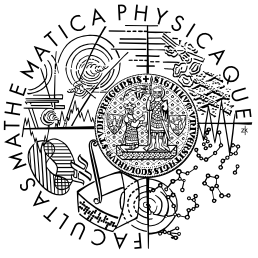


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Jan Slezák

Sociální pravidla pro multi-agentní hledání cest

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: prof. RNDr. Roman Barták, Ph.D.

Studijní program: Informatika – Umělá inteligence

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Děkuji prof. RNDr. Romanu Bartákovi, Ph.D. za vedení bakalářské práce, cenné rady a připomínky.

Název práce: Sociální pravidla pro multi-agentní hledání cest

Autor: Jan Slezák

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: prof. RNDr. Roman Barták, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Multi-agentní hledání cest je problém, kde se snažíme navigovat agenty do jejich cílů bez kolizí s dalšími agenty. K řešení problému využíváme distribuovaný přístup, který nespoléhá na centrální plánování ani přímou komunikaci mezi agenty. Místo centralizovaného přístupu zavedeme tzv. sociální pravidla, která omezují agenta ve vykonání některých akcí podle jeho lokálního okolí za cílem zamezení konfliktu mezi agenty. V této práci ukážeme jejich využitelnost v problému multi-agentního hledání cest. Tyto techniky byly navrhnuty a otestovány na různých testovací mapách pro různé počty agentů.

Klíčová slova: hledání cest, sociální pravidla, multi-agentní prostředí

Title: Social Laws for Multi-Agent Path Finding

Author: Jan Slezák

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: prof. RNDr. Roman Barták, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Multi-agent path-finding is a problem where we navigate agents to their destinations without collisions with other agents. We use a distributed approach to solve the problem that does not rely on centralized planning or direct communication between agents. Instead of a centralized approach, we introduce so-called social laws that restrict an agent from executing certain actions according to its local environment in order to avoid conflicts between agents. In this paper, we show their applicability in multi-agent pathfinding problem. These techniques have been designed and tested on various test maps for different numbers of agents.

Keywords: path-finding, social laws, multi-agent environment

Obsah

Úvod

Problematika multi-agentního hledání cest (v překladu Multi-Agent Path-Finding (MAPF)) je v současné době jednou z důležitých oblastí umělé inteligence. Zabývá se koordinací pohybu více agentů, kteří se nachází ve sdíleném prostředí a mají dosáhnout stanovených cílových pozic bez vzájemných kolizí. Tato problematika má mnoho praktických uplatnění v robotice, automatizovaných průmyslových skladech i v oblasti počítačových her.

V klasické variantě problému máme k dispozici startovní a cílové pozice jednotlivých agentů a snažíme se mezi nimi agenty koordinovaně navigovat. Řešením problému jsou plány pro jednotlivé agenty, které dostanou každého agenta do jeho cíle a zaručí, že se žádní dva agenti nesrazí.

Prostředí, v němž se agenti nacházejí, diskretizujeme a reprezentujeme grafem, kde všechny agentům dostupné pozice odpovídají vrcholům grafu. Sousední pozice pak propojují hrany grafu. Agenti se v grafu pohybují najednou v časových krocích, přičemž mohou zůstat v aktuálním vrcholu, nebo se přesunout do sousedního. Celá mapa, jak prostředí označujeme, je všem agentům předem známá.

Běžným přístupem k řešení problému je využití centralizovaných metod, kdy jedna centrální jednotka naplánuje cesty pro všechny agenty. Tyto metody fungují dobře pro menší instance problému, pro rozsáhlejší instance s mnoha agenty se však stávají neefektivními. Zároveň pracují s předpoklady, jako je znalost přesné lokace každého agenta v celém prostředí nebo bezchybné plnění všech pokynů od plánovače všemi agenty.

V reálných aplikacích však tyto předpoklady nemusí vždy platit. V těchto situacích se při řešení koordinace pohybu agentů uplatňují distribuované metody. V tomto přístupu agenti pracují autonomně a vycházejí při řešení problému pouze ze znalosti lokací ostatních agentů ve svém viditelném okolí. V tomto textu navíc pracujeme s předpokladem, že agenti mezi sebou nemají možnost vzájemné komunikace. Tím se výrazně zvyšuje efektivita celého systému, protože se eliminuje potřeba složitých komunikačních protokolů a snižuje se riziko chyby způsobené nesprávnou nebo ztracenou zprávou mezi agenty.

Z toho důvodu zavedeme tzv. sociální pravidla (v překladu social laws), která koordinují pohyb jednotlivých agentů. Můžeme si je představit jako dopravní předpisy, kterými se agenti řídí. Při jejich dodržování by se agenti měli dostat do svých cílů bez vzájemných srážek a zablokovaní. Chceme, aby se agenti dokázali vzájemně vyhýbat, aniž by mezi sebou museli jakkoliv komunikovat.

V této práci si nově představíme koncept sociálních pravidel v rámci multi-agentního hledání cest a na testovacích instancích problému ukážeme jejich využitelnost. Náš koncept pracuje na základě jen velmi mála informací z lokálního okolí agenta a umožňuje tak jednoduchou a efektivní implementaci v reálných autonomních robotech.

Struktura této práce je následující. V první kapitole si formálně popíšeme řešený MAPF problém. V druhé kapitole se zaměříme na současné metody řešící tento problém a představíme si využití sociálních pravidel v rámci obecného plánování. Ve třetí kapitole následně formálně zavedeme sociální pravidla v rámci distribuovaného přístupu k MAPF a navrhneme konkrétní sady pravidel. Ve čtvrté kapitole otestujeme hypotézy spojené s funkčností navržených sad na různých

typech instancí problému. V příloze je pak dostupný zdrojový kód aplikace MAPF Simulator for Social Laws i s uživatelskou dokumentací. Tato aplikace umožňuje testování vlastních sad sociálních pravidel na libovolně zvolených instancích problému.

1 Formulace problému

Začneme uvedením formálních definic a souvisejících pojmů z oblasti MAPF. Při popisu problému vycházíme z článku Stern et al. (2019).

1.1 Definice klasického MAPF

Nejprve definujeme to, co označujeme jako klasický MAPF. Vstupem do klasického problému MAPF s k agenty je trojice $\langle G, s, t\rangle$, kde: $G = (V, E)$ je neorientovaný graf, $s : [1 \dots k] \to V$ je funkce určující každému agentovi startovní vrchol a $t : [1 \dots k] \to V$ je funkce určující každému agentovi cílový vrchol. Dále předpokládáme, že čas je diskrétní a rozdělujeme ho do časových kroků. V každém tomto kroku se každý agent nachází v jednom z vrcholů grafu a může provést právě jednu *akci*.

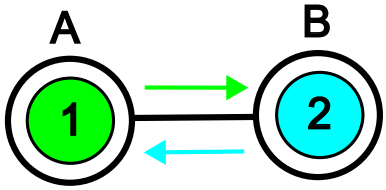
Akce v klasickém MAPF je funkce $a : V \to V$ taková, že $a(v) = v'$. To znamená, že pokud agent je ve vrcholu v a vykoná akci a, v dalším kroku bude ve vrcholu v'. Možné akce agenta můžeme rozdělit do dvou typů: *move* a *wait*. Akce *move* znamená, že se agent přesune z aktuálního vrcholu v do sousedního vrcholu v' v grafu V, tedy $(v,v') \in E$. Akce *wait* znamená, že agent nadále zůstává ve stejném vrcholu v.

Pro sekvenci akcí $\pi = (a_1 \dots a_n)$ a agenta i označíme $\pi_i[x]$ jako místo, kde se agent bude vyskytovat po vykonání x akcí podle π ze startovního vrcholu $s(i)$. Formálně můžeme zapsat jako $\pi_i[x] = a_x(a_{x-1}(\cdots a_1(s(i))))$. Takovou sekvenci akcí π nazveme **plánem** pro jednoho agenta i právě tehdy, když vykonáním této sekvence akcí ze startovního vrcholu $s(i)$ skončí agent v cílovém vrcholu $t(i)$. **Řešením** nazveme množinu k takových plánů, kde každému agentovi bude náležet jeden plán.

1.2 Konflikty mezi agenty

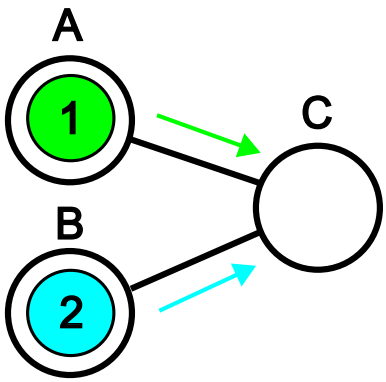
Při snaze o dosažení cílů mohou mezi sebou plány jednotlivých agentů kolidovat. V MAPF se snažíme najít taková řešení, kde mezi žádnými dvěma plány nedochází k žádným konfliktům, ty pak nazýváme **validními**. Existují různé typy konfliktů, my ale při řešení MAPF budeme uvažovat následující konflikty:

- **Výměnný konflikt** mezi plány π_i a π_j nastane, pokud si agenti v jednom kroku vymění pozice (viz Obrázek 1.1). To znamená, že existuje časový krok x takový, že $\pi_i[x+1] = \pi_j[x]$ a $\pi_j[x+1] = \pi_i[x]$.



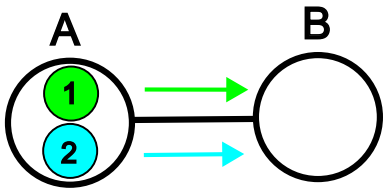
Obrázek 1.1 Výměnný konflikt.

- **Vrcholový konflikt** mezi plány π_i a π_j nastane, pokud se agenti nachází ve stejný čas ve stejném vrcholu (viz Obrázek 1.2). To znamená, že existuje časový krok x takový, že $\pi_i[x] = \pi_j[x]$.



Obrázek 1.2 Vrcholový konflikt.

- **Hranový konflikt** mezi plány π_i a π_j nastane, pokud agenti přecházejí po téže hraně ve stejný čas stejným směrem (viz Obrázek 1.3). To znamená, že existuje časový krok x takový, že $\pi_i[x] = \pi_j[x]$ a $\pi_i[x+1] = \pi_j[x+1]$.



Obrázek 1.3 Hranový konflikt.

Z definic můžeme usoudit, že zakázáním vrcholového konfliktu automaticky zakazujeme i hranový konflikt.

1.3 Chování agentů v cíli

Jelikož agenti mohou dorazit do cílů v různé časy, musíme definovat jejich chování v moment, kdy jednotliví agenti dosáhnou cíle, zatímco se jiní agenti stále pohybují po grafu. Tradičně existují dva způsoby:

- Agent zůstane v cíli.
- Agent v cíli zmizí.

V našem případě uvažujeme variantu, kde agent poté, co dorazí do cíle, zmizí, což znamená, že plán agenta nebude mít žádný konflikt s ostatními plány agentů po jeho dokončení.

1.4 Účelová funkce

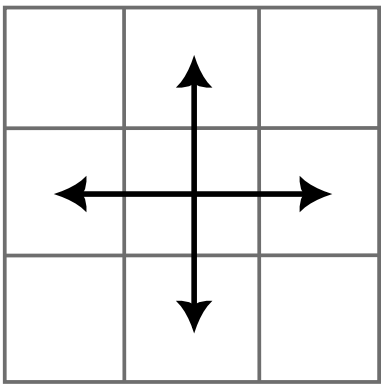
K evaluaci, jak dobré je konkrétní řešení instance MAPF problému, používáme účelovou funkci. Mezi dvě nejpoužívanější v klasickém MAPF patří:

- **Celková délka plánů (Makespan)** – počet časových kroků potřebných k tomu, aby všichni agenti dorazili do svých cílů. Pro konkrétní řešení $\pi = \{\pi_1 \dots \pi_k\}$, celková délka plánů π je definovaná jako $\max_{1 \leq i \leq k} |\pi_i|$.
- **Součet cen (Sum of costs (SoC))** – součet všech časových kroků potřebných k doražení do cíle všemi agenty. Součet cen π definujeme jako $\sum_{1 \leq i \leq k} |\pi_i|$.

Validní řešení s nejmenší možnou hodnotou zvolené účelové funkce nazveme **optimálním**.

1.5 MAPF grafy

V reálných oblastech, kde MAPF nachází využití, se většinou vyskytují prostředí, která se dají simulovat rovinnými grafy. Techniky sociálních pravidel můžeme uplatnit i v různých jiných grafech, ale v této práci budeme využívat standardní variantu grafu – mřížku se čtyřmi sousedy (viz Obrázek 1.4), pro kterou existuje celá řada referenčních řešení. Předpokládáme, že všechny hrany grafu mají stejnou délku a akce agenta trvají stejný časový úsek.



Obrázek 1.4 Možné přesuny agenta do sousedních vrcholů.

1.6 Neočekávané zpoždění

V klasické MAPF uvažujeme, že agenti provádějí akce přesně podle svého plánu, to ale v reálných situacích nemusí platit. Vlivem prostředí nebo nedokonalosti agentů může docházet k neočekávaným událostem, jednou z nich je zpoždění agenta. Když taková situace nastane, může dojít ke kolizím, i když původní plány byly bezkolizní (Barták et al., 2019).

Zaveďme důležité pojmy formálně:

Definice 1. *Zpoždění agenta v plánu π je definováno jako trojice $\langle \pi, i, t \rangle$, reprezentující agenta i, který v čase t nevykonal akci* move *z plánu π_i a místo toho zůstal v místě $\pi_i(t-1)$. (Atzmon et al., 2018)*

Definice 2 (robustnost)**.** *Plán, který můžeme vykonávat bez kolizí i po zpoždění agenta, označíme jako robustní. Formálně, pro plán π a zpoždění $D = \langle \pi, i, t \rangle$, nechť $D(\pi)$ je plán, který je ekvivalentní k π, kromě nahrazení π_i za*

$$\pi_i'(t) = \begin{cases} s_i & t = 0 \\ \pi_i(t) & t < t_D \\ \pi_i(t-1) & \textit{jinak.} \end{cases}$$

Plán je poté robustní vůči zpoždění D, pokud $D(\pi)$ je validní. Dále π je robustní vůči množině zpoždění $\mathcal{D}$ právě tehdy, když použitím jakékoliv podmnožiny $\mathcal{D}$ na π vznikne validní plán. (Atzmon et al., 2018)

2 Související práce

V této kapitole si představíme související práce a způsoby, jak je možné přistoupit k řešení problému MAPF. Zároveň přidáme kontext k využití sociálních pravidel v obecnějších plánovacích problémech.

2.1 Využití sociálních pravidel

V obecném pojetí plánování si sociální pravidla můžeme představit jako systém v prostředí s více agenty, který má za cíl koordinaci agentů k dosažení nějakého cíle. Tato pravidla regulují chování konkrétního agenta k prospěchu všech agentů jako celku.

Koncept sociálních pravidel použili v článku Shoham a Tennenholtz (1995) v kontextu umělých sociálních systémů v obecném multi-agentním prostředí. V tomto modelu sociální pravidla omezují akce agentů v systému. Přesněji řečeno, v době před vytvořením plánů pravidla zakáží agentům některé dříve povolené akce proto, aby se minimalizoval vznik konfliktů během vykonávání akcí. Robustní sociální pravidla zamezují vzniku konfliktů, ale zároveň nechají každému agentovi možnost dosáhnout svého cíle.

V tomto konceptu jsou sociální pravidla „domain-dependent“. Znamená to, že pro konkrétní multi-agentní prostředí, kde budeme pravila využívat, musíme předem navrhnout nová pravidla.

Využití sociálních pravidel bylo konkrétně zavedeno v plánovacím problému obsahujícím více agentů – Multi-Agent Planning (MAP) v systému Stanford Research Institute Problem Solver (STRIPS). Tento systém umožňuje definovat a řešit plánovací problémy pomocí formálního popisu světa, cílů a akcí. V článku autoři Karpas et al. (2017) představili sociální pravidla v upravené verzi Multi-Agent STRIPS (MA-STRIPS), kde každý agent má vlastní cíl.

Definice 3. *Problém MA-STRIPS lze formálně definovat jako čtveřici*

$$\Pi = \langle F, \{A_i\}_{i=1}^{n}, I, \{G_i\}_{i=1}^{n} \rangle$$

kde:

- F *je konečná množina výrokových proměnných (faktů o světě).*
- A_i *je množina akcí dostupných agentovi* i*.*
- $I \subseteq F$ *je počáteční stav.*
- $G_i \subseteq F$ *je cílový stav agenta* i*.*

Každá akce $a \in A_i$ *pro agenta* i *je definována jako čtveřice*

$$a = \langle pre(a), add(a), del(a) \rangle$$

kde:

- $pre(a) \subseteq F$*: Předpoklady, které musí být pravdivé, aby akce mohla být aplikována.*

- $add(a) \subseteq F$*: Efekty, které se přidají do stavu, když je akce vykonána.*
- $del(a) \subseteq F$*: Efekty, které se odeberou ze stavu, když je akce vykonána.*

Stav $s \subseteq F$ *je množina propozic, které jsou v tomto stavu pravdivé. Přechodová funkce pro akci* a *aplikovanou ve stavu* s *je definována jako:*

$$s' = (s \setminus del(a)) \cup add(a)$$

pokud $pre(a) \subseteq s$*; jinak akce* a *není ve stavu* s *aplikovatelná.*

Definice 4. *Sociální pravidla zde upravují konkrétní instanci MA-STRIPS problému a vytvoří novou, kde mohou:*

1. *odstranit existující akce*
2. *přidat nové předpoklady pro vykonání akce*
3. *přidat propozice do problému*
4. *přidat efekty akcí*
5. *přidat nové cíle pro agenty*

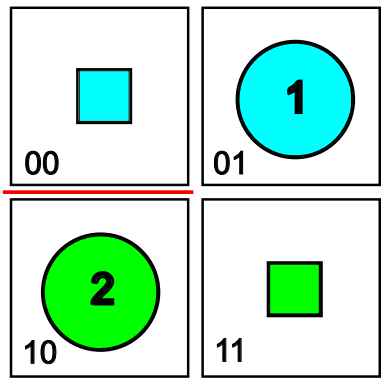
Příklad. Nechť sociální pravidla odstraňují akce jako červená čára na Obrázku 2.1 (čtverce označují cílové stavy agentů v dané barvě), pak:

$$A^l = \{\text{move}_{a2}(10{,}00), \text{move}_{a1}(10{,}00), \text{move}_{a2}(00{,}10), \text{move}_{a1}(00{,}10)\}\}$$

a výsledný MA-STRIPS vypadá následovně:

$$\Pi^l = \langle F, \{A_i \setminus A^l\}_{i=1}^n, I, \{G_i\}_{i=1}^n \rangle$$

(Nir et al., 2021)



Obrázek 2.1 Odstranění akcí z MA-STRIPS problému.

Další oblastí, kde byla sociální pravidla aplikována, je rozšířená verze klasického plánovacího problému o numerické plánování (Nir et al., 2023).

2.2 Centralizovaný přístup k MAPF

V centralizovaném přístupu předpokládáme, že máme jednu centrální jednotku – plánovač, který má celkovou znalost prostředí i všech agentů a jejich cílů. Plánovač má za úkol rozvrhnout všechny cesty agentů bez konfliktů se snahou minimalizovat účelovou funkci. Nalezení **optimálního řešení** (řešení s nejmenší možnou účelovou funkcí) klasického MAPF je NP-úplný problém (Surynek, 2010), proto se používají i algoritmy s horšími vlastnostmi řešení, které jsou však časově méně náročné. U algoritmů pro MAPF problém uvažujeme dvě vlastnosti – optimalitu a úplnost. Úplnost zaručuje nalezení řešení vždy, pokud existuje.

2.2.1 Neúplné suboptimální algoritmy

Do kategorie neúplných suboptimálních algoritmů spadají nejrychlejší algoritmy založené na prioritním plánování, kde je problém rozdělen na prohledávání jednotlivých agentů v časoprostoru v prioritním pořadí. Zmiňme například algoritmus Cooperative A* (CA*) a Windowed Hierarchical Cooperative A* (WHCA*), který snižuje náročnost prohledávání v časoprostoru pomocí dynamického okna. (Silver, 2005)

2.2.2 Úplné suboptimální algoritmy

Mezi úplné suboptimální algoritmy se řadí algoritmy, které jsou polynomiální a za určitých podmínek splňují úplnost, ale negarantují nejlepší kvalitu řešení. Uveďme například algoritmus Push and Swap, který splňuje úplnost pro grafy s alespoň dvěma volnými vrcholy. Algoritmus používá dvě základní operace: *push*, při níž se agenti pohybují směrem ke svým cílům až do okamžiku, kdy už nemohou pokračovat, a *swap*, která umožňuje dvěma agentům vyměnit si pozice, aniž by se změnila konfigurace ostatních agentů. (Luna; Bekris, 2011)

2.2.3 Úplné optimální algoritmy

Skupina úplné optimální algoritmy obsahuje algoritmy založené na různých principech, všechny z nich však zaručují optimalitu řešení:

- **Rozšířené verze A* algoritmu pro více agentů.**
- **ICTS (Increasing Cost Tree Seach)**. ICTS je dvouúrovňový algoritmus, který postupně zvyšuje limit ceny hledaného řešení. Na vyšší úrovni algoritmus zkoumá různé kombinace ceny pro jednotlivé agenty. Každý uzel ve stromě představuje konkrétní rozdělení ceny mezi agenty. Na nižší úrovni pak probíhá vyhledávání, které se snaží najít cesty pro agenty, jež splňují dané limity ceny bez vzájemných kolizí. (Sharon; Stern; Goldenberg et al., 2013)
- **CBS (Conflict-Based Seach)** a jeho varianty. CBS je dvouúrovňový algoritmus. Na vyšší úrovni probíhá vyhledávání ve stromu konfliktů (Conflict Tree, CT), což je strom založený na konfliktech mezi jednotlivými agenty. Každý uzel ve stromě představuje sadu omezení na pohyb agentů. Na nižší úrovni probíhají rychlá vyhledávání pro jednotlivé agenty, aby byla splněna

omezení daná uzlem stromu na vyšší úrovni. (Sharon; Stern; Felner et al., 2015)

- **Redukce na jiný problém.** MAPF můžeme převést na jiný problém ze třídy NP, pro který máme optimalizované způsoby řešení. Jedním z příkladů je převod na problém splnitelnosti (SAT). (Surynek et al., 2016)

2.3 Distribuovaný přístup k MAPF

Distribuovaný přístup nabízí alternativu k centrálnímu plánování. Využití nachází hlavně v situacích, kdy agenti nemají přehled o všech agentech v prostředí, nebo se nechovají deterministicky. Dalším důvodem použití je špatné škálování plánovacích algoritmů na rozsáhlé prostředí s mnoha agenty. U distribuovaných algoritmů se s tímto problémem nesetkáme, protože v tomto přístupu řešíme prevenci konfliktů v lokálním okolí. Autonomní agent si zde naplánuje vlastní cestu bez ohledu na ostatní agenty a během realizace si následně svou cestu upravuje podle agentů v jeho blízkosti.

V literatuře najdeme mnoho zmínek o distribuovaném (resp. decentralizovaném) řízení multi-agentních systémů, kde je povoleno předávat si některá data v globální míře. My předpokládáme, že distribuovaný přístup tyto informace sdílet neumožňuje. Agenti mají přístup pouze k mapě prostředí a informaci o aktuálním stavu jejich lokálního (viditelného) okolí.

Způsob, jak eliminovat konflikty, je zavedení komunikace mezi agenty. Agenti si mezi sebou v okolí sdělí podstatné informace a podle určitých pravidel vyjednávání si naplánují nové cesty, ve kterých nebudou vzájemné konflikty. V této práci ale zavedeme nový způsob řešení, jenž nevyužívá komunikaci, ale je založen na sociálních pravidlech specifikovaných kolektivně pro všechny agenty.

2.3.1 Způsoby řešení s komunikací agentů

Nyní si ještě představíme některé způsoby řešení problému založené na komunikaci mezi agenty. Tato metoda funguje na vzájemné výměně informací mezi agenty v lokálním okolí. Jde o podstatné informace, na základě kterých se agenti rozhodnou, jak dál pokračovat v kritické situaci. Pokud agenti při výměně naplánovaných cest objeví potenciální konflikt, musí se mu nějakým způsobem vyhnout. Přístupy, jak toho docílit, rozdělíme do dvou kategorií:

- **Přístupy založené na vyhledávání.** Nachází se zde adaptace známých vyhledávacích algoritmů pro distribuovaný přístup, zmiňme například PIBT (Priority Inheritance with Backtracking) (Okumura; Machida et al., 2022) a DBS (Deadlock-based Search) (Okumura; Bonnet et al., 2022).

- **Přístupy založené na hlubokém zpětnovazebním učení.** Zjednodušeně můžeme říci, že agenti se během trénovací fáze pomocí imitačního učení (imitation learning) naučí vyhýbací strategie od expertního centrálního algoritmu a následně v exekuční fázi je používají na základě lokálního okolí a informací od agentů. (Sartoretti et al., 2019)

3 Řešení MAPF pomocí sociálních pravidel

V této sekci si představíme nový přístup k řešení MAPF problému v distribuované verzi bez přímé komunikace mezi agenty. Předpokládáme klasickou verzi MAPF, kde je diskrétní čas a agenti své akce vykonávají ve stejný moment. Každý agent má k dispozici mapu prostředí, ale nezná pozice ostatních agentů, vidí je pouze ve svém lokálním okolí (tzn. má přehled o všech přítomných agentech a jejich pozicích ve svém lokálním okolí). Náš přístup se skládá z plánování tras a aplikací mechanismu k prevenci konfliktů agentů.

V první fázi si každý agent nezávisle naplánuje cestu do cílového vrcholu pomocí **A* algoritmu** (Algoritmus 1) bez ohledu na další agenty. A* algoritmus najde nejkratší cestu mezi startovním a cílovým vrcholem tím, že zohledňuje náklady na dosažení aktuálního uzlu a heuristický odhad zbývajících nákladů na dosažení cílového vrcholu. Jako F skóre pro aktuální uzel nazveme součet těchto hodnot.

Algoritmus 1 Algoritmus A*

```
1:  procedure AStar
2:      uzavrene ← {}
3:      otevrene ← {start}
4:      while otevrene není prázné do
5:          soucasny ← vyberSMinimalnímFSkóre(otevrene)
6:          přidejDoClosed(soucasny)
7:          if soucasny = cil then
8:              return úspěch
9:          end if
10:         for každého souseda soucasneho do
11:             if soused v uzavrene then
12:                 continue
13:             end if
14:             if soused v otevrene then
15:                 nechLepsiCestu()
16:                 continue
17:             end if
18:             přidejDoOpen(soused)
19:         end for
20:     end while
21:     return selhaní
22: end procedure
```

Ve druhé fázi každý agent pokračuje po naplánované cestě a řeší konflikty „za běhu“. Agent se v každém kroku musí rozhodnout, zda může v klidu pokračovat další akcí ze své naplánované trasy, nebo ji změní kvůli hrozícímu konfliktu. Za tímto účelem navrhneme **sociální pravidla**. Pokud se vykonaná akce bude lišit od předem naplánované akce nebo akce *wait*, musí si agent cestu do cíle znovu

naplánovat, k tomu využije opět A* algoritmus (Algoritmus 1).

3.1 Zavedení sociálních pravidel

Nyní zavedeme sociální pravidla v rámci distribuovaného přístupu k MAPF. Inspirací pro tato pravidla jsou dopravní předpisy, které známe z každodenního života. Pravidla cílí na prevenci vzniku konfliktů mezi agenty a současně mají umožnit agentům, aby se dostali do svých cílových vrcholů.

Každé pravidlo se skládá ze dvou částí:

- **předpoklady** – popis, co musí splňovat lokální okolí
- **příkaz** – udává další akce s pravděpodobností jejich uskutečnění

Prověření pravila znamená otestování svého lokálního okolí vůči požadavkům pravidla. Použitím pravidla myslíme výběr akce podle pravděpodobnostní distribuce v příkazu a následné její vykonání. Za sociální pravidla pak označíme uspořádaný seznam takových pravidel.

Agent tedy prověřuje sociální pravidla v daném pořadí a při prvním úspěchu pravidlo použije. Pokud neuspěje s žádným pravidlem, může pokračovat v předem naplánované cestě.

Nyní zavedeme sociální pravidla formálně.

Definice 5 (Sociální pravidla)**.** *Nechť A je množina akcí a F množina výrokových proměnných popisujících lokální okolí agenta, pak pravidlo je dvojice $P = \langle X, Y \rangle$ kde:*

- *$X \subseteq F$ (předpoklady, které musí být pravdivé pro vykonání příkazu),*
- *$Y = \{Z_1 \dots Z_n\}$, kde $Z_i = \langle a_i, p_i \rangle$, $a_i \in A$, $p_i \in [0,1]$, a zároveň platí $\sum_i p_i = 1$ (množina dvojic obsahující akci s danou pravděpodobností).*

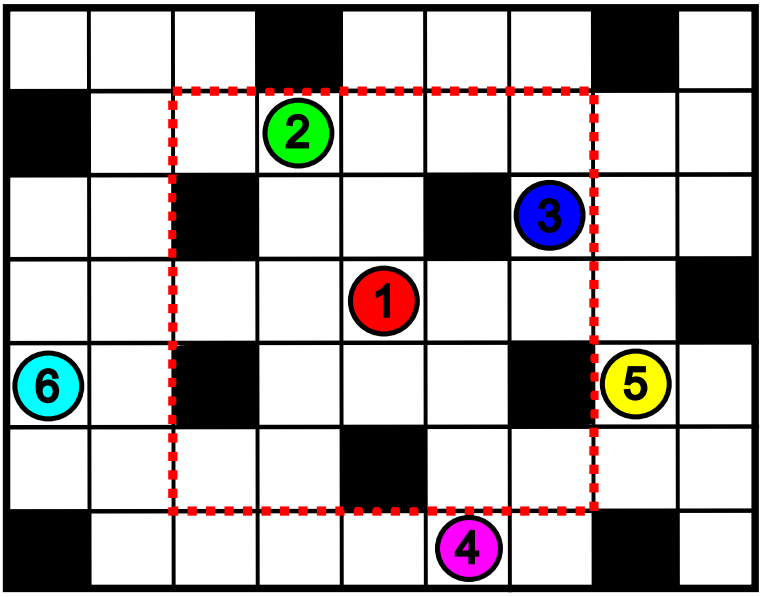
Nakonec sociální pravidla jsou úplně uspořádaná množina $SP = \{P_1 \dots P_n\}$, kde P_i je jedno pravidlo.

3.2 Implementace sociálních pravidel

V této sekci ukážeme implementaci sociálních pravidel pro námi řešené grafy – rovinné mřížkové grafy se čtyřmi sousedy. Lokální okolí agenta bude čtverec o velikosti $2d + 1$, kde d je vzdálenost agenta od hrany okolí. Agent se nachází uprostřed tohoto čtverce. V této práci budeme uvažovat, že $d = 2$ (viz Obrázek 3.1).

Syntax výrokových proměnných. Nejdříve očíslujeme vrcholy v lokálním okolí následujícím způsobem (vzdálenost počítáme maximovou metrikou):

- 0: vrchol, kde se nachází agent
- 1-8: vrcholy ve vzdálenosti 1
- 9-24: vrcholy ve vzdálenosti 2



Obrázek 3.1 Znázornění lokálního okolí agenta na mapě.

Vrchol 1 je ten, kam míří příští naplánovaná akce agenta (pokud je naplánovaná akce *wait*, sociální pravidla nepoužíváme) a 9 je další vrchol ve stejném směru. Číslování ostatních vrcholů funguje vzestupným způsobem ve směru pohybu vpravo po hranách čtverce se středem na pozici 0 a danou vzdáleností. Celkové očíslování je vidět na Obrázku 3.2. Výhoda volby takového číslování spočívá v kompatibilitě pravidel pro různé velikosti lokálního okolí.

23	24	9	10	11
22	8	1	2	12
21	7	0	3	13
20	6	5	4	14
19	18	17	16	15

Obrázek 3.2 Očíslování lokálního okolí agenta.

Každý vrchol se nachází v nějakém stavu, to popíšeme následujícími výrokovými proměnnými, které mají následující syntax a indikují:

- ***A*** (agent) – přítomnost agenta
- ***N*** (notAgent) – nepřítomnost agenta (tzn. je zde průchozí vrchol bez agenta nebo překážka)
- ***O*** (obstacle) – přítomnost překážky
- ***P*** (passable) – přítomnost průchozího vrcholu (tzn. může zde být agent i nemusí)

Výroková proměnná popisující lokální okolí je pak XY, kde X je výroková proměnná popisující vrchol a Y číslo vrcholu.

Syntax jedné části příkazu. Akce uvnitř části příkazu, po jejímž vykonání agent skončí ve vrcholu x, má následující syntax:

- ***S*** (stay): $x = 0$ (*wait* akce)
- ***F*** (forward): $x = 1$
- ***R*** (right): $x = 3$
- ***L*** (left): $x = 7$
- ***B*** (back): $x = 5$

Samotná část příkazu je pak XY, kde X je akce a Y přirozené číslo odpovídající pravděpodobnosti vyjádřené v procentech.

Syntax sociálních pravidel. Sociální pravidlo má syntax vyjádřenou regulárním výrazem:

$$(\backslash(X\backslash))^* > (\backslash(Y\backslash))^*$$

kde X je výroková proměnná popisující lokální okolí a Y je část příkazu.

Příklad:

(A1) (N3) (P3) (N4) (N13) > (R50) (S50)

Toto sociální pravidlo můžeme přeložit do běžné řeči jako: Pokud na pozici 1 je agent, na pozicích 3, 4 a 13 žádný není a pozice 3 je průchozí, tak jdi s padesátiprocentní pravděpodobností doprava a s padesátiprocentní pravděpodobností zůstaň stát.

Jazyk sociálních pravidel. V jazyce sociálních pravidel definujeme každé sociální pravidlo na jeden řádek. Každý takový řádek začíná znakem +, za ním následuje samotné sociální pravidlo. Pravidla se používají v pořadí, v němž jsou napsaná. Ostatní řádky se ignorují.

Použití sociálních pravidel. Předpokládáme, že agent má přístup k informacím o stavech vrcholů v jeho lokálním okolí. V každém kroku musí prověřit předpoklady všech zadaných pravidel. Předpoklady interpretujeme jako konjunkci všech výrokových proměnných, každá proměnná v předpokladu musí být tedy v daném okolí pravdivá. Agent použije první pravidlo s celým pravdivým předpokladem, tzn. vybere akci z pravděpodobnostní distribuce z příkazu. Pokud všechny předpoklady pravidel nejsou pravdivé, pravidla nepoužije, tzn. může pokračovat naplánovanou akcí.

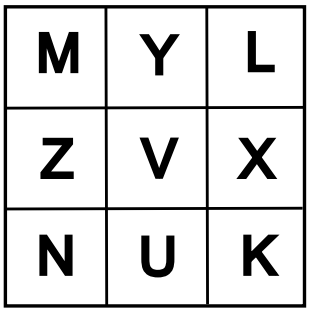
Implementace pravidel umožňuje vyjádřit všechny možné kombinace stavů vrcholů v lokálním okolí agenta a zároveň dovoluje přikázat agentovi všechny možné akce, případně pravděpodobnostní distribuci těchto akcí.

3.3 Návrh sociálních pravidel

V této sekci vytvoříme dva konkrétní návrhy sociálních pravidel, které později experimentálně otestujeme. Uvedli jsme, že tato pravidla jsou inspirována dopravními předpisy, a to nyní využijeme k návrhu pravidel. Konkrétní dopravní předpis, který zkusíme použít, je přednost zprava. Do našeho problému zasadíme přednost zprava vytvořením sady podmínek před vstupem agenta do neobsazeného vrcholu. Dále také zakážeme akce ústící do vrcholu, kde se v současném kroku nachází jiný agent.

Definice 6 (Přednost zprava)**.** *Předpokládejme, že agent A se nachází ve vrcholu U, sousední vrchol nazveme V. Dále nechť X je sousední vrchol V vlevo od U (z pohledu vrcholu V), nechť Y je sousední vrchol V vlevo od X (z pohledu vrcholu V) a nechť Z je sousední vrchol V vlevo od Y (z pohledu vrcholu V). (Označení vrcholů je vidět na Obrázku 3.3.) Pak agent A nesmí vstoupit do V, pokud se v X nachází jiný agent. Zároveň nesmí vstoupit do V, pokud se v Y nachází jiný agent a současně v Z žádný agent není.*

Definice 7 (Rozšířená přednost zprava)**.** *Předpokládejme stejné označení prostředků jako v definici (6) a navíc nechť K, L, M a N jsou v tomto pořadí společné sousední neoznačené vrcholy dvojic (U,X), (X,Y), (Y,Z) a (Z,U). (Označení vrcholů je vidět na Obrázku 3.3.) Pak v této rozšířené variantě povolujeme navíc oproti původní verzi (Definice 6) vstoupit agentovi A do V, pokud není jiný agent v X, a zároveň pokud v Y je agent, v K je překážka a v L překážka není. Dále povolujeme agentovi A vstoupit do V, pokud jsou agenti ve vrcholech X, Y a Z, současně v K je překážka a v L, M, N překážky nejsou.*



Obrázek 3.3 Označení vrcholů v definicích předností zprava.

Definice 8 (Zákaz vstupu do obsazeného vrcholu)**.** *Agentovi zakážeme v čase T akci, jejímž použitím skončí ve vrcholu U v čase T+1, pokud se jiný agent nachází v U v čase T.*

Definice 9. *Zablokovaní (deadlock) znamená neřešitelnou situaci, kdy si více agentů vzájemně zablokuje cestu. Jeho důvodem je zacyklené používání stejného sociálního pravidla.*

Za základní variantu předpisů uvažujeme kombinaci přednosti zprava a zákazu vstupu do obsazeného vrcholu. V rozšířené verzi nahradíme přednost zprava rozšířenou předností zprava. Za dodržení jedné z variant předpisů budeme designovat konkretní sociální pravidla.

Při konstrukci pravidel se snažíme o univerzální fungovaní pro všechny prostředí. Nechceme tedy pravidla, která by fungovala dobře v jednom prostředí, ale v jiném by naprosto selhávala.

První návrh (Program 1) obsahuje Deterministická sociální pravidla (DSP), která implementují základní variantu předpisů. Jejich dodržení zajistíme zkontrolováním tří kritických vrcholů v okolí agenta - vrcholy 1,2 a 9. Dosáhneme toho aplikací sociálních pravidel 1-9. Pokud se v některém z kritických vrcholů nachází jiný agent, musíme změnit naplánovanou akci. Při snaze o změnu akce ale musíme neustále kontrolovat podmínky k dodržení předpisů. Nejdříve ověříme možnost jít doprava, pokud neuspějeme, zkusíme možnost jít dozadu, poslední variantou je akce *wait*, která podmínky neporušuje nikdy.

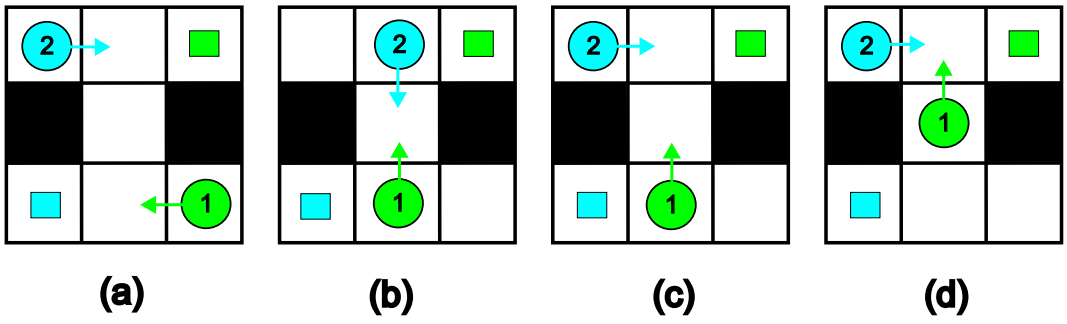
V prvním návrhu jsme zároveň přidali sociální pravidla ke koordinaci agentů a prevenci některých zablokování (pravidla 10-14).

Naše představa o fungování této sady pravidel je taková, že se agenti budou vyhýbat vzájemným obcházením vpravo z pohledu agenta.

Program 1 Návrh deterministických sociálních pravidel

```
 1: + (A1) (N3) (P3) (N4) (N13) > (R100)
 2: + (A1) (N5) (P5) (N6) (N17) > (B100)
 3: + (A1) > (S100)
 4: + (A2) (N3) (P3) (N4) (N13) > (R100)
 5: + (A2) (N5) (P5) (N6) (N17) > (B100)
 6: + (A2) > (S100)
 7: + (A9) (N3) (P3) (N4) (N13) > (R100)
 8: + (A9) (N5) (P5) (N6) (N17) > (B100)
 9: + (A9) > (S100)
10: + (A10) (N3) (P3) (N4) (N13) > (R100)
11: + (A12) (N3) (P3) (N4) (N13) > (R100)
12: + (A3) (N5) (P5) (N6) (N17) > (B100)
13: + (A11) (N3) (P3) (N4) (N13) > (R100)
14: + (A13) (N5) (P5) (N6) (N17) > (B100)
```

Druhý návrh (Program 2) obsahuje Stochastická sociální pravidla (SSP), která uplatňují rozšířenou verzi předpisů. Dodržení předpisů funguje na stejném principu jako v prvním návrhu, tato pravidla (1-33) však navíc přidávají možnost jít doleva. Zbytek sociálních pravidel (34-38) se opět stará o prevenci některých zablokování.



Obrázek 3.4 Příklad instance problému řešitelné pouze pomocí SSP.

Nyní ukážeme, že stochastičnost pravidel umožňuje lepší prevenci vzniku zablokování. Uvažujme například instanci na Obrázku 3.4 (šipky zde značí příští

naplánovanou akci pro agenta, čtverce cílový vrchol pro agenta v dané barvě). Instance začíná ve fázi (a), po první akci se dostane do fáze (b). Nyní potřebujeme zajistit přerušení symetrického chování mezi agenty 1 a 2. Deterministická pravidla to zajistit nedokáží, protože oba agenti vidí stejné lokální okolí, a proto v přístupu s DSP dojde k zablokování. SSP naopak umožňuje přejít do fáze (c) a rozbít tak symetrii. Následně můžou agenti pokračovat do fáze (d) a dále do svých cílů bez vzájemné kolize.

3.3.1 Vlastnosti sociálních pravidel

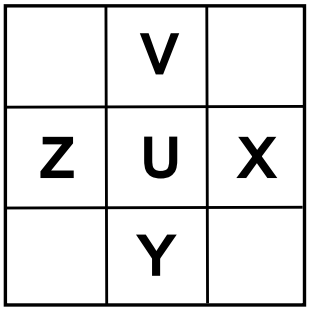
Lemma 1 (Bezkonfliktnost 1)**.** *Bezkonfliktnost zajistíme aplikováním přednosti zprava (Definice 6) a zákazu vstupu do obsazeného vrcholu (Definice 8).*

Důkaz. Označme přednost zprava jako PZ a zákaz vstupu na obsazený vrchol jako ZVOV. Předpokládáme, že na začátku je každý agent v jiném vrcholu. Pak musíme ošetřit oba zakázané konflikty:

1. Výměnný konflikt – neumožňuje ZVOV.
2. Vrcholový konflikt – mohou nastat 2 situace:
   (a) Agent se dostane do vrcholu, kde se již dříve nacházel jiný agent – neumožňuje ZVOV.
   (b) Agenti se akcí *move* dostanou do dříve neobsazeného vrcholu U ve stejný čas T.

   Označme sousedy U ve směru hodinových ručiček V, X, Y, Z (viz Obrázek 3.5). Nyní BÚNO vyberme vrchol V a předpokládejme, že platí PZ a ZVOV a že agent A je v čase $T-1$ ve V a vykoná akci, po které nastane konflikt s jiným agentem B ve vrcholu U. Pak B se musel nacházet v čase $T-1$ ve vrcholu X, Y nebo Z. Pokud B byl v X, B nemohl vykonat akci kvůli PZ. Pokud B byl v Z, A nemohl vykonat akci kvůli PZ. Pokud B byl v Y a zároveň X nebo Z byl obsazený, jeden z agentů nemohl vykonat akci kvůli PZ. Pokud byl B v Y a zároveň X a Z nebylo obsazené, ani jeden nemohl vykonat akci kvůli PZ. Spor s předpokladem.

□



Obrázek 3.5 Označení vrcholů v důkazu bezkonfliktnosti.

Lemma 2 (Bezkonfliktnost 2)**.** *Bezkonfliktnost také zajistíme aplikováním rozšířené přednosti zprava (Definice 7) a zákazu vstupu do obsazeného vrcholu (Definice 8).*

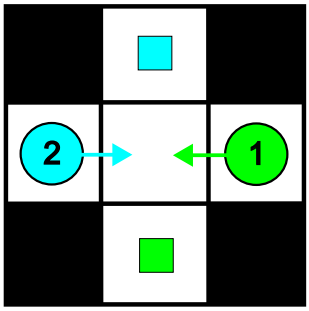
Důkaz. Analogicky podle důkazu lemmatu 1. □

Lemma 3 (Robustnost)**.** *Aplikováním zákazu vstupu do obsazeného vrcholu (Definice 7) zajistíme robustnost vůči zpoždění agenta.*

Důkaz. Zpoždění agenta ve vrcholu V v čase T může vyvolat konflikt jen v případě, kdyby jiný agent chtěl aplikování akce v čase T vstoupit do V, to mu ale zákaz vstupu do obsazeného vrcholu zakazuje. □

Oba naše návrhy implementují přednost zprava (popř. rozšířenou přednost zprava) a zákaz vstupu do obsazeného vrcholu, tudíž, pokud nedojde k zablokování, bude naše řešení bezkonfliktní a robustní vůči zpoždění agenta (podle Lemmat 1, 2 a 3).

Neřešitelnost některých instancí. Některé instance MAPF budou neřešitelné sociálními pravidly. Může dojít k zablokování, kdy dva agenti naproti sobě vidí symetrické lokální okolí a nemohou si dát přednost, ani se jinak vyhnout (viz Obrázek 3.6).



Obrázek 3.6 Příklad neřešitelné instance problému pomocí sociálních pravidel.

Program 2 Návrh stochastických sociálních pravidel

```
 1: + (A1) (N3) (P3) (N4) (N13) > (R50) (S50)
 2: + (A1) (N5) (P5) (N6) (N17) > (B50) (S50)
 3: + (A1) (N7) (P7) (N8) (N21) > (L50) (S50)
 4: + (A1) (N3) (P3) (A2) (A13) (N4) > (R50) (S50)
 5: + (A1) (N3) (P3) (O5) (A13) (P12) (N4) > (R50) (S50)
 6: + (A1) (N3) (P3) (A2) (P1) (O5) (A13) (P12) (A4) (P14) > (R50) (S50)
 7: + (A1) (N5) (P5) (A4) (A17) (N6) > (B50) (S50)
 8: + (A1) (N5) (P5) (O7) (A17) (P16) (N6) > (B50) (S50)
 9: + (A1) (N5) (P5) (A4) (P3) (O7) (A17) (P16) (A6) (P18) > (B50) (S50)
10: + (A1) > (S100)
11: + (A2) (N3) (P3) (N4) (N13) > (R50) (S50)
12: + (A2) (N5) (P5) (N6) (N17) > (B50) (S50)
13: + (A2) (N7) (P7) (N8) (N21) > (L50) (S50)
14: + (A2) (N1) (P1) (A8) (P7) (O3) (A9) (P24) (A2) (P10) > (F50) (S50)
15: + (A2) (N3) (P3) (A2) (A13) (N4) > (R50) (S50)
16: + (A2) (N3) (P3) (O5) (A13) (P12) (N4) > (R50) (S50)
17: + (A2) (N3) (P3) (A2) (P1) (O5) (A13) (P12) (A4) (P14) > (R50) (S50)
18: + (A2) (N5) (P5) (A4) (A17) (N6) > (B50) (S50)
19: + (A2) (N5) (P5) (O7) (A17) (P16) (N6) > (B50) (S50)
20: + (A2) (N5) (P5) (A4) (P3) (O7) (A17) (P16) (A6) (P18) > (B50) (S50)
21: + (A2) > (S100)
22: + (A9) (N3) (P3) (N4) (N13) > (R50) (S50)
23: + (A9) (N5) (P5) (N6) (N17) > (B50) (S50)
24: + (A9) (N7) (P7) (N8) (N21) > (L50) (S50)
25: + (A9) (N1) (P1) (A8) (A9) (N2) > (F50) (S50)
26: + (A9) (N1) (P1) (O3) (A9) (P24) (N2) > (F50) (S50)
27: + (A9) (N3) (P3) (A2) (A13) (N4) > (R50) (S50)
28: + (A9) (N3) (P3) (O5) (A13) (P12) (N4) > (R50) (S50)
29: + (A9) (N3) (P3) (A2) (P1) (O5) (A13) (P12) (A4) (P14) > (R50) (S50)
30: + (A9) (N5) (P5) (A4) (A17) (N6) > (B50) (S50)
31: + (A9) (N5) (P5) (O7) (A17) (P16) (N6) > (B50) (S50)
32: + (A9) (N5) (P5) (A4) (P3) (O7) (A17) (P16) (A6) (P18) > (B50) (S50)
33: + (A9) > (S100)
34: + (A8) (O7) (O2) (O9) (P3) (N3) (N4) (N13) > (R50) (S50)
35: + (A8) (O7) (O2) (O9) (P5) (N5) (N6) (N17) > (B50) (S50)
36: + (A10) (N3) (P3) (N4) (N13) (P2) > (R50) (S50)
37: + (A12) (N3) (P3) (N4) (N13) (P2) > (R50) (S50)
38: + (A3) (N5) (P5) (N6) (N17) > (B50) (S50)
```

4 Experimenty

V této kapitole budeme testovat sady sociálních pravidel navržené v kapitole 3.3 v simulovaném prostředí. Za tímto účelem byla naprogramována aplikace MAPF Simulator for Social Laws dostupná v příloze spolu s uživatelskou dokumentací (A.2).

V kapitole představíme hypotézy týkající se našich návrhů sociálních pravidel a následně je experimentálně potvrdíme, nebo vyvrátíme.

4.1 Instance

Instance MAPF problému, které budeme využívat, pochází z publikované sady MAPF benchmarků popsané v článku Stern et al. (2019). Jednotlivé instance obsahují mapu a k ní náležící množinu scénářů obsahujících startovní a cílové pozice agentů při jejich rovnoměrném nebo náhodném uspořádání v mapě.

4.2 Porovnání návrhů sociálních pravidel

Hypotézy. V této kapitole budeme ověřovat tři hypotézy. V první hypotéze předpokládáme, že pomocí našeho návrhu DSP jsou řešitelné instance problému s mapou bez překážek a s rozumným počtem agentů vzhledem k velikosti mapy. Řešitelné znamená, že za běhu nedojde k zablokování mezi agenty a všichni dorazí do svých cílů. Za rozumnou obsazenost mapy agenty uvažujeme obsazenost 5 procent všech vrcholů.

Druhá hypotéza je obdoba první formulovaná pro SSP. Ve třetí hypotéze budeme tvrdit, že na stejných instancích problému bude řešení za použití SSP lepší než řešení za použití DSP.

Nastavení. Hypotézy ověříme na mapě s názvem *empty-32-32*. Jde o prázdnou čtvercovou mřížku o velikosti 32x32. Sady pravidel otestujeme na 25 náhodně generovaných scénářích a různých počtech agentů. K evaluaci kvality řešení, zavedeme průměrný Makespan a průměrný SoC ze všech 25 instancí pro daný počet agentů. Úspěšnost určuje poměr vyřešených instancí z celkových 25.

Počet agentů	Úspěšnost	Průměrný Makespan	Průměrný SoC
10	1,00	44,1	227,2
20	0,96	50,3	501,4
30	0,92	55,7	836,6
40	0,84	59,6	1204,0
50	0,84	72,2	1670,6

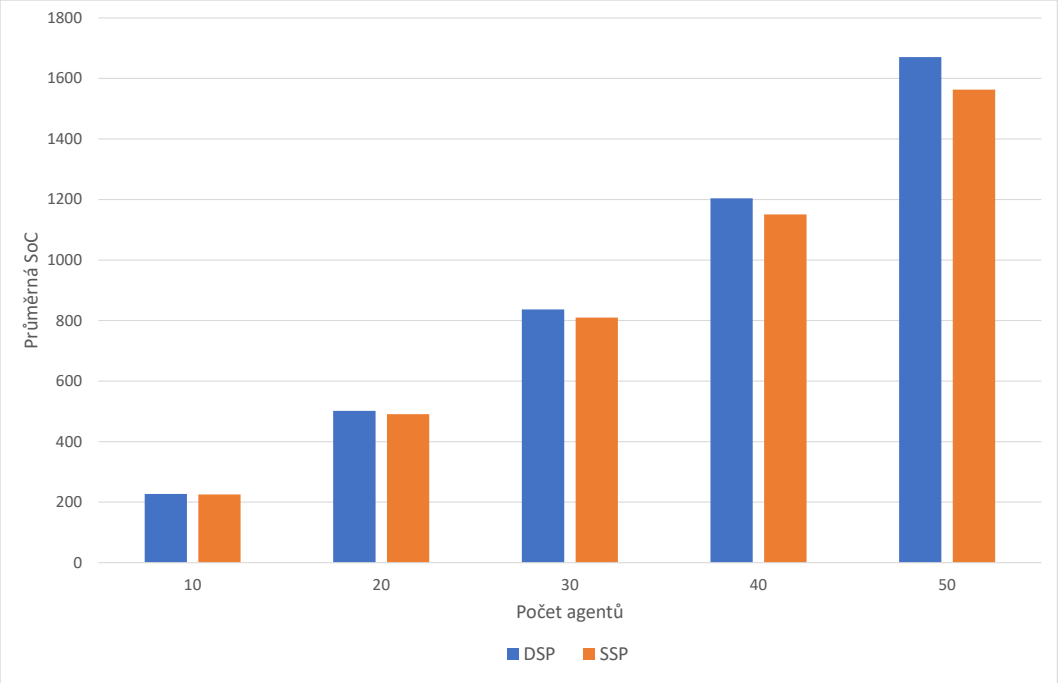
Tabulka 4.1 Výsledky DSP na mapě *empty-32-32*.

Počet agentů	Průměrný Makespan	Průměrný SoC
10	42,3	225,4
20	49,6	490,8
30	56,1	809,9
40	63,0	1150,8
50	66,8	1563,3

Tabulka 4.2 Výsledky SSP na mapě *empty-32-32*.

Výsledky. Z výsledků experimentu s DSP (viz Tabulka 4.1) je patrné, že stoprocentní úspěšnost byla pouze u instancí s 10 agenty, se zvyšujícím se počtem agentů se úspěšnost snižovala. První hypotézu proto zamítáme. Naopak SSP vyřešila všechny instance, proto druhou hypotézu potvrzujeme. Průměrné ceny řešení využívající SSP nalezneme v Tabulce 4.2.

Přehledné porovnání sad sociálních pravidel na vyřešených instancích vidíme v grafu na Obrázku 4.1. Z něj je zřejmé, že SSP je lepší pro všechny počty agentů, tudíž i třetí hypotézu potvrzujeme.



Obrázek 4.1 Porovnání výsledků DSP a SSP na mapě *empty-32-32*.

Diskuze. Z výsledků simulace můžeme usoudit, že návrh DSP nefunguje dobře ani na prázdných mapách. Přidáním překážek problém ještě zkomplikujeme, a proto budeme dále testovat pouze stochastickou variantu sociálních pravidel.

4.3 Testování v náhodném prostředí

Hypotéza. Celkově čtvrtou hypotézou je, že pomocí sady SSP lze vyřešit instance na mapě *random-32-32-20* s rozumným počtem agentů vzhledem k velikosti mapy. Uvedená mapa simuluje prostředí, kde se mohou náhodně vyskytovat překážky. Opět požadujeme, aby všechny instance byly vyřešené a nedošlo k žádným zablokováním mezi agenty.

Nastavení. Pro ověření hypotézy použijeme, jak bylo zmíněno, mapu *random-32-32-20*. Jde o čtvercovou mřížku o velikosti 32x32, kde 20 procent mapy zabírají náhodně rozmístěné překážky. Sadu pravidel opět otestujeme na 25 náhodných scénářích a různých počtech agentů. K ověření efektivity řešení zavedeme průměrný Makespan a průměrný SoC ze všech 25 instancí pro daný počet agentů.

Počet agentů	Průměrný Makespan	Průměrný SoC
10	45,8	255,2
20	59,7	570,4
30	67,2	930,2
40	80,8	1420,4
50	91,8	1914,2

Tabulka 4.3 Výsledky SSP na mapě *random-32-32-20*.

Výsledky. Simulace proběhla úspěšně na všech daných instancích problémů, čtvrtá hypotéza byla tudíž potvrzena. (Pro konkrétní naměřené výsledky řešení viz Tabulka 4.3.)

Diskuze. Výsledky ukazují, že ani náhodně rozmístěné překážky našemu způsobu řešení problém nedělají a na mapách s rozumným počtem agentů SSP fungují dobře.

4.4 Testování škálovatelnosti

Hypotézy. V páté hypotéze předpokládáme, že pomocí našeho návrhu SSP jsou řešitelné níže určené instance s prostředím, kde se nachází větší počet agentů. Šestou hypotézou je, že se poměr SoC optimálního řešení vůči SoC našeho řešení nebude se zvyšujícím se počtem výrazně měnit.

Nastavení. Hypotézy ověříme na 4 mapách. První dvě jsou již představené *empty-32-32* a *random-32-32-20*. Na těchto mapách otestujeme sadu pravidel na 25 náhodně generovaných scénářích a různých vyšších počtech agentů. Další dvě jsou mapy designované ve stylu reálného skladiště - mapa *warehouse-10-20-10-2-1* s uličkami o šířce 1 a mapa *warehouse-10-20-10-2-2* s uličkami o šířce 2. Zde otestujeme sadu pravidel na 5 náhodných scénářích a různých vyšších počtech agentů. K evaluaci zavedeme průměrný Makespan a průměrný SoC ze všech

simulovaných instancí pro daný počet agentů. Dále zavedeme hodnocení jako poměr průměrného SoC optimálního řešení a průměrného SoC našeho řešení ze všech simulovaných instancí pro daný počet agentů. Optimální řešení jsme získali z databáze řešení pro MAPF benchmarky představené v článku Shen et al. (2023).

Počet agentů	**Průměrný Makespan**	**Průměrný SoC**	**Hodnocení**
100	94,9	4576,5	0,47
150	156,3	10832,2	0,30
200	207,3	20312,3	0,21
250	270,0	33456,4	0,16
300	385,0	58683,0	0,12
350	518,0	94440,6	0,09
400	731,0	157456,7	0,06
450	950,8	240494,1	0,05
500	1417,7	417161,2	0,04

Tabulka 4.4 Výsledky SSP na mapě *empty-32-32* s vyšším počtem agentů.

Počet agentů	**Průměrný Makespan**	**Průměrný SoC**	**Hodnocení**
100	164,1	6280,4	0,37
150	259,0	15293,2	0,24
200	378,8	30665,0	0,17
250	566,3	60069,3	0,12
300	738,1	102401,6	0,10
350	1004,7	167805,0	0,08
400	1306,8	266376,0	0,11

Tabulka 4.5 Výsledky SSP na mapě *random-32-32-20* s vyšším počtem agentů.

Počet agentů	**Průměrný Makespan**	**Průměrný SoC**	**Hodnocení**
100	719,8	15425,8	0,54
300	1392,4	81033,8	0,30
500	2992,6	328372,0	0,13
700	7120,2	1038800,4	0,07

Tabulka 4.6 Výsledky SSP na mapě *warehouse-10-20-10-2-1*.

Výsledky. Simulace opět proběhla úspěšně na všech daných instancích problémů, a proto pátou hypotézu můžeme potvrdit. Z výsledků (viz Tabulky 4.4, 4.5, 4.6 a 4.7) je ale patrné, že cena řešení se s přibývajícím počtem agentů násobně zvyšuje oproti optimálnímu řešení, a proto šestou hypotézu musíme zamítnout.

Počet agentů	Průměrný Makespan	Průměrný SoC	Hodnocení
100	211,4	9866,0	0,95
300	233,6	30503,0	0,86
500	300,6	61432,6	0,72
700	481,6	109636,2	0,58
900	573,8	173583,2	0,51

Tabulka 4.7 Výsledky SSP na mapě *warehouse-10-20-10-2-2*.

Diskuze. Z výsledků je patrné, že kvalita řešení s aplikací SSP se výrazně zhoršuje na mapách s extrémně velkým počtem agentů vzhledem k velikosti mapy. Zároveň vidíme, že na mapě *warehouse-10-20-10-20-1* trvá dlouhou dobu než se větší množství agentů vzájemně vyhne (viz Tabulka 4.6), ale rozšířením uličky se hodnocení nalezeného řešení (viz Tabulka 4.7) výrazně zlepší. Můžeme z toho usoudit, že SSP dobře fungují na mapách, kde není těžké se pro agenty vzájemně vyhnout, zároveň však umožňují nalezení alespoň nějakého řešení i v obtížnějších instancích problému.

Závěr

V této práci jsme se zabývali problémem multi-agetního hledání cest (MAPF), což je problém hledání bezkolizních cest pro agenty ve sdíleném prostředí. Problém jsme nejprve formálně zadefinovali a představili si pojmy s ním spojené. Dále jsme se podívali na způsoby, kterými se dá problém řešit. Centralizovaný způsob dokáže najít optimální řešení, ale je neefektivní na větších instancích a při omezených informacích o ostatních agentech je nefunkční. Distributivní přístup naopak pracuje jen s lokální informací. V rámci tohoto přístupu jsme představili nový systém k řešení problému, který nevyužívá přímou komunikaci mezi agenty, ale spoléhá na tzv. sociální pravidla (v překladu social laws).

Sociální pravidla jsme v rámci MAPF problému formálně zavedli a představili jsme formální jazyk, ve kterém lze zadávat konkrétní sady pravidel do naprogramované aplikace MAPF Simulator for Social Laws. Následně jsme navrhli dvě sady sociálních pravidel a experimentálně je otestovali na různých instancích problému.

První sada (Program 1) deterministických pravidel (DSP) nefungovala zcela podle představ, a proto jsme ji po testování na prázdné mapě zavrhli. Druhá sada (Program 2) stochastických pravidel (SSP) díky lepší prevenci vzniku zablokování fungovala mnohem lépe.

Ukázali jsme, že SSP zvládají vyřešit i obtížné instance problému, kde se na malém prostoru nachází mnoho agentů. Zároveň fungují i v prostředí s náhodně rozmístěnými překážkami. Z pohledu kvality řešení jsou však nejefektivnější na mapách, kde agentům nedělá problém se vzájemně vyhnout v jejich lokálním okolí.

Předmětem dalšího studia by mohlo být vyzkoušení sociálních pravidel s větším lokálním okolím nebo přidání definovaných priorit mezi agenty, které by mohly nejen zefektivnit některá řešení problému ale také rozšířit množinu řešitelných instancí.

Literatura

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; Zhou, N.-F., 2018. Robust Multi-Agent Path Finding. In: *Proceedings of the 11th International Symposium on Combinatorial Search*. Sv. 9, s. 2–9. Č. 1.

Barták, R.; Švancara, J.; Škopková, V.; Nohejl, D., 2019. Multi-agent Path Finding on Real Robots. *AI Communications*. Roč. 32, č. 3, s. 175–189.

Karpas, E.; Shleyfman, A.; Tennenholtz, M., 2017. Automated Verification of Social Law Robustness in STRIPS. *Proceedings of the International Conference on Automated Planning and Scheduling*. Roč. 27, č. 1, s. 163–171.

Luna, R.; Bekris, K., 2011. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. *IJCAI International Joint Conference on Artificial Intelligence*, s. 294–300.

Nir, R.; Shleyfman, A.; Karpas, E., 2021. Learning-based synthesis of social laws in STRIPS. In: *Proceedings of the International Symposium on Combinatorial Search*. Sv. 12, s. 88–96. Č. 1.

Nir, R.; Shleyfman, A.; Karpas, E., 2023. Automated Verification of Social Laws in Numeric Settings. *Proceedings of the AAAI Conference on Artificial Intelligence*. Roč. 37, č. 10, s. 12087–12094.

Okumura, K.; Bonnet, F.; Tamura, Y.; Défago, X., 2022. *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. Offline Time-Independent Multi-Agent Path Planning. International Joint Conferences on Artificial Intelligence Organization. IJCAI-2022.

Okumura, K.; Machida, M.; Défago, X.; Tamura, Y., 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*. Roč. 310, s. 103752.

Sartoretti, G.; Kerr, J.; Shi, Y.; Wagner, G.; Kumar, T. K. S.; Koenig, S.; Choset, H., 2019. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. *IEEE Robotics and Automation Letters*. Roč. 4, č. 3, s. 2378–2385.

Sharon, G.; Stern, R.; Felner, A.; Sturtevant, N. R., 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*. Roč. 219, s. 40–66.

Sharon, G.; Stern, R.; Goldenberg, M.; Felner, A., 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*. Roč. 195, s. 470–495.

Shen, B.; Chen, Z.; Cheema, M. A.; Harabor, D. D.; Stuckey, P. J., 2023. Tracking progress in multi-agent path finding. *arXiv preprint arXiv:2305.08446*.

Shoham, Y.; Tennenholtz, M., 1995. On social laws for artificial agent societies: off-line design. *Artificial Intelligence*. Roč. 73, č. 1, s. 231–252.

Silver, D., 2005. Cooperative Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Roč. 1, č. 1, s. 117–122.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li J.and Atzmon, D.; Cohen, L.; Satish Kumar, T. K. S.; Boyarski, E.; Barták, R., 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In: *Proceedings of the International Symposium on Combinatorial Search*. Sv. 10, s. 151–158. Č. 1.

Surynek, P., 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. *Proceedings of the AAAI Conference on Artificial Intelligence*. Roč. 24, č. 1, s. 1261–1263.

Surynek, P.; Felner, A.; Stern, R.; Boyarski, E., 2016. *ECAI 2016*. Efficient SAT approach to multi-agent path finding under the sum of costs objective. IOS Press.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

DSP Deterministická sociální pravidla

MA-STRIPS Multi-Agent STRIPS

Makespan Celková délka plánů

MAP Multi-Agent Planning

MAPF Multi-Agent Path-Finding

SoC Sum of costs

SSP Stochastická sociální pravidla

STRIPS Stanford Research Institute Problem Solver

A Přílohy

A.1 Obsah elektronické přílohy

Elektronická příloha obsahuje:

- *MAPF Simulator for Social Laws* - adresář se zdrojovým kódem aplikace k testování sociálních pravidel
- *testovací vstupy* - adresář se vstupními soubory instancí problémů testovaných v experimentální části a soubory s návrhy sociálních pravidel

A.2 Uživatelská dokumentace k MAPF Simulator for Social Laws

MAPF Simulator for Social Laws je aplikace vytvořená pomocí frameworku Unity a napsaná v programovací jazyce C#. Slouží k simulaci konkrétních instancí MAPF problému s možností vložení vlastních sociálních pravidel. Aplikace zobrazuje průběh celé simulace problému a poskytuje uživateli její statistiky.

A.2.1 Jak aplikaci spustit

K sestavení aplikace je potřeba mít nainstalovaný framework Unity. Dále aplikace používá interní balíček TextMesh Pro a externí StandaloneFileBrowser (k dispozici na `https://github.com/gkngkc/UnityStandaloneFileBrowser`), které je potřeba před sestavením stáhnout. Nejdříve otevřeme složku se zdrojovými kódy v Unity, následně do projektu přidáme oba balíčky, pak můžeme aplikaci sestavit a spustit. Aplikace je otestovaná pro verze na Windows a MacOS Intel.

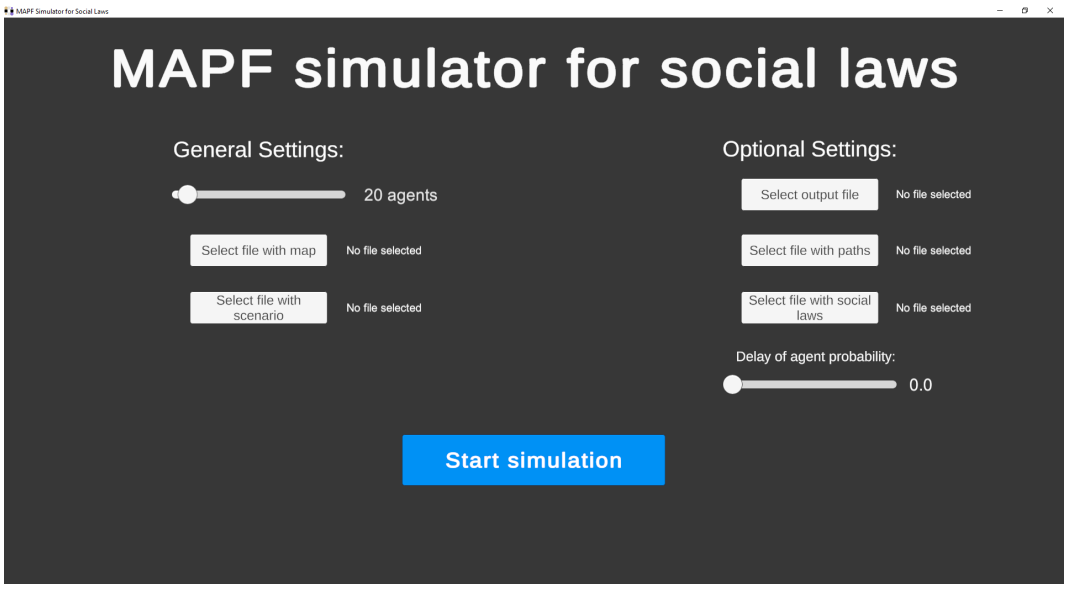
Další možností je nainstalování aplikace pro Windows pomocí instalačního souboru, který se nachází v podadresáři *Installer*.

A.2.2 Jak nastavit parametry

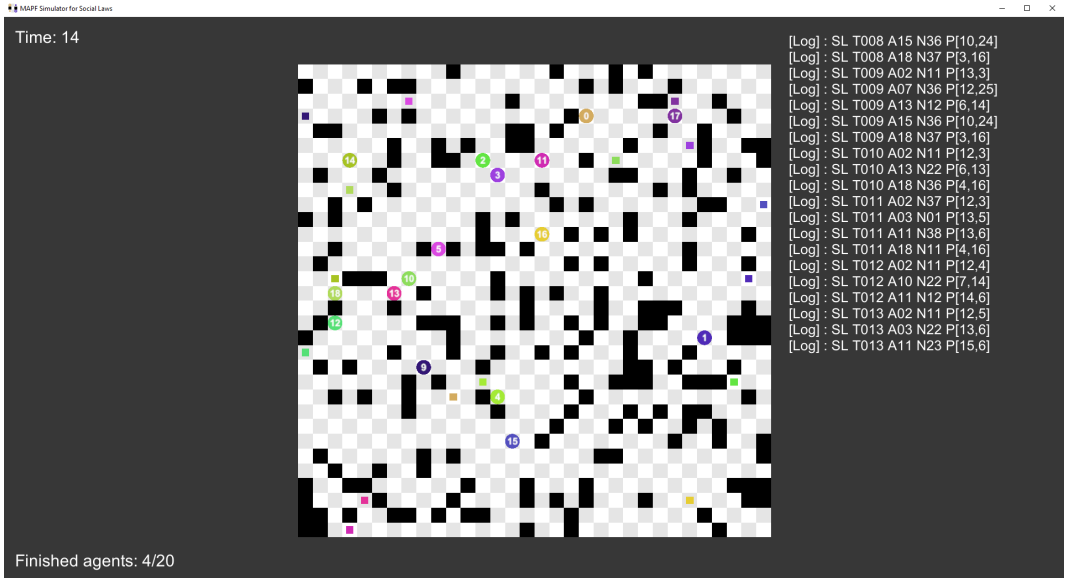
Po spuštění aplikace se dostaneme do menu s nastavením (viz Obrázek A.1). Zde lze nastavit jednotlivé parametry instance problému. Po kliknutí na odpovídající tlačítko můžeme vybrat konkrétní vstupní soubory s mapou, scénáři i sociálními pravidly z dialogového okna. Po úspěšném načtení souboru se zobrazí jeho název vedle tlačítka. Počet agentů lze nastavit posuvníkem. Menu případně umožňuje nastavit i pravděpodobnost zpoždění agenta, načtení souboru s předem nalezenými cestami pro agenty a výstupní soubor, kam se vypíší statistiky proběhlé simulace. Samotnou simulaci spustíme tlačítkem ve spodní části menu.

A.2.3 Běh simulace

Simulované prostředí můžeme vidět na Obrázku A.2. Mapa se skládá ze světlých políček označujících průchozí vrchol a černých označující překážku. Agenti jsou zobrazeni v rozdílných barvách a se specifickými čísly. Ke každému agentovi náleží



Obrázek A.1 Menu aplikace, kde lze nastavit parametry instance problému.



Obrázek A.2 Scéna aplikace, kde probíhá simulace běhu agentů.

jeden cíl zobrazený jako menší čtverec ve stejné barvě. V horní části okna je zobrazeno počítadlo časových kroků, dole pak statistika dokončených cest agentů. V pravé části okna se nachází výpis s informacemi potřebnými během simulace.

Simulace se po chvíli samovolně spustí. Zastavíme ji klávesou *Space* a následně posouváme po krocích šipkou doprava (*RightArrow*). Opět spustit automatický běh lze znovu klávesou *Space*. Restartovaní aplikace pro možnost zadaní nové instance problému provedeme klávesou *R*. Úplné vypnutí aplikace zařídíme klávesou *Esc*.