FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Hledání cest a přiřazování cílů ve zobecnění hry Pacman
Student:	Petr Šindlář
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je navrhnout řídící algoritmy pro zobecnění hry Pacman, poprvé představené v roce 1980, kde na rozdíl od původní verze bude Pacmanů více. Úkolem skupiny Pacmanů bude zkonzumovat všechny kuličky v bludišti definovaném na čtvercové síti. V bludišti jsou také přítomni pohybující se duchové, jimž je třeba se vyhýbat. Navíc je třeba se vyhýbat srážkám mezi Pacmany. Problém řízení Pacmanů kombinuje úlohu hledání cest a přiřazování cílů. Úkoly uchazeče budou následující:

1. Provede rešerši existujících algoritmů pro multi-agentní hledání cest a přiřazování cílů.

2. Navrhne vlastní či modifikuje existující algoritmus pro řízení Pacmanů

3. Algoritmus implementuje formou softwarového prototypu a provede jeho otestování v různých instancích hry s ohledem na rychlost zkonzumování kuliček.

–

[1] David Silver: Cooperative Pathfinding. AIIDE 2005: 117-122

[2] Hang Ma, Sven Koenig: Optimal Target Assignment and Path Finding for Teams of Agents. AAMAS 2016: 1144-1152

[3] Viliam Lisý, Branislav Bosanský, Michal Jakob, Michal Pechoucek: Goal-based Adversarial Search - Searching Game Trees in Complex Domains using Goal-based Heuristic. ICAART 2009: 53-60.

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 10. prosince 2020 v Praze.

Bakalářská práce

HLEDÁNÍ CEST A PŘIŘAZOVÁNÍ CÍLŮ VE ZOBECNĚNÍ HRY PACMAN

Petr Šindlář

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: doc. RNDr. Pavel Surynek, Ph.D.
11. února 2022

Odkaz na tuto práci: Petr Šindlář. *Hledání cest a přiřazování cílů ve zobecnění hry Pacman.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Obsah

Poděkování vi

Prohlášení vii

Abstrakt viii

Seznam zkratek ix

Úvod 1

1 Cíl práce 3

2 Teoretická část 5
2.1 Multi-agentní hledání cest . . . 5
2.1.1 Konflikty . . . 6
2.1.2 Kvalita řešení . . . 6
2.1.3 Přístupy k řešení . . . 7
2.1.4 Příklad MAPF . . . 7
2.2 Algoritmy pro řešení MAPF . . . 8
2.2.1 A* . . . 8
2.2.2 Suboptimální algoritmy . . . 9
2.2.2.1 CA* . . . 10
2.2.2.2 HCA* . . . 10
2.2.2.3 WHCA* . . . 11
2.2.3 Optimální algoritmy . . . 11
2.2.3.1 A* na k-agentním stavovém prostoru . . . 12
2.2.3.2 CBS . . . 12
2.3 Přiřazování cílů . . . 14
2.3.1 Příklad TAPF . . . 14
2.3.2 Existující algoritmy . . . 15
2.3.2.1 CBM . . . 15
2.3.2.2 CBS-TA . . . 16
2.4 Seřazování cílů . . . 17
2.4.1 Příklad MG-MAPF . . . 18
2.4.2 Existující způsoby řešení . . . 18
2.4.2.1 HCBS . . . 19
2.4.2.2 Aproximační algoritmus pro TSP . . . 19

3 Analýza **21**
3.1 Hra Pacman 21
3.2 Zobecnění hry Pacman pro účely práce 23

4 Návrh řešení a implementace **25**
4.1 Struktura programu 25
4.2 Vstup 26
4.3 Návrh algoritmu 26
4.3.1 Rozdělení mapy a přiřazení cílů 27
4.3.2 Seřazení cílů 30
4.3.3 Hledání cest 31
4.4 Výstup 37

5 Experimenty **39**
5.1 Porovnání algoritmů pro rozdělení mapy 39
5.2 Testování různého počtu Pacmanů a duchů na původní mapě . 40

Závěr **43**

Bibliografie **45**

Příloha **49**
.1 Ovládání softwarového prototypu 49

Obsah přiloženého média **51**

Seznam obrázků

2.1 Instance klasického a anonymního MAPF 7
2.2 Instance TAPF 15
2.3 Instance MG-MAPF 18

3.1 Originální mapa hry Pacman 22

5.1 Graf zobrazující rozsahy přidělených oblastí v obou variantách přiřazovacího algoritmu 39
5.2 Grafy zobrazující vliv počtu duchů a Pacmanů na dokončená řešení, makespan a dobu běhu algoritmu 41

Seznam pseudokódů

2.1 A* 9
2.2 Vyšší úroveň CBS 13

4.1 Hlavní úroveň algoritmu 27
4.2 Rozdělení mapy (BFS verze) 28
4.3 Rozdělení mapy (DFS verze) 29
4.4 Seřazení cílů 32
4.5 Hledání cest – metoda `findAgentPaths()` 33
4.6 Hledání cest – modifikovaný A* 35

Chtěl bych poděkovat především vedoucímu své práce panu doc. RNDr. Pavlovi Surynkovi, Ph.D. za cenné rady a konzultace, které mi pomohly lépe pochopit zkoumanou problematiku. Dále také děkuji své rodině za veškerou podporu během celého studia.

Prohlášení

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

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

V Praze dne 11. února 2022 .....

Abstrakt

Tato bakalářská práce se zabývá tématikou multi-agentního hledání cest a dvěma rozšiřujícími úlohami – přiřazováním a seřazováním cílů. Rešeršní část práce přibližuje tyto problémy a možnosti jejich řešení. Nastudované koncepty jsou aplikovány na zobecnění hry Pacman, ve kterém je libovolný počet Pacmanů a duchů. V práci je navržen a implementován algoritmus, který řeší instance tohoto zobecnění. K tomu jsou využity modifikace existujících algoritmů BFS, DFS, A* a CBS.

Klíčová slova multi-agentní hledání cest, přiřazování cílů, Pacman, algoritmy založené na A*, CBS

Abstract

This bachelor thesis deals with the topic of multi-agent path finding and two extending problems – target assignment and goal ordering. Research part of this work gives insight into these problems and ways of solving them. Studied concepts are applied to generalized version of Pacman game in which there can be more Pacmans and ghosts. This thesis contains the design and implementation of an algorithm that solves instances of this generalization using modifications of existing algorithms BFS, DFS, A* and CBS.

Keywords multi-agent path finding, target assignment, Pacman, A*-based algorithms, CBS

Seznam zkratek

ASP	Answer Set Programming
BFS	Breadth-First Search
CA*	Cooperative A*
CBM	Conflict-Based Min-Cost-Flow
CBS	Conflict-Based Search
CBS-TA	Conflict-Based Search–Task Assignment
DFS	Depth-First Search
G-TAPF	Generalized Target Assignment and Path Finding
HCA*	Hierarchical Cooperative A*
HCBS	Hamiltonian Conflict-Based Search
MAPF	Multi-Agent Path Finding
MG-MAPF	Multi-Goal Multi-Agent Path Finding
RRA*	Reverse Resumable A*
SAT	Boolean Satisfiability Problem
TAPF	Target Assignment and Path Finding
TASS	Tree-based Agent Swapping Strategy
TSP	Travelling Salesman Problem
WHCA*	Windowed Hierarchical Cooperative A*

Úvod

Multi-agentní hledání cest je známým problémem umělé inteligence a robotiky, který nalézá široké uplatnění v oblastech reálného světa – od letectví přes využití robotů ve skladech po počítačové hry. Různé aplikace vyžadují jiné formulace či rozšíření základní úlohy a jiné přístupy k řešení.

V práci je problém řešen v kontextu známé arkádové a počítačové hry Pac-man. V ní hráč ovládá stejnojmennou postavičku, kterou naviguje po 2D bludišti. Cílem je sesbírat všechny kuličky rozmístěné po bludišti a zároveň se vyhýbat čtyřem duchům. Pro účely práce jsou pravidla zobecněna, hlavní změnou je volitelný počet Pacmanů (agentů) i duchů. Situace vychází ze zábavné hry, může však modelovat i skutečnou situaci jako např. roboty, kteří mají za úkol vyklidit určitý prostor a vyhýbat se při tom lidem či jiným dynamickým překážkám.

Zavedením více Pacmanů se úloha komplikuje, protože kromě hledání cest s centralizovanou agentní koordinací (Pacmani se nesmí srazit navzájem mezi sebou ani s duchy) je potřeba řešit také přiřazení cílů (jakou část mapy má který z nich sesbírat) a jejich seřazení (v jakém pořadí přidělené kuličky vyzvednout). Zároveň je třeba se snažit, aby nalezené cesty byly co nejkratší.

V teoretické části práce jsou nejprve vysvětleny a definovány potřebné pojmy. Dále jsou popsány existující algoritmy, které jsou relevantní pro zadaný problém. V analytické části je přiblížena originální podoba hry Pacman a poté její úpravy pro tuto práci.

V praktické části je navržen způsob řešení a jsou popsány použité algoritmy a jejich modifikace. Celkové řešení je průběžně popsáno na modelové instanci. V poslední experimentální části jsou výsledky ověřeny a porovnány z hlediska rychlosti sesbírání kuliček na několika herních mapách s různými počty agentů a duchů.

Kapitola 1
Cíl práce

Hlavním cílem práce je navrhnout algoritmus pro řízení agentů ve zobecnění hry Pacman, které se od původní verze liší především větším počtem Pacmanů v herním bludišti. Ti se snaží spolupracovat a společně posbírat všechny kuličky co nejefektivnějším způsobem a vyhýbat se kolizím jak mezi sebou, tak s duchy, kteří hrají roli dynamických překážek.

Cílem teoretické části práce je seznámit se s tématikou multi-agentního hledání cest a přiřazování cílů, provést rešerši existujících řešení, a tato včetně potřebných definic a pojmů shrnout a popsat.

Cílem praktické části práce je pak výběr, modifikace vhodných dílčích algoritmů, jejich implementace v podobě terminálové aplikace s textovou vizualizací a jejich otestování na původní i vlastních herních mapách.

Kapitola 2
Teoretická část

V této kapitole budou představena teoretická východiska práce. Nejprve bude definováno multi-agentní hledání cest a s ním související pojmy. Následuje přehled způsobů a algoritmů pro jeho řešení, z nichž je většina založena na algoritmu A*. Dále budou popsána dvě rozšíření základní varianty úlohy, která do problematiky přináší přiřazení cílů agentům a jejich seřazení. Pro každé z nich budou také uvedeny možnosti řešení.

2.1 Multi-agentní hledání cest

Multi-agentní hledání cest (MAPF, *multi-agent path finding*) je zobecněním klasické úlohy hledání cesty jednoho agenta pro větší počet agentů. To přináší nové klíčové omezení: agenti se při souběžném následování nalezených cest nesmí srazit.

MAPF je v literatuře formálně definováno více způsoby, poprvé je zavedeno v [1] jako pohyb kamenů po grafu, další definice např. v [2, 3] lépe odpovídají problémům s roboty či agenty. Základní (neanonymní) MAPF problém [4, 5] lze definovat jako čtveřici (G, A, s, g). Prostor pro hledání je zadán grafem $G = (V, E)$, kde vrcholy v množině V jsou pozice, na kterých se mohou agenti nacházet, a hrany v množině E spojují sousední vrcholy a reprezentují tak možné přechody mezi nimi. $A = \{a_1, \ldots, a_n\}$ je množina n agentů. Zobrazení $s, g : A \to V$ přiřazují každému agentovi startovní a cílový vrchol.

V anonymní variantě MAPF [4, 6] jsou cíle zadány jako množina, nejsou přiřazeny jednotlivým agentům. Nezáleží v ní tedy na tom, který agent dorazí do kterého cíle, z tohoto pohledu jsou agenti zaměnitelní, tedy anonymní.

Prostor můžeme také zakreslit do dvourozměrné mřížky, ve které má každé pole 2^k sousedů (2^k-*connected grid* [4, 7]). Ta koresponduje se zadáním pomocí grafu – pole odpovídají vrcholům, hrany vedou do dosažitelných sousedů v povolených směrech. Klasickým příkladem je mřížka, po které se pohybujeme horizontálně a vertikálně (4-*connected grid*).

Pohyb agentů je vykonáván v časových krocích. Každý agent se v každém kroku nachází na jedné pozici a může provést jednu ze dvou akcí: čekání, kdy

v dalším časovém kroku zůstane ve stejném vrcholu, a pohyb, kdy přejde do sousedního vrcholu podél jedné z hran. Akce $\alpha : V \to V$ tedy přiřazuje vrcholu následníka v dalším kroku.

Plán i-tého agenta je posloupnost akcí $\pi_i = (\alpha_1, \ldots, \alpha_m)$, jejichž postupným provedením se agent přesune ze startovní pozice $s(a_i)$ do cílové pozice $g(a_i)$, tj. $\alpha_m(\alpha_{m-1}(\cdots\alpha_1(s(a_i)))) = g(a_i)$. [4] Jako cestu i-tého agenta pak můžeme označit posloupnost vrcholů $C_i = (\alpha_1(s(a_i)), ..., g(a_i))$, které při vykonávání plánu navštíví. Pojmy plán a cesta se v literatuře často zaměňují.

Řešením MAPF problému je množina π obsahující n plánů resp. cest pro jednotlivé agenty. Za validní řešení však považujeme pouze taková, která nemají žádné konflikty. [4, 5]

2.1.1 Konflikty

Konfliktem rozumíme nežádoucí situaci mezi dvěma cestami, která by vedla ke kolizi dvou agentů. V této práci požadujeme, aby nevznikaly tyto typy konfliktů:

- vrcholový konflikt, kdy se dle plánu mají dva agenti nacházet ve stejném čase na stejné pozici, tj. $C_i[t] = C_j[t]$,
- hranový konflikt, kdy se dle plánu mají dva agenti ve stejném čase přesunout mezi dvěma pozicemi ve stejném směru, tj. $C_i[t] = C_j[t]$ a zároveň $C_i[t+1] = C_j[t+1]$, či si pozice vyměnit, tj. $C_i[t] = C_j[t+1]$ a současně $C_i[t+1] = C_j[t]$.

Lze si povšimnout, že zákaz vrcholových konfliktů automaticky znamená také zákaz hranových ve stejném směru. Proto se často hranovým konfliktem myslí výměna pozic, pro kterou se případně pro rozlišení používá pojem *swapping* konflikt [4].

Dalšími typy jsou např. *following* konflikt, který značí těsné následování jednoho agenta druhým, či *cycle* konflikt, který obnáší několik agentů „rotujících“ dokola po stejných pozicích těsně za sebou.

2.1.2 Kvalita řešení

MAPF problém může mít několik validních řešení, která mohou být různě kvalitní. Kvalitu (cenu) řešení lze posuzovat podle nákladových funkcí. Nejpoužívanější dle [4, 8] jsou:

- *makespan* – počet časových kroků potřebných k tomu, aby se všichni agenti nacházeli ve své cílové destinaci (odpovídá délce cesty nejpomalejšího agenta), tj. $\max_{1 \le i \le k} |\pi_i|$,
- *sum of costs* – součet časových kroků potřebných k tomu, aby se všichni agenti nacházeli ve svém cíli (odpovídá součtu délek cest jednotlivých agentů), tj. $\sum_{1 \le i \le k} |\pi_i|$.

Další nákladovou funkcí je např. *fuel*, což je varianta *sum of costs*, která nezapočítává časové kroky, ve kterých agenti zůstávají na stejném místě, a odpovídá tak energii vynaložené na jejich pohyb. [9]

Optimální řešení je pak takové, které je nejlepší z pohledu minimalizace zvolené nákladové funkce.

2.1.3 Přístupy k řešení

K řešení MAPF problémů lze dle [5, 9] přistupovat dvěma způsoby s ohledem na konkrétní zadání a požadavky úlohy.

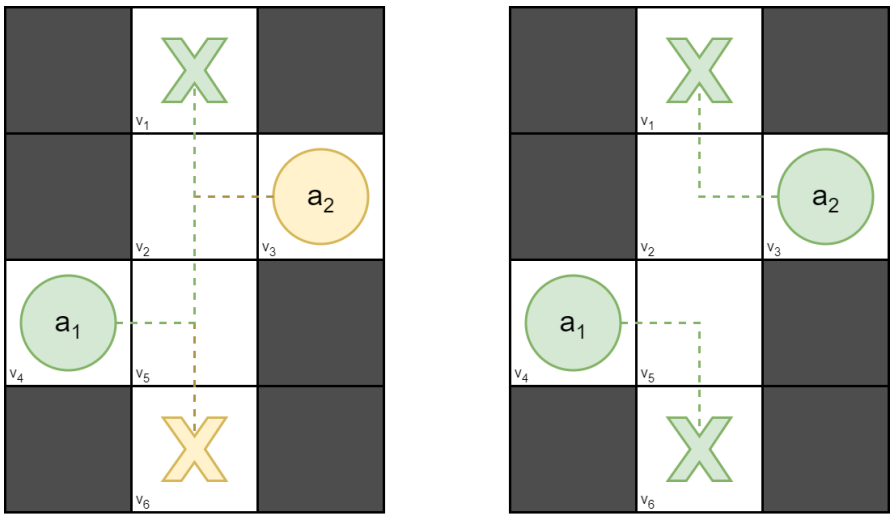
Distribuovaný (oddělený) přístup bere agenty jako jednotlivce, kteří si sami hledají svou cestu k cíli, mají vlastní výpočetní sílu a rozhodovací strategii. Ta může být kooperativní, kdy se agenti snaží spolupracovat, nebo nekooperativní, kdy jsou zaměřeni čistě na dosažení vlastního cíle. Komunikace mezi agenty může probíhat např. pomocí předávání zpráv.

Oproti tomu v centralizovaném přístupu jsou agenti plně ovládání jednou centrální výpočetní sílou, která má kompletní informace o agentech i prostředí. Takto můžeme nalézt všechna možná řešení.

V této práci bude k řešení přistupováno centralizovaně.

2.1.4 Příklad MAPF

Na obrázku 2.1 jsou na mřížkové mapě znázorněny dvě instance MAPF. Bílá pole značí průchozí pozice, tmavá pole překážky. Kruhoví agenti se mají dostat do cíle označeného symbolem „X“. Čárkovaně jsou naznačeny cesty agentů řešící problém.



Obrázek 2.1 Instance klasického a anonymního MAPF

Vlevo se jedná o klasické MAPF, agenti a jejich přiřazené cíle jsou odlišeni barevně. Nejkratší cesty agentů mají konflikt – při souběžném vykonávání plánu by si v čase $t = 1$ agenti chtěli vyměnit pozice mezi vrcholem v_2 a v_5, což

není možné. Řešením je nechat jednoho z agentů, např. a_2, čekat na počáteční pozici do času $t = 2$. V čase $t = 3$ již a_1 bude v cíli a nebude blokovat společnou „chodbu“. Obou cílů bude dosaženo v čase $t = 5$.

V anonymním MAPF vpravo nejsou agenti ani cíle rozlišeni. I v tomto případě by šlo použít stejné řešení, avšak zde si můžeme dovolit zvolit jiné přiřazení cílů, což má za následek řešení lepší – oba agenti budou v cíli již v čase $t = 2$.

2.2 Algoritmy pro řešení MAPF

Existuje mnoho algoritmů řešících MAPF. Můžeme je rozdělit do dvou skupin podle toho, jestli jsou schopny nalézt optimální řešení, nebo pouze nějaké, které se optimálnímu více či méně blíží. Algoritmy obou skupin, které pracují na principu prohledávání, často využívají algoritmus A* či nějaké jeho vylepšení.

2.2.1 A*

A* [10] je algoritmus pro informované hledání nejkratší cesty v grafu mezi zadaným startovním a cílovým vrcholem. Jde také použít pro prohledávání stavového prostoru, protože stavový prostor lze modelovat pomocí grafu. Je nejznámějším z *best-first search* algoritmů, které nejdříve prozkoumají nejvíce slibné vrcholy. Vrcholy jsou ohodnoceny evaluační funkcí $f(n) = g(n) + h(n)$, která k dosavadní délce cesty ze startu do vrcholu n, značené $g(n)$, přidává odhad délky cesty z n do cíle daný heuristickou funkcí $h(n)$. [11]

Heuristika je způsob výběru nejlépe vypadajícího rozhodnutí z více možností na základě dostupných informací. Výběr by měl být proveden dle jednoduchých kritérií, které ale zároveň umí dobře rozlišit mezi vhodnými a nevhodnými možnostmi. [12] Pro vzdálenosti na 2D mapě je vhodnou heuristickou funkcí např. Manhattanská metrika (také Manhattanská vzdálenost). Ta se vypočítá jako $h(a, b) = \sum_{i=1}^{n} |a_i - b_i|$, kde a a b jsou dva body a a_i resp. b_i značí souřadnice bodu v i-tém rozměru. [11]

Algoritmus A* je uveden v pseudokódu 2.1. A* si udržuje prioritní frontu otevřených vrcholů (`open`) čekajících na zpracování, vzdálenosti navštívených vrcholů od startu (`distSoFar`) a pro rekonstrukci cesty také eviduje, odkud byl který vrchol navštíven (`cameFrom`). Algoritmus vyzvedává z fronty `open` vrchol s nejnižší $f(n)$ hodnotou. Pokud se jedná o cílový vrchol, algoritmus končí a je možno vrátit hledanou cestu – funkce `reconstructPath` ji sestaví opačným průchodem od cílového vrcholu přes předchůdce až do startovního a obrácením této posloupnosti. V opačném případě je provedena expanze vrcholu. Pro každý z jeho sousedních vrcholů je vypočítána $g(n)$. Pokud daný soused ještě nebyl objeven nebo byl dosažen kratší cestou, uložíme $g(n)$ a vrchol, odkud byl dosažen, vypočteme prioritu $f(n)$ a přidáme jej do fronty. Pokud cíl nebyl nalezen a fronta je prázdná, není dosažitelný (např. kvůli překážkám) a algoritmus vrací tuto informaci.

Pro zaručení nalezení nejkratší cesty je potřeba, aby heuristika $h(n)$ byla přípustná, tedy že nikdy nevrátí horší odhad délky cesty do cíle, než je skutečná vzdálenost. Pokud je navíc konzistentní (monotónní), tj. pokud platí trojúhelníková nerovnost $h(a) \leq d(a,b) + h(b)$, kde $d(a,b)$ značí skutečnou vzdálenost mezi dvěma vrcholy (délku hrany mezi nimi), algoritmus expanduje každý vrchol maximálně jednou a nenavštíví tak již objevený vrchol kratší cestou. Konzistentní heuristika je automaticky také přípustná. Manhattanská vzdálenost splňuje obě tyto vlastnosti. [11]

Algoritmus A* je na konečných grafech úplný (skončí a najde řešení) a při použití přípustné heuristiky je také optimální (vždy najde nejkratší cestu). Důkaz je uveden v [10].

Input: graph, start node, goal node, heuristic function
Result: shortest path from start node to goal node

```
1  open ← empty priority queue;
2  distSoFar[start] ← g(start) = 0;
3  cameFrom[start] ← ∅;
4  priority ← f(start) = 0 + h(start);
5  insert (start, priority) into open;
6  while open not empty do
7  |  (priority, current) ← node from open with lowest f(n);
8  |  remove (priority, current) from open;
9  |  if current is goal then
10 |  |  return reconstructPath(cameFrom, goal);
11 |  end
12 |  neighbors ← set of nodes accessible from current;
13 |  foreach node next in neighbors do
14 |  |  newDist ← distSoFar[current] + d(current, next);
15 |  |  if distSoFar[next] = ∅ or newDist < distSoFar[next] then
16 |  |  |  distSoFar[next] ← newDist;
17 |  |  |  newPrio ← f(next) = newDist + h(next);
18 |  |  |  insert (next, newPrio) into open;
19 |  |  |  cameFrom[next] ← current;
20 |  |  end
21 |  end
22 end
23 return no path found;
```

Pseudokód 2.1: A*

2.2.2 Suboptimální algoritmy

Suboptimální algoritmy se dle [5, 9] zaměřují na rychlejší nalezení cest pro všechny agenty výměnou za méně optimální řešení. Jsou obvykle používány v úlohách s velkým množstvím agentů, kde je nejlepší řešení velmi těžké nalézt, nebo tam, kde je více preferována rychlost před nejlepším výsledkem. Tuto

kategorii můžeme dle dále rozdělit do tří skupin.

Algoritmy založené na principu prohledávání se snaží o nalezení řešení, které je kvalitou blízko optimálnímu, avšak pro některé případy jej vůbec nenajdou. Patří mezi ně např. algoritmy CA*, HCA* a WHCA* představené v [13], které jsou blíže popsány dále.

Algoritmy založené na pravidlech naopak upřednostňují rychlost a jistotu nalezení řešení na úkor jeho kvality. Obsahují specifická pravidla pro pohyb agentů v různých situacích a pro jejich fungování je obvykle nutné, aby graf, na kterém je úloha formulována, splňoval určité vlastnosti. Patří zde např. TASS [14], který vždy najde řešení na stromových grafech, či BIBOX [15] pro *bi-connected* grafy, které zůstanou propojené po odebrání jakéhokoli vrcholu.

Hybridní algoritmy kombinují hledání i pravidla. Zavádí omezení na to, kterými směry se mohou agenti vydat, což snižuje šance na konflikt a také počet následníků každého vrcholu. Cesty jsou tedy v důsledku delší. Tento přístup funguje na velkých otevřených mapách, ale obecně nenajde řešení vždy, protože agenti mohou uváznout v bezvýchodné situaci.

2.2.2.1 CA*

CA* (*Cooperative A**) plánuje cesty agentů po jednom. Hledání probíhá na zadaném prostoru doplněném o dimenzi času. Každá cesta je po nalezení zaznamenána do rezervační tabulky, čímž jsou vrcholy cesty v určitých časech považovány za zablokované a tedy se jim následující agenti během hledání vlastní cesty musí vyhnout.

Nevýhodou CA* je fakt, že jsou cesty hledány v určitém předem definovaném pořadí. Může se tak stát, že jeden agent zablokuje svou cestou cestu dalším. To lze částečně řešit pomocí prioritizace agentů, kdy pořadí hledání závisí na jednotlivých prioritách a může se v průběhu algoritmu měnit.

2.2.2.2 HCA*

HCA* (*Hierarchical CA**) zlepšuje předchozí algoritmus pomocí abstrakce stavového prostoru a jiné heuristiky. Vzdálenosti do cíle se odhadují na prostoru, kde ignorujeme časovou dimenzi a ostatní agenty. Jedná se tak o odhady délky zbývající cesty v perfektním scénáři, kdy nenastane žádný konflikt s jiným agentem. Počítat tuto heuristiku v každém otevřeném vrcholu by bylo neefektivní a zpomalilo by celkový algoritmus. Proto jsou hodnoty heuristiky pro HCA* získávány pomocí algoritmu RRA* (*Reverse Resumable A**).

Ten provádí hledání pomocí A* na abstrakci prostoru, avšak v opačném směru od cíle $g(a_i)$ ke startovní pozici $s(a_i)$ agenta. RRA* běží nejen do dosažení této startovní pozice, ale až do expanze vrcholu, pro který odhadujeme vzdálenost k cíli. A* garantuje, že při použití konzistentní heuristiky známe nejlepší vzdálenost od začátku do daného vrcholu ve chvíli, kdy je tento vrchol expandován. RRA* samotné používá jako heuristiku Manhattanskou vzdálenost.

RRA* si navíc ukládá již expandované vrcholy a jejich hodnoty, a tak při dalším požadavku na heuristiku od HCA* může vrátit již dříve vypočíta-

nou hodnotu, pokud vrchol, ve kterém heuristiku požadujeme, již byl během předchozích požadavků navštíven.

2.2.2.3 WHCA*

WHCA* (*Windowed HCA**) je dalším rozšířením, které je zaměřeno na eliminaci nevýhod předchozích algoritmů. Jednou z nich je již dříve zmíněné pořadí agentů, kdy některé problémy nelze s fixním pořadím vyřešit. Další nevýhodou je, že po dosažení cíle agenti stojí na místě a mohou tak blokovat cestu jiným. Předchozí algoritmy také hledají kompletní cestu do cíle, což vede k nutnosti přeplánovat, pokud se vyskytne nějaká nepředvídatelnost. Lepším přístupem může být omezení hledání na menší úseky.

WHCA* řeší všechny najednou zavedením okénka, které limituje kooperativní hledání s rezervační tabulkou na určitý počet kroků w. Každý agent nalezne částečnou cestu o délce w a poté se po ní hned vydá. Okénko se v pravidelných intervalech posouvá a hledají se navazující částečné cesty, přičemž můžeme měnit priority agentů.

Po dosažení w kroků jsou po zbytek hledání ostatní agenti ignorováni. To odpovídá hledání v abstraktním stavovém prostoru z HCA* a tedy není nutno skutečné hledání dokončit, lze využít heuristiky RRA*. Zavádí se speciální hrany, které vedou z vrcholů dosažených po w krocích přímo do cíle a jejichž délka je dána heuristikou RRA*. To v důsledku omezuje skutečné hledání pouze na w kroků.

Jakmile agent dorazí do cíle, lze s ním dále pracovat, pokud změníme jeho úkol na dokončení cesty pomocí speciální hrany. Každých w kroků končí speciální hranou, takže cíle vždy dosáhne a dostává tak „manévrovací“ prostor o délce w na uvolnění cesty jiným.

2.2.3 Optimální algoritmy

Optimální algoritmy se zaměřují na nalezení nejlepšího řešení. Jejich použití se dle [5, 9] vyplatí především pokud je počet agentů k malý, jelikož stavový prostor MAPF roste exponenciálně vzhledem právě ke k. Tuto kategorii lze také rozdělit do tří skupin.

MAPF je možné redukovat na (vyjádřit jako) jiné známé problémy v informatice, které víme, jak řešit. Graf, pozice agentů i jejich omezení lze například zakódovat jako logickou formuli a poté řešit SAT (problém splnitelnosti booleovské formule) [16]. Tyto redukce jsou výhodné pro menší grafy, u větších zabere mnoho času samotný převod na jiný problém, což je neefektivní.

Také samotný A* (či algoritmy založené na něm) lze použít pro MAPF, avšak je potřeba prohledávat tzv. k-agentní stavový prostor namísto prostoru daného grafem v zadání úlohy, viz kapitola 2.2.3.1.

Třetí skupinou jsou algoritmy, které nejsou primárně založené na A*. Tyto prohledávají ve dvou úrovních, kdy vyšší úroveň se stará o zamezení konfliktů jednotlivých agentů pomocí omezujících podmínek předávaných nižší úrovni, která má na starost hledání cest jednotlivých agentů s ohledem na tyto pod-

mínky. V nižší úrovni lze na hledání cest použít A*, ale také jiné algoritmy schopné téhož. Do této skupiny patří např. CBS.

2.2.3.1 A* na k-agentním stavovém prostoru

Algoritmus A* lze podle [5] přímo použít pro MAPF za předpokladu, že jím prohledáváme tzv. k-agentní stavový prostor. Ten reprezentujeme grafem, ve kterém vrcholy (stavy) jsou možná rozmístění k agentů do $|V|$ pozic původního grafu a hrany (přechody mezi stavy) jsou možné kombinace akcí jednotlivých agentů, které nevedou ke konfliktům. Počáteční stav má všechny agenty umístěné ve svých startovních vrcholech a koncový stav v cílových. Na takto zadaném stavovém prostoru lze MAPF vyřešit optimálně.

Jako heuristiku pro tento A* lze použít součet heuristik jednotlivých agentů. Jinými variantami jsou např. heuristiky *sum* resp. *maximum of individual costs* (dle toho, co minimalizujeme), které pro každého agenta počítají ideální délku cesty do cíle za předpokladu neexistence jiných agentů.

Jeho velkou nevýhodou je však příliš velký faktor větvení (číslo vyjadřující počet následníků každého vrcholu). Na 4-*connected* mřížce, kde každý agent má k dispozici pět akcí (pohyb do čtyř směrů a čekání), je faktor větvení k-agentního stavového prostoru $b = 5^k$, což pro např. 20 agentů dělá $9{,}53 \times 10^{14}$ možných rozmístění agentů v kroku následujícím po startovním stavu. Vůbec vygenerovat všechny tyto možnosti by bylo výpočetně náročné. A* navíc potřebuje otevřené vrcholy uchovávat ve frontě `open`, kterou by pro takový počet stavů nebylo možné udržovat v paměti.

Redukovat tuto nevýhodu pomáhají různá vylepšení jako např. *independence detection*, která spočívá v rozdělení agentů do skupin, plánování pro jednotlivé skupiny a sloučení konfliktních skupin do jedné větší, čímž je exponent faktoru větvení zmenšen na počet agentů v největší skupince, či techniky, díky kterým je možno se vyhnout generování stavů nepotřebných k nalezení optimálního řešení.

2.2.3.2 CBS

CBS (*Conflict-Based Search*) [9] rozděluje MAPF na množství jednoagentních hledání cest s omezeními. Omezením rozumíme trojici (a_i, v, t), která vyjadřuje, že se agent a_i nesmí vyskytovat na pozici v v čase t.

Vyšší úroveň algoritmu prohledává tzv. strom konfliktů (*conflict tree*). Každý jeho vrchol obsahuje množinu omezení, řešení problému a cenu tohoto řešení dle nákladové funkce. Kořen stromu nemá žádná omezení. Cílový vrchol je takový, který obsahuje validní tj. bezkonfliktní řešení.

V kořenovém vrcholu `root` nalezneme cesty pro všechny agenty, vypočteme cenu tohoto řešení a vložíme jej do prioritní fronty `open`. Vrcholy jsou zpracovávány v pořadí podle jejich ceny. Po vyzvednutí je provedena validace řešení, protože cesty agentů mohou mít konflikty. Konflikt je čtveřice (a_i, a_j, v, t) vyjadřující agenty a_i, a_j ve stejném čase na stejné pozici. Jakmile je při kontrole nalezen první konflikt, není třeba kontrolovat dál a je nutno jej vyřešit. Pokud žádný konflikt neexistuje, je vrchol cílový a řešení může být vráceno.

Konflikt je řešen rozdělením vrcholu na dva potomky. Potomci zdědí stejnou množinu omezení, jako má rodič, a přidají se omezení vzniklé v důsledku konfliktu, tedy (a_i, v, t) do jednoho a (a_j, v, t) do druhého. Protože se omezení takto přidávají k již existujícím, není nutno do každého vrcholu kopírovat celou množinu, stačí si uložit pouze nově přidané omezení a k ostatním se dostat průchodem stromu od aktuálního vrcholu do kořene.

Potomci také přeberou nalezené cesty z rodiče, ale aktualizují cestu agenta, kterému bylo přidáno omezení. Nižší úroveň algoritmu, kterou může být A*, tak navíc kontroluje omezení dané vyšší úrovní a nepřidává do množiny otevřených vrcholů ty, ve kterých se agent nesmí vyskytovat. Pokud byla úspěšně nalezena aktualizovaná cesta, která respektuje nové omezení, je potomek vložen do fronty `open`.

Algoritmus CBS je úplný a optimální, jak je dokázáno v [9].

```
Input: instance of MAPF
Result: set of non-conflicting paths, one for each agent
1  open ← empty priority queue;
2  root ← new node;
3  root.constraints ← ∅;
4  root.solution ← find paths for all agents by lowLevelSearch();
5  root.cost ← costFunction(root.solution);
6  insert root to open;
7  while open is not empty do
8      current ← node with lowest cost from open;
9      remove current from open;
10     validate paths in current.solution until a conflict is found;
11     if no conflict was found then
12         return current.solution;
13     end
14     conflict ← first found conflict in current.solution;
15     foreach agent a_i in conflict do
16         next ← new node;
17         next.constraints ← current.constraints + (a_i, v, t);
18         next.solution ← current.solution;
19         update path of agent a_i in next.solution by
            lowLevelSearch(a_i);
20         next.cost ← costFunction(next.solution);
21         if updated path of agent a_i was found then
22             insert next to open;
23         end
24     end
25 end
```

Pseudokód 2.2: Vyšší úroveň CBS

2.3 Přiřazování cílů

Úloha přiřazení cílů sama o sobě odpovídá přiřazovacímu problému [17] z oblasti kombinatorické optimalizace. Je dáno n agentů (pracovníků) a n cílů (úkolů) spolu s funkcí, která každé dvojici agent-cíl dává číslo vyjadřující cenu tohoto přiřazení. Cena může reprezentovat např. obtížnost, čas, který agent potřebuje na to dorazit do daného cíle, či jiné náklady. Úkolem je přiřadit každému agentovi cíl a zároveň minimalizovat celkovou cenu.

Ekvivalentní problém lze v pojmech teorie grafů vyjádřit jako hledání maximálního párování v ohodnoceném bipartitním grafu tak, aby součet vah hran byl minimální.

V kontextu plánování cest je v [18] definována úloha rozšiřující MAPF – přiřazování cílů a hledání cest (TAPF, *target assignment and path finding*). V ní je třeba brát ohled na to, do kterých cílů mají agenti povoleno dorazit, a pokud mají více možností, přiřadit jim jeden konkrétní.

Formálně lze vyjít z definice MAPF – je zadán graf $G = (V, E)$, množina n agentů $A = \{a_1, ..., a_n\}$ a zobrazení $s : A \to V$ zadávající agentům startovní pozice. V TAPF jsou navíc agenti rozděleni do k disjunktních týmů, $t_1, ..., t_k$. Tým t_i je tvořen n_i agenty, každý agent patří do právě jednoho týmu. Každému týmu je přiděleno n_i cílových vrcholů. Přiřazení cílů agentům je prosté zobrazení φ_i, které je dáno permutací $1 \dots n_i$ a přiřazuje každému agentovi z týmu t_i jeden z cílových vrcholů tohoto týmu.

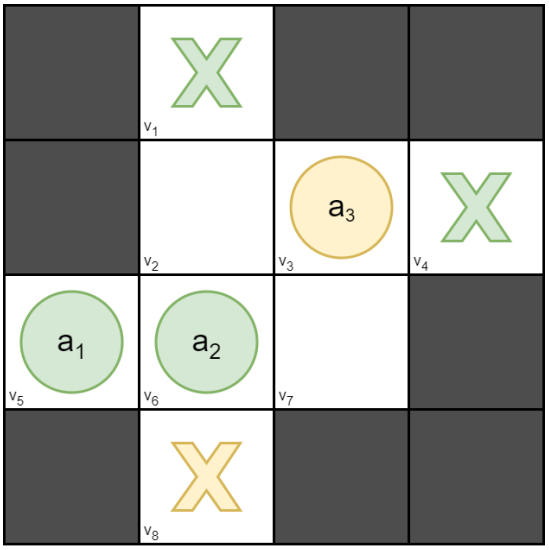
Součástí procesu řešení TAPF je nalezení vhodného přiřazení cílů, které spolu s nekolidujícími cestami agentů minimalizuje použitou nákladovou funkci. Přiřazení není nutno v řešení explicitně uvést, protože je zřejmé z cest agentů.

TAPF je zobecněním obou MAPF variant. Když je v každém týmu jediný agent a tedy počet týmů je stejný jako počet agentů, $k = n$, jedná se o klasické (neanonymní) MAPF, kde má každý agent přidělen svůj cíl. Pokud naopak všichni agenti patří do jediného týmu, $k = 1$, jde o anonymní MAPF, kde můžou agenti dorazit do kteréhokoli z cílů. TAPF navíc popisuje i situace mezi těmito extrémy, kdy $1 < k < n$ a agenti v rámci jednoho týmu jsou zaměnitelní, ale agenti z různých týmů nejsou.

2.3.1 Příklad TAPF

Na obrázku 2.2 je znázorněna instance TAPF. V úloze jsou tři agenti, a_1 a a_2 v zeleném týmu a a_3 ve žlutém. Každý tým má odpovídající barvou vyznačeny své cílové pozice. Agent a_3 je v týmu sám a musí tak obsadit jediný týmový cíl, agenti zeleného týmu mají na výběr dvě možná přiřazení cílových vrcholů.

Úloha má dvě možná řešení: $\{C_1 = (v_6, v_2, v_1), C_2 = (v_2, v_3, v_4), C_3 = (v_7, v_6, v_8)\}$ a $\{C_1 = (v_6, v_2, v_3, v_4), C_2 = (v_2, v_1), C_3 = (v_7, v_6, v_8)\}$. Optimálním je první z nich, jehož *makespan* je nižší, agenti budou ve svých cílech v čase $t = 3$.



Obrázek 2.2 Instance TAPF

2.3.2 Existující algoritmy

Přiřazovací problém lze řešit např. maďarskou metodou (algoritmem) [17]. Situaci můžeme zapsat do $n \times n$ matice, kde v polích jsou uvedeny ceny odpovídajících dvojic agent-cíl. Pokud je v počtech agentů a cílů nepoměr, můžeme méně početnou skupinu doplnit na stejný počet falešnými a nastavit všechny jejich ceny na 0.

Metoda je postavena na myšlence, že pokud přičteme (či odečteme) k jednomu řádku či sloupci stejné číslo, optimální přiřazení bude stejné jako pro původní hodnoty. Algoritmus se tak snaží na pozicích matice vytvořit nuly do té doby, než jich lze vybrat n tak, že každá je v jiném řádku i sloupci. Pozice nul pak určují optimální přiřazení agentů k cílům.

Lze tedy přiřazení provést zvlášť jednorázově a dále řešit úlohu jako klasické MAPF. Jiným způsobem je použít algoritmy jako např. CBM, které obě části řeší najednou, či CBS-TA, které zkouší více možností přiřazení.

2.3.2.1 CBM

CBM (*Conflict-Based Min-Cost-Flow*) [18] je hierarchický algoritmus o dvou úrovních pro TAPF, který využívá myšlenek algoritmů anonymního i klasického MAPF. Optimálního řešení je dosaženo díky tomu, že přiřazení cílů a plánování cest není rozděleno do dvou fází, ale probíhá zároveň. Používanou nákladovou funkcí je *makespan*.

Jako vyšší vrstva je použit algoritmus CBS. Ten je mírně upraven tak, že se stará o vyřešení konfliktů mezi různými týmy (namísto jednotlivých agentů). Nižší vrstvou, které zbývá na starost nalezení cest včetně přiřazení cílů a také vyhýbání se konfliktům v rámci týmu, je algoritmus pro nalezení maximálního toku v síti. Nápad pochází z [3], kde je ukázáno, že MAPF koresponduje s problémy toků v sítích, včetně způsobu, jakým jej na tento problém převést.

Původní graf TAPF úlohy je potřeba převést na tzv. časovou síť o T krocích. Vrcholy i hrany v síti mají kapacity o velikosti 1, které udávají, jak velký tok jimi může procházet. Ze zdrojů (startovních pozic agentů) vytéká tok o velikosti 1 a do stoků (cílových pozic) musí stejně velký tok dotéct.

První časový krok je vytvořen následovně. Každý vrchol $v \in V$ původního grafu je rozdělen na dva, výstupní v_0^{out} a vstupní v_1^{in}. Mezi nimi vede ve stejném směru hrana (v_0^{out}, v_1^{in}) (symbolizuje čekání agenta na stejném místě mezi časovými kroky). Každá hrana $(u, v) \in E$ původního grafu je reprezentována dvěma pomocnými vrcholy w, w' a je zapojena mezi vstupní a výstupní vrcholy pomocí hran $(u_0^{out}, w), (v_0^{out}, w), (w, w'), (w', u_1^{in}), (w', v_1^{in})$. To zabraňuje hranovým konfliktům, protože mezi (w, w') může vést pouze tok o velikosti 1 a tedy i mezi původními (u, v) může projít pouze jeden agent.

Tato síť jednoho časového kroku je za sebou T-krát zřetězena pomocí hran (v_t^{in}, v_t^{out}) mezi vstupními a výstupními vrcholy, což zabraňuje vrcholovým kolizím (jen jeden agent může stát na daném místě v čase t). Pokud je v takové síti nalezen tok ze startovních do cílových vrcholů, získáváme jak přiřazení agentů k cílům, tak rovnou i cesty do nich.

Omezení vyšší vrstvy zakazující výskyt agentů v určitých vrcholech či přechod po určitých hranách jsou reflektována odebráním odpovídajících hran v časové síti.

Nižší vrstva CBM iterativně zkouší, zda existuje maximální tok o velikosti n_i (počet agentů v týmu) jednotek v časové síti o T krocích, kde T se postupně zvyšuje. T může začít na hodnotě *makespanu* rodičovského vrcholu ve vyšší CBS vrstvě, protože nové řešení v aktuálním CBS vrcholu bude nejméně stejně tak nákladné. Pokud řešení není nalezeno do dané horní hranice T, je vrácen neúspěch.

Pro snížení počtu konfliktů mezi týmy ve vyšší vrstvě je možno navíc zavést vylepšení v nižší vrstvě, které hranám v časové síti přidává cenové ohodnocení. Ceny začínají na 0 a jsou zvýšeny těm hranám, které odpovídají cestám agentů z jiného týmu. Tím jsou tyto pozice pro aktuální tým znevýhodněny (ale ne úplně zakázány). Algoritmus nižší vrstvy potom hledá maximální tok o nejnižší ceně.

2.3.2.2 CBS-TA

CBS-TA (*Conflict-Based Search–Task Assignment*) [19] rozšiřuje klasické CBS tak, že prohledává více stromů omezení, každý s jiným přiřazením agentů k cílům.

Kromě grafu G, množiny n agentů $\{a_1, \ldots, a_n\}$, jejich startovních pozic a množiny m cílových pozic $\{g_1, \ldots, g_m\}$ je vstupem do algoritmu také binární matice o rozměru $n \times m$, jejíž prvky a_{ij} vyjadřují, zda agent a_i může dostat přiřazen cíl g_j. Pomocí ní můžeme popsat různé situace v TAPF, navíc je možno popsat také situace s více agenty než cíli nebo naopak.

Vrcholy stromu konfliktů oproti běžnému CBS navíc nesou informaci o aktuálním přiřazení, pro které se snažíme najít řešení. Při vytváření prvního kořene je nutno najít nejlepší přiřazení. Dle binární matice je vytvořena ma-

tice cen: pokud $a_{ij} = 1$, najdeme nejkratší cestu agenta a_i k cíli g_j (nehledě na ostatní agenty) a použijeme její cenu, pokud $a_{ij} = 0$, nastavíme odpovídající prvek v matici cen na nekonečno. Tato matice cen je poté použita v algoritmu pro nalezení optimálního přiřazení (lze použít např. maďarská metoda). Výsledek je uložen do množiny přiřazení a použit v prvním kořeni.

Jakmile je vyšší úrovní zpracováván kořen stromu konfliktů, vytvoří se kořen nového stromu konfliktů s jiným přiřazením. Další nejlepší přiřazení je získáno z nejlevnějšího z množiny přiřazení systematickým zakazováním určitých dvojic a naopak vynucováním jiných. Tímto způsobem můžeme dostat i více různých přiřazení, které všechny uložíme v množině přiřazení a do nového kořene použijeme to s nejnižší cenou.

Dále algoritmus pracuje jako klasické CBS – hledají se cesty agentů do cílů daných přiřazením, při konfliktních cestách se vrcholy stromu konfliktů rozdělují na dva potomky. Teprve až v aktuálním stromu nelze dojít k řešení, algoritmus přeskočí do dalšího kořene v pořadí a zkouší tak hledat cesty pro jiné přiřazení.

2.4 Seřazování cílů

Definice MAPF a TAPF počítají s jedním cílem pro každého agenta. Nepostihují tak třídu problémů, kde agenti mají více cílových lokací, které musí postupně navštívit. Tento nedostatek doplňují následující rozšíření.

G-TAPF (*generalized TAPF*) [20] dále zobecňuje dříve zmíněné TAPF. Popisuje řešený problém jako trojici (G, R, T). Graf $G = (V, E)$ stejně jako v předchozích formulacích popisuje vrcholy a přechody mezi nimi. R je množina agentů, každý agent je rozlišen dvojicí (s, t) vyjadřující startovní pozici $s \in V$ a typ cílů resp. úkolů, které může navštívit resp. vykonat. T je množina skupin cílů, skupiny se skládají ze seznamu cílů a deadlinu vyjadřující čas, do kdy musí být všechny cíle dané skupiny splněny. Cíle jsou popsány svou pozicí $g \in V$ a typem t.

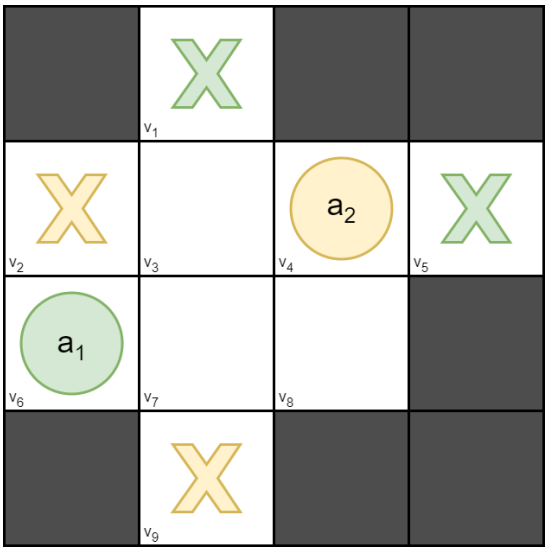
Týmy agentů z TAPF tak můžeme rozlišit pomocí typů, navíc zde není žádný limit na počet cílů daného typu. S touto formulací lze modelovat nové situace jako například pořadí skupin, kdy je nejprve potřeba navštívit všechny cíle jedné skupiny před započetím další, časové limity na skupiny cílů či checkpointy resp. pořadí jednotlivých cílů, kdy před finální destinací je nutno navštívit jiné.

MG-MAPF (*multi-goal multi-agent path finding*) [21] je přímočařejším rozšířením MAPF, které oproti definici uvedené v kapitole 2.1 mění pouze zobrazení zadávající cílový vrchol na zobrazení $g : A \to 2^V$. Oborem hodnot je nyní potenční množina množiny vrcholů, tedy je možno agentovi přiřadit jako jeho cíle libovolný počet a libovolnou kombinaci vrcholů.

V úlohách, kde mají agenti zadáno více cílů, dává smysl řešit jejich seřazení, a to i pokud nejsou specifikovány vnější požadavky na jejich pořadí. Například může být vhodné cíle seřadit tak, aby bylo co nejméně nutné vracet po již jednou navštívených částech mapy, případně aby části, po kterých se vrátit musíme, byly co nejkratší.

2.4.1 Příklad MG-MAPF

Ukázková instance MG-MAPF je uvedena na obrázku 2.3. V úloze jsou dva agenti, z nichž oba mají zadány své dva cíle. Je nutno rozhodnout, v jakém pořadí je mají obejít.



Obrázek 2.3 Instance MG-MAPF

Optimálním seřazením je (v_1, v_5) pro agenta a_1 a (v_2, v_9) pro agenta a_2, které vede k optimálnímu celkovému řešení např. $\{C_1 = (v_7, v_3, v_1, v_3, v_4, v_5)$, $C_2 = (v_3, v_2, v_3, v_7, v_9)\}$ s dokončením v čase $t = 6$. Opačné seřazení by znamenalo nejkratší možné cesty o délkách $|C_1| = 7$ a $|C_2| = 6$, obě horší o jeden krok.

Pokud by přiřazení cílů agentům nebylo zadáno, bylo by nejlepší jejich množiny cílů prohodit – tak by byli oba blíže své množině a mohli ji tak obejít rychleji.

2.4.2 Existující způsoby řešení

Formulace G-TAPF byla zavedena s myšlenkou použití v ASP (*answer set programming*). [20] Jedná se o typ deklarativního programování, kde je řešená úloha vyjádřena jako logický program. V [22] je zmíněna podobnost se SAT – zatímco SAT pracuje s logickými formulemi, do kterých musí být instance problému zakódována (nejčastěji pomocí jiného programu), ASP problém popisuje pomocí kolekce pravidel. V obou přístupech je pak reprezentace úlohy předána příslušnému řešiči, který v případě SAT nalezne ohodnocení, pro která je formule splnitelná, a v případě ASP nalezne „množiny odpovědí“, tedy řešení úlohy.

2.4.2.1 HCBS

HCBS (*Hamiltonian CBS*) [21] je rozšířením algoritmu CBS pro MG-MAPF. Protože agenti musí navštívit více cílů, algoritmus hledá nejkratší hamiltonovské cesty. Hamiltonovská cesta pokrývající podmnožinu $U \subseteq V$ vrcholů grafu je definována jako taková posloupnost vrcholů, která obsahuje každý z vrcholů $u \in U$ (a může obsahovat další vrcholy nepatřící do U).

Běžné CBS je algoritmus o dvou vrstvách. Vyšší vrstva HCBS funguje stejně jako v CBS, ale k nižší vrstvě, která hledá nejkratší cestu do jednoho konkrétního cílového vrcholu, HCBS přidává nádstavbu, která má na starost seřazení množiny cílů. To je provedeno opět algoritmem A*, který v této části prohledává stavový prostor možných permutací cílů daného agenta.

Heuristikou pro tento A* je cena minimální kostry pokrývající množinu cílových vrcholů. Ta je spodním odhadem pro délku hamiltonovské cesty pokrývající tuto stejnou množinu. Kostrou je myšlen podgraf původního grafu G, který je stromem s kořenem v aktuálním vrcholu a obsahuje všechny cílové vrcholy z množiny. Cenou kostry grafu myslíme počet hran v tomto stromě. Pro nalezení minimální kostry grafu lze použít např. Borůvkův nebo Jarníkův algoritmus.

Hledání vhodné permutace začíná ve stavu se startovní pozicí agenta a žádnými pokrytými cíli. Stav je vložen do prioritní fronty s prioritou $f(n) = g(n) + h(n)$, kde $g(n)$ je délka již nalezené částečné cesty a $h(n)$ zmíněná heuristika odhadující délku zbývající cesty. V každé iteraci je vyzvednut nejlevnější vrchol z prioritní fronty. Pro každý ještě nenavštívený cíl je nejnižší vrstvou (také stejná jako v běžném CBS) nalezena cesta k němu s ohledem na omezení daná nejvyšší vrstvou. Vytvoří se nový stav s aktuální pozicí v tomto cíli, nalezenou cestou a množinou ještě nenavštívených cílů, je mu vypočtena priorita a je vložen do fronty. Hledání končí, jakmile je vyzvednut stav se všemi cíli již navštívenými – ukládané částečné cesty společně tvoří cestu celkovou.

2.4.2.2 Aproximační algoritmus pro TSP

Na seřazení cílů můžeme také nahlížet jako na problém obchodního cestujícího (TSP, *travelling salesman problem*). [23] V TSP je zadán graf, jehož vrcholy reprezentují města a hrany cesty mezi nimi. Hrany jsou nezáporně ohodnocené – cesty mají délku. Řešíme otázku, jakou nejkratší cestou je možné objet všechna města právě jednou a vrátit se do počátečního místa, což odpovídá hledání nejkratší hamiltonovské kružnice (kružnice je cesta v grafu, která začíná i končí ve stejném vrcholu). Metrický TSP je pak varianta, ve které navíc pro vzdálenosti v grafu platí trojúhelníková nerovnost.

TSP je NP-těžký problém kombinatorické optimalizace, pro který není znám algoritmus, který by jej vyřešil v polynomiálním čase. Pro metrickou variantu však existuje 2-aproximační algoritmus [23], který najde kružnici nejhůře dvakrát delší, než je ta optimální.

Nejdříve je potřeba najít minimální kostru grafu. Počáteční vrchol hledané kružnice zvolíme jako kořen kostry. Z kořene kostru projdeme pomocí prohledávání do hloubky (DFS, *depth-first search*) a při průchodu si ukládáme

pořadí navštěvovaných vrcholů – tento krok je někdy ekvivalentně popisován jako zdvojení hran kostry a nalezení eulerovského tahu. Během prohledávání projdeme každou hranu dvakrát, jednou při vstupu do potomka aktuálního vrcholu a podruhé při návratu ze zanoření.

V získané posloupnosti se opakují vrcholy, proto není hamiltonovskou kružnicí. Posledním krokem je tak její zkrácení – z posloupnosti vynecháme opakované návštěvy vrcholů. Tím se z každého vrcholu přesuneme vždy do dalšího neprozkoumaného a trojúhelníková nerovnost zaručuje, že tato cesta nebude horší než případná původní cesta přes již navštívené vrcholy. Na konec posloupnosti přidáme počáteční vrchol a tato již je hledanou hamiltonovskou kružnicí.

Kapitola 3
Analýza

V této kapitole budou uvedeny základní informace o originální hře Pacman včetně jejích pravidel, detailů herní mapy a podrobností ohledně pohybu Pacmanů a duchů. Následně bude popsán Multi-Pacman – zobecněná varianta řešená v této práci.

3.1 Hra Pacman

Pacman je dodnes známá původně arkádová hra, kterou vyvinul v 80. letech 20. století japonský vývojář Tóru Iwatani pod společností Namco. Článek [24] detailně popisuje nejen samotnou hru, ale také myšlenku za jejím zrodem, její vývoj, zajímavosti a možné strategie.

Hráč se pohybuje s postavou Pacmana (žlutým kruhem s výsečí znázorňující ústa) a má za úkol sníst všechny kuličky v herním bludišti. V tom se mu snaží zabránit čtyři různobarevní duchové, se kterými se Pacman nesmí srazit. Pokud se mu povede vysbírat všechny kuličky, postupuje do další úrovně, kterých je v původní hře 256. Pokud se s duchem srazí, ztrácí jeden život a Pacman i duchové jsou navráceni do původních pozic, úroveň však pokračuje dále. Hra končí, pokud hráč ztratí všechny životy.

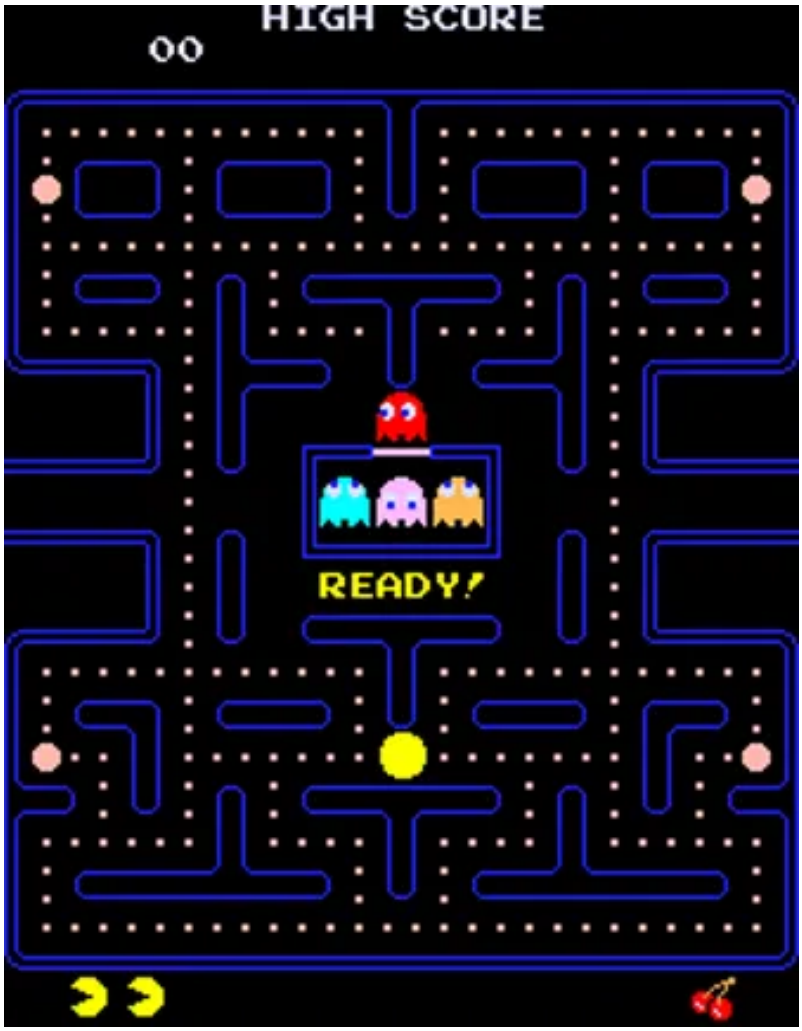
Herní bludiště je mřížka o velikosti 31×28 polí. Pole mohou být buď nepřístupné (zdi) nebo přístupné. Pouze po přístupných polích se lze pohybovat a jen v nich se mohou vyskytovat kuličky. Rozmístění zdí tvoří z přístupných polí úzké chodby, které nikdy nejsou slepé. Uprostřed mapy není levý a pravý okraj jedné chodby ohraničen zdí – tvoří tak tunel, kterým je možno projít a objevit se na opačné straně. Nad bludištěm je zobrazeno aktuální skóre, pod ním zbývající životy.

Barevně rozlišení duchové mají každý svou strategii pro polapení Pacmana. Červený duch Pacmana přímo pronásleduje, růžový se ho snaží překvapit nadběhnutím, světle modrý duch je nepředvídatelný a oranžový Pacmana sleduje stejně jako červený, pokud se Pacman nachází daleko, jakmile se však přiblíží, oranžový duch „ztratí zájem“ a Pacmanovi se vyhýbá.

Zároveň duchové střídají tři módy chování. Popsané strategie platí v pro-

následovacím (*chase*) módu, který převažuje po většinu hry. V definovaných intervalech se střídá s módem rozptýlení (*scatter*), který trvá pár sekund a při kterém duchové přestanou cílit na Pacmana a pohybují se každý k jednomu okraji mapy. Vystrašený (*frightened*) mód může vyvolat hráč sebráním velké energetické kuličky, která Pacmana posílí a duchové se na několik vteřin stanou zranitelnými. Během této doby se s nimi Pacman může srazit a tím ducha vrátit na startovní pozici, čímž jej na chvíli vyřadí ze hry. Změnu módu lze poznat podle toho, že duchové změní svůj směr – jindy směr změnit nemohou, vždy pokračují tím, který si zvolili na křižovatce.

Zajímavostí je, že v poslední 256. úrovni je pravá polovina obrazovky zaplněna náhodnými symboly, které jsou způsobeny přetečením osmibitové proměnné pro úroveň – nejvyšší reprezentovatelné číslo v 8 bitech je 255. Tuto úroveň tak nelze dohrát.



Obrázek 3.1 Originální mapa hry Pacman [24]

3.2 Zobecnění hry Pacman pro účely práce

V práci je řešena zobecněná verze hry Pacman nazvaná Multi-Pacman, která se od původní hry liší v několika aspektech. Stěžejní změnou je možnost více Pacmanů v herním bludišti. To úlohu povyšuje na multi-agentní problém, ve kterém Pacmani na sesbírání kuliček spolupracují. Kromě kolizí s duchy je nutné se také vyhýbat kolizím se sebou navzájem. Navíc je potřeba rozhodnout, jakou část mapy přidělit kterému Pacmanovi, aby spolupráce mohla probíhat současně a efektivně. V klasické verzi by stačilo řešit pouze seřazení kuliček a hledání cest od jedné k další pro jednoho agenta.

Další změnou je také volitelný počet duchů a jejich chování. Duchové představují pohyblivé překážky, které v každém časovém kroku blokují jiný vrchol, jejich větší množství tak znamená více omezení pro hledané cesty Pacmanů. Přestože má v původní hře každý z duchů vlastní chování a strategii pro odchycení Pacmana, všichni se pohybují (v pronásledovacím módu) deterministicky. Je tak možno předvídat jejich přesné pozice a rovnou se jim při plánování vyhýbat. Z tohoto důvodu jsou v Multi-Pacmanovi odebrány individuální strategie duchů a namísto nich se všichni duchové chovají náhodně.

Tato náhodnost spočívá ve výběru směru, kam se duch dále vydá, pomocí generátoru náhodných čísel. Je zachována vlastnost, že duchové mění směr pouze na křižovatkách (nemohou jej tedy změnit uprostřed chodby). Také je zachováno, že se nemohou vrátit stejným směrem, ze kterého přišli (jediný případ, kdy je tomu jinak, nastává, když mapa obsahuje slepé chodby – pokud se v ní duch ocitne, nezbývá mu jiná možnost, než se vrátit; v originálních mapách hry se slepé chodby nevyskytují). Na křižovatce se třemi resp. čtyřmi východy se tak rozhodují ze dvou resp. tří možností.

Poslední změnou je odebrání speciálních energetických kuliček. Účel této funkcionality je spíše herní, umožňuje hráči interagovat s nepřítelem. Není důležitá pro řešení Multi-Pacmana jakožto multi-agentní úlohy a v případě potřeby je lehce implementovatelná.

Kapitola 4

Návrh řešení a implementace

V této kapitole bude popsána struktura implementovaného programu a následně návrh algoritmu, který řeší Multi-Pacmana. Jednotlivé části algoritmu budou popsány pseudokódy a důkladně okomentovány s poukázáním na změny oproti původním algoritmům.

4.1 Struktura programu

Hlavní třídou programu je třída `Solver`, ve které je uložena reprezentace zadané instance problému a také mezivýsledky jednotlivých částí algoritmu a nastavení běhu programu (vizualizace, debug mód atd.). Dále jsou v ní implementovány metody navrhovaného algoritmu, které tak mají k reprezentaci problému přístup.

Herní mapa je v programu reprezentována třídou `GameMap`, která si pamatuje rozměry, celkový počet kuliček a jednotlivá políčka mapy. Pole `Tile` zná své souřadnice, zda je na něm kulička k sesbírání a který Pacman či duch na něm aktuálně stojí. `GameMap` slouží k záznamu aktuálního stavu polí při běhu algoritmu (např. kterému Pacmanovi je již dané pole přiřazeno) a také je využita pro vizualizaci mapy při skutečném vykonávání plánů.

Třída `Graph` popisuje topologii herní mapy jakožto grafu. Uchovává (průchozí) vrcholy `Node`, které jsou jednoznačně určeny souřadnicemi příslušného pole. Hrany v grafovém vyjádření herní mapy jsou neorientované a všechny mají stejnou délku, poskytují tak pouze informaci o tom, které vrcholy spolu sousedí. Z toho důvodu je namísto hran rovnou ukládána množina sousedů každého vrcholu.

Vzhledem k charakteru originálních map hry, které jsou tvořeny převážně úzkými chodbami a neobsahují téměř žádné otevřené oblasti, je výhodné si graf uložit také na vyšší úrovni abstrakce. Do té jsou jako vrcholy brány pouze takové vrcholy původního grafu, které mají 1, 3 nebo 4 sousedy, tj. takové, které končí slepou uličku nebo jsou křižovatkami. V algoritmech používajících tuto abstrakci stačí provádět rozhodování, kterým směrem se vydat, pouze v těchto hlavních vrcholech a v ostatních jen následovat daný směr, dokud nedorazíme

do dalšího hlavního vrcholu. Hrany abstrakce spojují hlavní vrcholy a jejich délky jsou různé. Opět jsou ukládány jako množina sousedů každého hlavního vrcholu a navíc je ke každému sousedovi uložena cesta, jak se k němu v původním grafu dostat.

Program tak využívá hierarchickou reprezentaci grafu – v místech, kde není nutno brát v potaz každý vrchol, se pro rychlejší navigaci grafem používá abstrakce a skutečná verze se použije teprve když je potřeba přesných cest.

Třídy `Pacman` a `Ghost` modelují entity figurující v úloze. Obě si pamatují svůj identifikátor a aktuální vrchol, na kterém stojí. Pacmani navíc udržují jim přiřazenou oblast, seřazené cíle a naplánovanou cestu, duchové pak směr, kterým se pohybují.

Dále se v programu vyskytují pomocné třídy, které jsou popsány v části, ve které figurují.

4.2 Vstup

Vstupem programu je popis problému v textovém souboru, jehož cesta je zadána jako argument při spouštění z příkazové řádky. Načtení souboru a inicializaci potřebných tříd má na starost metoda `init()` třídy `Solver`.

Formát vstupního souboru je následující: na první řádce jsou dvě celá čísla vyjadřující počet řádků a sloupců herní mapy. Za nimi následuje samotná mapa. Neprůchozí pole (zdi) jsou značeny symbolem `#`, všechna ostatní pole jsou průchozí. Pokud na poli startuje Pacman, je zakreslen jako `P`, kuličky k sesbírání jsou značeny jako `.` (tečka), pro prázdná pole je použita obyčejná mezera. Po mapě následuje číslo vyjadřující počet duchů a za ním tento počet řádků s jejich pozicemi. Pozice duchů jsou na rozdíl od Pacmanů uvedeny souřadnicemi, protože duchové se mohou vyskytovat na stejných místech jako kuličky (Pacmani vždy startují na poli bez kuličky). Souřadnice polí jsou číslovány od nuly v obou směrech, tj. první pole vlevo nahoře má souřadnice [0, 0], kde první značí řádek a druhá sloupec, ve kterém se pole nachází.

Duchové nesmí začínat na stejné pozici jako některý z Pacmanů, to by znamenalo kolizi ještě před spuštěním algoritmu. Jako nevalidní vstup jsou dále považovány takové mapy, ve kterých některé kuličky nejsou dosažitelné (tj. jsou úplně odděleny zdmi), protože koncovou podmínkou algoritmu je sesbírání všech cílů.

Nepovinným vstupem je dále soubor s nastavením programu, pomocí kterého je možno ovládat např. vizualizaci nebo která varianta grafu se má použít. Pokud není poskytnut, použije se nastavení výchozí.

4.3 Návrh algoritmu

Hlavní myšlenkou celkového algoritmu je rozdělit jej do tří částí, které na sebe navazují a vždy výsledek předchozí části je součástí vstupu do části následující. Nejprve je potřeba vyřešit rozdělení mapy na menší oblasti pro každého Pacmana. Dalším krokem je získat posloupnost kuliček vyskytujících se na přidělených oblastech, která vyjadřuje, v jakém pořadí bude Pacman své cíle

navštěvovat. Poslední část algoritmu se stará o koordinovaný pohyb bludištěm, tedy o hledání cest mezi cíli a vyhýbání se kolizím.

Navrhovaný algoritmus je suboptimální, protože řeší jednotlivé části postupně a zvlášť. Pro optimální řešení by bylo nutno vzít v potaz, že spolu podúlohy souvisí a tak např. i horší přiřazení cílů (z pohledu funkce hodnotící kvalitu) může vést k celkově lepšímu řešení, pokud při takovém přiřazení nastává méně kolizí při plánování. Bylo by tak nutné systematicky prozkoumávat různé kombinace přiřazení, seřazení a plánování. Suboptimální přístup byl zvolen také proto, že v plánovací fázi náhodný pohyb duchů brání nalezení kompletních cest, je potřeba střídat hledání cest částečných a vykonávání plánu (viz dále). Není tak možné znát celkové délky cest ještě před započetím pohybu Pacmanů.

Nejvyšší vrstva algoritmu je uvedena v pseudokódu 4.1. Je spuštěna zavoláním metody `solve()` na instanci `Solver`.

```
   Input: instance of Multi-Pacman
   Result: set of paths for each agent
 1 solve():
 2 |  assignArea();
 3 |  orderGoals();
 4 |  while gameMap.foodCount ≠ 0 do
 5 |  |  solution ← planPaths();
 6 |  |  if solution.type = impossible then
 7 |  |  |  return solution not found;
 8 |  |  else if solution.type = partial then
 9 |  |  |  add first solution.validUntil nodes from paths from
   |  |  |    solution to agents' plans;
10 |  |  else
11 |  |  |  add full paths from solution to agents' plans;
12 |  |  end
13 |  |  execute agents' plans;
14 |  end
15 |  return full paths traveled by agents;
```

Pseudokód 4.1: Hlavní úroveň algoritmu

4.3.1 Rozdělení mapy a přiřazení cílů

Algoritmy pro přiřazení cílů z teoretické části všechny předpokládají přidělení jednoho cíle každému agentovi, v řešené úloze je však potřeba přidělit Pacmanům množinu kuliček. Podúloha je tak řešena z pohledu rozdělení herní mapy na menší oblasti pro každého Pacmana. Chtěli bychom, aby měli s procházením oblastí přibližně stejnou práci, tedy aby oblasti byly přibližně stejně rozlehlé. Dalším předpokladem je, že startovní vrchol Pacmana patří do jeho oblasti a ta se dále rozrůstá z tohoto vrcholu. Pokud by tomu tak nebylo, bylo by při plánování cest nejprve nutné Pacmany dovést do nejbližšího vrcholu přidělené části, což by jistě prodloužilo celkové cesty.

První variantou řešení je algoritmus prohledávání do šířky (BFS, *breadth-first search*) modifikovaný tak, aby prohledával z více zdrojů najednou. Je popsán v pseudokódu 4.2. Každý Pacman má svou vlastní frontu otevřených vrcholů, díky čemuž je možné mezi frontami iterovat a oblasti rozšiřovat stejnou rychlostí pro všechny. Při použití společné fronty by se rychleji expandovaly oblasti, které se větví do více směrů. Tato varianta pracuje na klasické reprezentaci grafu.

```
Input: instance of Multi-Pacman
Result: disjoint areas of game map, one for each agent
assignArea():
    queues ← map of individual queues for each agent;
    closed ← empty set of visited nodes;
    foreach (id, pacman) in Pacmans do
        insert pacman.position into queues[id];
    end
    while all queues aren't empty do
        foreach (id, queue) in queues do
            if queue is empty then continue;
            current ← front of queue;
            remove current from queue;
            neighbors ← set of nodes accessible from current;
            foreach next in neighbors do
                if next not in closed and next hasn't been assigned to
                 agent's area then
                    insert next into queue;
                    insert next into Pacmans[id].assignedArea;
                end
            end
            insert current into closed;
        end
    end
```

Pseudokód 4.2: Rozdělení mapy (BFS verze)

Výhodou tohoto přístupu je, že oblasti jsou disjunktní a nezpůsobují tak už předem možné konflikty v plánovací části. Nevýhodou je fakt, že pro rovnoměrné rozdělení je nutné, aby už startovní pozice Pacmanů byly zvoleny s podobnými vzdálenostmi mezi sebou. Oblasti se totiž rozšiřují do té doby, než narazí na jinou – v extrémním případě, kdy by Pacman startoval na pozici obklíčen jinými, by jeho oblast rozšiřována vůbec nebyla.

Druhá varianta se snaží tuto nevýhodu eliminovat. Používá upravený algoritmus DFS operující na abstraktní reprezentaci grafu, který je také spuštěn z více zdrojů najednou, a vlastní pravidla pro to, kterým směrem se v rozhodujících vrcholech vydat. Je popsán pseudokódem 4.3.

V této variantě jsou oblasti rozšiřovány po větších částech (celých chodbách mezi dvěma rozhodujícími vrcholy), proto je použita prioritní fronta,

kde priorita vyjadřuje velikost oblasti přiřazené Pacmanovi. Oblasti a zásobníky jsou inicializovány startovním vrcholem agentů (řádky 4–8). Zásobníky jsou použity spíše jako „historie“ zanoření do rozhodujících vrcholů než jako zásobníky v běžném DFS, protože na ně nejsou ukládáni všichni sousedé.

V cyklu algoritmu je nejdříve z prioritní fronty vyzvednut Pacman s aktuálně nejmenší přiřazenou oblastí, z jeho zásobníku je pak vyzvednut aktuální vrchol, ve kterém se algoritmus nachází. Sousední vrcholy v abstraktním grafu jsou možnosti, do kterých lze oblast dále rozšířit (řádky 10–14).

```
   Input: instance of Multi-Pacman
   Result: areas of game map, one for each agent
 1 assignArea():
 2 |  pq ← empty priority queue;
 3 |  stacks ← map of individual stacks for each agent;
 4 |  foreach (id, pacman) in Pacmans do
 5 |  |  insert [0, id] into pq;
 6 |  |  insert pacman.position into stacks[id];
 7 |  |  insert pacman.position into Pacmans[id].assignedArea;
 8 |  end
 9 |  assignedFoodCount ← 0;
10 |  while assignedFoodCount ≠ gameMap.foodCount do
11 |  |  (priority, id) ← node from pq with lowest priority;
12 |  |  remove (priority, id) from pq;
13 |  |  current ← front of stacks[id];
14 |  |  options ← set of higherGraph nodes accessible from current;
15 |  |  bestOption ← getBestOption(id, current, options);
16 |  |  path ← path between current and bestOption;
17 |  |  if hasUnclaimedFood(path) then
18 |  |  |  insert bestOption to stacks[id];
19 |  |  |  claimPath(path, id);
20 |  |  |  insert (priority + path size, id) into pq;
21 |  |  else if checkStack(stacks[id]) then
22 |  |  |  remove top of stacks[id];
23 |  |  |  insert (priority, id) into pq;
24 |  |  else if notAssignedTo(path, id) then
25 |  |  |  insert bestOption to stacks[id];
26 |  |  |  claimPath(path, id);
27 |  |  |  insert (priority + path size, id) into pq;
28 |  |  else
29 |  |  |  remove top of stacks[id];
30 |  |  |  if stacks[id] not empty then
31 |  |  |  |  insert (priority, id) into pq;
32 |  |  |  end
33 |  |  end
34 |  end
```

Pseudokód 4.3: Rozdělení mapy (DFS verze)

Nejlepší možnost je vybrána seřazením dle následujících pravidel: nejprve jsou preferovány chodby, které ještě nebyly přiřazeny nikomu, a poté chodby, které nebyly přiřazeny aktuálnímu Pacmanovi (ale mohly být přiřazeny jiným). V těchto skupinkách jsou dříve umístěny ty, které obsahují kuličky, více možností je rozděleno dle délky chodby, kde přednost má kratší, a stále nerozhodné seřazení je definitivně rozhodnuto výchozím pořadím, v jakém jsou sousedé uloženi.

Pokud chodba k nejlepší možnosti ještě není přiřazena a jsou v ní zároveň kuličky, je přiřazena aktuálnímu Pacmanovi, vrchol je umístěn na zásobník a Pacman je umístěn do fronty se zvýšenou prioritou, protože má zabranou větší oblast (řádky 17–20). Dokud z expandovaných vrcholů existují nepřiřazené chodby s kuličkami, algoritmus takto rozšiřuje oblasti Pacmanů a zanořuje je hlouběji do DFS průchodu.

Jakmile taková chodba není k dispozici, je zkontrolován zásobník Pacmana, jestli nepřiřazená chodba s kuličkami nevede z nějakého již navštíveného vrcholu. Pokud ano, aktuální vrchol je odebrán ze zásobníku a prohledávání se tak vrátí o jednu pozici zpět (řádky 21–23). Takto se vrací do doby, než se nachází ve vrcholu, odkud vede hledaná chodba.

Pokud ani na zásobníku není nalezen vrchol s vhodným směrem, požadavek na nejlepší možnost je zmírněn a oblast se z aktuálního vrcholu rozšíří chodbou, která ještě není přiřazena aktuálnímu Pacmanovi, ale mohla již být přiřazena jinému nebo neobsahuje kuličky (řádky 24–27). Jestliže ani taková chodba neexistuje, algoritmus se vrací dle zásobníku o pozici zpět (řádek 29). Cyklus je zastaven ve chvíli, kdy jsou rozřazeny všechny vrcholy s kuličkou.

Tato varianta umožňuje přiřadit jeden vrchol více Pacmanům, rozdělené oblasti se tak mohou částečně překrývat. Díky tomu není náchylná na počáteční rozmístění Pacmanů, zato však může způsobit více kolizí v plánovací části, pokud se bude více Pacmanů snažit dostat na stejné místo.

Algoritmus oproti klasickému DFS mění pouze pořadí, v jakém jsou vrcholy expandovány. Část, která seřazuje možnosti dle pravidel, nezávisí na velikosti vstupu, protože možnosti jsou vždy maximálně čtyři (vrchol může mít hrany vedoucí do max. čtyř směrů). Teoretickým nejhorším případem, kdy algoritmus poběží nejdéle, je situace, kdy je každému Pacmanovi přiřazena celá mapa a všechny oblasti se plně překrývají. V takovém případě by běžel stejně dlouho, jako bychom spustili DFS pro každého zvlášť a nechali jím prohledat celou mapu. Tato situace by však nastala pouze ve chvíli, kdy by byla některá kulička nedosažitelná (zcela oddělena zdmi) a algoritmus tak nemohl skončit ve chvíli, kdy jsou všechny kuličky přiřazeny.

Experimentální porovnání obou variant se nachází v kapitole 5.

4.3.2 Seřazení cílů

Seřazení cílů je provedeno pomocí aproximačního algoritmu pro TSP z kapitoly 2.4.2.2, který je použit na oblasti z předchozí části.

Nejprve chceme najít minimální kostru dané oblasti. V úloze používáme neohodnocený graf (resp. si jej lze představit jako ohodnocený, kde všechny

hrany mají stejnou délku 1). Protože kostra grafu je strom pokrývající všechny vrcholy a strom s n vrcholy má vždy $n - 1$ hran [25], má každá kostra stejný počet hran a tedy v neohodnoceném grafu jsou všechny kostry minimální. Stačí tak najít nějakou (z případných více možných) koster.

K tomu je možno použít DFS, které navštíví a uzavře každý vrchol právě jednou, čímž sestrojuje strom pokrývající všechny vrcholy a tedy kostru. Protože dalším krokem aproximačního algoritmu je DFS průchod nalezené kostry a zaznamenávání pořadí navštěvovaných vrcholů, můžeme provést oba kroky najednou.

Posledním krokem je zkrácení posloupnosti vrcholů. Cíle Pacmanů jsou pouze vrcholy s kuličkami (nemusí obejít celou přidělenou oblast), proto jsou z posloupnosti vynechány nejen již navštívené vrcholy, ale také vrcholy bez kuličky. I tento krok lze provést současně při průchodu oblasti, když budeme do posloupnosti ukládat pouze první návštěvy vrcholů s kuličkou. V (Multi-)Pacmanovi není nutno se po sesbírání všech cílů vrátit do počáteční pozice, proto na konec posloupnosti není znovu přidán startovní vrchol.

Pokud je pro přiřazení použit klasický graf, nemáme o oblastech žádnou další informaci a proto je i seřazení prováděno na něm. Jestliže je přiřazení provedeno na abstraktní reprezentaci, oblasti jsou rozšiřovány po celých chodbách mezi dvěma rozhodovacími vrcholy, a tak i seřazení je možno provést průchodem abstraktního grafu oblasti a ve vrcholech využít informaci o délkách cest do následníků. Např. můžeme změnit pořadí expanze vrcholů, které se ve výchozím stavu řídí uložením sousedů, aby se řídilo délkou chodby do následníka. Nejdříve chceme s Pacmanem navštívit nejkratší chodby, protože se po nich případně budeme muset vracet, a po vysbírání nejdelší chodby skončit.

Varianta s hierarchickou reprezentací grafu je uvedena v pseudokódu 4.4. Oproti běžnému DFS je rozdíl v tom, že na zásobník ukládá dvojici aktuálního a předchozího vrcholu (řádek 2), aby bylo možné z abstraktního grafu získat chodbu mezi těmito vrcholy (řádky 8–12) a uzavřít ji spolu s uložením kuliček, na které po cestě natrefíme (řádky 13–21). Sousedé aktuálního vrcholu jsou seřazeni a výsledné pořadí je obráceno (řádky 22–24), protože DFS používá zásobník a tak nejdříve vyzvedne poslední vložený vrchol. První část podmínky na řádku 27 zaručuje, že algoritmus prochází pouze oblast přiřazenou konkrétnímu Pacmanovi v předchozí části.

4.3.3 Hledání cest

V poslední plánovací části je pro hledání nekolidujících cest použit algoritmus CBS s patřičnými modifikacemi.

Jak již bylo naznačeno v pseudokódu 4.1, plánování je z hlavní úrovně voláno opakovaně tak dlouho, dokud nejsou sesbírány všechny kuličky nebo řešení není možné nalézt. Používané CBS může vrátit částečné řešení, což je rozdílné od klasického CBS vracejícího úplné cesty agentů. Je to zapříčiněno možnými kolizemi s duchy, které nejde řešit na vyšší úrovni CBS stejně jako konflikty Pacmanů – nemáme možnost řešení rozdělit na dvě, v jednom zakázat přístup Pacmanovi a v druhém duchovi, protože duchy neovládáme, je potřeba se jim za každou cenu vyhnout.

```
   Input: instance of Multi-Pacman,
          areas of game map assigned to Pacmans
   Result: ordering of goals to visit for each Pacman
 1 orderGoals():
 2     foreach pacman in Pacmans do
 3         stack ← empty stack;
 4         closed ← empty set;
 5         insert (pacman.position, pacman.position) into stack;
 6         while stack not empty do
 7             (current, prev) ← top pair from stack;
 8             remove (current, prev) from stack;
 9             if prev = current then path ← current;
10             else
11                 path ← path from prev to current;
12             end
13             if all nodes from path aren't in closed then
14                 foreach node in path do
15                     if node has food then insert node into
                        pacman.orderedGoals;
16                     insert node into closed;
17                 end
18                 insert current into closed;
19             end
20             neighbors ← neighbors of current in higher graph;
21             sort neighbors by given rules;
22             reverse order of neighbors;
23             foreach next in neighbors do
24                 path ← path from current to next;
25                 if next in pacman.assignedArea and all nodes from
                    path aren't in closed then
26                     insert (next, current) into stack;
27                 end
28             end
29         end
30     end
```

Pseudokód 4.4: Seřazení cílů

Pro vrcholy stromu konfliktů je použita pomocná třída `CBSNode`. Obsahuje sadu omezení platných pro tento vrchol, řešení respektující tato omezení a jeho cenu. Řešení je reprezentováno další pomocnou třídou `Solution`, která uchovává typ řešení (úplné, částečné či nenalezené), do jaké délky cest je validní (využito v případě částečného řešení) a množinu cest pro jednotlivé Pacmany.

Vyšší úroveň CBS není nutné příliš měnit oproti klasické verzi uvedené v pseudokódu 2.2. Jediným doplněním je přidání kontroly mezi řádky 9–10, jestli je řešení ve vyzvednutém vrcholu typu nenalezeno. Pokud ano, nižší úroveň nenašla pro některého Pacmana přípustnou cestu, jelikož je ze všech stran obklíčen duchy, a tak není možné pokračovat.

```
Input: instance of Multi-Pacman, ordered sets of goals to visit
Result: full or partial set of paths visiting goals in given order
findAgentPaths(constraints, allAgents, agentId):
    solution ← new solution;
    solution.type ← full;
    foreach (id, pacman) in Pacmans do
        if not allAgents and id ≠ agentId then continue;
        insert (id, empty path) into solution.paths;
        goals ← pacman.orderedGoals;
        current ← pacman.position;
        while goals ≠ ∅ do
            goal ← front of goals;
            if goal doesn't have food then
                remove goal from goals;
                continue;
            end
            timestep ← length of solution.paths[id] + gameTime;
            (reachedGoal, path) ←
              AStar(id, timestep, current, goal, constraints);
            if not reachedGoal then
                solution.type ← partial;
                if solution.paths[id] is empty and path is not empty
                  then insert path to solution.paths[id];
                else
                    solution.type ← impossible;
                    return solution;
                end
                break;
            else
                insert path into solution.paths[id];
                current ← front of goals;
                remove goal from goals;
            end
        end
    end
    foreach (id, path) in solution.paths do
        if path is empty then
            remove (id, path) from solution.paths;
            remove Pacmans[id] from Pacmans;
        end
    end
    if solution.type is partial then
        solution.validUntil ← length of shortest path in
          solution.paths;
    end
    return solution;
```

Pseudokód 4.5: Hledání cest – metoda `findAgentPaths()`

V takovém případě CBS vrátí do hlavní úrovně tuto informaci, která přeruší opakované plánování a skončí algoritmus s tím, že pro tuto konfiguraci (mapa, rozmístění Pacmanů a duchů a *seed* náhodného generátoru) není možné sesbírat všechny kuličky.

Pro nalezení řešení CBS volá nižší úroveň uvedenou v pseudokódu 4.5. Metoda `findAgentPaths()` potřebuje kromě instance Multi-Pacmana s již seřazenými cíli také množinu omezení `constraints` (z právě prohledávaného vrcholu ve vyšší úrovni). Argumenty `allAgents` a `agentId` ovlivňují, jestli bude metoda hledat řešení pro všechny Pacmany, či pouze jednoho uvedeného (tato varianta je využita při aktualizaci řešení po přidání nových omezení v důsledku konfliktu).

V cyklu je iterováno všemi Pacmany. Procházíme jejich posloupnost cílů, přičemž pokud aktuální cíl již neobsahuje kuličku, přeskočíme jej (řádky 9–14). Vypočteme, v jakém časovém kroku se nacházíme a pro nalezení cesty k ještě nesebranému cíli zavoláme algoritmus A* (řádky 15–16). Ten vrátí informaci, zda bylo cíle dosaženo, a cestu.

Řádky 17–24 popisují, co se děje, pokud cíle dosaženo nebylo. Víme, že po cestě hrozí kolize s duchem, a tak je nastavením typu řešení na částečné indikováno, že nalezené cesty všech Pacmanů budou validní pouze do časového kroku před tím, ve kterém hrozí první kolize. Pokud již Pacman má ve svém řešení nějaké cesty, nalezená nedokončená cesta je zahozena. Pokud v řešení žádné cesty zatím nemá, stále je nutno Pacmana někam navigovat, i když se nedosáhlo jeho cíle. Modifikovaný A* se stará i o tuto situaci, viz dále. Když tedy nalezená cesta není prázdná, je v tomto případě také přidána do řešení. Jestliže prázdná je, nastala situace, kdy není možno Pacmana nikam posunout a při stání na místě bude sražen duchem – řešení je nastaveno na typ nenalezeno a vráceno.

Pokud A* úspěšně nalezl cestu do cíle, je cesta přidána do řešení, aktuální pozice Pacmana se nastaví na nalezený cíl, který je odebrán z posloupnosti a pokračuje hledání do dalšího cíle (řádky 25–29).

Jakmile tento cyklus proběhne pro všechny Pacmany, smažou se případné prázdné cesty z řešení. Pokud v řešení zůstala prázdná cesta, znamená to, že Pacman už nemá další cíle k hledání, splnil svou část úlohy a je tak možno smazat i samotného Pacmana (řádky 32–37). Pokud byl v průběhu typ řešení nastaven na částečné, vypočte se, kolik kroků v nalezených cestách je validních podle délky nejkratší z nich (řádky 38-40). Nakonec je řešení vráceno vyšší vrstvě.

O hledání jednotlivých cest se stará modifikovaná verze A* uvedená v pseudokódu 4.6. Protože v Multi-Pacmanovi mohou Pacmani místo pohybu čekat na místě, je nutno do mapy předchůdců `cameFrom` ukládat také čas výskytu v daném vrcholu (řádek 4), aby čekání bylo bráno v potaz při rekonstrukci cest. Do prioritní fronty spolu s aktuálním vrcholem ukládáme také aktuální časový krok, který v cyklu potřebujeme znát.

Hledací cyklus je rozšířen několika způsoby. Na řádku 10 jsou vygenerovány možné pozice duchů v následujícím časovém kroku. Protože se duchové pohybují náhodně, musíme při plánování počítat se všemi možnými pozicemi, na kterých se mohou vyskytnout.

```
Input: instance of Multi-Pacman
Result: shortest path from start to goal or to the closest node to goal
AStar(agentId, startTime, start, goal, constraints):
    open ← empty priority queue;
    distSoFar[start] ← g(start) = 0;
    cameFrom[(start, startTime)] ← (start, startTime);
    priority ← f(start) = 0 + h(start);
    insert (priority, start, startTime) into open;
    while open not empty do
        (prio, current, time) ← node from open with lowest f(n);
        remove (prio, current, time) from open;
        generatePossibleGhostPositions(time + 1);
        if current is goal then
            goalTime ← time;
            break;
        end
        neighbors ← set of nodes accessible from current;
        insert current into neighbors;
        foreach node next in neighbors do
            newDist ← distSoFar[current] + d(current, next);
            if not ((time + 1, next) ∈ constraints and
              constraints[(time + 1, next)] = agentId) and
              (distSoFar[next] = ∅ or next = current or
              newDist < distSoFar[next]) then
                distSoFar[next] ← newDist;
                newPrio ← f(next) = newDist + h(next);
                insert (newPrio, next, time + 1) into open;
                cameFrom[(next, time + 1)] ← (current, time);
            end
        end
    end
    (reachedGoal, path) ←
      reconstructPath(start, goal, startTime, goalTime, cameFrom);
    if not reachedGoal and path is empty then
        while open not empty do
            (prio, cur, time) ← node from open with lowest f(n);
            remove (prio, cur, time) from open;
            (reached, newPath) ←
              reconstructPath(start, cur, startTime, time, cameFrom);
            if newPath not empty and current ≠ start then
                return (reachedGoal, newPath);
            end
        end
        return (reachedGoal, empty path);
    end
    return (reachedGoal, path);
```

Pseudokód 4.6: Hledání cest – modifikovaný A*

Metoda `generatePossibleGhostPositions()` možné pozice získá z možných pozic v předchozím časovém kroku, pokud žádný předchozí krok nebyl vygenerován, použijí se skutečné pozice duchů. Možné pozice narůstají velmi rychle, protože v každé křižovatce má duch dvě až tři možnosti a jelikož musíme počítat se všemi, je to jako by duchů přibývalo. V každém dalším časovém kroku je jich více a více, což znemožňuje nalézt plné cesty do cíle a což je důvodem, proč je nutné v hlavní úrovni řešení střídat plánování a vykonávání plánů.

Na řádku 16 mezi sousedy aktuálního vrcholu vkládáme taky samotný aktuální vrchol, protože je povolena akce čekání. Upravená podmínka na řádku 19 vyjadřuje, v jakém případě můžeme sousední vrchol uvažovat v hledání – k požadavkům z klasického A* je přidáno, že soused nesmí být v dalším časovém kroku v `constraints` (omezení daná vyšší vrstvou) a že soused může být aktuální vrchol.

Na řádku 21 je jako heuristika místo Manhattanské vzdálenosti použita její upravená verze, která funguje pro toroidní mapy. Pacmanovské mapy jsou toroidní v místech tunelů. Dle [26] se vypočítá jako

$$h(a,b) = \sum_{i=1}^{n} \min(|a_i - b_i|, (a_i + mapSize_i) - b_i, (b_i + mapSize_i) - a_i)$$

pro body a, b, kde a_i resp. b_i značí souřadnice bodu v i-tém rozměru a $mapSize_i$ je rozměr mapy v i-tém rozměru.

Po nalezení cíle je zavolána metoda `reconstructPath()`. Ta dle mapy předchůdců zjistí cestu ze startu do cíle (řádek 27) a navíc tyto cesty kontroluje s vygenerovanými možnými pozicemi duchů. Pokud hrozí kolize, je cesta zkrácena pouze na část, která kolizi předchází. Metoda vrací cestu a informaci, zda dosáhla cíle (tj. zda nebyla zkrácena).

Pokud cíle dosaženo nebylo a cesta je prázdná (řádky 28–38), znamená to, že duch blokuje hned první krok cesty k cíli (tj. vyskytuje se hned vedle Pacmana). Přesto je nutné Pacmanovi najít nějakou cestu. K tomu je využita fronta `open`, ve které ještě zbývají další slibné expandované vrcholy při prvotním hledání cíle. Když se k některému z nich lze pohnout (rekonstruovaná cesta k tomuto vrcholu není prázdná), je z A* vrácena tato cesta. Když nelze, Pacman je obklíčen ze všech stran a celá instance nemá řešení.

Pokud tento problém nenastal, je vrácen výstup `reconstructPath()` z řádku 27.

Po návratu z metody `findAgentPaths()` CBS pokračuje běžným postupem – nalezené cesty jsou kontrolovány na konflikty Pacmanů mezi sebou a případné konflikty jsou vyřešeny, dokud není nalezeno bezkonfliktní řešení. Takové je vráceno do nejvyšší vrstvy (pseudokód 4.1).

Je-li řešení částečné, jsou do plánů Pacmanů přidány nalezené cesty pouze do počtu `validUntil` kroků, jinak jsou přidány v plné délce. Následně jsou plány skutečně vykonány – Pacmani se po časových krocích posouvají dle naplánovaných cest, duchové se v křižovatkách dle generátoru náhodných čísel rozhodnou pro konkrétní směr. Protože plánování počítalo se všemi možnými pozicemi duchů, nikdy se s Pacmany nesrazí.

Během vykonávání plánu jsou Pacmanům z množiny cílů odebírány takové, které již nemají kuličku (ať už tím, že ji Pacman právě sebral, nebo byla sebrána jiným např. při odchýlení od vlastních cílů kvůli ducha). Po vykonání plánů se zkontroluje, zda jsou všechny kuličky sebrány. Pokud ne, plánovací část se opakuje znovu – tentokrát s menší posloupností zbývajících cílů pro každého Pacmana a s aktualizovanými pozicemi duchů (po odkrokování známe jejich skutečné pozice a tak je možno se „synchronizovat“ s realitou a v dalším plánování generovat nové možné pozice od aktuálních). Jakmile jsou sesbírány kuličky všechny, algoritmus končí a vrací finální celkové cesty agentů.

4.4 Výstup

Výstupem programu je textový soubor obsahující výsledky jednotlivých částí algoritmu – tedy seznam oblastí přidělených Pacmanům, seřazené posloupnosti cílů nacházejících se v jejich oblastech, a celkové cesty, které vedly k sesbírání všech kuliček (resp. cesty do kroku, ze kterého není možné dále pokračovat).

Při použití debugovacího či měřícího módu jsou součástí výstupního souboru zprávy popisující běh algoritmu či měřené hodnoty. Pro lepší vizualizaci řešené situace je dalším výstupem jednoduchá textová animace v konzoli zobrazující vykonávání plánů.

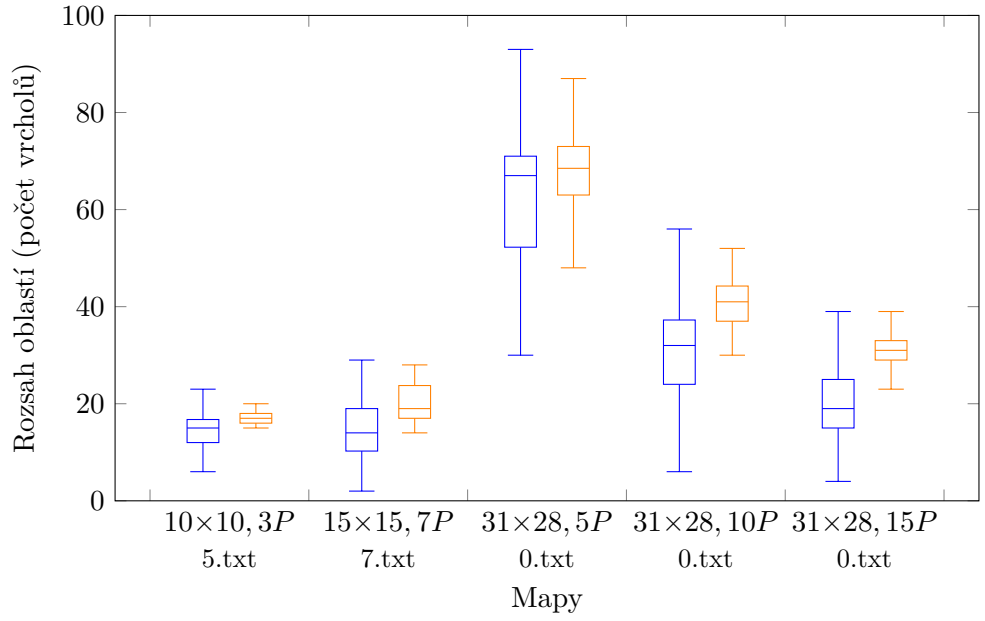
Spouštění, ovládání a nastavení programu je blíže popsáno v příloze a na přiloženém médiu.

Kapitola 5
Experimenty

V této kapitole jsou uvedeny výsledky testování programu na různých herních mapách s různými počátečními konfiguracemi (počty a rozmístění Pacmanů a duchů).

5.1 Porovnání algoritmů pro rozdělení mapy

Na obrázku 5.1 je uveden krabicový graf, který porovnává rozsahy velikostí přidělených oblastí při použití dvou variant algoritmu pro rozdělení mapy. Modře je zakreslena varianta BFS pracující na normální reprezentaci grafu, oranžově modifikované DFS na hierarchické reprezentaci.



Obrázek 5.1 Graf zobrazující rozsahy přidělených oblastí v obou variantách přiřazovacího algoritmu

Měření bylo provedeno na dvou vlastních mapách a na originální mapě s různými počty Pacmanů, konkrétní rozměry mapy a počet Pacmanů jsou uvedeny na horizontální ose. Bylo provedeno 10 měření, každý sloupec tak zobrazuje $10 \times n$ hodnot velikostí oblastí, kde n je počet Pacmanů.

Z grafu lze vyčíst, že rozptyl mezi velikostmi jednotlivých oblastí je větší při použití BFS varianty. Protože BFS rozšiřuje oblasti pouze do doby, než narazí na jinou oblast, mohou vznikat velké rozdíly způsobené počátečním rozmístěním Pacmanů – např. pokud se Pacman vyskytuje v blízkosti jiných, bude jeho přidělená oblast velmi malá.

Varianta DFS pokračuje s expandováním oblastí i jakmile narazí na oblast cizí, je také povolen překryv oblastí a rozšiřuje se vždy ta oblast, která je aktuálně nejmenší. Důsledkem je menší rozptyl, což znamená podobné velikosti oblastí pro všechny Pacmany a tak je práce rovnoměrněji rozdělena.

5.2 Testování různého počtu Pacmanů a duchů na původní mapě

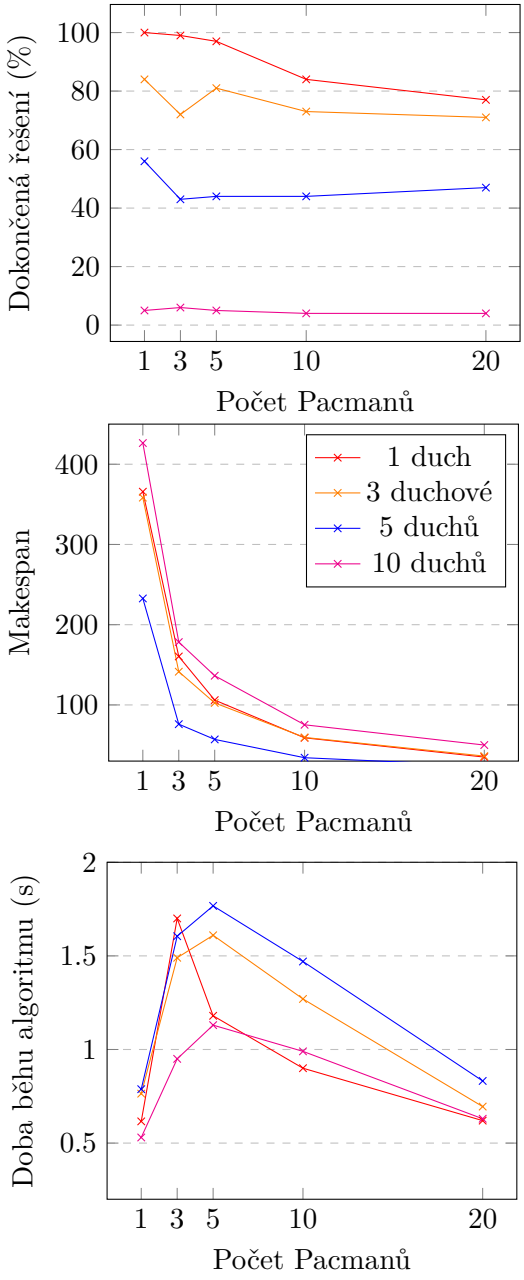
Následující tři grafy ukazují závislosti mezi počty Pacmanů a duchů a procentuální úspěšností dokončených řešení, makespanem a dobou, jak dlouho algoritmus úlohu řeší. Počet Pacmanů je znázorněn na vodorovné ose a počty duchů vyjadřují různé barvy křivek.

Pro každou kombinaci počtu Pacmanů a duchů bylo vygenerováno 100 variant jejich rozmístění a tyto se program pokusil vyřešit. Pro rozdělení oblastí byla použita varianta pracující na hierarchické reprezentaci mapy.

Z prvního grafu, který zobrazuje úspěšná řešení v procentech, je zřejmé, že zvyšující se počet duchů rapidně snižuje šanci na vyřešení. To dává smysl především v situacích, kdy je více duchů než Pacmanů, protože mohou Pacmany snadněji obklíčit z obou stran chodby. Úspěšnost se snižuje také se zvyšujícím počtem Pacmanů.

Druhý graf zobrazuje průměrný počet časových kroků potřebných k sesbírání všech kuliček, který je počítán pouze z úspěšných řešení. Křivky naznačují, že tento průměrný počet příliš nezávisí na počtu duchů. Větší počet Pacmanů však makespan velmi snižuje, což je dáno vzájemnou spoluprací Pacmanů – každý vysbírává jemu přidělenou oblast, celkově je tak dokončeno rychleji.

Dle třetího grafu dosahuje algoritmus nejvyšších dob běhu pro tři až pět Pacmanů, poté se doby běhu pro větší počty snižují. Také hodnoty pro jednoho Pacmana jsou nízké, což může být způsobeno tím, že mezi jednoho Pacmana není třeba mapu rozdělovat (připadne mu celá).



Obrázek 5.2 Grafy zobrazující vliv počtu duchů a Pacmanů na dokončená řešení, makespan a dobu běhu algoritmu

Závěr

V teoretické části práce byly uvedeny definice a vysvětleny pojmy týkající se multi-agentního hledání cest a dvou jeho rozšiřujících variant, ve kterých se také řeší přiřazení a seřazení cílů agentům. Pro každou ze tří podúloh byly představeny alespoň dva existující algoritmy pro jejich řešení. V analytické části byla popsána původní hra a její zobecnění pro tuto práci.

V praktické části byly nastudované přístupy zkombinovány do víceúrovňového algoritmu, který využívá jako své součásti algoritmy BFS, DFS, A* a CBS. Celkový algoritmus a jeho implementace byla popsána a vysvětlena včetně patřičných úprav potřebných pro řešenou zobecněnou verzi Pacmana.

V experimentální části byly porovnány dva algoritmy pro rozdělení herního bludiště a byla provedena měření pro různé počty Pacmanů a duchů.

Bibliografie

1. KORNHAUSER, D.; MILLER, G.; SPIRAKIS, P. Coordinating Pebble Motion On Graphs, The Diameter Of Permutation Groups, And Applications. In: *25th Annual Symposium on Foundations of Computer Science*. 1984, s. 241–250. ISBN 0-8186-0591-X. Dostupné z DOI: 10.1109/SFCS.1984.715921.
2. RYAN, Malcolm. Graph Decomposition for Efficient Multi-Robot Path Planning. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. 2007, s. 2003–2008. ISBN 978-1-57735-298-3.
3. YU, Jingjin; LAVALLE, Steven M. Multi-agent path planning and network flow. In: *Algorithmic foundations of robotics X*. Springer, 2013, s. 157–173. ISBN 978-3-642-36279-8.
4. STERN, Roni; STURTEVANT, Nathan R.; FELNER, Ariel et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In: *Twelfth Annual Symposium on Combinatorial Search*. AAAI Press, 2019, s. 151–158. ISBN 978-1-57735-808-4.
5. FELNER, Ariel; STERN, Roni; SHIMONY, Solomon Eyal et al. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In: *Tenth Annual Symposium on Combinatorial Search*. AAAI Press, 2017, s. 29–37. ISBN 978-1-57735-790-2.
6. MA, Hang; KOENIG, Sven; AYANIAN, Nora et al. Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios. 2017. Dostupné z arXiv: 1702.05515 [cs.AI].
7. RIVERA, Nicolás; HERNÁNDEZ, Carlos; HORMAZÁBAL, Nicolás et al. The 2^k Neighborhoods for Grid Path Planning. *Journal of Artificial Intelligence Research*. 2020, roč. 67, s. 81–113. Dostupné z DOI: 10.1613/jair.1.11383.
8. SURYNEK, Pavel; FELNER, Ariel; STERN, Roni et al. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In: *Proceedings of the Twenty-second European Conference on Artificial Intelligence*. 2016, s. 810–818. ISBN 978-1-61499-671-2.

9. SHARON, Guni; STERN, Roni; FELNER, Ariel et al. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*. 2015, roč. 219, s. 40–66. ISSN 0004-3702. Dostupné z DOI: 10.1016/j.artint.2014.11.006.

10. HART, Peter E; NILSSON, Nils J; RAPHAEL, Bertram. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*. 1968, roč. 4, č. 2, s. 100–107.

11. RUSSELL, Stuart J.; NORVIG, Peter. *Artificial Intelligence: a modern approach*. 3. vyd. Pearson, 2009. ISBN 978-0-13604-259-4.

12. PEARL, Judea. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., 1984. ISBN 978-0-201-05594-8.

13. SILVER, David. Cooperative pathfinding. In: *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2005, s. 117–122.

14. KHORSHID, Mokhtar M; HOLTE, Robert C; STURTEVANT, Nathan R. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In: *Proceedings of the Fourth Annual Symposium on Combinatorial Search*. 2011, s. 76–83. ISBN 978-1-57735-537-3.

15. SURYNEK, Pavel. A novel approach to path planning for multiple robots in bi-connected graphs. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, s. 3613–3619. ISBN 978-1-424-42788-8.

16. SURYNEK, Pavel. Towards optimal cooperative path planning in hard setups through satisfiability solving. In: *12th Pacific Rim International Conference on Artificial Intelligence*. 2012, s. 564–576. ISBN 978-3-642-32695-0.

17. KUHN, Harold W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*. 1955, roč. 2, č. 1-2, s. 83–97. Dostupné z DOI: 10.1002/nav.3800020109.

18. MA, Hang; KOENIG, Sven. Optimal Target Assignment and Path Finding for Teams of Agents. In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. 2016, s. 1144–1152. ISBN 978-1-4503-4239-1.

19. HÖNIG, Wolfgang; KIESEL, Scott; TINKA, Andrew et al. Conflict-Based Search with Optimal Task Assignment. In: *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*. 2018, s. 757–765. ISBN 978-1-5108-6808-3.

20. VAN, Nguyen; PHILIPP, Obermeier; SON, Tran Cao et al. Generalized Target Assignment and Path Finding Using Answer Set Programming. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. 2017, s. 1216–1223. Dostupné z DOI: 10.24963/ijcai.2017/169.

21. SURYNEK, Pavel. Multi-Goal Multi-Agent Path Finding via Decoupled and Integrated Goal Vertex Ordering. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021, sv. 35, s. 12409–12417. Č. 14. ISBN 978-1-57735-866-4.

22. LIERLER, Yuliya. What is answer set programming to propositional satisfiability. *Constraints*. 2017, roč. 22, č. 3, s. 307–337. Dostupné z DOI: 10.1007/s10601-016-9257-7.

23. LAPORTE, Gilbert. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*. 1992, roč. 59, č. 2, s. 231–247. ISSN 0377-2217. Dostupné z DOI: 10.1016/0377-2217(92)90138-Y.

24. PITTMAN, Jamey. The Pac-Man Dossier. In: *Game Developer* [online]. 2009 [cit. 2021-12-06]. Dostupné z: https://www.gamedeveloper.com/design/the-pac-man-dossier.

25. CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L. et al. *Introduction to Algorithms, Third Edition*. 3. vyd. The MIT Press, 2009. ISBN 978-0-262-03384-8.

26. AMIT, Patel. *Map representations* [online] [cit. 2021-02-01]. Dostupné z: http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html.

Příloha

.1 Ovládání softwarového prototypu

Softwarový prototyp byl naprogramován v jazyce C++. Byl kompilován a testován na linuxové distribuci Ubuntu 18.04 pomocí kompilátoru G++ verze 10.3.0.

Zdrojové kódy se nachází na přiloženém médiu ve složce code. Pro kompilaci je doporučeno použít program cmake, který dle souboru CMakeLists.txt vytvoří Makefile pro program make. Přesné příkazy k použití jsou uvedeny na přiloženém médiu v souboru readme.txt.

Prototyp se spouští z příkazové řádky s jedním parametrem, kterým je cesta ke vstupnímu souboru. Vzorové vstupní soubory se nachází v adresáři input. Nastavení programu je možno měnit pomocí souboru settings.txt, kterým je možno ovládat: generátor náhodných čísel (jestli se má *seed* vygenerovat nebo použít dodaný), použitou reprezentaci grafu (normální či hierarchická), vizualizaci v konzoli (zda krokování čeká na vstup uživatele klávesou Enter nebo krokuje po určitém počtu milisekund), použití unikátních symbolů pro Pacmany a duchy pro jejich rozlišení, mód (klasický, debugovací či měřící) a náhodné rozmisťování zvoleného počtu Pacmanů a duchů. Způsoby a možnosti pro nastavení jsou popsány komentáři přímo v souboru settings.txt.

Do vstupních map je možné pozice Pacmanů přímo zakreslit a počet duchů a jejich souřadnice uvést pod mapou (jako je tomu v souboru 8.txt), nebo využít náhodného rozmisťování.

Vizualizace řešené instance v konzoli nejprve zobrazí hierarchickou reprezentaci mapy, poté zvýrazní oblasti přidělené jednotlivým Pacmanům a nakonec je možno krokovat naplánované cesty a sledovat pohyb Pacmanů a duchů při sbírání kuliček.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `code` ..... adresář se spustitelnou formou implementace
  - `include` ..... zdrojové kódy implementace
  - `input` ..... adresář se vstupními soubory
  - `output` ..... adresář pro výstupní soubory
  - `src` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF