algoritmus je tedy s náhodným vybíráním výchozích pozic agentů připraven na jakékoliv rozestavení a jejich cílené určování jej může jedině zoptimalizovat.

Základní kostra algoritmu se skládá z nekonečné smyčky tří hlavních úkonů: naplánování následujícího kroku všemi agenty, vyřešení kolizí těchto naplánovaných kroků a vykonání naplánovaných kroků všemi agenty. Agenti mezi sebou sdílí model prostředí, kam zapisují objevené překážky. Díky tomu vědí agenti o všech překážkách, které se nacházejí (nacházely) v dohledu kteréhokoliv agenta.

3.1 Rozdělení prostoru

Jak vyplývá z úvodu této kapitoly, správné rozdělení prostoru je pro co nejlepší fungování algoritmu klíčové. Rozdělení prostoru závisí na jeho rozměrech, na počtu agentů, kteří budou prozkoumávat, a na jejich dohledu. Předpokládejme, že algoritmus dělí prostor na stejně velké oblasti[1]. Oblasti po takovém rozdělení tvoří (téměř) pravidelnou mřížku. Platí pravidlo, že těchto oblastí musí být stejně nebo více, než je agentů, aby si mohl každý agent přivlastnit svůj prostor.

Obecně agenti v této práci prozkoumávají vertikálními cestami a je proto nutné správně zvolit šířku oblastí. Oblasti by neměly být příliš úzké, aby je agenti nemuseli příliš často měnit, s čímž mohou být spjaty různá zdržení a zbytečné kroky. Zároveň ale nesmí být oblast ani moc široká, aby případné zdržení agenta překážkami při průzkumu nevedlo k velmi dlouhému nezmapování pozic na konci oblasti, které by se při menších rozměrech oblastí nacházely v jiné oblasti a mohly by tak být dávno prozkoumány jinými agenty. Z těchto důvodů mají oblasti v této práci šířku uzpůsobenou pro dvě vertikální průzkumné cesty agenta. Maximální šířka oblasti tedy je:

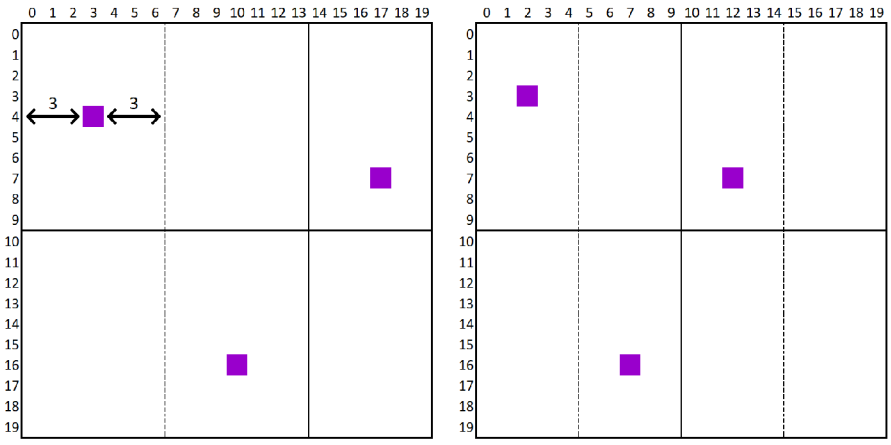
$$\textit{maximální šířka oblasti} = 2 * (2 * \textit{dohled agenta} + 1) \tag{3.1}$$

Na kolik oblastí se celý prostor rozdělí, je vypočítáno těmito rovnicemi:

$$\textit{počet oblastí horizontálně} \doteq \frac{\textit{šířka prostředí}}{2 * (2 * \textit{dohled agenta} + 1)} + 0.5 \tag{3.2}$$

$$\textit{počet oblastí vertikálně} \doteq \frac{\textit{počet agentů}}{\textit{řádek}} + 0.5 \tag{3.3}$$

Rovnice 3.2 pomocí šířky prostředí a dohledu agenta spočítá, kolik oblastí horizontálně musí minimálně být, aby se jimi zaplnila celá šířka prostředí. Rovnice 3.3 pomocí počtu agentů a počtu oblastí na řádku vypočítá počet oblastí vertikálně, aby jich celkem bylo minimálně stejně jako agentů. Výsledek dělení v obou rovnicích je nutné zaokrouhlit nahoru, což je zde znázorněno přičtením konstanty 0,5 a zaokrouhlením výsledku.



Obrázek 3.1: Ukázka rozdělení prostoru 20x20 v systému 3 agentů s dohledem 3. Fialová – agenti, pevná čára – hranice oblasti, přerušovaná čára – rozdělení oblasti na 2 průchody.

[1]Velikost oblastí se může lišit o jeden řádek či sloupec, aby byl vyplněn celý prostor i přes případnou nedělitelnost rozměrů celého prostředí počtem oblastí na řádku nebo ve sloupci

Příklad rozdělení prostoru na oblasti je na obrázku 3.1. V levé části obrázku je šířka levých oblastí dána jejich maximální šířkou a šířka pravých oblastí je dána zbytkem. Takto se v první řadě prostor rozdělí, aby bylo jasné, kolik oblastí horizontálně bude. Jejich šířky se pak srovnají a výsledkem je rozdělení v pravé části obrázku. Protože jsou agenti 3, je zde potřeba ještě oblasti vertikálně rozdělit, aby nebylo porušeno pravidlo o minimálním počtu oblastí vůči počtu agentů.

3.2 Rozdělení agentů do oblastí

Prostor je rozdělený na stejně velké oblasti a agenti si musí tyto oblasti rozdělit k průzkumu. Agent zná pozice jednotlivých oblastí, dobu, která uplynula od jejich posledního prozkoumání, a jejich obsazenost. Obsazenost může nabývat 3 stavů: volná oblast, agent přichází do oblasti a prozkoumávaná oblast. Před plánováním, kterou oblast půjde agent prozkoumat, si musí agent všechny oblasti ohodnotit. Výsledkem ohodnocení je skóre oblasti, které se skládá ze vzdálenosti agenta od oblasti a době od posledního prozkoumání oblasti. Čím nižší skóre oblast má, tím agenta zajímá více. Výpočet skóre se provádí rovnicí:

$$skóre = |x_{poz} - x_{roh}| + |y_{poz} - y_{roh}| + 2 * šířka * výška - 2 * ppo \tag{3.4}$$

Kde x_{poz} a y_{poz} jsou souřadnice aktuální pozice agenta, x_{roh} a y_{roh} jsou souřadnice nejbližšího rohu oblasti, do kterého je možné se dostat, *šířka* a *výška* jsou rozměry celého prostředí a *ppo* je zkratka „poslední prozkoumání oblasti“, jejíž číslo značí počet kroků simulace, které uběhly od posledního opuštění oblasti kterýmkoli agentem. Aby doba od posledního prozkoumání oblasti splňovala nepřímou úměrnost, že čím je vyšší, tím nižší je ohodnocení oblasti, a tedy pro agenty atraktivnější, je doba od posledního prozkoumání odečítána od konstanty, za kterou zde byl zvolen dvojnásobek obsahu celé plochy. Navíc je ještě pro rychlejší pokrývání celého prostoru dána době od posledního prozkoumání oblasti větší váha vynásobením této hodnoty konstantou 2. Výsledná hodnota je sečtena s Manhattanskou vzdáleností agenta od nejbližšího rohu oblasti.

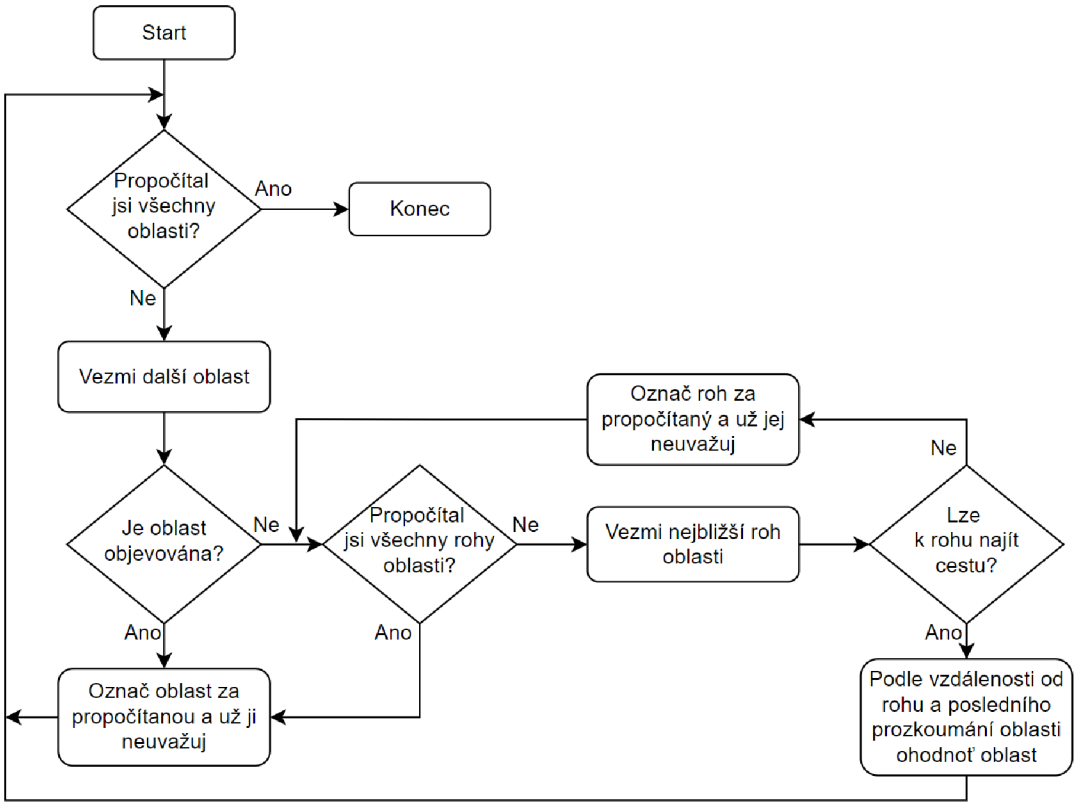
Celý proces ohodnocování oblastí agentem je znázorněn v diagramu na obrázku 3.2.

Jakmile má agent ohodnoceny všechny neobsazené oblasti, postupuje podle diagramu na obrázku 3.3 při výběru jedné konkrétní oblasti, do které se vydá.

Agenti se tedy navzájem vyzývají ke změně cílové oblasti, pokud je některý blíže než aktuálně přicházející. Agent, který je takto ke změně vyzván, provede znovu všechny kroky diagramů 3.2 a 3.3. Do možných oblastí ale nezahrnuje tu, do které právě cestoval, protože z jeho přesměrování vyplývá, že je některý agent blíž.

3.3 Průběh prozkoumávání oblasti

Prozkoumávání je hlavní aktivitou agenta. Každý agent v jeden okamžik prozkoumává maximálně jednu oblast a každou oblast prozkoumává maximálně jeden agent. Každá oblast má takovou šířku, aby ji agent prozkoumal za 2 vertikální průzkumné cesty. Šířka oblasti ale může být menší z důvodu většího počtu oblastí na řádku a vyrovnání jejich šířek. Prozkoumávání oblasti má 3 fáze: první průzkumná cesta, otočení a druhá průzkumná cesta. Po vykonání všech 3 fází agent označí oblast za prozkoumanou a hledá si další.



Obrázek 3.2: Diagram postupu agenta při ohodnocování oblastí.

Průzkumné cesty

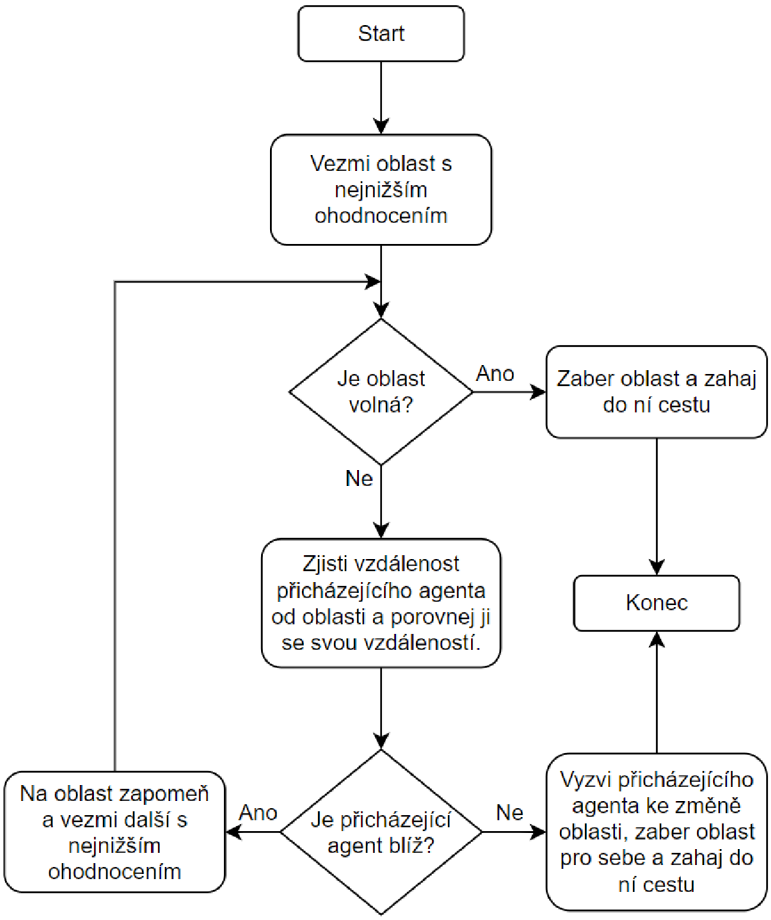
Průzkumné cesty v tomto algoritmu představují „oficiální“ prozkoumávání a vždy probíhají vertikálním pohybem agenta. Kromě toho ale agent zapisuje svoje okolí i při jakékoli jiné činnosti. Průzkumná cesta zpravidla vede přes vertikální středovou čáru jedné z polovin aktuálně prozkoumávané oblasti. Jediným důvodem pro odchýlení se od středu mohou být překážky. Diagram rozhodování agenta při průzkumu je na obrázku 3.4.

Horizontální vzdálenost počátečního bodu cesty od okraje oblasti se spočítá:

$$\textit{horizontální vzdálenost poč. bodu cesty od okraje oblasti} = \textit{šířka oblasti}/4 \tag{3.5}$$

Tato vzdálenost se pak buď přičte k levému okraji oblasti nebo odečte od pravého okraje oblasti podle toho, kterou vertikální polovinu oblasti jde agent objevovat. Výsledkem je pozice u horizontálního okraje prozkoumávané oblasti. Může se ale stát, že se na této pozici nachází překážka. V takovém případě se agent pokouší posouvat počáteční pozici vertikálně směrem od okraje, dokud nenarazí na volnou pozici – ta je skutečným počátkem cesty.

Jakmile se agent nachází na středu vertikální poloviny oblasti, pohybuje se vertikálně směrem od počátku cesty. Cestu mu může zkřížit buď překážka, nebo okraj prozkoumávané oblasti. Pokud agent narazí na překážku, určí nejbližší volný bod za překážkou, najde do něj cestu kolem překážky a cestu postupně vykoná. Může se také stát, že za překážkou není žádný volný bod, který by ještě ležel v aktuálně prozkoumávané oblasti. V takovém případě se zachová stejně, jako by došel přímo k okraji oblasti – ukončí průzkumnou cestu.



Obrázek 3.3: Diagram postupu agenta při výběru oblasti pro průzkum z ohodnocených oblastí.

V případě, že ukončená cesta byla první cestou, tak následuje přesun na začátek druhé cesty, neboli otočení. V případě ukončení druhé průzkumné cesty označí agent oblast za prozkoumanou a vydá se prozkoumat další.

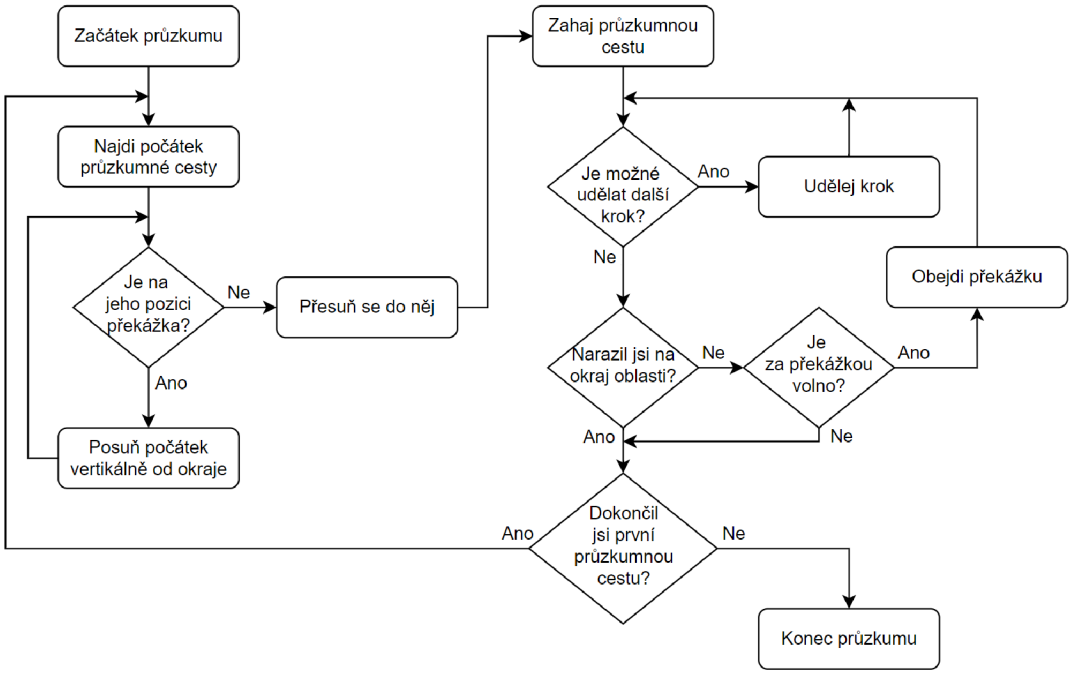
Otočení

Otočení probíhá po ukončení první průzkumné cesty. Agent buď došel na horizontální okraj prozkoumávané oblasti, nebo jej zastavila překážka, za kterou nenalezl volný bod. V obou případech si určí počátek druhé cesty stejně, jako si určoval počátek první cesty. Pokud se na počátku druhé průzkumné cesty nachází překážka, posouvá jej agent stejným způsobem, jako je popsáno v předchozím odstavci. Pakliže má agent nalezen počátek druhé průzkumné cesty, vyhledá si do něj trasu, vykoná ji a průzkumnou cestu započne.

3.4 Hledání cesty

Tato sekce je částečně inspirována zdroji [2] a [1].

Agenti často potřebují najít nejkratší cestu do nějakého bodu na mapě. Pro tento účel je v algoritmu zahrnuto plánování nejkratší cesty pomocí algoritmu A*. Algoritmus A*



Obrázek 3.4: Diagram kroků agenta při prozkoumávání oblasti.

ohodnocuje pozice na mapě a podle nejníže ohodnocených pozic nalezne optimální cestu. Ohodnocení se provádí jednoduchou rovnicí:

$$f(n) = g(n) + h(n) \tag{3.6}$$

Kde n je pozice na mapě, $f(n)$ je ohodnocení pozice n, $g(n)$ je délka trasy od počátku do bodu n a $h(n)$ je výsledek heuristické funkce bodu n. Jako heuristická funkce zde byla zvolena Manhattanská vzdálenost bodu od cíle.

Algoritmus A* začne prohledávat prostor od aktuální pozice agenta – startu cesty. Všechny pozice kolem startu ohodnotí podle výše zmíněné rovnice a uloží si vypočítané hodnocení do fronty OPEN, která bude po celou dobu hledání obsahovat body s jejich ohodnocením určené k expanzi. Poté je ve frontě OPEN nalezen bod s nejnižším ohodnocením (pokud je jich více, je vybrán ten, který je z nich ve frontě první). Tento bod je expandován a do fronty jsou přidány všechny jeho sousední body, které ještě nebyly expandovány a nenachází se ve frontě OPEN[2]. Proces se opakuje, dokud se expandovaný bod nerovná požadovanému cíli. Poté následuje zpětné hledání optimální cesty po expandovaných bodech pomocí jejich hodnocení, konkrétně pomocí hodnoty $g(n)$ neboli vzdálenosti bodu od startu. Začíná se od cíle a vždy je do výsledné cesty přidán ten sousední bod, jehož $g(n)$ je ze všech sousedních bodů nejmenší. Tímto způsobem je postupně nalezena optimální cesta až ke startu.

Výslednou trasu si agent zapamatuje a postupně vykoná.

[2]Díky výběru bodů z fronty podle jejich pořadí, v jakém byly vloženy, se nemůže stát, že by u již ohodnocených bodů existovalo lepší ohodnocení způsobené lepší cestou k nim.

3.5 Kolize agentů

Kolize agentů jsou detekovány po naplánování příštího kroku každým z nich. Při samotném plánování následujících kroků totiž agenti neuvažují pozice ostatních agentů. Mohou nastat 3 typy kolizí popsané v následujících odstavcích.

Průchod agentů skrz sebe

První typ kolize nastává v případě, že si 2 agenti mění pozice, takže prochází skrz sebe, ale v žádném momentě nesdílí stejnou pozici. Řešením tohoto typu kolize je kompletní vzájemná výměna všech informací, které agenti mají. Vypadá to tedy tak, že 2 agenti stojící vedle sebe si nezávisle na sobě naplánují krok na pozici toho druhého. Poté je tato kolize detekována, agenti si vymění cíle a přehodnotí svůj původní krok za starým cílem na nový krok za novým cílem. Tímto přístupem je navíc docíleno lehké optimalizace ušetřením jednoho kroku.

Naplánované kroky na stejnou pozici

Tento typ kolize by také mohl být pojmenován jako kolize při lichém počtu pozic mezi agenty. Výsledkem plánování agentů je pak krok na stejnou pozici. Možných řešení je více. Jeden agent by mohl pokračovat v cestě a druhý by se od své cesty odchýlil, takže by normálně pokračujícímu agentovi uvolnil průchod. Toto řešení by se ale muselo vypořádat s možnými nástrahami okolního prostředí kolem agentů a s případnou nemožností se odchýlit od trasy kvůli úzkému průchodu mezi překážkami. Proto bylo zvoleno jiné jednodušší řešení: Jeden agent se na krok zastaví a druhý agent udělá krok podle původního plánu. V příštím kroku pak nastane první typ kolize popsaný v předchozím odstavci. Nebo se také může stát, že cesty agentů svírají pravý úhel, a tedy už žádná kolize nenastane.

Naplánované kroky na stejnou pozici s jedním stojícím agentem

Poslední typ je podstatou stejný jako typ předchozí, ale řešení je jiné. Kolize nastává, jakmile spolu dva agenti sousedí. Jeden agent má z jakéhokoli důvodu naplánován krok na svoji aktuální pozici (má naplánováno stání) a druhý agent chce udělat krok na jeho pozici. Agent, který se chce dostat na pozici stojícího agenta, obdrží oznámení o překážce. Překážejícího agenta totiž opravdu vidí jako překážku v modelu prostředí a naplánuje si cestu, kterou stojícího agenta – překážku – obejde.

3.6 Optimalizace algoritmu

Optimalizace algoritmu se zaměřují na případy, které čas od času nastávají a mohou výrazně zpomalit prozkoumávání a zhoršit efektivitu algoritmu dočasně, nebo dokonce trvale. Mimo několika malých optimalizací algoritmus obsahuje dvě podstatné optimalizace popsané v následujících podsekcích.

Detekce uzavření agenta a uvolnění oblasti

V dynamickém prostředí se může stát, že dynamické překážky agenta uzavřou, a ten nebude schopen pokračovat a dokončit prozkoumávání přidělené oblasti. Bez jakéhokoli řešení této situace by agent stál a čekal až překážky, které ho uzavřely, zmizí a s jeho přidělenou oblastí

by ostatní agenti nekalkulovali při rozhodování, kterou oblast prozkoumat dál. Zbytek oblasti, kterou by agent ze své pozice neviděl, by po celou dobu agentova uvěznění obsahoval staré neaktuální informace.

Pro eliminaci tohoto problému má každý agent možnost restartu. Agent se sám restartuje, pokud nenajde žádnou možnou cestu do cíle, což znamená, že je uzavřen překážkami. Restart zahrnuje mimo jiné uvolnění právě prozkoumávané oblasti pro možnost vstupu a průzkumu ostatními agenty. Restart agentovi také zruší všechny cíle, takže v následujících krocích se bude rozhodovat „od začátku“, tj. znovu si spočítá, kterou oblast půjde prozkoumávat. Kvůli svému uzavření ale agent při počítání zjistí, že nenajde cestu do žádné z oblastí a znovu se restartuje. Tento proces se opakuje, dokud dynamika prostředí agentovi neumožní volný pohyb.

Zapomínání na dlouho neobjevené překážky

Tato optimalizace se může zdát zbytečná nebo dokonce nežádoucí. Ale dlouho neobjevená oblast s neaktuálními pozicemi překážek je agentům k ničemu, a je v takovém případě lepší o žádné překážce nevědět, než vědět o překážkách, které už na svých pozicích pravděpodobně dávno nejsou. Agenti mohou mít například objeveny překážky, které brání v cestě do některých oblastí. Bez zapomínání na dlouho neobjevené překážky by agenti při plánování cesty zjistili, že do těchto oblastí cesta neexistuje, a ani by ji nezahájili, takže by se nikdy nedozvěděli, že je cesta volná a oblast by zůstala neprozkoumaná do konce běhu algoritmu.

Aby bylo možné zapomínání provádět, potřebují agenti znát, kdy byla každá pozice naposledy zmapována. U každé pozice si tedy pro tento účel ukládají časové razítko, kdy naposledy byla tato pozice spatřena. Pokud je doba od posledního objevení pozice příliš vysoká, vynulují se agentům o této pozici veškeré informace. Doba, po které zapomenutí informací o pozici nastane, se vypočítá podle rovnice:

$$t = \frac{\textit{šířka} * \textit{výška}}{\textit{počet agentů}} \tag{3.7}$$

Kde t je čas (počet kroků běhu algoritmu), po kterém agenti zapomenou informace o pozici, a *šířka* a *výška* jsou rozměry celého prostředí. Výsledkem rovnice je v podstatě prostor, který připadá jednomu agentovi k prozkoumání, a před uplynutím této doby by tedy teoreticky měla být každá pozice objevena.

Kapitola 4

Implementace algoritmu

Program simulující prostředí a algoritmus řídící agenty je naprogramován v jazyce Java. Implementace zahrnuje uživatelské rozhraní, na které se vykreslují pozice překážek a agentů v reálném čase, kompletní ovládání prostředí od generování překážek po jejich dynamiku, agenta a jeho chování a kontroléry, které celý program spojují a režírují.

4.1 Celková struktura programu

Použitou architekturou je Model-view-controller.

Model

Model má obecně za úkol provádět veškerou logiku programu. To znamená kompletní rozhodování agenta, generování a dynamiku překážek a pomocné třídy, například třída reprezentující souřadnice.

Model obsahuje třídy `Agent`, `Graph`, `Node`, `Environment`, `EntitiesInMap`, `NumbersInMap` a `Coordinates`. Třída `Agent` reprezentuje agenta a obsahuje celé jeho chování. Třídy `Graph` a `Node` reprezentují oblasti, na které je prostředí pro agenty rozděleno. `Node` symbolizuje jednu oblast, `Graph` má uloženy všechny oblasti a poskytuje agentům informaci o tom, ve které oblasti se nacházejí[1]. Třída `Environment` reprezentuje celé prostředí a provádí veškeré operace s překážkami. Pomocné třídy `EntitiesInMap` a `NumbersInMap` jsou dvourozměrná pole vylepšená o několik operací. `EntitiesInMap` obsahuje *boolean* hodnoty a používá se například pro reprezentaci překážek objevených agenty tak, že na indexech v poli, které jsou identické se souřadnicemi v prostředí, je v případě překážky uložena hodnota *true*. `NumbersInMap` obsahuje stejné metody jako `EntitiesInMap`, ale hodnotami jsou zde čísla. Používá se například pro ukládání doby uplynulé od posledního objevení pozic v prostředí. K ukládání souřadnic slouží třída `Coordinates`, která umožňuje porovnání 2 objektů tohoto typu.

View

Část architektury nazývaná *View* zahrnuje uživatelské rozhraní a vykreslování aktuálního dění na mapě.

[1] Inspirací pro názvy těchto tříd byl Reebův graf (ukázka na obrázku 2.4), kde `Graph` je celý graf jako celek a `Node` je jeden uzel grafu.

Spadají sem třídy `Window` a `MyCanvas`. Třída `Window` obsahuje kompletní definici uživatelského rozhraní. Třída `MyCanvas` reprezentuje plátno, na které se vykresluje běh simulace s pozicemi agentů a překážek v reálném čase.

Controller

Kontroléry se starají o řízení běhu programu. Předávají informace z modelu do view a naopak. Celá implementace obsahuje 2 kontroléry: `Controller` a `AgentsController`.

Třída `Controller` reprezentuje hlavní kontrolér, který je ve třídě `Main` vytvořen a spuštěn. Obsahuje reakce na stisknutí tlačítek v uživatelském rozhraní a nekonečnou hlavní smyčku programu. V každé iteraci této smyčky kontrolér nejdříve zkontroluje stavy tlačítek a případně na ně adekvátně zareaguje. Pokud simulace běží, kontrolér nechá prostředí provést krok (možná změna překážek), pak nechá provést krok `AgentsController` (pohyb agentů), získá z pohybu agentů informace a předá je do uživatelského rozhraní na vykreslení. Nakonec si od uživatelského rozhraní zjistí aktuálně zadanou rychlost simulace z posuvníku a podle rovnice 4.1 vypočítá časovou délku kroku, na kterou se běh programu před dalším krokem zastaví.

$$\textit{časová délka kroku } [ms] = \frac{1000}{\textit{počet kroků za sekundu}} \tag{4.1}$$

Hodnota *počet kroků za sekundu* je hodnota získaná z posuvníku v uživatelském rozhraní a může nabývat hodnot 0,5 - 180.

Třída `AgentsController` má za úkol obsluhovat agenty. Při započnutí simulace algoritmu vytvoří všechny agenty. Po zbytek běhu simulace čeká na výzvu od třídy `Controller` k vykonání kroku. Krok kontroléru agentů se skládá z vyzvání agentů k naplánování následujícího kroku, detekce kolizí naplánovaných kroků a vyzvání agentů k vykonání naplánovaných kroků. Třída `AgentsController` provádí ukládání překážek z dohledů agentů do objektu třídy `EntitiesInMap`.

Kolize agentů jsou detekovány dvěma kompletními průchody všech dvojic agentů, kde v prvním průchodu se detekují a řeší kolize naplánování kroku na stejnou pozici a v druhém průchodu se detekují a řeší kolize výměny pozic agentů. Při nalezení kolize jsou agenti podle situace vyzváni ke změně plánu a celý proces detekce kolizí se zopakuje. Detekování kolizí má zabudovanou pojistku proti zacyklení v podobě čítače detekovaných kolizí. Jakmile čítač přesáhne určitou hodnotu, jsou všichni agenti ve všech stávajících kolizích restartováni. Jakmile program projde přes detektor kolizí bez detekování jediné kolize, je možné vyzvat všechny agenty k vykonání naplánovaného kroku.

4.2 Uživatelské rozhraní a vykreslování

Uživatelské rozhraní není tématem této práce, a proto na něj nebyl kladen přílišný důraz. Bylo vytvořeno pomocí Java Swing s využitím Abstract Window Toolkit (AWT). Uživatelské rozhraní se skládá z menu v pravé části okna a vykreslovací oblasti, která zabírá zbytek prostoru okna. Tato oblast obsahuje 2 karty – „Aktuální překážky“ a „Objevené překážky“. Na kartě „Aktuální překážky“ lze v reálném čase pozorovat aktuální mapu se všemi překážkami. Na kartě „Objevené překážky“ lze v reálném čase pozorovat mapu tak, jak ji vidí agenti, tedy se zmapovanými pozicemi překážek tam, kde byly naposledy agenty spatřeny. Na této kartě se také promítá zapomínání překážek popsané v sekci 3.6. Ukázka vzhledu aplikace je k vidění na obrázku 4.1.

Definice okna

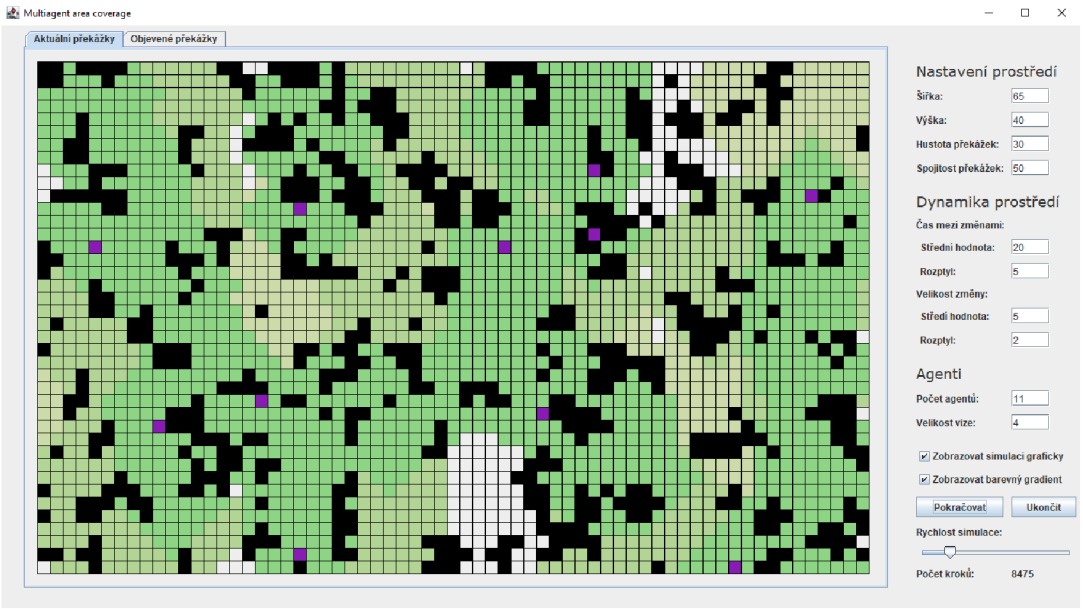
Okno aplikace je reprezentováno třídou `Window`, která dědí z třídy `JFrame`. Prostor v okně je rozdělen na několik panelů typu `JPanel`. Karty jsou vytvořeny pomocí `JTabbedPane`, který obsahuje 2 panely reprezentující právě zmíněné 2 karty.

Menu je složeno z klasických komponent `JLabel`, `JTextField` a jiné. Rozložení komponent je řešeno pomocí `GridBagConstraints`, který nabízí potřebné možnosti rozmísťování komponent. O komunikaci mezi aplikací a tlačítky se částečně starají události vyvolané stiskem tlačítek a částečně se o ni stará hlavní kontrolér `Controller`. Po stisku tlačítka je pomocí události uloženo, které tlačítko bylo stisknuto, a případně je změněn nápis na tlačítku. Kontrolér se pak na začátku každé iterace hlavní smyčky programu okna ptá na naposledy stisknuté tlačítko a podle odpovědi zareaguje. Zároveň si kontrolér v každé iteraci zjišťuje stav zaškrtávacích políček, která zapínají a vypínají celkové vykreslování dění simulace a vykreslování barevného gradientu na nedávno prozkoumaných buňkách. Zmíněným barevným gradientem je přechod ze syté barvy do bílé, kde sytější barva značí prozkoumání buňky před kratší dobou, než je tomu u méně sytých barev. Díky tomuto gradientu je tak při prvním pohledu na prostředí zřetelné, jak dobře agenti pokrývají prostor.

Poslední částí uživatelského rozhraní je okno s výsledky simulace, které se objeví po jejím ukončení. Toto okno se vyvolává funkcí `JOptionPane.showMessageDialog()`.

Vykreslování

Vykreslování se provádí na plátna, která jsou reprezentována třídou `MyCanvas` dědící z `JPanel`. Pro každou kartu vykreslených informací je vytvořeno jedno plátno. Vykreslování je definováno metodou `paintComponent()`. Pro správné vykreslení je nejdříve potřeba znát velikost jednoho čtverečku prostředí. Ta je určena výpočtem šířky i výšky čtverečku s ohledem na rozměry prostředí a rozměry plátna. Výsledným rozměrem čtverečku je menší z šířky a výšky. Při vykreslování se pak pomocí velikosti jednoho čtverečku určují absolutní pozice jednotlivých čar mřížky , překážek, agentů a gradientu jimi prozkoumané plochy na plátně.



Obrázek 4.1: Ukázka okna aplikace při běžící simulaci.

4.3 Implementace prostředí

Prostředí, ve kterém se agenti pohybují, je nedílnou součástí této práce. Bez překážek tvořených prostředím by nebylo možné algoritmus řádně otestovat a případně i nasadit do situací v reálném životě, kde se překážky na požadované ploše zpravidla vyskytují.

Po kliknutí na tlačítko „Start“ se vytvoří instance třídy `Environment` a předají se jí všechny parametry zadané uživatelem, které se týkají prostředí, tj. celková výška a šířka, hustota a spojitost překážek, střední hodnota a rozptyl času mezi změnami prostředí a střední hodnota a rozptyl velikosti změny prostředí. Třída `Environment` se pak po celý běh simulace algoritmu stará o veškeré operace s prostředím a překážkami.

Generování překážek

Generování překážek probíhá před započnutím simulace. Každý shluk překážek (skupina překážek, které se dotýkají minimálně jednou stranou) vždy obsahuje jednu kořenovou překážku a z ní vygenerované listové překážky. Klíčovým krokem před generováním je vypočítání počtu kořenových překážek. Počet kořenových překážek k v celém prostředí se počítá rovnicí:

$$k = \textit{šířka} * \textit{výška} * \frac{\textit{hustota}}{100} * (1 - \frac{\textit{spojitost}}{100}) * 0.2 \tag{4.2}$$

Kde *šířka* a *výška* jsou rozměry prostředí, *hustota* je hustota překážek v intervalu $\langle 0, 100)$ zadaná uživatelem a *spojitost* je spojitost překážek v intervalu $\langle 0, 100)$ zadaná uživatelem. Celkový počet kořenových překážek tedy přímo úměrně závisí na velikosti prostředí a zadané hustotě a nepřímo úměrně na zadané spojitosti, protože při nižší spojitosti je potřeba vygenerovat více kořenových překážek a naopak. Konstanta 0,2 slouží pouze ke zmenšení výsledného počtu na pětinu původní velikosti a je experimentálně zjištěná.

Každé kořenové překážce se náhodně přidělí pozice na mapě a počet listových překážek. Aby nebyl počet listových překážek každé kořenové překážky stejný, je počítán pomocí normálního rozdělení pravděpodobnosti, které pracuje se střední hodnotou μ a rozptylem ρ:

$$\mu = \frac{\textit{šířka} * \textit{výška} * 0.3}{k} \tag{4.3}$$

$$\rho = \mu * 0.2 \tag{4.4}$$

Kde *šířka* a *výška* jsou rozměry prostředí a k je výsledek z rovnice 4.2. Konstanty 0,3 a 0,2 jsou výsledkem zkoušení a pozorování výsledků jako v předchozím případě.

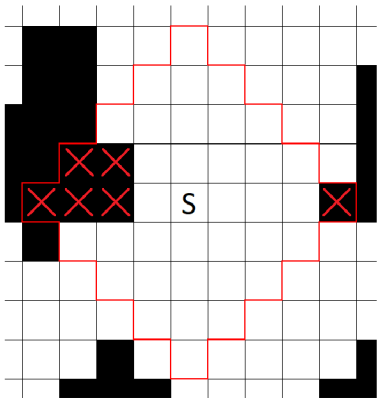
Při generování listových překážek z kořenové překážky se náhodně vybere jedna z již vygenerovaných překážek ve shluku (při první iteraci je ve shluku pouze kořenová překážka), náhodně se vybere směr a vedle vybrané překážky se ve vybraném směru vytvoří nová listová překážka.

Dynamika prostředí

Dynamika prostředí se řídí uživatelem zadaným parametrem pro časovou délku mezi jednotlivými změnami a parametrem pro velikost změny. Při vytváření instance třídy `Environment` se pomocí normálního rozdělení pravděpodobnosti a parametrů, které určují časovou délku

mezi jednotlivými změnami, rovnou spočítá, za kolik kroků simulace nastane první změna. Při každém kroku simulace se pak tato hodnota zmenšuje.

Jestliže nastal krok, ve kterém má být provedena změna prostředí, určí se nejdříve střed (ohnisko) této změny náhodným výběrem kterékoliv pozice v prostředí. Následně se pomocí normálního rozložení pravděpodobnosti a parametrů, které zadal uživatel, spočítá velikost změny. Kolem středu se vytvoří kosočtverec, který změna zasáhne.



Obrázek 4.2: Prostor kolem středu S zasažený změnou o velikosti 4. Bílá – volné pozice, černá – překážky, červené křížky – rušené překážky.

Všechny překážky v tomto kosočtverci jsou odstraněny. Nové překážky v oblasti změny jsou generovány stejně, jako jsou na začátku programu generovány všechny překážky v prostředí. V rovnicích 4.2 a 4.3 se místo šířky a výšky prostředí používá obsah kosočtverce, který se spočítá rovnicí:

$$obsah\ kosočtverce = \frac{(2 * velikost\ změny + 1)^2}{2} + 0.5 \tag{4.5}$$

4.4 Implementace rozdělování agentů do oblastí

K rozdělování agentů do oblastí slouží dvě pomocné třídy: `Graph` a `Node`.

Třída `Graph` agentům pomáhá s detekcí, ve které oblasti se právě nacházejí. Pro každý běh simulace existuje vždy jiná ale pouze jedna její instance a mají ji uloženou všichni agenti. Třída se stará o rozdělení prostoru na oblasti, které budou agenti prozkoumávat. Tvoření těchto oblastí, které jsou reprezentovány právě třídou `Node`, probíhá hned v konstruktoru třídy `Graph`. Její hlavní funkcí je pak za běhu programu inkrementovat dobu od posledního prozkoumání ve všech instancích `Node`. Agenti si od instance této třídy mohou vyžádat kompletní seznam všech oblastí v prostředí nebo pouze oblast, ve které se právě nacházejí.

Třída `Node` reprezentuje jednu z oblastí, na které je prostor rozdělen, a které si agenti přivlastňují na průzkum. Obsahuje svůj unikátní identifikátor, souřadnice levého horního a pravého spodního rohu, dobu od posledního kompletního průzkumu některým z agentů, odkaz na agenta, který tuto oblast prozkoumává, nebo do ní cestuje, a stav oblasti, který může nabývat tří hodnot: *volná oblast*, *agent přichází do oblasti* a *oblast je prozkoumávána agentem*.

4.5 Implementace agentů

Implementace agentů je hlavní částí celého programu a celá se nachází v třídě `Agent`. Agent si v atributech ukládá svoje unikátní identifikační číslo, svoji pozici, cílovou pozici, do které má právě namířeno, naplánovanou cestu do cílové pozice typu pole souřadnic a aktuálně naplánovaný krok. Dále má práci s oblastmi uloženou instanci třídy `Graph` a pokud nějakou oblast prozkoumává, tak ji má uloženou jako instanci třídy `Node`. Pomocí instance pomocné třídy `EntitiesInMap` má přístup ke sdílenému modelu prostředí se zmapovanými pozicemi překážek všemi agenty. V poslední řadě má uloženy informace o aktuálních činnostech: právě vykonávanou aktivitu (cestování, průzkum), fázi průzkumu (žádná, první cesta, otočení, druhá cesta), směr průzkumu (žádný, nahoru, dolů), roh oblasti, do kterého právě cestuje v rámci cestování nebo i v rámci průzkumu (žádný, levý horní, pravý horní, levý spodní, pravý spodní), a *boolean* hodnotu informující, zda právě obchází překážku. Pomocí všech výše zmíněných atributů agent provádí svoje veškeré plánování a rozhodování.

Krok agenta

Každý krok agenta začíná jeho plánováním. Základ tvoří zjištění, zda agent cestuje, nebo prozkoumává. Podle toho, jestli právě cestuje, nebo prozkoumává, se zavolají příslušné funkce, které vrátí souřadnice příštího kroku, nebo nevrátí nic, což může znamenat přepnutí agentovy činnosti nebo nějakou chybu a tudíž setrvání na stejné pozici. Agent si souřadnice příštího kroku uloží a skončí plánování. Při případné kolizi detekované kontrolérem agentů agent krok přeplánuje. Po vyřešení všech kolizí je agent kontrolérem vyzván, aby krok vykonal, což v praxi znamená přepsání aktuální pozice naplánovanou pozicí, která je následně vynulována a připravena na další plánování.

Průzkumná fáze

Plánování následujícího kroku při aktivní průzkumné fázi začíná ověřením, jestli průzkum už probíhá, nebo má právě začít, což agent zjistí podle toho, že v atributu aktuální průzkumné fáze má uloženou hodnotu „žádná“. Pokud má průzkum právě začít, podle atributu „cílový roh oblasti“ agent určí, ve kterém rohu se právě nachází, a vydedukuje směr první průzkumné cesty (nahoru nebo dolů) a cílový roh této cesty. Tím byla zahájena průzkumná fáze agentovy aktivity.

Další plánování závisí na aktuální průzkumné fázi a případném obcházení překážky.

- **Průzkumné cesty** – Agent se podívá na pozici před sebou podle směru, kterým prozkoumává, a pokud se na ní nenachází překážka, naplánuje na ni krok. V případě nárazu na překážku přejde agent k obsluze obcházení překážky.
- **Otočení** – Agent si podle toho, jestli má uloženou nějakou cílovou pozici ve svých atributech, odvodí, zda otočka právě začala, nebo probíhá. Pokud právě začala, tak si agent spočítá počáteční bod druhé průzkumné cesty. Tento výpočet je více popsán v sekci 3.3. Poté si do tohoto bodu najde cestu, případně jej posune. Při nárazu na překážku během vykonávání otočky cestu přeplánuje. Po doražení do cíle (počátku druhé průzkumné cesty) se ještě agent pokusí posunout vertikálně k okraji, kdyby se tam už náhodou překážky, které ho při plánování donutily posunout cíl od okraje, nenacházely, aby měl průzkumnou cestu co nejkompletnější.

- **Obcházení překážky** – Při obcházení překážky si agent najde nejbližší volný bod za překážkou a naplánuje do něj cestu. I zde platí, že při nárazu na překážku, která brání naplánované cestě, dochází k přeplánování cesty a případnému posouvání cíle obcházení.

Průzkumná fáze je ukončena, jakmile agent v průběhu druhé průzkumné cesty narazí na horizontální hranu prozkoumávané oblasti. Poté si přepne aktivitu na cestování a rovnou přejde na plánování v rámci cestování.

Cestovací fáze

Agent využívá cestovací fázi pro přesun mezi právě prozkoumanou oblastí a oblastí, kterou jde prozkoumat jako další. Nejprve si musí určit, kterou oblast půjde objevovat. To si spočítá pomocí algoritmu popsaném v sekci 3.2, kde pro ukládání jednotlivých oblastí a jejich skóre slouží třída `HashMap`, která ukládá dvojici klíč a hodnota. Je použito celkem 6 těchto map, každá v klíči obsahuje identifikátory oblastí a v hodnotě obsahují postupně: instance daných oblastí, skóre, souřadnice nejbližšího rohu, název nejbližšího rohu (který ze čtyř rohů je dán souřadnicemi), vzdálenost od nejbližšího rohu a cestu do nejbližšího rohu. 4 mapy s posledními 4 zmíněnými hodnotami jsou zde pouze pro optimalizační účely, protože všechny jejich hodnoty se používají již pro monitorování, do kterých oblastí se agent může vůbec dostat, a jsou při získání uloženy pro pozdější použití.

Do určování oblasti, kterou agent půjde prozkoumávat, také patří případné domluvy mezi agenty, pokud některému vychází do dané oblasti kratší cesta než jinému. Agent, který do oblasti právě putuje, si pak musí přehodnotit cíl. Nejdříve se takový agent restartuje, aby se mu vrátily všechny atributy do výchozích hodnot, a dále postupuje od začátku celého plánování kroku. Dokud si agent nevybere v rámci aktuálního kroku novou oblast na prozkoumání, odkaz na něj se bude stále nacházet v oblasti, kam původně cestoval. Při výběru nové oblasti tedy agent na tento fakt musí dbát a svoji původní oblast do možných oblastí ani nezahrnout, protože agent, který ho donutil k přesměrování, ji má určitě blíž.

Jakmile má agent vybranou oblast k prozkoumání, pomocí jejího identifikátoru si ze všech map získá potřebná data k započnutí cesty.

Pokud agent zná cestu, v každém kroku z ní vezme první souřadnici v pořadí, což je souřadnice jeho příštího kroku. Pokud se na její pozici nachází překážka, je cesta přeplánována. Jinak postupně cestu vykoná a po dosažení cílové pozice v nové oblasti může začít prozkoumávat.

Kolize agentů

Na kolizi agenta upozorní kontrolér agentů, který má za úkol kolize detekovat. O typech kolizí a jejich řešení se píše v sekci 3.5.

Řešením kolize, kdy si dva agenti mění pozice, je výměna téměř všech hodnot, které mají agenti uložené v atributech. Tuto operaci provádí jeden ze střetnutých agentů za oba. Agenti si vymění tyto hodnoty:

- Souřadnice cílové pozice, na kterou putují.
- Odkaz na oblast, do které cestují nebo kterou prozkoumávají (pokud taková je). Zároveň je potřeba těmto oblastem změnit odkaz na agenta, který je na ně navázán.
- Naplánovanou cestu do cílové pozice (pokud nějaká je).

- Všechny informace o aktuálních aktivitách a fázích – hlavní aktuální činnost, fáze průzkumu, směr průzkumu, cílový roh oblasti a *boolean* hodnota, zda agent právě obchází překážku.

Po vyměnění všech těchto hodnot je nutné, aby si oba agenti znovu naplánovali následující krok.

Při kolizi, kdy si dva agenti naplánují krok na stejnou pozici a zároveň jsou oba v pohybu, je potřeba jednoho agenta zastavit. Agent, který bude zastaven, není vybrán podle žádného pravidla. Pro zastavení agenta většinou stačí, aby si agent přepsal naplánovaný krok jeho aktuální pozicí. V situacích, kdy agent vykonává nějakou cestu (do nové oblasti nebo i obcházení překážky v rámci průzkumu), si agent musí přidat svůj původně naplánovaný krok zpět do naplánované cesty.

V posledním typu kolize, kdy se agenti nachází vedle sebe, jeden plánuje v příštím kroku stát a druhý agent plánuje vkročit na tutéž pozici, obdrží zablokovaný agent zprávu o agentovi v cestě. Blokující agent je pro tuto chvíli vložen jako překážka do modelu zmapovaných pozic překážek typu `EntitiesInMap`. Agent dále postupuje již podle klasických pravidel: pokud již vykonává naplánovanou trasu, přeplánuje ji, nebo pokud právě prozkoumává, naplánuje si trasu kolem překážky.

Kapitola 5

Vyhodnocení

Vyhodnocení chování agentů a jejich schopnost mapovat prostředí prozradí, jak dobře algoritmus funguje pro různé parametry dynamiky prostředí a agentů. Pro vyhodnocení algoritmu popsaného touto bakalářskou prací je využito porovnání zjištěných hodnot pro různé parametry simulace. Vedle toho jsou ještě počítány optimální hodnoty, které obsahují nejlepší teoretické výsledky. Hodnotami, pomocí kterých se algoritmus vyhodnocuje, jsou:

- **Průměrný počet kroků simulace mezi dvěma prozkoumáními jedné pozice** – Každá pozice má uloženo, kolik kroků simulace uplynulo od doby, kdy byla naposledy spatřena některým agentem. Jakmile se tato pozice dostane znovu do dohledu jakéhokoli agenta, do statistik se zaznamená hodnota, kterou před vynulováním obsahovala. Z těchto hodnot se na konci simulace v rámci každé pozice udělá aritmetický průměr a z průměrů všech pozic se udělá celkový aritmetický průměr, který napoví, jak často byly pozice mapovány.
- **Průměrný počet kroků simulace mezi dvěma prozkoumáními jedné oblasti** – Oblasti, na které mají agenti celou plochu rozdělenou, si také ukládají počet kroků simulace od posledního prozkoumání a opuštění agentem. Tato hodnota především udává, jak dobře si agenti vybírají oblasti k prozkoumání, a jestli nějakou dlouho neopomíjejí. Jakmile je oblast prozkoumána a agent ji opustí, je hodnota v čítači zaznamenána před tím, než se čítač vynuluje. Podobně, jako u jednotlivých pozic, se pak na konci simulace udělá v rámci každé oblasti aritmetický průměr těchto hodnot. Průměrné hodnoty ze všech oblastí jsou nakonec zprůměrovány do jednoho výsledku.
- **Průměrný počet kroků simulace mezi dvěma prozkoumáními celé plochy** – Každá pozice v prostředí má uložen nejen počet kroků simulace od posledního prozkoumání agentem, ale také *boolean* hodnotu. Výchozí stav této hodnoty je *false*. Vždy, když se pozice dostane do dohledu kteréhokoliv agenta, přepíše se na *true*. Po každém kroku všech agentů se aplikace pokusí detekovat, zda již mají všechny pozice hodnotu nastavenou na *true* a případně pak do statistik uloží počet uplynulých kroků od posledního tohoto stavu – prozkoumání celé plochy – a vrátí všechny hodnoty zpět do výchozího stavu na *false*. Kvůli nedokonalosti algoritmu v občasném minutí některých pozic z důvodu obcházení překážek není trváno na 100% prozkoumání celé plochy, ale pouze na 95%.

Výše popsané hodnoty získané jednou z každé simulace mohou být následně porovnávány s optimálními hodnotami – nejlepší (nejmenší) výše zmíněné hodnoty, kterých lze

teoreticky dosáhnout. Z porovnání vzejdou procenta (vydělením optimálních hodnot reálně získanými), která jsou výsledkem vyhodnocení. Optimální průměrný počet kroků simulace mezi dvěma prozkoumáními jedné pozice, jedné oblasti a celé plochy se počítá rovnicemi:

$$pozice = celá\ plocha \tag{5.1}$$

$$oblast = \frac{počet\ oblastí}{počet\ agentů} * \frac{\sum_{n=1}^{počet\ oblastí} \frac{šířka(n) * výška(n) - (\frac{(2 * dohled\ agentů - 1)^2}{2} + 0.5)}{2 * dohled\ agentů - 1}}{počet\ oblastí} \tag{5.2}$$

$$celá\ plocha = \frac{šířka * výška - (\frac{(2 * dohled\ agentů + 1)^2}{2} + 0.5) * počet\ agentů}{(2 * dohled\ agentů + 1) * počet\ agentů} \tag{5.3}$$

Výsledky rovnic 5.1 a 5.3 se rovnají, protože při optimálním prozkoumávání celé plochy je každá pozice v rámci jednoho průzkumu plochy prozkoumána právě jednou a optimální počet kroků mezi dvěma prozkoumáními je tedy roven optimálnímu počtu kroků pro prozkoumání celé oblasti. Ve výpočtu 5.2 se n rovná identifikátorům jednotlivých oblastí. Výpočet v čitateli se tedy provádí pro každou oblast v prostředí, protože se oblasti mohou lehce lišit svými rozměry. V rovnicích 5.2 a 5.3 se používají stejné části výpočtů: $2 * dohled\ agentů + 1$ vyjadřuje úhlopříčku kosočtverce agentova dohledu a $\frac{(2 * dohled\ agentů - 1)^2}{2} + 0.5$ vyjadřuje počet pozic, které kolem sebe agent v jeden moment vidí (obsah kosočtverce).

5.1 Měření

Pro účely měření bylo do aplikace implementováno automatické vyhodnocování algoritmu. Tomuto vyhodnocování se v kódu nastaví minimální a maximální hodnota jednotlivých parametrů simulace a jejich velikost změny (hodnota změny se může k parametrům přičítat nebo je násobit). Simulace byla spuštěna celkem pro 560 různých kombinací parametrů dynamiky prostředí a agentů. Pro každou kombinaci simulace proběhla pětkrát a výsledky se zprůměrovaly. Počet kroků simulace byl pro každou kombinaci odvozen od počtu agentů, protože menší počty agentů potřebují více času na vícenásobné prozkoumání prostředí a naopak větší počty agentů potřebují méně času na prozkoumání. Pro větší počty agentů je také lepší nižší počet kroků kvůli hardwarové náročnosti. Počet kroků v jedné simulaci se počítá rovnicí:

$$počet\ kroků = 20000 - \frac{počet\ agentů}{maximální\ počet\ agentů} * 15000 \tag{5.4}$$

Z rovnice vyplývá, že pro jednoho agenta (tj. minimální počet agentů) bude kroků 20000 a pro maximální počet agentů bude kroků simulace pouze 5000.

V následujících tabulkách jsou použity tyto symboly a názvy:

μ_{cas}	– Střední hodnota normálního rozdělení intervalu mezi změnami prostředí.
μ_{vel}	– Střední hodnota normálního rozdělení velikosti změny prostředí.
Dohled	– Velikost dohledu agenta (poloměr dohledu)
Pozice	– Průměrná časová délka mezi dvěma prozkoumáními jedné pozice v prostředí.
Oblast	– Průměrná časová délka mezi dvěma prozkoumáními jedné oblasti, na které je prostředí rozděleno.

Plocha – Průměrná časová délka mezi dvěma prozkoumáními 95% celé plochy prostředí.
Průzkum – Průměrný počet kroků, který agenti strávili prozkoumáváním.
Cestování – Průměrný počet kroků, který agenti strávili cestováním.
Stání – Průměrný počet kroků simulace, který agenti strávili stáním na stejné pozici.
FPS – Snímky za sekundu, konkrétně počet kroků simulace za sekundu.

V předchozím seznamu jsou uvedeny střední hodnoty pro počítání pomocí normálního rozdělení pravděpodobnosti. Toto rozdělení ale k fungování ještě potřebuje rozptyl. Rozptyl se v obou případech počítá vynásobením střední hodnoty konstantou 0,15.

Všechna měření, jejichž výsledky následují, byla provedena na prostředí o pevně nastavených rozměrech 80x80, hustotě překážek na 30 a spojitosti překážek na 50.

Vliv počtu agentů a intervalu měnění překážek na rychlost průzkumu

$\mu_{vel} = 8$, dohled = 4					
Počet agentů	μ_{cas}	10	20	40	80
1	Pozice	772,4	765,79	786,09	724,76
	Oblast	1689,34	1665,07	1649,36	1457,22
	Plocha	6762,9	8131,2	11435	11401,4
2	Pozice	384,39	373,51	374,44	373,03
	Oblast	869,87	792,91	789,14	766,24
	Plocha	3543,43	4919,68	8003	9570,8
5	Pozice	149,04	147,31	148,35	148,69
	Oblast	285,85	273,17	269,5	271,21
	Plocha	2206,22	3852,78	6148,7	7144,2
15	Pozice	48,49	47,85	48,03	48,06
	Oblast	97,94	92,25	91,556	92,01
	Plocha	1877,67	3307,63	3416,2	0

Tabulka 5.1: Efektivita algoritmu při pevném dohledu a velikosti změny prostředí

Tabulka 5.1 zobrazuje výsledky algoritmu pro měnící se počet agentů a časový interval mezi změnami. Naopak pevnými parametry jsou zde dohled agentů a střední hodnota velikosti změny prostředí.

Pokud je algoritmus správně implementovaný, měl by narůstající počet agentů rovnoměrně snižovat dobu potřebnou k objevení jednotlivých pozic, oblastí, i celé plochy. Jestliže zde například jednomu agentovi trvalo prozkoumat jednu pozici průměrně 762 kroků a dvěma agentům to samé trvalo 376 kroků, je rychlost prozkoumávání zvýšena 2,027krát. Rychlost je tedy dokonce více než dvakrát vyšší. Pokud se porovná rychlost při 1 agentovi a rychlost při 15 agentech, z dat lze spočítat dokonce 15,875násobné zrychlení. Z toho vyplývá, že v jednoagentní simulaci musí agent udělat více vedlejších kroků, které samy o sobě nevedou k žádnému novému prozkoumávání, ale například k přesunu do jiné oblasti. Oproti tomu v systému patnácti agentů se často stává, že agent, který doprozkoumá svoji oblast, nemá kam jinam jít než znovu do své původní oblasti, kterou tak začne prozkoumávat bez „zbytečného“ cestování. Data z prozkoumávání celých oblastí vykazují podobně dobré statistiky, jako data z jednotlivých pozic. Ovšem data z prozkoumávání celé plochy už tak dokonalá nejsou. V posledním sloupci na posledním řádku dokonce nejsou vůbec žádná,

což znamená, že agenti za celou simulaci nenavštívili více než 94% všech pozic. O tomto problému více pojednává následující odstavec.

Při pohledu na vliv frekvence změn prostředí na délku prozkoumávání není v datech pro jednotlivé pozice a oblasti poznat žádná změna. Ovšem v rámci prozkoumávání celé plochy zde nastala neočekávaná reakce. Čím větší interval mezi změnami je, tím menší by měla být doba, kterou zabere průzkum celé plochy, protože agenti, kteří své cesty plánují, nebudou tak často překvapováni novými překážkami a jejich cesty budou přeplánovávány méně. Zde ale nastal opačný případ. Jeho vysvětlením bude pravděpodobně fakt, že tím, že se překážky nemění příliš často a agenti je obcházejí pořád stejnými cestami, zůstávají na ploše stále stejné neprozkoumané pozice mnohem déle, než když se překážky mění často a agenti chodí po různých cestách. Algoritmus totiž neobsahuje žádné zpětné procházení minutých pozic a tak se stává, že se kvůli obcházení překážky agent nepodívá na prostor na druhé straně překážky. Tento problém s nevyplňováním celého prostoru patří do kategorie nejdůležitějších budoucích vylepšení algoritmu.

Vliv počtu agentů a jejich dohledu na rychlost průzkumu

$\mu_{cas} = 20$, $\mu_{vel} = 8$					
Počet agentů	Dohled	3	4	5	6
1	Pozice	905,91	765,79	618,31	535,47
	Oblast	1913,24	1665,07	1303,77	1239,34
	Plocha	7854,6	8131,2	6040,7	3216,96
2	Pozice	478,29	373,51	309,79	266,9
	Oblast	1009,91	792,91	628,97	621,48
	Plocha	5840,83	4919,68	3927,58	2189,89
5	Pozice	191,83	147,31	121,26	103,26
	Oblast	374,21	273,17	260,33	257,42
	Plocha	4124,07	3852,78	3767,83	1698
15	Pozice	62,83	47,85	38,25	32,28
	Oblast	129,1	92,25	83,92	83,51
	Plocha	3606	3307,63	2834,6	1138,89

Tabulka 5.2: Efektivita algoritmu při pevném intervalu mezi změnami a velikosti změny prostředí

Stejně, jako v tabulce 5.1, i v tabulce 5.2 se téměř dokonale přímo úměrně zrychluje průzkum s narůstajícím počtem agentů. Stejně tak se při změnách dohledu agentů neděje nic nepředvídatelného a platí přímá úměra mezi velikostí dohledu a rychlosti průzkumu. Jediná zdánlivě podivná data v tabulce se nachází vpravo v trojřádku pro patnáct agentů, kde mezi dohledy 5 a 6 není u jednotlivých pozic, a zejména u oblastí, takový rozdíl jako v předchozích sousedních dvojicích. Obecně zde platí, že při vyšším počtu agentů s vyšším dohledem se už i při dokonalém rozprostření v prostoru dohledy agentů chvílemi překrývají, a proto zrychlení mezi dohledy 5 a 6 už není takové. Hodnota délky průzkumu oblastí je mezi dohledy 5 a 6 s patnácti agenty dokonce téměř stejná, a to proto, že zde už zřejmě nedošlo k úpravě rozměrů jednotlivých oblastí. Naopak v rámci celé plochy zvýšení dohledu na 6 pomohlo mnohem více v prozkoumávání jinak míjených pozic při obcházení překážek.

Vliv počtu agentů a velikosti změny prostředí na rychlost průzkumu

$\mu_{cas} = 20$, dohled $= 4$					
Počet agentů	μ_{vel}	2	4	8	16
1	Pozice	752,79	736,05	765,79	733,19
	Oblast	1075,46	1315,09	1665,07	1852,65
	Plocha	–	12476	8131,2	7984
2	Pozice	383,15	378,18	373,51	390,87
	Oblast	574,33	701,05	792,91	959,65
	Plocha	–	8794,1	4919,68	4066,55
5	Pozice	144,03	143,89	147,31	151,97
	Oblast	194,91	236,67	273,17	323,06
	Plocha	–	8650	3852,78	2259,03
15	Pozice	46,52	46,99	47,85	48,88
	Oblast	72,68	82,98	92,25	105,61
	Plocha	–	6322,5	3307,63	1694,58

Tabulka 5.3: Efektivita algoritmu při pevném dohledu agentů a intervalu mezi změnami.

Počet agentů i v tabulce 5.3 stále velice úspěšně ovlivňuje rychlost průzkumu. Vedle toho z tabulky zcela jednoznačně vyplývá lehké zpomalení směrem doprava v rámci průzkumu jednotlivých pozic a větší zpomalení v rámci průzkumu oblastí. Tento jev lze vysvětlit jednak tím, že se směrem vpravo mění více překážek a agentům často kazí naplánované cesty, ale druhak se při větších poloměrech změn prostředí často stává, že nově vytvořené překážky uzavřou agenta, a to se pak také projeví na naměřených hodnotách. U hodnot průzkumu plochy jako celku je naopak znát výrazný pokles směrem doprava. Důvod je přesně opačný než v tabulce 5.1. Větší velikost změn totiž znamená častější změny překážek a různé cesty agentů, kteří tyto překážky obcházejí. Z toho pak vyplyne, že pozice minuté agentem při jednom průchodu jsou většinou prozkoumány hned při dalším průchodu. Naopak ve sloupci se střední hodnotou velikosti změny 2 dokonce nebylo ani jednou prozkoumáno alespoň 95% celé plochy, protože při tak malém poloměru změny prostředí jsou změny téměř nepatrné a překážky často zůstávají na svých původních pozicích po celý běh simulace. Agenti tak vynechávají stále stejné pozice.

Vliv počtu agentů a dynamiky prostředí na aktivity agentů a hardwarovou náročnost

Tabulka 5.4 obsahuje informace o aktivitách agentů během simulací pro různé parametry dynamiky prostředí. Vedle znázornění doby strávené těmito aktivitami tabulka uvádí přibližnou hardwarovou náročnost simulace.

Počet kroků simulace, které agenti strávili jednotlivými aktivitami je silně ovlivněn počtem kroků simulace, na kolik byla spuštěna. Jak je uvedeno v úvodu této kapitoly, počet kroků simulace se s přibývajícími agenty zmenšuje. Pro zde uvedené počty agentů jsou počty kroků, na které je simulace spouštěna, následující: 1 agent – 20000 kroků, 2 agenti – 18500 kroků, 5 agentů – 16250 kroků a 15 agentů – 8750 kroků. Součet provedených kroků při jednotlivých aktivitách dá dohromady právě celkový počet kroků simulace. Na první pohled je zřejmé, že velikost změny prostředí výrazně ovlivňuje délku stání agentů, protože s většími změnami prostředí jsou agenti častěji uzavíráni překážkami, a musí proto

Dohled agentů = 4							
Počet agentů	μ_{cas}	10		20		40	
	μ_{vel}	2	16	2	16	2	16
1	Průzkum	16918	15225,6	16833	15344,8	16907	15933
	Cestování	2329,2	3506,8	2408,2	3221,2	2330,2	2852,6
	Stání	2,8	517,6	8,8	684	12,8	464,4
	FPS	7936,5	880,3	8771,9	931,1	4000	787,4
2	Průzkum	15122,5	14156,4	15242	14093,6	15345,8	14674,3
	Cestování	3376,6	3663,4	3248,9	3162,6	3144,8	3447,3
	Stání	0,9	680,2	9,1	1243,8	9,4	378,4
	FPS	7812,5	464,7	7575,8	553,1	2487,6	462,1
5	Průzkum	16097,2	14173,7	16045,2	14345,5	15949,6	14475,1
	Cestování	61,7	1205,8	123,4	1005,9	229	887
	Stání	91,1	870,5	81,4	898,6	71,4	887,9
	FPS	3703,7	88,9	1592,4	128,5	837,5	72
15	Průzkum	8530,8	7499,1	8404	7554,5	8437	7697,5
	Cestování	83,6	824,7	224,4	754,9	188,2	662,6
	Stání	135,6	426,2	121,6	440,6	124,8	389,9
	FPS	598,8	68,8	411,9	80,2	233	98,5

Tabulka 5.4: Aktivity agentů při různých parametrech dynamiky prostředí a náročnost simulace na hardware

stát a čekat, až překážky kolem nich zmizí. Na krocích strávených cestováním je v souvislosti s frekvencí změn prostředí vidět, že čím méně se prostředí mění, tím agent plánuje a provádí lepší cesty. Na době strávené prozkoumáváním nemá interval změny prostředí téměř žádný vliv. Z tabulky také lze vyčíst, že agenti ve víceagentním systému (zde myšleny systémy o 5 a více agentech) stojí více času než agenti, kteří jsou na prozkoumávání sami, nebo třeba jen 2. Hodnota stání agentů může být u systému s více agenty vyšší proto, že se agenti musí vypořádávat s kolizemi, jejichž řešením může být stání jednoho z agentů. Druhé vysvětlení vyššího počtu stání v systému více agentů může být větší pravděpodobnost, že bude některý agent uzavřen objevujícími se překážkami, a to se pak projeví do celkového průměru této hodnoty.

Hardwarové nároky byly měřeny na klasickém stolním počítači s procesorem AMD Ryzen 7 3700x a 16GB RAM. Simulace je samozřejmě výkonnostně náročnější pro větší počty agentů. Zároveň je ale také obecně náročnější při větších velikostech změn. K tomu pravděpodobně dochází kvůli špatné optimalizaci chování agenta, který je uzavřen překážkami (což se při větších velikostech změn děje relativně často). Agent v takové pozici se totiž pokouší postupně najít cestu do všech v tu dobu volných oblastí. Ani hledání cesty samo o sobě není pro hardware nijak jednoduchý proces, a pokud je agent uzavřen, tak může v rámci plánování jednoho jediného kroku postupně hledat cestu například do 20 různých cílů. Jinak simulace jede relativně dobře, nejmenší hodnotou FPS bylo 68,8.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a implementovat algoritmus plánující trasy multirobotického systému v dynamickém prostředí za účelem jeho prozkoumávání. Tohoto cíle bylo nakonec dosaženo, i když se v algoritmu stále nachází prostor ke zlepšení.

Na začátku práce proběhlo seznámení se stručnou teorií k multiagentním systémům, ta byla následně doplněna o dvě nastudované metody skupinového plánování cest. Poté byl navrhnut algoritmus pracující v dynamickém prostředí, který plánuje multiagentnímu systému cesty za účelem plošného pokrytí. Algoritmus řeší rozdělování prostoru na podoblasti, konflikty agentů a i případné přeplánování jejich tras. Po navržení byl celý algoritmus implementován do samostatné aplikace, která umí přijímat uživatelské vstupy a vykreslovat chování algoritmu. Nakonec byly naměřeny a okomentovány hodnoty pro různé parametry dynamiky prostředí a agentů.

Výsledky práce jsou velmi dobré, jelikož například zrychlení průzkumu mezi 1 agentem a 15 agenty bylo dokonce 15,875násobné. Také je na výsledcích patrný vliv dynamiky prostředí, který u častěji se měnícího prostředí zpomaluje celkový průzkum ale pomáhá s prozkoumáním jinak míjených pozic. Aplikace, ve které je algoritmus naimplementován, selhává velice vzácně. Agenti během průzkumu nezůstávají bezdůvodně stát a nenechávají si svoje plány narušit jakýmikoli situacemi, které v prostředí nastávají.

Nejdůležitějším budoucím vylepšení algoritmu by bylo určitě doprozkoumávání pozic, které byly při průzkumu minuty. To je momentálně největší vadou algoritmu. Tyto pozice totiž často zůstávají neprozkoumané velmi dlouho, dokud dynamika prostředí nedonutí agenty kolem nich projít. Dalším možným vylepšením je respektování průzkumu, který agent „omylem“ provede při přesunu mezi oblastmi. Respektováním se myslí zbytečné neprozkoumávání částí oblastí, kterými cestující agent před chvílí prošel. Kromě těchto větších vylepšení je v algoritmu spousta prostoru na optimalizaci a zefektivňování.

Literatura

[1] *A* search algorithm* [online]. Wikipedia, The Free Encyclopedia., březen 2021 [cit. 2021-04-13]. Dostupné z: https://en.wikipedia.org/w/index.php?title=A*_search_algorithm&oldid=1015185222.

[2] *Metody řešení úloh prohledáváním stavového prostoru* [online]. FIT VUT v Brně, únor 2021 [cit. 2021-04-13]. Snímky 70-86. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FIZU-IT%2Flectures%2FIZU%2F2021izu_2.pdf.

[3] Brooks, R. A. Intelligence without representation. *Artificial Intelligence*. 1991, sv. 47, č. 1, s. 139–159. DOI: https://doi.org/10.1016/0004-3702(91)90053-M. ISSN 0004-3702. Dostupné z: https://www.sciencedirect.com/science/article/pii/000437029190053M.

[4] Ge, S. S. a Fua, C. Complete Multi-Robot Coverage of Unknown Environments with Minimum Repeated Coverage. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, s. 715–720. DOI: 10.1109/ROBOT.2005.1570202.

[5] Kong, C. S., Peng, N. A. a Rekleitis, I. Distributed coverage with multi-robot system. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* 2006, s. 2423–2429. DOI: 10.1109/ROBOT.2006.1642065.

[6] Oprea, M. Applications of Multi-Agent Systems. In: Reis, R., ed. *Information Technology*. Boston, MA: Springer US, 2004, s. 239–270. ISBN 978-1-4020-8159-0.

[7] Parasumanna Gokulan, B. a Srinivasan, D. An Introduction to Multi-Agent Systems. In:. červenec 2010, sv. 310, s. 1–27. DOI: 10.1007/978-3-642-14435-6_1. ISBN 978-3-642-14434-9.

[8] Zbořil, F. *Plánování a komunikace v multiagentních systémech*. 2004. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: http://www.fit.vutbr.cz/~zborilf/PhD/thesis.pdf.

[9] Zbořil, F. *Úvod do agentních a multiagentních systémů* [online]. FIT VUT v Brně, prosinec 2020 [cit. 2021-04-23]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FAGS-IT%2Flectures%2FAGS_20_01b.pdf.