VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

KOOPERACE MULTIAGENTNÍ SKUPINY V DYNAMICKÉM PROSTŘEDÍ

COOPERATION OF MULTIAGENT GROUP IN DYNAMIC ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PAVEL BALAJKA**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Balajka Pavel, Bc.**
Program: Informační technologie
Obor: Inteligentní systémy
Název: **Kooperace multiagentní skupiny v dynamickém prostředí**
Cooperation of Multiagent Group in Dynamic Environment
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s pravidly soutěže Multiagent Contest tak, jak byly vyhlášeny pro rok 2020.
2. Ve vhodném modelovacím nástroji, jako je Prometheus, JACK, nebo v podobném navrhněte scénáře, cíle, role a další patřičné části multiagentního systému, který má úspěšně řešit zadaný problém.
3. Tyto scénáře realizujte v systému JaCaMo.
4. Ověřte funkčnost systému v porovnání s fungováním týmů, které se tento rok soutěže zúčastnili.
5. Diskutuje pozitiva a negativa vašeho návrhu, a také rozeberte silné a slabé stránky systému JaCaMo

Literatura:

- Padgham, L., Winikoff. M.:Developing Intelligent Agent Systems: A Practical Guide, Wiley, 2004
- Wooldridge, M.: An Introduction to MultiAgent Systems, Wiley, 2009

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání

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

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Práce shrnuje důkladné prostudování pravidel a mechanismů soutěže *Multiagent Contest* pro rok 2020 a následný návrh a implementaci multiagentní skupiny, která se bude snažit v simulaci zmíněné soutěže uspět s co nejlepšími výsledky. Návrh byl tvořen dle metodologie *Prometheus* od obecné roviny (scénáře, cíle) až ke konkrétním detailům (plány, zprávy). Implementace je uskutečněna v systému *JaCaMo*, který je v práci také stručně popsán. Jsou rozebrány zejména jeho klady a zápory. Nakonec je vlastní řešení porovnáno s ostatními týmy, které se zmíněné soutěže tento rok účastnily.

Abstract

The thesis summarises thorough analysis of rules and mechanisms of *Multiagent Contest* for year 2020 followed by design of multiagent group which will try to succeed in a simulation of the mentioned contest with the best results. The design was created according to the *Prometheus* metodology from general view (scenarios, goals) to specific details (agent plans, messages). Implementation is realized through *JaCaMo* system, which is also briefly described in this paper. Then the pros and cons of *JaCaMo* system are analyzed. Lastly the implemented solution of this work is compared with other teams that participated in the mentioned agent competition this year.

Klíčová slova

multiagentní systém, agent, prometheus, soutěž, kooperace, jacamo

Keywords

multiagent system, agent, prometheus, contest, cooperation, jacamo

Citace

BALAJKA, Pavel. *Kooperace multiagentní skupiny v dynamickém prostředí.* Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

Kooperace multiagentní skupiny v dynamickém prostředí

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana docenta Františka Zbořila. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....

Pavel Balajka

16. května 2021

Obsah

Kapitola 1

Úvod

Práce předpokládá alespoň základní znalosti agentních systémů jak je sepsal např. Wooldridge [13]. Text je prolínán i grafickou dokumentací, zejména v oblasti návrhu, ze které lze taktéž vyčíst užitečné informace o součástech systému. V diagramech je kladen důraz na přehlednost. Téměř všechny použité obrázky a diagramy jsou vlastní tvorbou. U převzatých obrázků je vždy uveden zdroj v jejich popise.

Každoročně probíhá soutěž agentních systémů *Multi-Agent Programming Contest* popsaná v kapitole 2. Cílem práce je navrhnout a implementovat agentní systém, který bude úspěšně řešit problém definovaný soutěží. Je také vhodné aby byl navržený systém nejen přijatelný, ale také schopný porazit ostatní týmy v soutěži.

Kapitola 3 pojednává o detailním návrhu systému pomocí metodologie *Prometheus*, která poskytuje podrobný návod, jak postupovat při návrhu jednotlivých částí agentního systému, v jakém pořadí a jak následně části sestavovat dohromady.

Pro implementaci byl vybrán systém *JaCaMo* o jehož vlastnostech a součástech informuje kapitola 4. Je zde obecný úvod, o jaký typ systému se jedná, na čem je založen, jaké jsou jeho součásti a jak spolu dohromady fungují, atd. Ke konci kapitoly jsou shrnuty některé stinné i světlé stránky systému *JaCaMo*.

V další kapitole 5 jsou podrobněji rozepsány některé zajímavé implementační problémy a jejich řešení spolu s důležitými častmi kódu, které stojí za zmínění. Pozdější část kapitoly se věnuje návrhům pro zlepšení dosavadní implementace řešení a jejich detailnějšímu rozboru.

Poslední kapitola 6 zahrnuje testování a celkové porovnání úspěšnosti implementovaného řešení vzhledem k realizacím jiných týmů, které se účastnily soutěže. Jsou zde stručně popsány týmy účastníků a jak si vedly v soutěži.

Kapitola 2

Soutěž Multi-Agent Programming Contest

Smyslem soutěže je rozšířit povědomí a zájem o vývoj multiagentních systémů. *Multi-Agent Programming Contest* existuje již od roku 2005 [1], kdy se poprvé představila se scénářem *CLIMA VI Contest*. Od té doby se soutěž opakuje každým rokem, ale dochází k úpravám nebo i úplným změnám scénáře. Například pro roky 2006 a 2007 museli účastníci pracovat se scénářem *Goldminers Scenario* a v letech 2008, 2009, 2010 se potýkali s *Cow Herding Scenario*. Pro roky 2019, 2020 a 2021 je nutné řešit scénář *Agents Assemble*, což je i předmětem této práce. Popisy jednotlivých scénářů i se zdrojovými kódy pro jejich spuštění jsou uloženy v repozitářích na GitHubu[1].

Ve scénáři *Agents Assemble* spolu soupeří 2 týmy agentů, jejichž cílem je prozkoumávat neznámou oblast a získávat kostky k sestavování požadovaných útvarů. Agenti se pohybují po mřížce a mohou na sebe připojovat kostky ze všech svých 4 stran. S pomocí dalšího agenta na sebe může agent připojovat kostky do komplexnějších útvarů. V náhodně dlouhých časových intervalech scénář generuje úkoly, které v sobě nesou informace, jaké útvary kostek jsou potřeba k jejich splnění, kolik vítězných bodů získá tým za jeho splnění, atd.

2.1 Prostředí

Agenti se pohybují po obdélníkové mřížce. Rozměry mřížky nejsou agentům známy. Agenti vnímají pouze relativní pozice objektů vztažené k nim samotným. Mřížka se opakuje vodorovně i svisle, tj. pokud agent projde přes pravý okraj mapy, objeví se a bude pokračovat od levého okraje.

Na každé buňce mřížky se může vyskytovat objekt, který může kolidovat s jinými objekty (tj. agenti, kostky, překážky). Pouze na začátku simulace agent sdílí stejnou buňku s jedním agentem druhého týmu (pro zajištění spravedlnosti). Jakmile se jeden z agentů pohne, nemohou se později znovu překrývat.

Agenti

Agenti neznají svou absolutní pozici v prostředí. Mohou pouze určovat relativní umístění jiných objektů vzhledem k vlastní pozici. Tedy mohou zjistit pouze jak daleko a kterým směrem se nacházejí jiné objekty či agenti, které agent záhlédne. Vzdálenost, na kterou

[1] https://github.com/agentcontest

mohou agenti zpozorovat jiné objekty, je taktéž omezena parametrem scénáře. Nevidí tedy celou mapu, ale jen blízké objekty.

Objekty a terén

Existuje několik typů objektů, které se mohou vyskytovat v buňce mřížky:

- **Agent**: Popsán v podkapitole 2.1, sekce Agenti.
- **Dávkovač**: Dávkovače mohou být různých typů, které jsou specifikovány parametry scénáře. Jeho použitím dostane agent kostku stejného typu jako je dávkovač.
- **Kostka**: Stavební bloky, které agenti mohou vzít a připojovat k sobě za účelem tvorby sestav pro plnění zadaných úkolů scénáře. Kostky mohou být různých typů.
- **Značka**: Neblokující objekt, který tedy může sdílet buňku i s dalšími objekty. Značka indikuje, že v oblasti proběhne *akce čištění*.
- **Tabule úkolů**: Neblokující objekt. Agent se musí nacházet u tabule, aby mohl přijmout úkol.

Dále představuje každá buňka terén určitého typu:

- **Prázdno**: Žádná speciální vlastnost. Nemá-li buňka žádný terén, jedná se o *prázdno*.
- **Cílová oblast**: Pokud chce agent odeslat sestavu za účelem splnění úkolu, musí se nacházet na buňce tohoto typu.
- **Překážka**: Neprůchozí buňka. Agent na ni nemůže vstoupit a ani se otočit tak, aby se zde nacházela některá z jeho připojených kostek. Překážku je možné odstranit pomocí *akce čištění*.

Události

Prostředí náhodně generuje události. Momentálně jediná událost, která může být vygenerována je speciálně upravená **akce čištění**. Je-li tato akce vyvolaná prostředím, provede se na náhodné pozici s náhodnou velikostí. Navíc po dokončení se v okolí středu akce náhodně vygenerují *překážky*.

Úkoly

Prostředí náhodně generuje úkoly pro agenty. Každý úkol má své jméno, počet vítězných bodů za splnění, požadovanou sestavu a deadline, po kterém již úkol nebude možné splnit. Každý agent může v jeden čas přijmout a plnit pouze jeden úkol. Pro přijetí úkolu se musí agent nacházet v blízkosti úkolové tabule. Pokud pak bude chtít agent úkol odevzdat, musí nacházet v *cílové oblasti*.

2.2 Vjemy agentů

Na začátku simulace a pak každé kolo pošle server každému agentovi jeho vjemy ve formátu JSON[2]. Na začátku simulace se jedná o data jako např. jméno agenta, jméno týmu, vzdálenost dohledu agenta, atd. Vjemy získávané v průběhu simulace jsou již mnohem detailnější. Ze zmíněného JSON souboru jsou pro návrh významné následující vjemy:

[2]Formát dat JavaScript Object Notation

- **Stav agenta**: Informace, zda je agent deaktivovaný. K deaktivaci agenta dojde, když je agent zasažen *akcí čištění*. Deaktivovaný agent okamžitě upustí všechny připojené kostky a po daný počet kol nemůže provést žádnou akci.
- **Energie agenta**: Energii potřebuje agent k provedení *akce čištění*. Pokud nemá dostatek energie, akci nelze provést. Energie se agentovi částečně doplňuje při každém pohybu.
- **Aktuální úkol**: Úkol, který agent přijal. Obsahuje jméno úkolu, deadline po kterém již nebude možné úkol splnit a požadovanou formaci kostek, tedy jak musejí být kostky napojeny na agenta pro úspěšné splnění úkolu.
- **Zjištěn nový úkol**: Uměle vytvořený vjem. Agent si musí zapamatovat jména úkolů z minulého kroku a porovnat je se úkoly v aktuálním kroku. Pokud najde nové jméno, byl zjištěn nový úkol.
- **Připojené kostky**: Pozice a typy kostek, které jsou připojeny k agentovi.
- **Zpozorován objekt**: V okolí agenta se nachází určitý významný objekt či terén. Obsahuje informace o pozici objektu či terénu a případně i typu. Dle objektu/terénu se vjem dále dělí na:
  - Zpozorován dávkovač
  - Zpozorován nepřátelský agent
  - Zpozorována akce čištění
  - Zpozorována cílová oblast
  - Zpozorována kostka
  - Zpozorována překážka

2.3 Akce agentů

V každém kroku může agent provést přesně jednu akci. Akce se poté na serveru provádějí v náhodném pořadí, takže se např. může stát, že pokud chtějí v jednom kroku dva agenti vstoupit na stejnou pozici v prostředí, podaří se akce pouze tomu, který bude mít větší štěstí a jehož akce se vyhodnotí dříve. Druhý agent bude mít smůlu a jeho akce selže z důvodu již obsazené pozice, na kterou chtěl vkročit. Všechny akce mají navíc stejnou pravděpodobnost náhodného selhání. Každá akce má několik parametrů. Přesný počet závisí na typu akce. Pořadí parametru určuje jeho význam. Jedná se vždy o řetězcové nebo číselné hodnoty.

- **Zůstaň (skip)**: Stejné chování jako by agent neprovedl žádnou akci.
- **Pohni se (move)**: Agent se pohne ve směru specifikovaným parametrem (n|s|e|w). Akce selže pokud v požadovaném směru agenta (nebo jeho připojené kostky) blokuje jiný neprůchozí objekt.
- **Otoč se (rotate)**: Otočí agenta (se všemi jeho připojenými kostkami) ve směru specifikovaným parametrem (cw|ccw). Akce může selhat pokud otočení připojených kostek blokuje nějaký objekt.
- **Zvedni kostku (attach)**: Zvednutí kostky ze směru specifikovaného parametrem (n|s|e|w). Akce selže pokud není co zvednout a nebo pokud kostka je již připojena k jinému agentovi.
- **Polož kostku (detach)**: Položení kostky ve směru specifikovaného parametrem (n|s|e|w). Akce selže pokud agent nemá co položit.
- **Připoj kostku (connect)**: Připojení kostky jednoho agenta do sestavy jiného agenta. Pro úspěšné připojení kostky musejí oba agenti specifikovat v prvním parametru svého partnera, se kterým chtějí připojení provést a v dalších dvou parametrech pozici kostek

(x, y), které k sobě chtějí připojit (každý specifikuje souřadnice pouze své kostky k připojení).

- **Odpoj kostku (disconnect)**: Odpojení dvou připojených kostek v sestavě agenta. V prvních dvou parametrech je potřeba uvést pozici (x, y) první kostky a dalších dvou parametrech pozici (x, y) druhé kostky pro odpojení.
- **Použij dávkovač (request)**: Získání nové kostky z dávkovače. Pro úspěšné provedení akce musí být agent hned vedle dávkovače a musí specifikovat parametrem (n|s|e|w) směr, kterým se od něj dávkovač nachází. Akce selže pokud se na pozici dávkovače již nachází kostka.
- **Odešli sestavu (submit)**: Splnění úkolu specifikovaného parametrem. Agent musí mít přijatý úkol, který chce splnit. Akce selže pokud sestava připojená k agentovi neodpovídá požadované sestavě v úkolu nebo pokud se agent nenachází v cílové oblasti.
- **Akce čištění/Vyčisti (clear)**: Agent se připraví k provedení *akce čištění* na pozici specifikované parametry (x, y). K úspěšnému vyvolání *akce čištění* musí mít agent dostatek energie, musí akci úspěšně provést několik kol za sebou a zacílená pozice musí být dostatečně blízko, aby na ni agent dohlédl. *Akce čištění* má za následek zničení všech zasažených kostek, proměnu ovlivněných překážek na prázdný terén a deaktivaci všech agentů, kteří se v zasažené oblasti zrovna nacházejí.
- **Přijmi úkol (accept)**: Agent přijme úkol specifikovaný parametrem a nahradí tak jakýkoliv předchozí úkol, který přijal. Pro úspěšné přijetí úkolu se agent musí nacházet v blízkosti tabule úkolů.

Kapitola 3

Návrh systému

Cílem práce je navrhnout a implementovat multiagentní systém, který se bude řídit pravidly a mechanismy soutěže *Multi-Agent Programming Contest* (kapitola 2). Je však vhodné aby měl i co největší pravděpodobnost k vítězství nad všemi ostatními soupeři. Nestačí tedy jen navrhnout agentní systém, který bude schopen v soutěži fungovat, ale jeho chování musí vést k lepším výsledkům než u potenciálních soupeřů.

Existuje hned několik metodologií dle kterých lze navrhnout a implementovat multiagentní systém. Mezi nejvlivnější patří *Gaia*, *Tropos* a *Prometheus*, který je klasifikovaný jako nejvhodnější na porozumění pro lidi bez znalosti agentních systémů [7]. Pro účely této práce byla vybrána metodologie *Prometheus*. Další podkapitoly se věnují popisu použité metodologie a jejím jednotlivým fázím návrhu.

Obsažené diagramy byly modelovány v internetové aplikaci **Diagrams.net**[1].

3.1 Metodologie Prometheus

Jedná se o detailní postup specifikace a návrhu agentních systémů. Definuje spoustu artefaktů jako např. funkcionality, které se využijí pouze při návrhu dalších součástí systému, a nebo deskriptory plánů agentů, ze kterých se přímo čerpá při implementaci. Díky strukturování artefaktů lze proces i částečně automatizovat což umožňuje existenci nástrojů s jejichž pomocí lze snadněji navrhnout agentní systém. Takovým nástrojem je např. *PDT*[2].

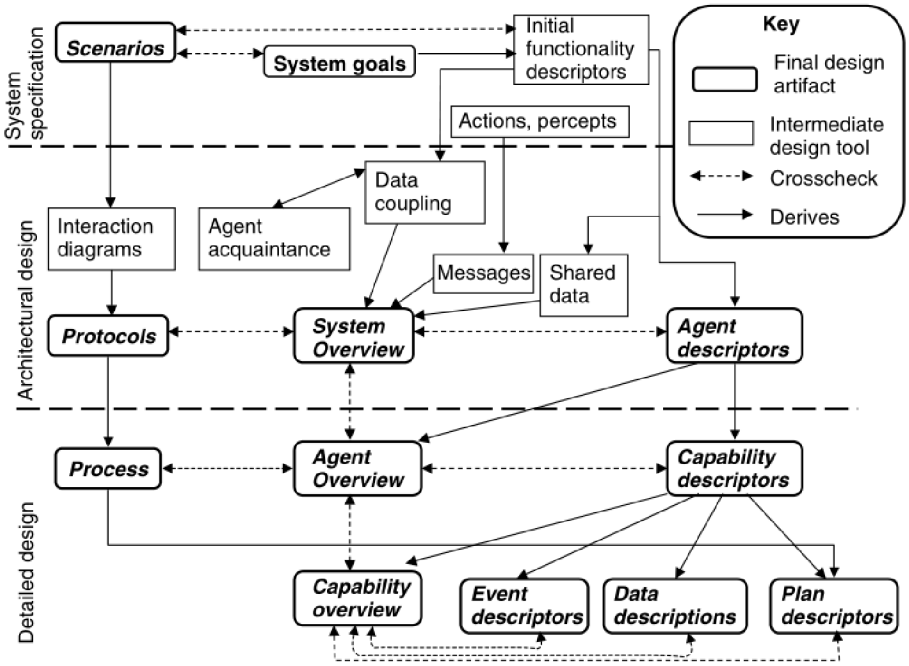
Metodologie *Prometheus* rozděluje návrh na 3 fáze, které je nutné dělat postupně. Artefakty v jednotlivých fázích však mohou být (někdy i musejí být) vytvářeny paralelně [9]. Detail lze vidět na diagramu 3.1.

1. **Specifikace systému**: Zaměřuje se na zjištění cílů, akcí, vjemů a celkových funkcionalit systému. Spadá sem zejména návrh scénářů a cílů agenta.

2. **Architektonický návrh**: Na základě předchozí fáze rozděluje agenty do rolí a popisuje komunikaci mezi nimi. Specifikují se zde zejména protokoly a deskriptory zpráv a agentů.

3. **Detailní návrh**: Zaměřuje se na vnitřní strukturu agentů. Vznikají zde detaily agenta jako např. deskriptory jeho schopností a plánů.

[1] https://app.diagrams.net/ - dříve známo jako *draw.io*
[2] Prometheus Design Tool - https://sites.google.com/site/rmitagents/software/prometheusPDT



Obrázek 3.1: Fáze návrhové metodologie Prometheus [9]

Před započetím návrhu touto metodologií byl vytvořen diagram případů užití (obr. 3.2), který vnáší do návrhu pohled na všechny aktéry ve scénáři a posloužil jako pevný základ pro specifikaci systému. Diagramem případu užití lze krásně vystihnout, které hlavní cíle agent může mít. Zakomponováním samotných akcí agenta a jejich provázáním skrze závislosti vzniknou i podcíle [6].

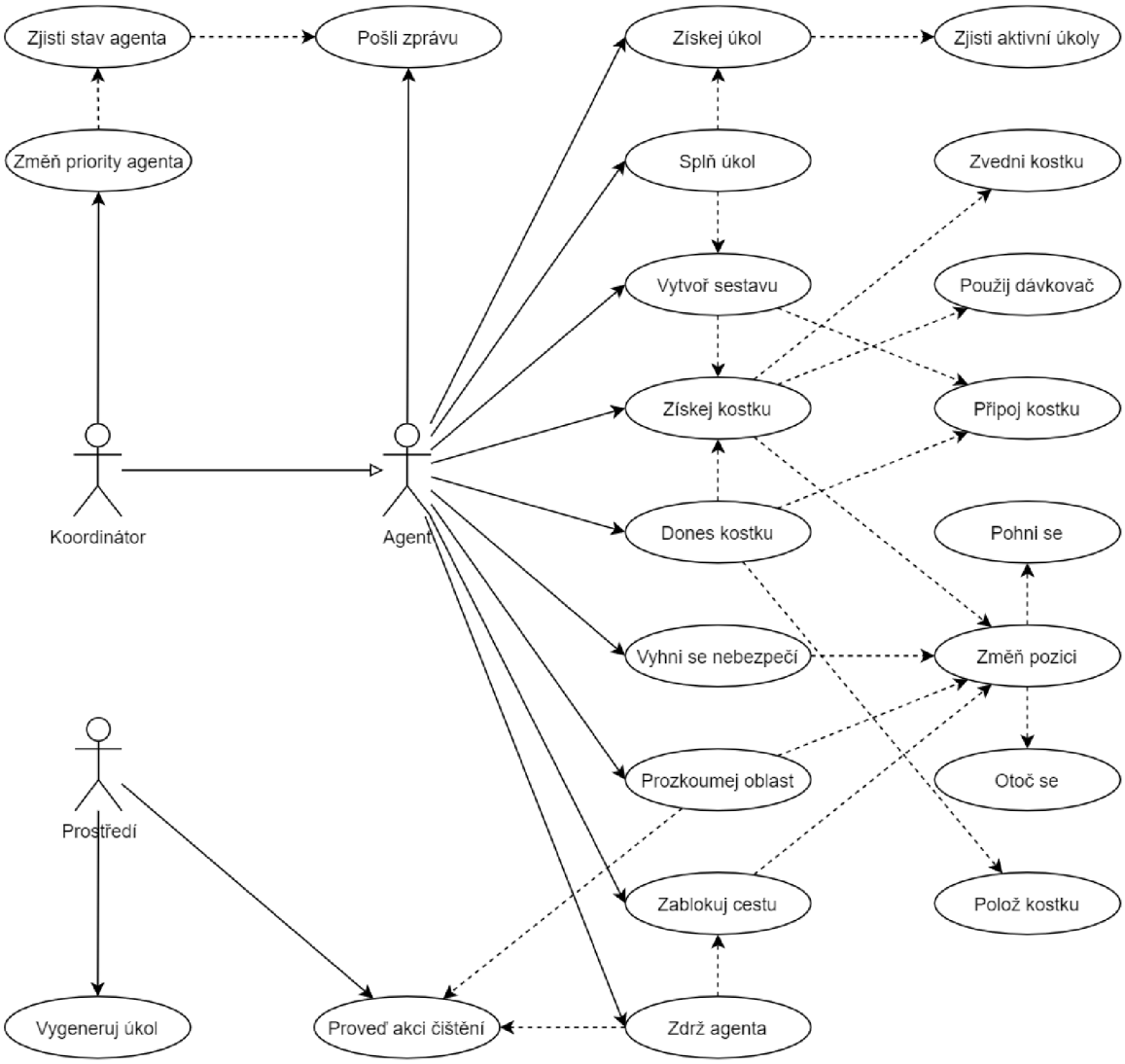
3.2 Specifikace systému

Podkapitola popisuje prvotní fázi návrhu, kdy se z obvykle textového popisu [9] vytvářejí prvotní artefakty. Je stěžejní zcela porozumět požadavkům na systém aby nedošlo k chybným předpokladům již na počátku návrhu. Fáze se zaměřuje zejména na určení cílů agentů a na popis scénářů, které v systému mohou nastat. Shlukováním cílů s podobnou oblastí působnosti se získají funkcionality. Dále se stanoví rozhraní agentů, jakým způsobem vnímají okolí a jakými akcemi reagují. Při specifikaci systému je často potřeba přeskakovat od jedné části návrhu k jiné neboť se je možné úpravami objevovat nové prvky. Přidáním jednoho cíle můžeme např. zcela objevit nové scénáře.

Akce a vjemy agentů jsou již dány pravidly soutěže (podkapitoly 2.2 a 2.3), a proto není potřeba je dále určovat. Zbývá tedy zjistit cíle agentů, odvodit funkcionality systému a sepsat scénáře, které v soutěži mohou proběhnout. Artefakty se zde vytvářejí paralelně protože tvorbou různých scénářů lze nalézt další neobjevené cíle a funkcionality systému. Naopak přidání dalšího cíle může vést k množině nových scénářů.

3.2.1 Specifikace cílů

K určení cílů agentů výrazně pomůže diagram případů užití na obr. 3.2 jehož jednotlivé případy užití lze interpretovat jako samotné cíle agentů. Krátkým uvážením, *jak* realizovat



Obrázek 3.2: Diagram případů užití scénáře *Agents Assemble*. Všechny čárkované šipky představuji vztah «include» [6] a pro přehlednost je u nich toto označení vynecháno.

jednotlivé cíle agenta, se dále určí i podcíle, které odpovídají vazbě «include» ve zmíněném diagramu případů užití. Tím je získán graf cílů a jejich závislostí (obr. 3.3). Cíle pomohou ke specifikaci funkcionalit a ke konstrukci scénářů.

3.2.2 Funkcionality systému

Shlukováním cílů, akcí a vjemů s podobnou oblastí působnosti, mohou být vytvořeny tzv. *funkcionality*, které pomohou systém dále rozčlenit. Navržený diagram funkcionalit se nachází na obr. 3.4. Funkcionality pomohou při sepisování scénářů a hlavně při určování rolí agentů v další fázi návrhu.

3.2.3 Scénáře

Scénáře obvykle popisují posloupnosti událostí a akcí, které ve specifikovaném systému mohou nastat. Také pomáhají k nalezení dalších či opomenutých cílů. Dále v architektonickém návrhu jsou na scénářích založeny interakce mezi agenty.

Kvůli jednoduchosti jsou vynechány některé triviální scénáře jako např. zapamatování nově nalezeného objektu a případné rozeslání této informace ostatním *pozičně synchronizovaným*[3] agentům. Dle pravidel a specifikace soutěže se vybízejí následující scénáře:

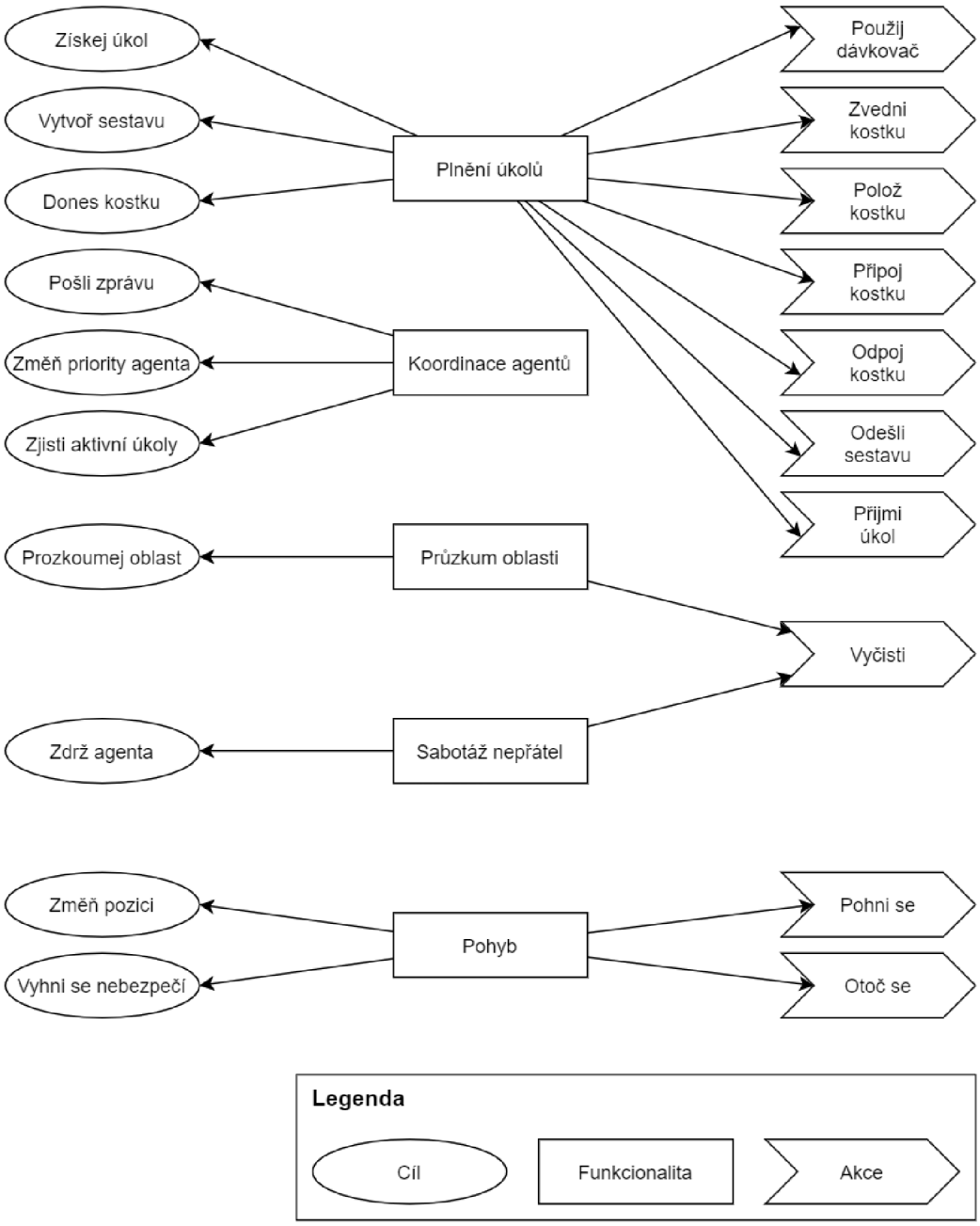
Přehodnocení priorit

- **Spouštěče**: Koordinátor získá vjem "zjištěn nový úkol"
  nebo koordinátor zjistí, že už se úkol nestihne dokončit před deadlinem
  nebo agent zpozoruje dávkovač na pozici, kterou ještě nemá zapamatovanou.
- **Popis**: Nastane-li některý ze spouštěčů, musí dojit k přehodnocení, které úkoly se agenti budou snažit dokončit, které již vzdají a celkově co mají aktuálně dělat. Důvodů pro přehodnocení může být hned několik. Např. zbývá nedostatek času na dokončení úkolu, nebo byl zjištěn lehčí úkol a nebo byl nalezen nový dávkovač, který týmu umožňuje plnit větší škálu úkolů. Koordinátor si vyžádá informace o pozici a připojených kostkách od všech ostatních agentů, prohlédne si aktuální úkoly a s ohledem na již nalezené dávkovače sdělí agentům, jak mají dále konat.
- **Postup**:

	Typ	**Krok**	**Funkcionalita**	**Poznámka**
1.	cíl	zjisti aktivní úkoly	koordinace agentů	-
2.	cíl	zjisti stav agenta	koordinace agentů	koordinátor získá informace od ostatních agentů
3.	jiné	počkej na odpovědi	-	-
4.	jiné	přepočítej priority	-	koordinátor z dostupných informací určí nové priority pro každého agenta
5.	cíl	změň priority agenta	koordinace agentů	koordinátor upraví priority ostatních agentů

[3] Agenti se již setkali a mohli si sjednotit referenční bod, od kterého určují pozice zapamatovaných objektů a sebe sama.

Obrázek 3.3: Cíle a podcíle agentů odpovídající diagramu případů užití (obr. 3.2)



Obrázek 3.4: Diagram funkcionalit systému

Přijetí a splnění úkolu se složitější sestavou

- **Spouštěč**: Agent dostane instrukci od koordinátora k přijetí úkolu.
- **Popis**: Koordinátor agentovi sdělí, který úkol má přijmout a začít jej plnit. Pokud jde o složitější sestavu, kterou agent není schopen vytvořit sám, musí koordinátor vybrat dalšího vhodného agenta (nebo agenty) a sdělit jim, jaké kostky mají přinést a připojit. Agenti se následně již mezi sebou informují, jak se správně napozicovat aby došlo k rychlému připojení kostek, a tedy k rychlému dokončení sestavy. Agent se sestavou nakonec dojde do cílové oblasti a sestavu odešle jako splněný úkol.
- **Postup**:

	Typ	**Krok**	**Funkcionalita**	**Poznámka**
1.	akce	přijmi úkol	plnění úkolů	-
2.	cíl	pošli zprávu	koordinace agentů	agent dá vědět, které kostky potřebuje připojit do sestavy
3.	jiné	počkej na donesení kostky	-	-
4.	cíl	vytvoř sestavu	plnění úkolů	-
5.	vjem	zprozorována cílová oblast	-	-
6.	cíl	změň pozici	pohyb	agent dojde do cílové oblasti
7.	akce	odešli sestavu	-	-

Pomoc agentovi s tvorbou sestavy

- **Spouštěč**: Agent dostane od koordinátora instrukce aby pomohl jinému agentovi s tvorbou sestavy.
- **Popis**: Scénář nastane ve chvíli, kdy agent dostane prioritní instrukce od koordinátora, aby se účastnil vytváření sestavy, která bude navázána na jiného agenta. Jde o případ, kdy už si jeden agent sám navázat kostku nemůže, tak potřebuje aby mu jiný agent kostku donesl a navázal.
- **Postup**:

	Typ	**Krok**	**Funkcionalita**	**Poznámka**
1.	cíl	získej kostku	plnění úkolu	-
2.	cíl	pošli zprávu	koordinace agentů	agenti se informují, jak se napozicovat, aby byla kostka co nejrychleji napojena
3.	cíl	změň pozici	pohyb	-
4.	cíl	připoj kostku	plnění úkolu	-
5.	cíl	pošli zprávu	koordinace agentů	agent dá vědět koordinátorovi, že kostka byla připojena

Zdržení nepřátelského agenta

- **Spouštěč**: Agentovi se změní priorita na sabotáž nepřátelských agentů.
- **Popis**: Jsou-li agentovi priority změněny tak, aby se věnoval sabotáži nepřátelských agentů, agent se nejprve v dané oblasti snaží nalézt nepřátelské agenty a následně jim blokovat cestu jak sebou samým tak prováděním akcí čištění.

- **Postup**:

	Typ	Krok	Funkcionalita	Poznámka
1.	cíl	prozkoumej oblast	průzkum oblasti	-
2.	vjem	zpozorován nepřátelský agent	-	-
3.	cíl	zdrž agenta	sabotáž nepřátel	-

Nalezení dávkovače

- **Spouštěč**: Agent při pohybu nalezne dávkovač.
- **Popis**: Agent při pohybu nalezne nový dávkovač o čemž dá okamžitě vědět koordinátorovi, což vede ke scénáři "Přehodnocení priorit". Stejný scénář lze použít pro nalezení cílové oblasti, akorát se dávkovač nahradí cílovou oblastí. Nový dávkovač lze nalézt obecně při pohybu agenta, ale zde je ilustrováno pouze na průzkumu oblasti.
- **Postup**:

	Typ	Krok	Funkcionalita	Poznámka
1.	cíl	prozkoumej oblast	průzkum oblasti	-
2.	vjem	zpozorován dávkovač	-	-
3.	cíl	pošli zprávu	koordinace agentů	agent informuje koordinátora o pozici dávkovače

Vyhnutí se nebezpečí

- **Spouštěč**: Zpozorována akce čištění v blízkosti agenta.
- **Popis**: Jelikož se akce čištění objevují náhodně a nebo mohou být vyvolány agenty, lze na ně narazit kdekoliv. Při zpozorování akce čištění musí agent dočasně změnit směr svého pohybu (případně zcela zastavit, pokud se jinak vyhnout nelze) aby se co nejrychleji dostal z oblasti akce čištění a již do oblasti nevstupoval dokud akce nezmizí. Poté se vrací ke své předchozí činnosti. Scénář "Vyhnutí se nebezpečí"může nastat prakticky kdykoliv, ale zde je ilustrován pouze na průzkumu oblasti.
- **Postup**:

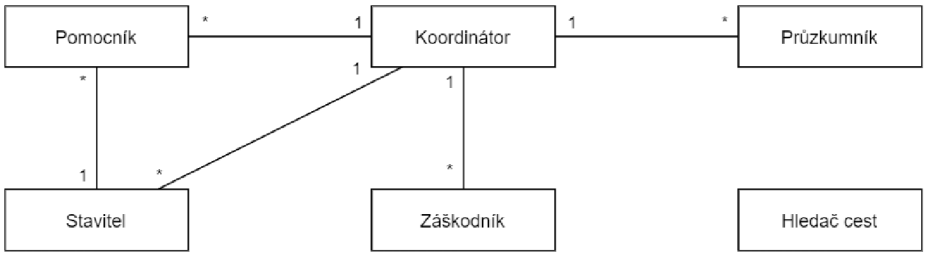
	Typ	Krok	Funkcionalita	Poznámka
1.	cíl	prozkoumej oblast	průzkum oblasti	-
2.	vjem	zpozorována akce čištění	-	-
3.	cíl	vyhni se nebezpečí	pohyb	-
4.	jiné	nebezpečí zmizí	-	-
5.	jiné	návrat k předchozímu cíli	-	-

3.3 Architektonický návrh

Na základě navržených funkcionalit systému lze rozčlenit agenty do rolí a následně jednotlivé role popsat. Urči se zde i protokoly mezi agenty a zprávy které si spolu posílají. Popis komunikace vychází ze scénářů sepsaných při specifikaci systému. Hlavním produktem této fáze návrhu by měl být popis přehledu systému.

3.3.1 Role agentů

Nebylo by moudré aby se v multiagentním systému snažil každý agent dělat všechno sám, a proto se v metodologii *Prometheus* přiřazují agentům role [9]. Každý agent se pak dle své role může věnovat svým úkonům, které jsou obvykle rozdílné od činností jiných rolí.



Obrázek 3.5: Počty agentů v systému dle vzájemné závislosti

Role mohou být agentům přiděleny neměnně po celou jejich životnost a nebo i dynamicky, kdy agent v průběhu svého fungování může vystřídat hned několik rolí [14]. Systém lze navrhnout i tak, že agent bude mít v jeden okamžik i více rolí. Např. budeme-li chtít aby spolu agenti komunikovali přes centrální uzel, který se bude starat o režii této komunikace, vznikne nám role **koordinátora**. Avšak nechceme aby koordinátor jen tak stál na místě a nedělal nic jiného než určoval úkoly ostatním agentům. Proto agent s rolí koordinátora bude mít přidělenu ještě jinou roli, např. průzkumníka který se už bude pohybovat a prozkoumávat oblast.

Kromě koordinátora jsou potřeba i další role. V první řadě je nutné skládat kostky do sestav a plnit úkoly což bude práce **stavitele**. Při tvorbě složitějších sestav kdy je potřeba kostky k sobě připojovat, což nezvládne jeden agent sám, bude staviteli přidělen **pomocník**. Agenti dohlédnou jen do určité vzdálenosti kolem sebe a aby mohli stavitelé vytvářet sestavy, je potřeba **průzkumníka** aby nalézal dávkovače odpovídajícího typu. V soutěži soupeří dva týmy agentů proti sobě a mohou si navzájem škodit jak blokováním pohybu tak i zneaktivněním pomocí akce čištění. Je tedy vhodné aby se agenti chopili i role **záškodníka**. Jelikož se všichni agenti potřebují pohybovat a systém již počítá s více rolemi na agenta, tak pohyb a jiné generické úkony zastane role **hledač cest**.

Koordinátor

- **Popis**: Nesamostatná role - agent má vždy ještě nejméně jednu jinou roli. Koordinuje ostatní agenty, rozděluje jim role a úkoly, dává instrukce, které cíle mají dále preferovat. Slouží jako ústředna pro komunikaci mezi agenty (sdílení pozic významných objektů).
- **Kardinalita**: 1
- **Životnost**: od začátku až do konce simulace
- **Inicializace**: počáteční rozdělení rolí a úkolů ostatním agentům
- **Zánik**: -
- **Funkcionality**: koordinace agentů
- **Využívá data**: pozice agentů, pozice dávkovačů, pozice cílových oblastí, připojené kostky agentů, aktivní úkoly
- **Produkuje data**: pozice agentů, pozice dávkovačů, pozice cílových oblastí, role a priority agentů
- **Cíle**: změň priority agenta, zjisti aktivní úkoly, pošli zprávu
- **Reaguje na vjemy**: zjištěn nový úkol
- **Akce**: -
- **Interakce**: rozdělení rolí a priorit ostatním agentům, sdílení nově nalazených věcí mezi agenty

Hledač cest

- **Popis**: Základní a také nesamostatná role pro všechny agenty. Stará se o zajištění bezpečné a nejrychlejší cesty k aktuálně vyžadované pozici.
- **Kardinalita**: 1 pro každého existujícího agenta
- **Životnost**: od začátku až do konce simulace
- **Inicializace**: -
- **Zánik**: -
- **Funkcionality**: pohyb
- **Využívá data**: pozice agentů (svoje), pozice překážek
- **Produkuje data**: směr pohybu
- **Cíle**: změň pozici, vyhni se nebezpečí
- **Reaguje na vjemy**: zpozorována překážka, zpozorována akce čištění
- **Akce**: pohni se, otoč se
- **Interakce**: -

Stavitel

- **Popis**: Snaží se splnit zadaný úkol. Sestavuje kostky do potřebné formace. Často využívá pomocníka k připojování kostek.
- **Kardinalita**: vždy alespoň 1 pokud je možné získat kostky ke splnění některého aktivního úkolu
- **Životnost**: od přidělení role stavitele do dokončení sestavy nebo do odebrání role stavitele
- **Inicializace**: přijetí nového úkolu, zapamatování si přidělených pomocníků
- **Zánik**: zbavení se připojených kostek
- **Funkcionality**: plnění úkolu
- **Využívá data**: pozice dávkovačů, pozice cílových oblastí, připojené kostky, pozice agentů (pomocník)
- **Produkuje data**: pozice agentů (svoje), připojené kostky
- **Cíle**: získej úkol, vytvoř sestavu, pošli zprávu
- **Reaguje na vjemy**: zpozorován dávkovač, zporozována kostka, zpozorována cílová oblast, připojené kostky, aktuální úkol
- **Akce**: zvedni kostku, polož kostku, připoj kostku, odpoj kostku, použij dávkovač, odešli sestavu, přijmi úkol
- **Interakce**: úprava rolí a priorit od koordinátora, získávání a připojování kostek s pomocníkem

Pomocník

- **Popis**: Pomáhá staviteli s tvorbou sestavy. Získává pro něj a připojuje k němu kostky za účelem sestavení požadovaného vzoru ke splnění úkolu stavitele. Předpokladem přidělení role pomocníka je, že jsou se stavitelem již *pozičně synchronizovaní.*
- **Kardinalita**: 0 až 2 pro každého stavitele
- **Životnost**: od přidělení role pomocníka po dokončení sestavy stavitele nebo do odebrání role pomocníka
- **Inicializace**: zapamatování si přiděleného stavitele
- **Zánik**: zbavení se připojených kostek
- **Funkcionality**: plnění úkolu

- **Využívá data**: pozice dávkovačů, pozice agentů (stavitel)
- **Produkuje data**: pozice agentů (svoje), připojené kostky
- **Cíle**: dones kostku, pošli zprávu
- **Reaguje na vjemy**: zpozorován dávkovač, zporozována kostka, připojené kostky
- **Akce**: zvedni kostku, polož kostku, připoj kostku, odpoj kostku, použij dávkovač
- **Interakce**: úprava rolí a priorit od koordinátora, získávání a připojování kostek se stavitelem

Záškodník

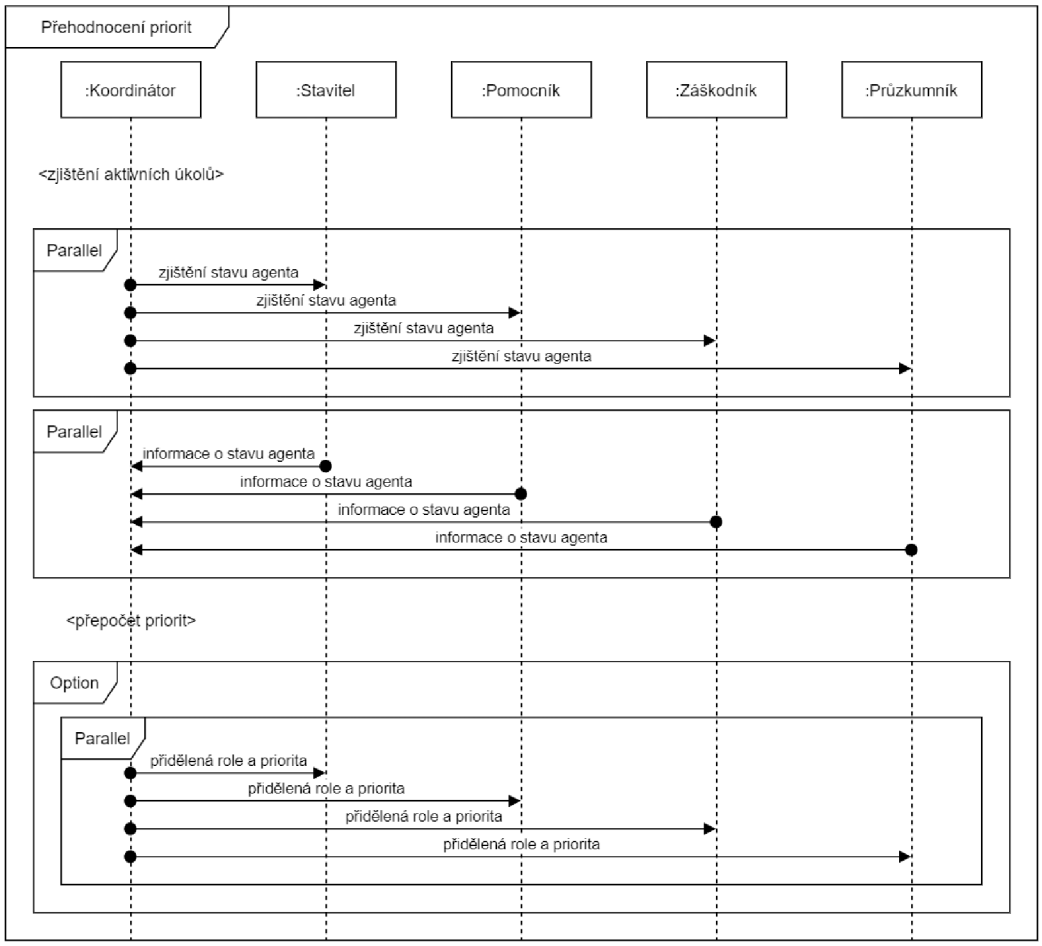
- **Popis**: Vyhledává nepřátelské agenty, kterým se snaží stát v cestě a používat na ně akce čištění.
- **Kardinalita**: 0 až n
- **Životnost**: od přidělení role záškodníka do odebrání role záškodníka
- **Inicializace**: aktuální nebo naposledy zpozorovaná pozice nepřátelských agentů
- **Zánik**: -
- **Funkcionality**: sabotáž nepřátel, průzkum oblasti
- **Využívá data**: pozice agentů (nepřátelských), energie agenta
- **Produkuje data**: -
- **Cíle**: zdrž agenta, prozkoumej oblast
- **Reaguje na vjemy**: zpozorován nepřátelský agent, energie agenta
- **Akce**: vyčisti
- **Interakce**: úprava rolí a priorit od koordinátora, informace o pozicích nepřátel od koordinátora

Průzkumník

- **Popis**: Snaží se projít pozice, které ještě žádný spojenecký agent nezmapoval a najít tak co nejvíce užitečných objektů (dávkovače, cílové oblasti). V případě, že mu v cestě stojí překážka, provede akci čištění.
- **Kardinalita**: 0 až n
- **Životnost**: od přidělení role průzkumníka do odebrání role průzkumníka
- **Inicializace**: aktuálně prozkoumané pozice
- **Zánik**: -
- **Funkcionality**: průzkum oblasti
- **Využívá data**: prozkoumané pozice
- **Produkuje data**: pozice dávkovačů, pozice cílových oblastí, pozice agentů (nepřátelských)
- **Cíle**: prozkoumej oblast
- **Reaguje na vjemy**: zpozorován dávkovač, zpozorována cílová oblast, zpozorována překážka, energie agenta
- **Akce**: vyčisti
- **Interakce**: úprava rolí a priorit od koordinátora, informace o pozicích důležitých věcí (dávkovače, cílové oblasti, nepřátelští agenti) s koordinátorem

3.3.2 Komunikace mezi agenty

Je potřeba navrhnout protokoly, skrze které spolu agenti budou komunikovat. Z vytvořených protokolů se získají a podrobněji rozepíší detaily o zprávách, které si agenti posílají.

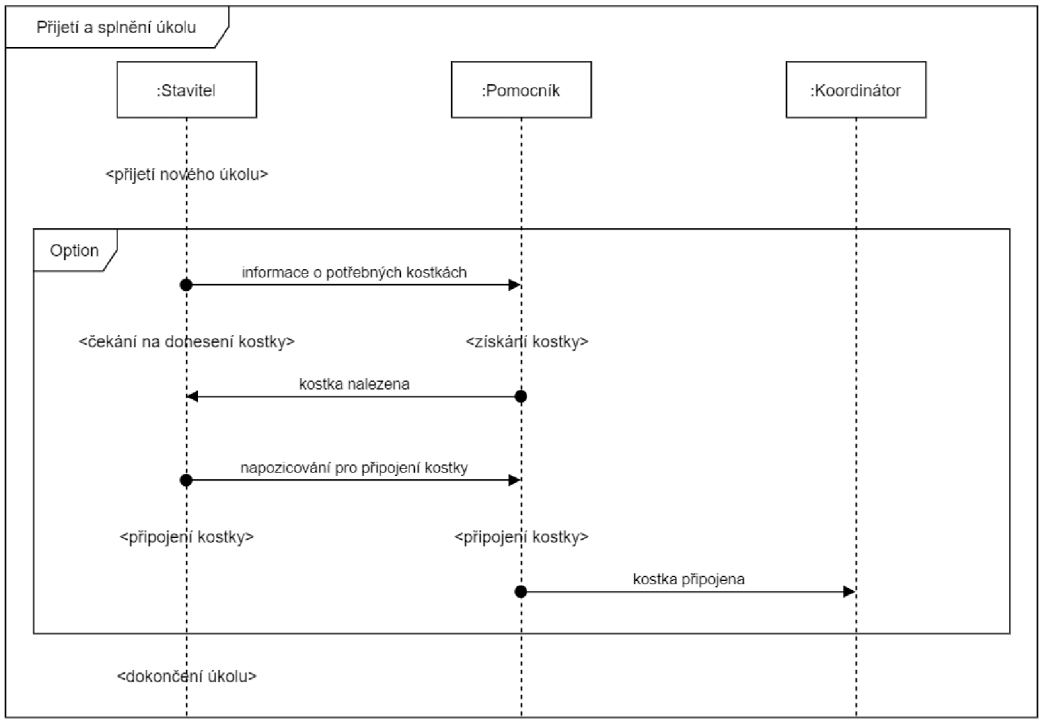


Obrázek 3.6: Komunikační protokol - přehodnocení priorit

Předlohou pro tuto část návrhu jsou scénáře, ze kterých lze interakce mezi agenty vyčíst, pokud jsou správně sepsány.

Nejsou zde zmíněny triviální zprávy, které si agenti posílají. Jedním případem je nalezení nového významného objektu a oznámení této skutečnosti ostatním agentům. Další případ nastane, když se dva agenti ze stejného týmu poprvé setkají. Jelikož agenti neznají svou absolutní pozici v prostředí, ale znají pouze pozice objektů kolem nich, musejí si určit *referenční bod* ke kterému budou vztahovat pozice všech významných objektů, které si chtějí zapamatovat. Když se tedy dva agenti potkají, dojde k **poziční synchronizaci** tak, že jeden přijme *referenční bod* druhého agenta a přepočítá pozice všech zapamatovaných objektů k novému *referenčnímu bodu*, který právě přijal. Agenti pak vztahují pozice svých zapamatovaných objektů ke společnému bodu. Nejen že si mohou nasdílet všechny významné objekty, které doposud našli, ale pro další průběh scénáře již budou znát svoji vzájemnou polohu a mohou si tedy sdílet pozice významných objektů ihned po nalezení.

Ze specifikovaných scénářů vyplývají dvoje interakce mezi agenty, a tedy dva protokoly. Jedním je komunikace koordinátora a ostatních agentů při přehodnocování priorit (vizte obr. 3.6) a dalším je interakce stavitele a pomocníka při tvorbě sestavy za účelem splnění úkolu (vizte obr. 3.7). Deskriptory zpráv, které agenti používají v protokolech, jsou uvedeny níže.



Obrázek 3.7: Komunikační protokol - přijetí a splnění úkolu

Zjištění stavu agenta

- **Protokol**: Přehodnocení priorit
- **Popis**: Požadavek na zjištění aktuálních informací o agentovi.
- **Odesilatel**: Koordinátor
- **Příjemce**: Stavitel, Pomocník, Záškodník, Průzkumník
- **Účel**: Zjištění informací o agentech k efektivnímu přehodnocení rolí a priorit.
- **Přenesené informace**: -

Informace o stavu agenta

- **Protokol**: Přehodnocení priorit
- **Popis**: Odpověď na požadavek koordinátora o zjištění aktuálních informací o agentovi.
- **Odesilatel**: Stavitel, Pomocník, Záškodník, Průzkumník
- **Příjemce**: Koordinátor
- **Účel**: Informování koordinátora o stavu agentů k efektivnímu přehodnocení rolí a priorit.
- **Přenesené informace**: pozice agenta, připojené kostky, aktuální cíle, energie agenta

Přidělená role a priorita

- **Protokol**: Přehodnocení priorit
- **Popis**: Přidělení (nové) role agentovi a případná úprava jeho cílů
- **Odesilatel**: Koordinátor
- **Příjemce**: Stavitel, Pomocník, Záškodník, Průzkumník
- **Účel**: Zefektivnění operací agentů za účelem jistější výhry

- **Přenesené informace**: role, aktuální cíle

Informace o potřebných kostkách

- **Protokol**: Přijetí a splnění úkolu
- **Popis**: Stavitel informuje pomocníka, které kostky od něj potřebuje donést.
- **Odesilatel**: Stavitel
- **Příjemce**: Pomocník
- **Účel**: Stavitel potřebuje donést a připojit kostky do sestavy, které si sám připojit už nedokáže.
- **Přenesené informace**: druh kostky

Kostka nalezena

- **Protokol**: Přijetí a splnění úkolu
- **Popis**: Informuje stavitele, že pomocníkovi se podařilo obstarat potřebnou kostku.
- **Odesilatel**: Pomocník
- **Příjemce**: Stavitel
- **Účel**: Domluvení se na místě setkání pro nejrychlejší napojení kostky.
- **Přenesené informace**: pozice agenta, připojené kostky

Napozicování pro připojení kostky

- **Protokol**: Přijetí a splnění úkolu
- **Popis**: Sdělení pomocníkovi, na které pozici se setká se stavitelem a kam kostku připojí.
- **Odesilatel**: Stavitel
- **Příjemce**: Pomocník
- **Účel**: Pomocník potřebuje vědět, jak má kostku připojit aby byla sestava vytvořena správně.
- **Přenesené informace**: pozice

Kostka připojena

- **Protokol**: Přijetí a splnění úkolu
- **Popis**: Pomocník po připojení kostky dá vědět koordinátorovi, že svůj úděl splnil.
- **Odesilatel**: Pomocník
- **Příjemce**: Koordinátor
- **Účel**: Přidělení další práce (role a priorit) pomocníkovi.
- **Přenesené informace**: -

3.3.3 Přehled systému

Po určení rolí agentů v systému a komunikaci mezi nimi jsou tyto informace spojeny dohromady v přehledu systému, který zachycuje celkovou architekturu agentního systému. Je třeba zohlednit informace o prostředí, na jaké vjemy budou agenti reagovat, a které akce mohou agenti v rámci prostředí podniknout.

Jsou zde využity vytvořené protokoly, zprávy a deskriptory agentů ze specifikace systému, které daly možnost vzniku přehledu systému popsaném diagramem na obr. 3.8. Pře-

hled zaobaluje agenty rozdělené do rolí a ukazuje, na které vjemy reagují, které akce používají a jak spolu agenti komunikují.

3.4 Detailní návrh

Metodologie *Prometheus* popisuje jak z předchozího návrhu použít protokoly pro návrh procesů a přehled systému s deskriptory agentů pro návrh přehledu samotných agentů a identifikaci jejich schopností. Ze schopností lze dále popsat plány a data agentů a události, které je ovlivňují.

Procesy mají pouze pomoci k vyobrazení postupu realizace plánů, a proto jsou z návrhu vypuštěny. Postupy jsou popsány přímo v deskriptorech plánů v podkapitole 3.4.2. Jelikož plány agentů nejsou příliš složité a je možné je navrhnout a popsat bez využití událostí, jsou z návrhu vynechány také schopnosti agenta. Prvky v přehledech agentů tedy reprezentují samotné plány, nikoliv schopnosti.

3.4.1 Přehledy agentů

S využitím přehledu systému jako rozhraním a deskriptorů agentů jako vnitřním popisem chování agenta, je možné již konstruovat přehledy agentů. Vyobrazí se zde, jak agenti reagují a pracují s jednotlivými vjemy, jak se uvnitř chovají a ve které akce jejich chování vyústí. Diagramy 3.10 až 3.15 mají společnou legendu 3.9 a znázorňují přehledy jednotlivých rolí agentů.

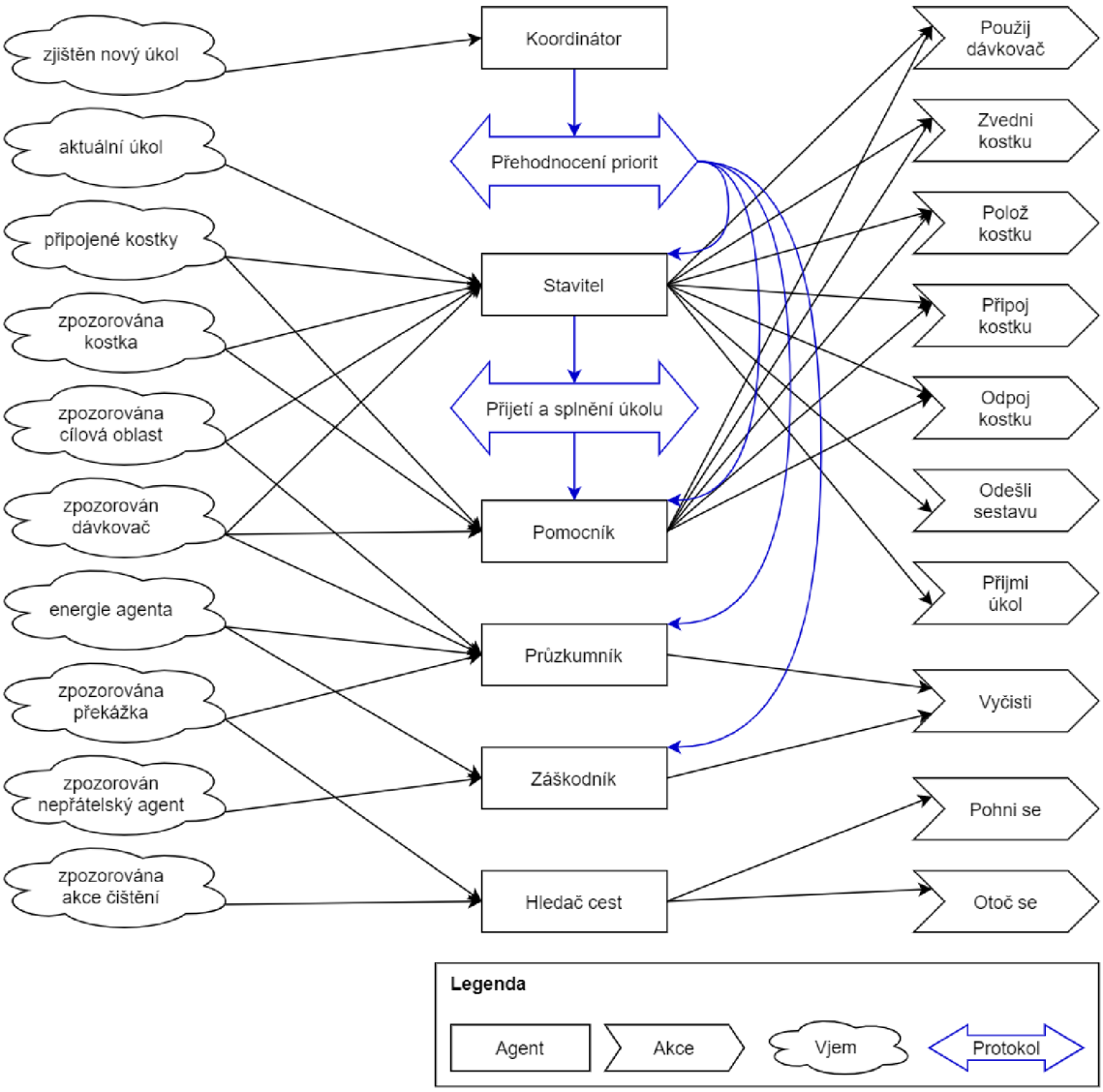
3.4.2 Plány agentů

Jednotlivé plány agentů lze vidět na diagramech 3.10 až 3.15. Níže jsou podrobněji popsány včetně postupů, takže je již lze využít pro implementaci ve BDI[4] [10] frameworcích, které podporují plány jako např. *JACK*[5] nebo *Jason*[6] a další [9].

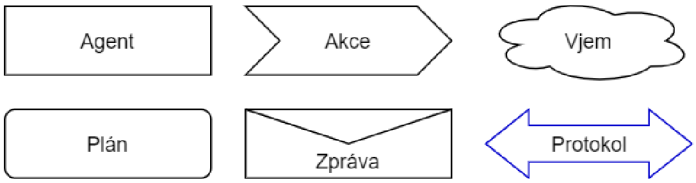
Rozdělení rolí a přepočet priorit

- **Agent**: Koordinátor
- **Popis**: Koordinátor získá od všech ostatních agentů potřebné informace (pozice významných věcí, nepřátel, připojených kostek) a na základě zjištěných informací propočítá, které role a aktuální priority budou pro každého agenta nejvhodnější.
- **Spouštěče**: zjištěn nový úkol, zjištěna nová pozice významné věci
- **Vstupní data**: aktivní úkoly
- **Výstupní data**: nové role a priority pro agenty
- **Cíl**: Zefektivnění plnění úkolů a získání výhody nad soupeřem
- **Postup**:
    1. Zaslání zprávy ostatním agentům aby poslali koordinátorovi potřebné informace a vyčkání na odpovědi.
    2. Vyhodnocení, zdali je potřeba agent stavitel dle aktivních úkolů a nalezených dávkovačů. Případné zvážení, zdali stavitel potřebuje pomocníky a kolik jich

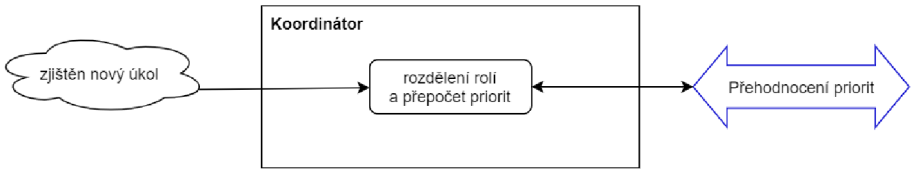
[4] **B**elief-**D**esire-**I**ntention
[5] https://aosgrp.com/products/jack/
[6] https://github.com/jason-lang/jason



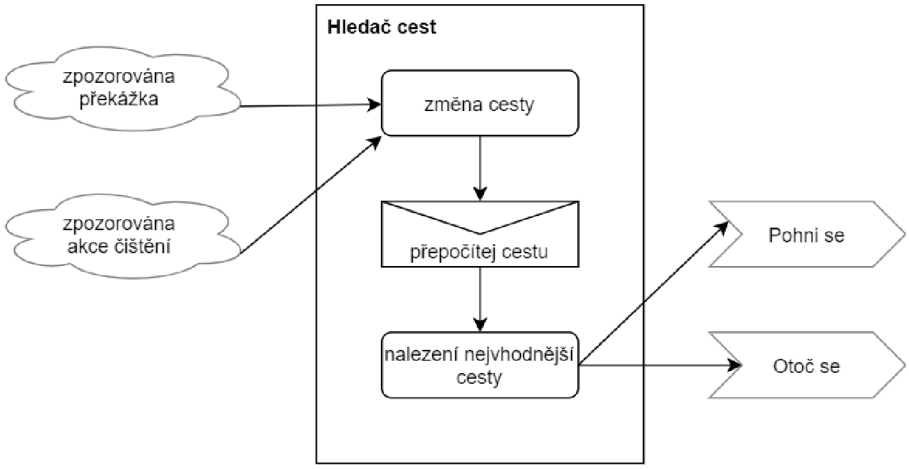
Obrázek 3.8: Diagram přehledu systému



Obrázek 3.9: Legenda k diagramům 3.10 až 3.15



Obrázek 3.10: Diagram přehledu agenta - koordinátor



Obrázek 3.11: Diagram přehledu agenta - hledač cest

bude. Případné rozdělení rolí stavitel a pomocník nejvhodnějším agentům (zpravidla nejblíže u potřebných dávkovačů).

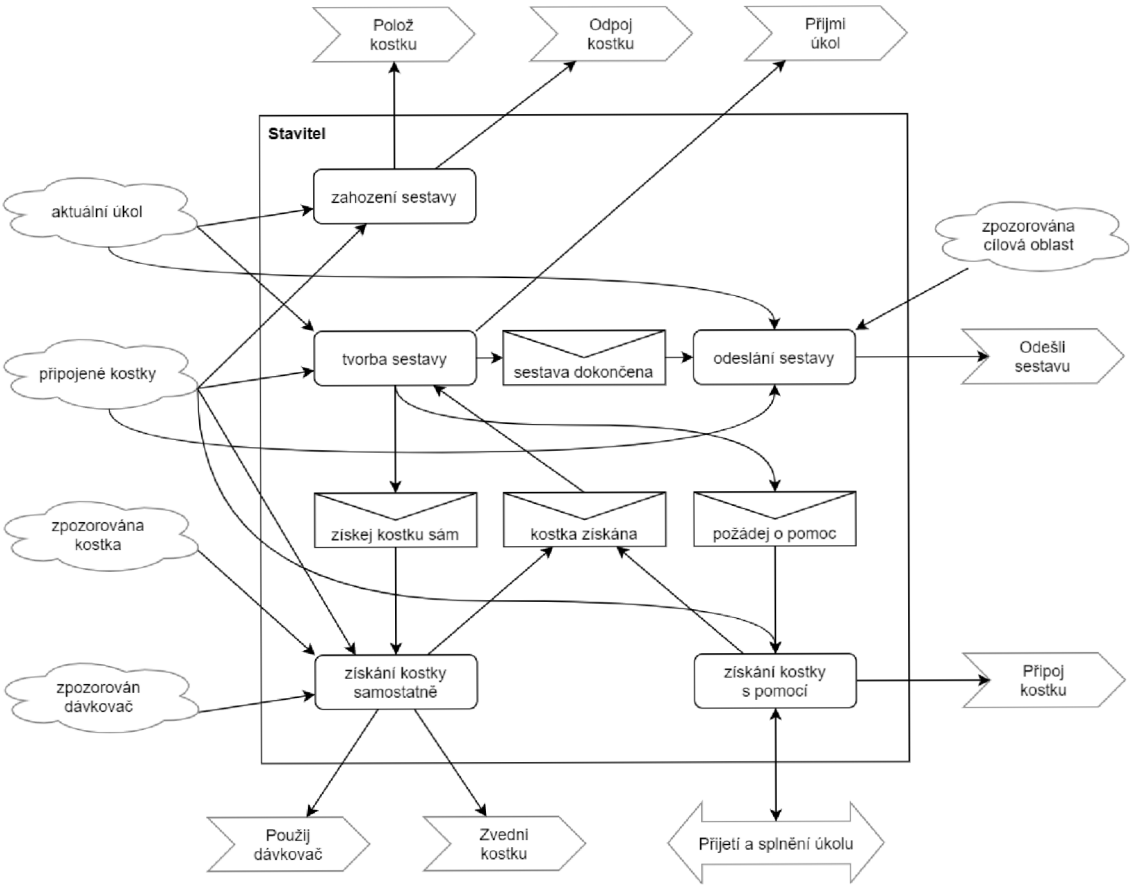
3. Vyhodnocení, zdali je potřeba agent záškodník na základě aktuálně nalezených nepřátelských agentů. Případné přidělení role nejvhodnějším agentům (zpravidla nejblíže nepřátelům).
4. Přidělení role průzkumník ostatním agentům s informací, do které oblasti (kterým směrem) se mají vydat.

Nalezení nejvhodnější cesty

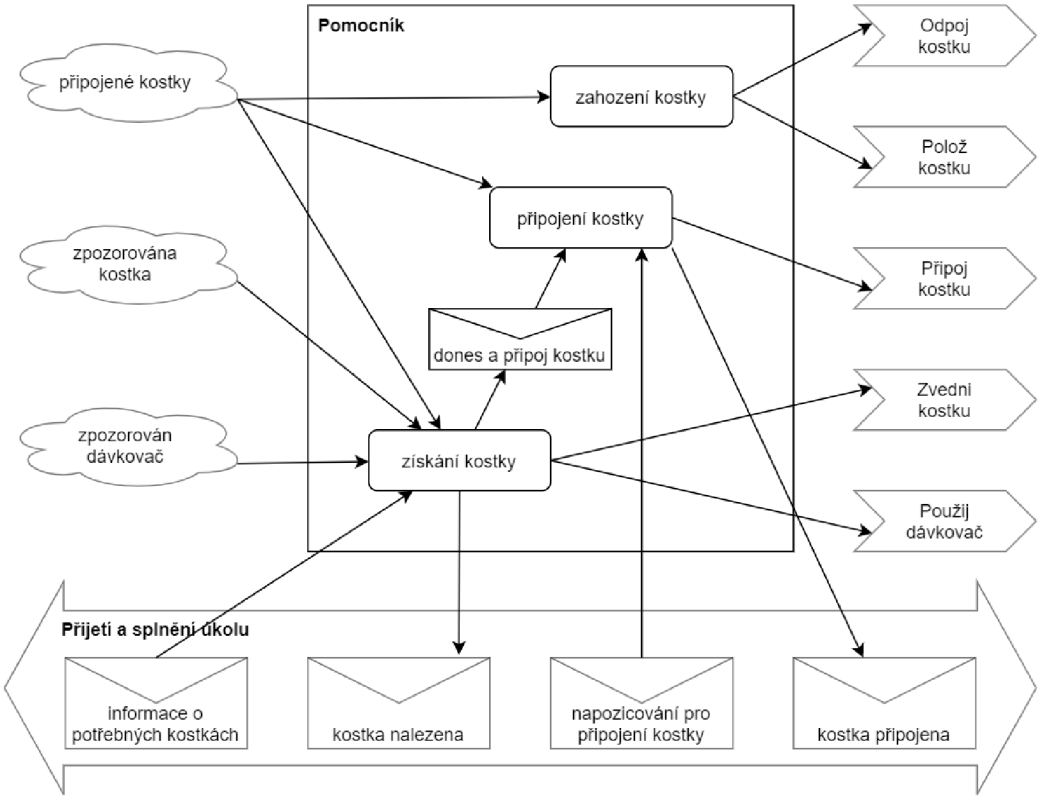
- **Agent**: Hledač cest
- **Popis**: Určení směru a nalezení nejrychlejší cesty k aktuálnímu cíli s ohledem na překážky a akce čištění.
- **Spouštěče**: potřeba přepočítání cesty, zjištěn nový cíl
- **Vstupní data**: pozice cíle
- **Výstupní data**: cesta k cíli a aktuální směr pohybu
- **Cíl**: Dopravení agenta na požadovanou pozici
- **Postup**:
  1. Vytvoření co nejkratší cesty mezi agentem a cílem s ohledem na vyhnutí se známým překážkám a blízkým akcím čištění.

Změna cesty

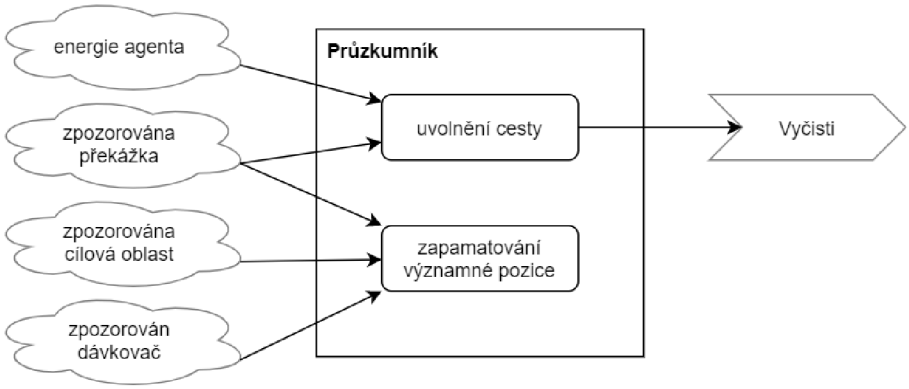
- **Agent**: Hledač cest



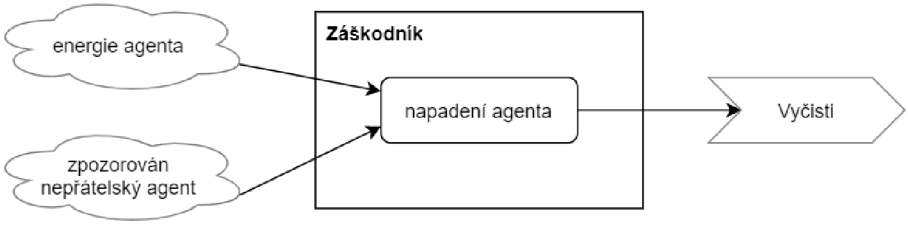
Obrázek 3.12: Diagram přehledu agenta - stavitel



Obrázek 3.13: Diagram přehledu agenta - pomocník



Obrázek 3.14: Diagram přehledu agenta - průzkumník



Obrázek 3.15: Diagram přehledu agenta - záškodník

- **Popis**: Potřeba přepočítání aktuální cesty z důvodu vyhnutí se překážce či nebezpečí.
- **Spouštěče**: zpozorována akce čištění v cestě, zpozorována překážka v cestě
- **Vstupní data**: pozice nebezpečné/neprůchozí oblasti
- **Výstupní data**: potřeba přepočítání cesty
- **Cíl**: Vyhnutí se nebezpečí či překážce
- **Postup**:
    1. Ověření, zdali je překážka či nebezpečí opravdu v cestě agenta.
    2. Upozornění sám sebe na přepočet cesty agenta.

Tvorba sestavy

- **Agent**: Stavitel
- **Popis**: Plán pro vytvoření sestavy pro nový nebo aktuální úkol. Určí, kterou kostku bude chtít stavitel obstarat sám a kterou bude chtít od pomocníka.
- **Spouštěče**: přijetí nového úkolu, získání kostky, připojení kostky
- **Vstupní data**: aktuální úkol, připojené kostky
- **Výstupní data**: rozhodnutí o kompletnosti sestavy a nebo o potřebě získání další kostky
- **Cíl**: Vytvořit sestavu dle zadaného úkolu
- **Postup**:
    1. Porovnání již vytvořené sestavy (připojených kostek) s požadovanou sestavou v zadaném úkolu.
    2. Rozhodnutí o další akci (odevzdání sestavy nebo získání další kostky samostatně či od pomocníka).

Získání kostky samostatně

- **Agent**: Stavitel
- **Popis**: Agent dojde k požadovanému dávkovači a získá potřebnou kostku sám. Případ může nastat jen tehdy, pokud má agent na některé své straně ještě místo na zvednutí další kostky.
- **Spouštěče**: rozhodnutí agenta o získání kostky samostatně
- **Vstupní data**: typ potřebné kostky
- **Výstupní data**: informace o získání kostky
- **Cíl**: Získání potřebné kostky
- **Postup**:
    1. Nalezení nejbližšího dávkovače požadovaného typu.
    2. Dopravení se k dávkovači.
    3. Použití dávkovače.
    4. Zvednutí kostky.

Získání kostky s pomocí

- **Agent**: Stavitel
- **Popis**: Agent sdělí svému přidělenému pomocníkovi aby mu donesl požadovanou kostku.
- **Spouštěče**: rozhodnutí agenta o získání kostky s pomocí

- **Vstupní data**: typ požadované kostky, zvolený pomocník
- **Výstupní data**: informace o připojení kostky
- **Cíl**: Získání a připojení potřebné kostky
- **Postup**:
    1. Stavitel pošle zprávu pomocníkovi, kterou kostku mu má donést a poté počká na odpověď.
    2. Stavitel pošle pomocníkovi informace o místě setkání a přesné pozici, na kterou má kostku připojit.
    3. Stavitel se vydá na místo setkání a počká na úspěšné připojení kostky.

Odeslání sestavy

- **Agent**: Stavitel
- **Popis**: Agent dojde do cílové oblasti a odešle vytvořenou sestavu.
- **Spouštěče**: informace o dokončení sestavy
- **Vstupní data**: připojené kostky, aktuální úkol
- **Výstupní data**: informace o úspěšném odeslání sestavy
- **Cíl**: Odeslání dokončené sestavy
- **Postup**:
    1. Ověření, zdali vytvořená sestava opravdu odpovídá požadované sestavě v úkolu.
    2. Nalezení a dopravení se do cílové oblasti.
    3. Odeslání sestavy.

Zahození sestavy

- **Agent**: Stavitel
- **Popis**: Zbavení se kostek, které nejsou vyžadovány aktuálním úkolem (případně i všech).
- **Spouštěče**: informace od koordinátora, přijat nový úkol
- **Vstupní data**: aktuální úkol, připojené kostky
- **Výstupní data**: -
- **Cíl**: Odpojení a zahození všech nepotřebných kostek
- **Postup**:
    1. Porovnání připojené sestavy s požadovanou v aktuálním úkolu.
    2. Odpojení a položení kostek.

Získání kostky

- **Agent**: Pomocník
- **Popis**: Obdržení informací od stavitele o potřebné kostce a její následné získání.
- **Spouštěče**: obdržení zprávy o potřebě kostky
- **Vstupní data**: připojené kostky, typ potřebné kostky
- **Výstupní data**: informace o získání kostky
- **Cíl**: Získání požadované kostky
- **Postup**:
    1. Nalezení nejbližšího dávkovače požadovaného typu.

2. Dopravení se k dávkovači.
3. Použití dávkovače a zvednutí kostky.
4. Odeslání zprávy staviteli, že byla kostka získána.

Připojení kostky

- **Agent**: Pomocník
- **Popis**: Pomocník připojí kostku do sestavy svého stavitele na danou pozici dle instrukcí od stavitele.
- **Spouštěče**: informace o úspěšném získání kostky
- **Vstupní data**: připojené kostky
- **Výstupní data**: informace o připojení úspěšném kostky
- **Cíl**: Připojit kostku do sestavy
- **Postup**:
  1. Pomocník vyčká na zprávu od stavitele ohledně pozice pro připojení kostky.
  2. Pomocník vyrazí na pozici sjednanou se stavitelem.
  3. Jakmile je na sjednané pozici i stavitel, pomocník připojí svou kostku do jeho sestavy.
  4. Pomocník informuje stavitele, že byla kostka připojena.

Zahození kostky

- **Agent**: Pomocník
- **Popis**: Zahození kostky v případě, že už není potřeba.
- **Spouštěče**: informace od koordinátora
- **Vstupní data**: připojené kostky
- **Výstupní data**: -
- **Cíl**: Odhození kostky za účelem uvolnění agenta
- **Postup**:
  1. Položení kostky.

Zapamatování významné pozice

- **Agent**: Průzkumník
- **Popis**: Účelem průzkumníka je zejména prozkoumávat oblast a nalézt nové významné věci, jejichž pozici pak sdílí s ostatními agenty skrze koordinátora.
- **Spouštěče**: zpozorován dávkovač, zpozorována cílová oblast, zpozorována překážka
- **Vstupní data**: pozice nově zpozorované věci
- **Výstupní data**: -
- **Cíl**: Zmapování prostředí
- **Postup**:
  1. Zapamatování si pozice nově zpozorované věci.
  2. Informování koordinátora o nově zpozorované věci.

Uvolnění cesty

- **Agent**: Průzkumník

- **Popis**: Brání-li překážka v prozkoumávání dané oblasti, průzkumník se ji pokusí odstranit aby mohl pokračovat v prozkoumávání oblasti.
- **Spouštěče**: zpozorována překážka v cestě
- **Vstupní data**: energie agenta, pozice zpozorované překážky
- **Výstupní data**: -
- **Cíl**: Odstranění překážky, která brání dalšímu průzkumu oblasti
- **Postup**:
    1. Kontrola, zdali má agent dostatek energie na provedení akce čištění.
    2. Provedení akce čištění cílenou na překážku v cestě.

Napadení agenta

- **Agent**: Záškodník
- **Popis**: Záškodník se z aktuálního pohybu nepřátelského agenta pokusí předpovědět, kterým směrem se nepřítel pohybuje. Následně se záškodník napozicuje tak, aby mohl provést akci čištění namířenou na cestu nepřítele s co největší pravděpodobností na zásah.
- **Spouštěče**: zpozorován nepřátelský agent
- **Vstupní data**: pozice nepřátelského agenta, energie agenta
- **Výstupní data**: -
- **Cíl**: Zpomalení nepřátelského týmu v plnění úkolů
- **Postup**:
    1. Analýza pohybu nepřátelského agenta a předpověď jeho následné pozice.
    2. Ověření, zdali má agent dostatek energie na provedení akce čištění a případná předpověď, za jak dlouho bude agent mít dostatek energie.
    3. Napozicování se k úspěšnému zásahu nepřátelského agenta akcí čištění (do pozicování se započítá i potřebná energie).
    4. Použití akce čištění na určenou pozici.

Kapitola 4

Systém JaCaMo

Skupina akademických pracovníků sestavila systém pro multiagentní programování z tří již existujících a fungujících rozdílných technologií, které ale dohromady splňují všechny požadavky pro vývoj i sofistikovanějších multiagentních systémů [2]. Z názvů použitých technologií pak vzniklo pojmenování *JaCaMo*. Ke konstrukci samotných agentů je využit programovací jazyk ***Ja****son*[1]. O artefakty a přejímání vjemů z prostředí se stará ***CA****rtAgO*[2]. Struktury a organizace agentů má v režii ***Mo****ise*[3].

Jason je interpret rozšířené verze jazyku *AgentSpeak* a vychází z programovacího jazyku *Java*. Jedná se o *BDI*[4] architekturu agentního programování [4].

CArtAgO[5] je nástroj pro spouštění a adaptaci virtuálních prostředí (i virtuálních aplikací) pro multiagentní systémy. Je založen na interakci agentů a artefaktů, které se dají chápat jako dynamicky generované nástroje a zdroje pro agenty ke splnění jejich nejen idividuálních, ale i společných akcí [11].

Moise je organizační model používaný k definování a strukturování rolí, skupin, aj. v multiagentních systémech. Smyslem modelu je jeho využití nejen samotnými agenty, kteří se podle něj mají řídit, ale také pomoc s režií skupin v nadřazeném/řídícím systému [8].

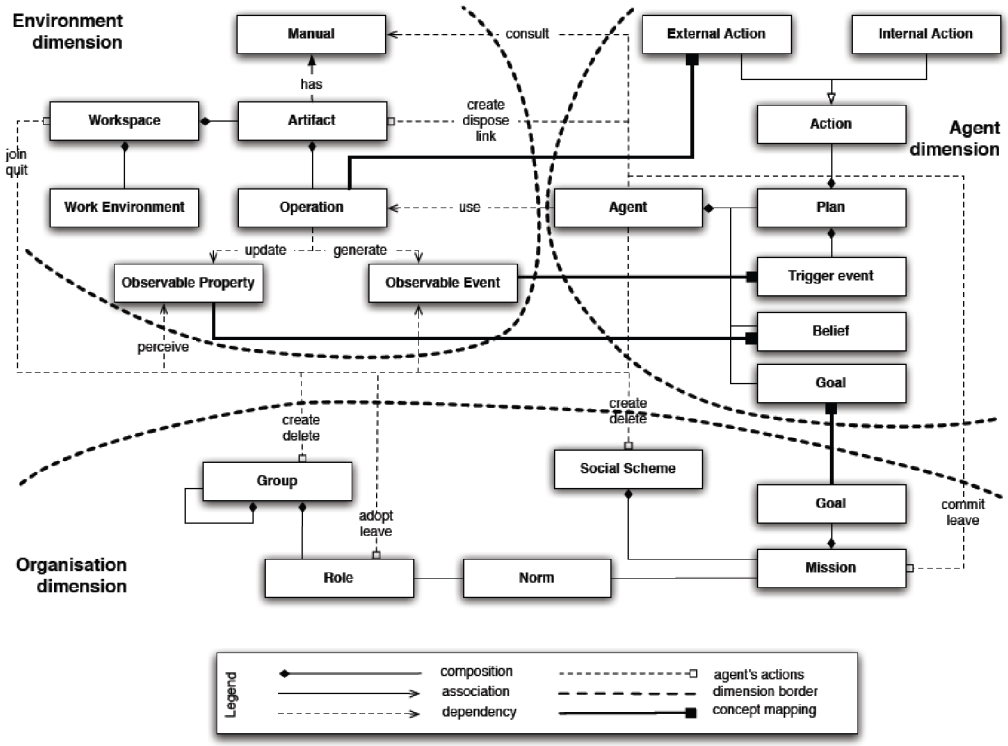
4.1 Hlavní rysy

Každá ze tří zmíněných nezávislých platforem, které tvoří systém JaCaMo, přináší vlastní abstrakce a modely. Proto JaCaMo spojuje dohromady a vytváří klíčový meta-model [2], který bere v úvahu všechny abstrakce zpřístupněné každou zahrnutou platformou. Zmíněný meta-model byl realizován s cílem definovat závislosti, propojení a hlavně koncepční mapování a souhru mezi všemi různými abstrakcemi dostupnými v zahrnutých podsystémech a jejich meta-modely.

4.1.1 JaCa model

Základy systému JaCaMo jsou postaveny na programovacím modelu *JaCa* [3]. V JaCa modelu je agentní systém navržen jako sada agentů, kteří spolupracují ve sdíleném prostředí. Vytváření aplikace v modelu JaCa znamená programování agentů na jedné straně a za-

[1] http://jason.sourceforge.net/
[2] http://cartago.sourceforge.net/
[3] http://moise.sourceforge.net/
[4] **B**eliefs - **D**esires - **I**ntentions
[5] Common ARTifact infrastructure for AGents Open environments



Obrázek 4.1: Rozdělení dimenzí v meta-modelu systému JaCaMo [3]

pouzdření dynamiky prostředí a logiky řízení úkolů, které je třeba provést, na druhé straně. Stojí za zmínku, že se jedná o endogenní představu o prostředí, tj. prostředí zde je součástí softwarového systému, který je ještě potřeba vyvinout [12].

4.1.2 Dělení na dimenze

JaCaMo se skládá a spoléhá na souhru tří dimenzí ilustrovaných na obrázku 4.1. **Dimenze prostředí**, **dimenze organizace** a **dimenze agentů**. Do dimenze prostředí spadá zejména pracovní prostředí, artefakty a operace nad nimi. Dimenze agentů se věnuje přímo agentům, jejich plánům, akcím a dalším náležitostem, které musí agent obsahovat. Organizační dimenze spravuje hlavně organizační strukturu agentů, jejich role a případně skupiny.

Interakce mezi jednotlivými dimenzemi probíhá synergicky, tj. že spolu zároveň reagují dva prvky. Interakce mezi agenty a prostředím probíhá z pohledu agenta skrze artefatky prostředí podle požadovaných akcí a obdržených vjemů. Interakce mezi organizací a prostředím je založena organizací řízenou infrastruktury pro Moise. Základní myšlenka představovala navržení organizační struktury jako část prostředí, kde jsou situování agenti. Organizační schémata, pravidla prostředí, plány, úkoly aj. jsou vytvářeny pomocí artefaktů [2].

4.2 Souhrn silných a slabých stránek systému

Jako každý systém, i JaCaMo má své plusy a mínusy. Nejedná se pouze o sloučení výhod a nevýhod využitých subsystémů jelikož jejich syntézou jsou některé jejich aspekty elimi-

novány či upozadněny a naopak systém nabude několika nových rysů, které lze rozebírat a hodnotit.

Hlavním negativem, které logicky plyne ze sloučení více systémů, je **navýšení složitosti** návrhu a implementace. Vzniká nutnost alespoň minimální znalosti všech zahrnutých subsystémů. Také je potřeba porozumět vazbám mezi jednotlivými subsystémy a celkovému fungování výsledného systému. S rozsáhlejším systémem narůstá také množství režie, kterou je nutné zaopatřit.

JaCaMo umožňuje **programovat agenty** a agentní skupiny včetně definování případných rolí a struktury skupin. Dále poskytuje možnost k implementaci detailního aparátu pro usnadnění **interakce s prostředím**. Všechny uvedené funkcionality zaobaluje do jednoho společného systému.

Oproti jiným programovacím jazykům, jako např. *Jason*, nachází systém *JaCaMo* podporu ve vývojovém prostředí *Eclipse*[6] za použití vyvinutého pluginu, které je spolu s návodem na jeho zprovoznění k dostání na URL adrese `http://jacamo.sourceforge.net/eclipseplugin/tutorial/` nebo také přístupný z oficiálních stránek projektu *JaCaMo* [3]. Vývojové prostředí pak přizpůsobí své uživatelské rozhraní pro přehledné zobrazení struktury projektu vyvíjeného v systému *JaCaMo*.

Výhody i nevýhody lze vidět na porovnání systému *JaCaMo* s jazykem *Jason*. Vhodné řešení problému definovaného soutěží v kapitole 2 lze docílit i implementací v samotném *Jasonu*. Implementace by se mohla zdát jednoduší jelikož je potřeba být obeznámen pouze s jazykem *Jason*, nicméně neexistuje žádný aparát, který by definoval či alespoň informoval o organizaci a struktuře implementovaných agentů. I když by vznikl funkční multiagentní systém se správou rolí, samotný *Jason* nezná koncept rolí a ani koncept organizace či členění agentů do skupin. Zmíněné chování by se dalo pouze simulovat náležitou implementací správných podmínek a plánů. Oproti tomu systém *JaCaMo* přímo definuje strukturu používaných rolí agentů za pomoci *Moise*.

Dále lze opět srovnáním s jazykem *Jason* ukázat na důležitost subsystému *Cartago*, díky kterému lze upravovat implementaci rozhraní systému *JaCaMo* mezi agenty a okolím. Takže už pak nevzniká potřeba větší modifikace komunikace mimo systém *JaCaMo*. Naproti tomu v jazyce *Jason* není k dispozici modifikovatelné rozhraní a je tedy potřeba vytvořit adaptér, který bude komunikovat skrze externí protokol a předávat vjemy a akce agentů.

[6] `https://www.eclipse.org/`

Kapitola 5

Implementace

Navzdory možným náznakům, kapitola nepojednává o celkové realizaci multiagentního systému. Obecný rozbor celkové implementace by byl příliš nudný a téměř o ničem nevypovídající. Naopak příliš dětailní popis celé implementace by působil spíše rušivým dojmem a poskytoval mnoho zbytečných informací. Proto se kapitola zaměřuje zejména na detailní rozbor pouze některých zajímavých a důležitých částí systému. Pojednává jednak o již implementovaných aspektech, ale také se věnuje i vhodným vylepšením, které ještě nebyly do systému zakomponovány.

Pro realizaci řešení byl vybrán právě zmíněný systém *JaCaMo*, zejména díky svým výhodám oproti obyčejné implementaci v samostatném *Jasonu*. I když je drtivá většina kódu napsaná právě v jazyce *Jason*.

Propojení samotného systému JaCaMo s jádrem soutěže skýtá mimo jiné problémy také potřebu implementovat vlastní komunikační protokol, který není předmětem práce. Organizátoři soutěže totiž implementovali jádro systému soutěže v programovacím jazyce *Java* a přidali jen hrstku možností komunikace mezi agenty a systémem. Mezi možnostmi je sice zahrnut i již implementovaný protokol s jazykem *Jason*, což je ve své podstatě základ systému *JaCaMo*, ale ten bohužel nelze použít i pro *JaCaMo*. Proto se implementace této práce odráží od řešení týmu *MLFC*, který se v aktuálním roce 2021 soutěže zúčastnil a umístil se na druhém místě. Bylo využito pouze již vytvořeného komunikačního protokolu mezi agenty a systémem *JaCaMo* a souvisejícími strukturami. Implementace agentů byla samozřejmě nahrazena za vlastní spolu s organizačním souborem pro definici rolí agentů.

Návod, jak spustit jejich řešení, má zmíněný tým uveden v souboru `README.md` na GitHubu https://github.com/autonomy-and-verification-uol/mapc2020-lfc. Stačí pouze do adresáře `src` vložit soubory této práce a nahradit již existující. Dojde tak k nahrazení organizační struktury rolí a hlavně k přepsání souboru `agent.asl`, který obsahuje základní chování agentů a zahrnuje do sebe i plány, záměry a jiné náležitosti z dalších souborů. Poté stačí již pokračoval dle návodu spuštění řešení přes vývojové prostředí *Eclipse*[1].

5.1 Zajímavé aspekty a řešené problémy vlastní implementace

Byly vybrány pouze některé části implementace, které stojí za zmíňku. Detailní informace o běžných a jednoduchých aktivitách jsou mnohdy příliš nezajímavé pro detailnější rozbor.

[1] https://www.eclipse.org/

5.1.1 Pohyb a reakce na nebezpečí

Významným aspektem chování agentů je schopnost reagovat na možná nebezpečí v prostředí. Tedy jak agent zareaguje, pokud se vyskytne v oblasti označené pro provedení akce čištění a nebo pokud do takové oblasti míří. Ke správnému pochopení je však nejprve vhodné alespoň stručně zmínit, jak se agent v prostředí pohybuje.

Agent ve své bázi znalostí vždy uchovává svůj aktuální cíl včetně informací o jeho pozici. Vždy se snaží najít co nejkratší cestu k cíli. Pokud by měl narazit na překážku, pokusí se ji obejít. Zejména při obcházení se ale může stát, že agent uvázne. Agent si pamatuje své předchozí pozice několik kroků do minulosti, díky čemuž může identifikovat, zdali se pohybuje stále ve stejném okolí překážky místo toho, aby se přibližoval k cíli. V takovém případě agent usoudí, že uváznul a dočasně si určí nový cíl v náhodném směru a vzdálenosti od své aktuální pozice. Ač se jedná o velmi primitivní metodu řešení uváznutí, která ještě navíc spoléhá na náhodu, prokázala se ve většině simulací jako dostačující. Bylo by však vhodné ji ještě vylepšit alespoň o kontrolu, zdali cesta k novému dočasnému náhodně vybranému cíli nemíří opět přes blízké překážky a nebo znova do místa, kde se agent zaseknul.

Pokud agent ve své blízkosti zpozoruje oblast označenou pro akci čištění, chová se k ní jako by se jednalo o neodstranitelnou překážku a bude se ji snažit obejít tak, aby ani žádná z jeho připojených kostek nezasahovala do nebezpečné oblasti. V případě, že se agent v označené oblasti rovnou vyskytne, jeho nejvyšší prioritou se stane ústup na nejbližší bezpečné pole, odkud bude již normálně pokračovat dále ke svému původnímu cíli.

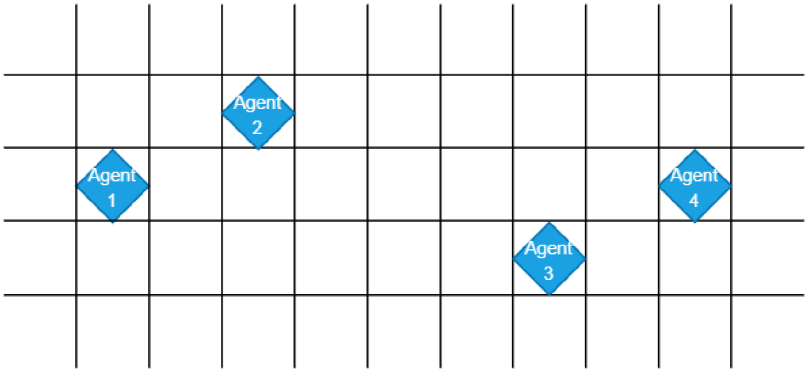
Může se stát, že agent nestihne uniknout do bezpečí včas a akce čištění zasáhne některé z jeho připojených kostek, což má za následek vymazání zasažené kostky. V každém kroku si agent ověřuje, které kostky jsou k němu připojeny a zdali stále existují, a tak je snadné zjistit, jestli některá kostka byla vymazána. Agent v tu chvíli přehodnotí své priority a zamíří opět k náležitému dávkovači pro získání nové kostky.

5.1.2 Synchronizace pozic

Nejen že agenti neznají svoji absolutní pozici v prostředí a ani nemají ponětí o umístění ostatních agentů, ale dokonce ani nemohou jednoduše určit totožnost agenta, pokud nějakého potkají. Vidí pouze do jakého týmu zpozorovaný agent přísluší.

K úspěšnému plnění úkolů potřebují alespoň dva agenti znát relativní pozice vůči sobě navzájem aby mohli kompletovat kostky do požadovaných sestav, k čemuž jeden agent nestačí. Budou-li agenti znát svá relativní umístění vůči sobě, skýtá to pro ně i další výhody jako např. sdílení nalezených a zapamatovaných významných objektů (dávkovače, cílové oblasti, aj.) a nebo domlouvání se na společném místě setkání za účelem předání či připojení požadované kostky.

O synchronizaci údajů o vzájemných pozicích se agenti pokoušejí vždy, když ve svém okolí spatří jiného agenta ze stejného týmu. Jelikož neznají jméno viděného agenta a ani jej nemohou nijak jednoduše identifikovat, tak všichni agenti posílají v každém kroku koordinátorovi zprávu s informacemi, jestli vidí spojeneckého agenta a případně na jaké pozici. Koordinátor se poté snaží v získaných datech hledat navzájem inverzní souřadnice spatřených spojenců. Např. pokud jeden agent vidí spojence na pozici `[1, 3]` a další agent agent jej vidí na opačných souřadnicích `[-1, -3]`, tak koordinátor může vyvodit, že se zrovna tito dva agenti na zmíněných pozicích vzájemně vidí. Koordinátor pak informuje agenty, že se vzájemně vidí a pošle jim jméno jejich spatřeného spojence aby si mohli vytvořit *synchronizační bod*, dle kterého pak mohou odvozovat své vzájemné relativní pozice.



Obrázek 5.1: Případ nejednoznačné identifikace při synchronizaci pozic agentů. *Agent 1* i *Agent 3* pošlou koordinátorovi zprávu, že vidí jiného agenta ze stejného týmu na souřadnicích `[2, 1]`. *Agent 2* a *Agent 4* pošlou invertované souřadnice. Jelikož koordinátor nalezne více shod a nemá žádný způsob, jak mezi nimi agenty rozlišit, tak pro aktuální krok zúčastněné agenty ze synchronizace pozic vynechá a počká si na přívětivější situaci.

Může nastat situace, že koordinátor obdrží informace se shodnými souřadnicemi spatřených spojenců od více agentů. Např. že dva agenti vidí spojence na souřadnicích `[2, 1]` jako na obrázku 5.1. Pak se naskytuje vícero možností, kteří agenti se navzájem vidí a nelze je jednoznačně odlišit. V takovém případě koordinátor raději nerozhoduje a čeká na pozdější jednoznačnou situaci.

Tým *MLFC*, jehož řešení pomohlo postavit základy pro tuto práci, řešil problém se synchronizací agentů tak, že definoval roli kartografa pro jediného agenta. Ten měl kromě prozkoumávání oblasti a zapamatovávání si významných objektů, také za úkol obesílat zprávami spojence, které potká a kteří pak synchronizovali svou pozici skrze kartografa.

5.1.3 Koordinace agentů

Každý agent má své vlastní vyhodnocovací moduly, které definují jeho chování a dle kterých jedná. Agenti se samozřejmě mohou dorozumívat spolu navzájem, ale aby nešlo o chaotickou nebo jen stěží kontrolovatelnou komunikaci, vnikla potřeba přenechávat určité rozhodovací problémy na jediného agenta, ke kterému půjdou všechny potřebné informace a který bude ostatní agenty koordinovat.

Nejprve je zapotřebí určit, který agent se zhostí koordinace. Jelikož koordinátor jako role nemusí nijak interagovat s prostředím, ale pouze komunikuje s ostatními agenty, lze ji přiřadit kterémukoliv agentovi. Pro jednoduchost je vždy vybrán právě první agent z týmu. Ještě je však potřeba tuto skutečnost dát na vědomí i ostatním agentům ze stejného týmu. Pomocí komunikační funkce `.broadcast` se všichni spojenci dozví, kdo je koordinátor a z jejich odpovědí naopak odesilatel získá jména svých podřízených agentů.

Koordinace probíhá v každém kroku simulace, a to tak že nejprve všichni agenti pošlou koordinátorovi souhrn využitelných informací jako např. zapamatované pozice zajímavých objektů, aktuální role, připojené kostky, aj. Na základě určitých směrodatných informací pak koordinátor rozhoduje o přerozdělování rolí. Nejprve zhodnocuje pro každého stavitele, zdali si má ponechat svou roli a nebo mu bude odebrána, např. z důvodu blížícího se deadlinu na odevzdání úkolu. Poté se koordinátor snaží vybrat další adepty pro roli stavitele (a jeho pomocníka) dle požadovaných kritérií. Stavitel musí být schopen splnit alespoň jeden z existujících úkolů což znamená, že musí splňovat 3 rozhodující kritéria:

- znalost pozic všech dávkovačů jejichž kostky úkol vyžaduje,
- znalost alespoň jedné cílové oblasti, do které následně půjde úkol odevzdat,
- synchronizace s alespoň jedním spojencem, který může být pomocníkem.

Úkoly pro stavitele se vybírají s ohledem na množství času, které je vymezeno k jejich splnění a také s ohledem na již přidělené úkoly jiným stavitelům. Koordinátor také limituje počet stavitelů na maximálně 1/3 celkového počtu agentů v jednom týmu aby zbyl prostor pro další role. Pokud by měla celá 1/3 agentů roli stavitele, znamenalo by to existenci minimálně další 1/3 rolí pomocníků, což už dává dohromady 2/3 agentů věnujících se plnění úkolů. Ostatní role by tedy byly rozděleny mezi zbývající 1/3 agentů.

Je nutné vzít na vědomí, že agenti vykonávají svou činnost paralelně bez jakékoliv záruky synchronního provozu pokud není zajištěn programátorem. Mohlo by tedy dojít k tomu, že některý agent již bude rozhodnutý o své akci aniž by koordinátor dokončil koordinaci ve stejném kroku. Proto agenti před rozhodováním, kterou akci provedou, nejprve vyčkají dokud jim koordinátor nepošle všechny potřebné informace a nedá pokyn k pokračování.

5.1.4 Reakce na selhání akce

V poskytnutém prostředí má každá akce agenta určitou šanci na selhání. Nelze tedy již při provádění akce provádět změny v bázi znalostí agenta, které předpokládají úspěšné provedení akce (např. označit kostku jako připojenou při provádění akce `connect`). Naštěstí v každém kroku prostředí poskytuje každému agentovi vjemy s informacemi o jeho poslední akci a zdali proběhla úspěšně. Dalo by se říct, že některé akce mají tedy dvě fáze. Jednak samotné provedení a v následujicím kroku pak i kontrolu, zdali akce byla úspěšně provedena a lze tedy náležitě změnit bázi vědomostí agenta.

Všechny případy, kdy je potřeba kontrolovat výsledek akce, souvisí s úkoly a připojováním kostek. Prostředí vůbec neposkytuje informaci o skutečnosti, zda byl úkol agentem přijat, i když by dle oficiální dokumentace soutěže tento vjem měl být poskytnut, pokud agent úkol úspěšně přijal. Je tedy potřeba kontrolovat výsledek akce přijetí a odevzdání úkolu. Stav si pak musí agent sledovat sám. Další kontroly poslední akce se týkají připojování, sbírání a pokládání kostek. Opět kvůli nekonzistenci vjemů z prostředí, které agentovi sice pošle vjemy určující, které kostky jsou k němu připojeny, ale také agent uvidí i kostky všech ostatních blízkých agentů, jako by byly připojeny k němu samotnému, což činí zmíněný vjem prakticky nepoužitelný. Agent si tedy opět informace související s operacemi nad kostkami musí kontrolovat sám, a tedy i hlídat si stav aktuálně připojených kostek.

5.2 Plány a návrhy na zlepšení

Dle výsledků testování v kapitole 6, v implementovaném řešení je stále spousta místa pro zlepšení. Dále jsou popsány plány na vylepšení některých významných aspektů jejichž realizace by jistě vedla k markantnímu zvýšení efektivnosti vyhotoveného řešení. Většina z uvedených nápadů bohužel nebyla implementována zejména z časových důvodů.

5.2.1 Vylepšení již implementovaných aspektů

Z již vyhotoveného řešení by si zasloužilo další péči zejména několik částí jejichž implementace byla zamýšlena jen jako dočasná než se přejde k efektivnějšímu a mnohdy také časově náročnějšímu způsobu realizace.

Propracovanější koordinace

V aktuální stavu řešení ověřuje koordinátor při rozdělování rolí pouze zdali agenti splňují či nesplňují kritéria pro úpravu priorit a rolí. Bylo by však vhodné do rozhodování zakomponovat i vektorové údaje, a to konkrétně:

- aktuální umístění agentů vzhledem k potřebným dávkovačům, cílům a potenciálním pomocníkům, následované kalkulací pozičně nejvhodnějšího agenta pro vhodný úkol
- deadliny a odměny za splnění úkolů pro výpočet, zdali se úkol vyplatí plnit
- pozice nepřátelských agentů

Zhodnocení průběhu přijatého úkolu

Každý úkol, který stavitel přijme, obsahuje informaci o počtu kroků, do kdy je nutné/možné úkol úspěšně splnit a odevzdat. Bylo by vhodné, aby si každý stavitel průběžně propočítával, zdali je ještě možné úkol splnit vzhledem k aktuálnímu stavu, který se určí podle aktuálně připojených kostek a vzdáleností od dalších potřebných dávkovačů a od následné nejbližší cílové oblasti. Pokud stavitel vypočte, že už úkol není možné do požadovaného počtu kroků splnit a nebo že už je pravděpodobnost na splnění příliš nízká, tak úkol zahodí. Koordinátor se v následujícím kroku rozhodne, co se stavitelem bude dál.

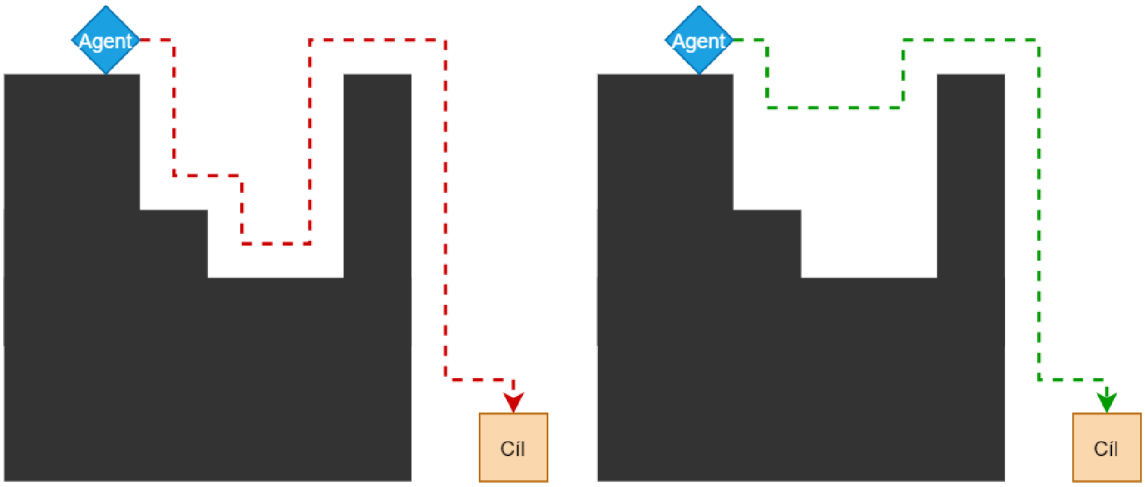
Strategické shromažďování stavitelů v cílové oblasti

Pokud stavitel získá všechny kostky, které mohl nasbírat svépomocí, odebere se do nejbližší cílové oblasti, kde počká na pomocníka se zbývajícími kostkami. Pokud se stavitel neocitne v nebezpečné zóně označené pro akci čištění, tak zůstane stát na stejném místě, dokud se svým pomocníkem nepřipojí zbývající kostky a dokud úspěšně neodevzdá právě plněný úkol. Zmíněné chování však může mít za následek blokaci další spojeneckých stavitelů, kteří přijdou čekat na své pomocníky do stejné cílové oblasti. Bylo by vhodné vylepšit implementaci tak, aby stavitelé ve stejné oblasti čekali s většími odstupy mezi sebou, aby měli pomocníci prostor na průchod a připojení dodatečných kostek. Případně aby byl potřebný prostor vytvořen jakmile se přiblíží jeden z pomocníků.

Vylepšení pohybu agenta v prostředí

Samotný pohyb k aktuálnímu cíli nemá mnoho potenciálu ke zlepšení. Avšak obcházení překážek a reakce na blízké nebezpečí je vhodným kandidátem pro úpravu. Kvůli akcím čištění prováděným z prostředí i od agentů se celá plocha, po které se agenti pohybují, s časem mění. Je tedy zbytečné zapamatovávat si pozice překážek, které již za několik kroků nemusí být na stejném místě. Proto má smysl pracovat pouze s překážkami v blízkém okolí, do kterého agent vidí.

Zlepšení pohybu k požadovanému cíli spočívá zejména v rozšíření oblasti, ve které agent reaguje na překážky. Momentálně si agent v každém kroce určí směr, kudy chce jít, ale pokud je v daném směru překážka, zvolí alternativní cestu. Zmíněným způsobem je reálné vnímání překážek z hlediska pohybu omezeno pouze na ty, které s agentem ve stejném kroce přímo sousedí. Může se tak stát, že se agent bude držet u zdi a zbytečně si prodlouží cestu a nebo se v horším případě zasekne. Což by se nemuselo stát, pokud by agent bral v potaz i vzdálenější překážky, které vidí jsou v jeho zorném poli a náležitě plánoval vhodnější cestu s ohledem na blízkou neprůchozí oblast. Případ, kdy by zmíněné chování mohlo pomoci je ilustrován na obrázku 5.2.



Obrázek 5.2: Ilustrace pohybu agenta okolo překážky k aktuálnímu cíli. Nalevo je vyobrazena cesta, kterou by agent použil v aktuálním řešení práce jelikož při obcházení překážky bere ohled pouze na své blízké okolí. Napravo je ideálnější cesta, kterou by agent volil, pokud by bral v potaz celé své okolí, do kterého je schopen vidět (aktuálně na vzdálenost 5 polí). Jakmile by udělal jeden krok níž, už by se do jeho zorného pole dostaly všechny blízké překážky potřebné k usouzení, že cesta směrem šikmo dolů je slepá.

Další oblast pro vylepšení pohybu spočívá v optimalizaci metody řešení uváznutí agenta. Samotná detekce uváznutí funguje téměř spolehlivě tak, jak je popsána v sekci 5.1.1. Problémy se však mohou naskytnout při řešení uváznutí, a to zejména již při vybírání nového dočasného cíle, kam se bude agent snažit docestovat. Momentálně se cíl v případě zaseknutí určuje zcela náhodně, pouze s podmínkou, že musí být v určité vzdálenosti od agenta. Může tedy nastat situace, že nově určený cíl vznikne ve stejném a nebo velmi blízkém směru jako agentův aktuální cíl, což bude mít za následek jen další zcela stejnou formu uváznutí. Bylo by vhodnější, aby se při určování nového dočasného cíle bral ohled i na aktuální cíl agenta. Nový cíl by musel mít dostatečně odlišný směr, aby se zamezilo opětovnému uváznutí agenta ve stejné oblasti.

Efektivita pohybu lze ještě více zvýšit zakomponováním používání akce čištění, pokud bude agentovi stát v cestě menší odstranitelná skupina překážek. Jelikož provedení akce čištění zabere předem daný počet kroků a dohromady může agent pročistit pouze 5 polí (zaměřené pole a všechny přímo sousedící buňky), je potřeba náležitě zhodnotit, zdali bude výhodnější překážku obejít a nebo si uvolnit cestu pomocí akce čištění.

Role sabotéra

V aktuálním řešení role sabotéra není dokončena, takže ji koordinátor zatím nikomu nepřiřazuje. Alespoň k základnímu fungování role, aby mohla být přidělována, je potřeba doimplementovat:

- **Chování sabotéra** ve smyslu následování blízkých nepřátelských agentů s případným umístěním se do směru jejich cesty a blokováním pohybu. Agent by měl zvažovat, zdali je vhodné na nepřítele použít akci čištění a zohledňovat šanci na zásah. Prioritou by pro sabotéra měli být nepřátelští agenti s již připojenými kostkami.

- **Rozhodování koordinátora** zda roli sabotéra přidělit či ponechat zvažovanému agentovi na základě spatřených nepřátel v okolí agenta.

5.2.2 Přidání nových funkcionalit

Předchozí sekce této kapitoly se věnovala *úpravě* a *optimalizaci* již vytvořeného řešení. Nyní se jedná o *přidání* zcela nových aspektů chování agentů za účelem zvýšení jejich úspěšnosti v soutěži. Následující návrhy na vylepšení řešení představují sémanticky netriviální a implementačně časově náročné úpravy již vytvořeného řešení, jejichž správnou funkčnost je nutné při případné realizaci řádně otestovat.

Chování agenta bez cíle

V případě, že koordinátor nepřidělil některému agentovi vhodnou roli nebo cíl, tak bude takovému agentovi přidělena automaticky role průzkumníka. Agent si poté náhodně generuje pozice, ke kterým se snaží dorazit, a tím prozkoumává okolní oblast.

Vzniká zde rozsáhlý prostor pro celou škálů možných návrhů, jakou činnost by agent mohl vykonávat v případě poskytnutí zmíněné volnosti. Možností lze vymyslet spousty, avšak je potřeba u každé řádně zvážit případný přínos a náročnost vyhotovení. Mezi nejvhodnější varianty patří:

- **Cílený průzkum** dosud neobjevených oblastí spočívající v implementaci určitého algoritmu pro volby nových vhodných pozic k prozkoumání. Nemělo by se jednat o jednoduchý postup, který prostě vybere nějaký bod v prostředí, na kterém agent doposud nebyl. Algoritmus by měl aplikovat určitou strategii prozkoumávání zatím neobjevené oblasti a ideálně takovým způsobem, aby se společným úsilím několika průzkumníků zvýšila efektivita průzkumných agentů. V ideálnějším případě by měl algoritmus zvažovat také pohyb v již prozkoumané oblasti za účelem mapování aktuálních překážek a nepřátelských agentů. Agenti se vždy pohybují v prostředí, které má konečnou velikost, takže by měli počítat i s případem, kdy už budou mít prozkoumáno celé prostředí a nebudou existovat další neobjevené pozice.
- **Příprava** na obdržení role. Přímo se vybízí, aby si agent nasbíral různé kostky z dávkovačů pro případ, že bude obdařen rolí stavitele či pomocníka a třeba již bude mít u sebe některé kostky, které jsou potřeba ke splnění přiděleného úkolu. Současně by na již přidělené kostky měl přihlížet i koordinátor během rozdělování rolí a úkolů agentům, aby mohl připravenému staviteli přidělit zrovna takový úkol, který bude moci rychleji splnit vzhledem k již připojeným kostkám.

Ovlivňování nepřátel planými akcemi čištění

Na úspěšné provedení akce čištění potřebuje agent soustavně vykonávat tuto akci několik po sobě jdoucích kroků. Použití akce stojí agenta energi, ale pouze v případě, že ji provede úspěšně. Je možné kdykoliv vykonávání akce čištění přerušit aniž by se agentovi odečetla energie za její použití. Už při započetí akce čištění se v prostředí začne zobrazovat v zaměřené oblasti všem agentům upozorňující označení, které vypovídá, že se v označené oblasti brzy provede akce čištění. Tedy pokud ji agent úspěšně dokončí. Lze předpokládat, že v řešení jiných účastníků soutěže bude implementována adekvátní obranná reakce na akci čištění, tedy konkrétně už na objevení označení oblasti. Lze předpokládat, že podobně jako v řešení této práce, se budou i agenti jiných týmů snažit co nejdříve dostat z nebezpečné oblasti a budou se jí chtít vyhnout.

Stálo by tedy za úvahu, naprogramovat záškodníka tak, aby byl nejen schopen provádět akce čištění, ale aby jimi mohl i "strašit"nepřátelské agenty. Jelikož agent může akci v jednom kroku začít aby se v prostředí objevilo varovné označení a klidně ji hned v dalším kroku zdarma přestat provádět, mohl by tak nepřátelské agenty v podstatě znehybnit a nebo dokonce i navádět úplně jinam, než původně chtěli jít.

O účinnosti vylepšení tohoto rázu lze diskutovat. Je zcela založeno na předpokladu, který však nemusí být pravdivý a agent by tak odváděl zbytečnou práci a jen se zdržoval. Na druhou stranu pokud bychom chtěli uvedený předpoklad vyvracet, je potřeba si položit směrodatnou otázku, zdali by vůbec mohl některý tým v soutěži být tak odvážný aby si dovolil ignorovat označení pro provedení akce čištění, která budou vyvolána agentem.

Formace záškodníků

Dalším ambicióznějším návrhem by byla implementace spolupráce agentů takovým způsobem, aby se v určitých případech mohli seřazovat do formací (nejefektivněji nejspíš pouze do linií), které by efektivněji zamezily pohybu nepřátelských agentů. Obyčejnou překážku by mohl agent odstranit pomocí akce čištění, ale agenti na rozdíl od překážek zůstanou neprůchozí i když jsou zasaženi akcí čištění. Další výhodu představují nové možnosti, jak mohou agenti dohromady využít akci čištění k pokrytí mnohem větší oblasti.

Návrh je ambiciózní zejména z důvodu, že by bylo potřeba řešit hned několik problémů zároveň. Mezi ty nejzávažnější spadá:

- **Selhání akce** jednoho z agentů ve formaci. Kvůli existenci určité šance na selhání jakékoliv akce agenta, je potřeba počítat s `n` krát tak velkou šancí na selhání akce pro formaci `n` agentů. Myšleno tak, že pokud selže akce alespoň jednoho agenta ve formaci, musí na něj ostatní agenti v dalším kroku počkat a nebo náležitě upravit své chování či formaci, aby došlo k co nejmenší ztrátě pohyblivosti celé formace.
- **Náročnější pohyb** přes neprůchozí překážky a objekty. Logicky čím rozsáhlejší bude vytvořená formace, tím hůře s ní bude možné v prostředí konzistentně pohybovat. Zejména kvůli roztroušeným překážkám. Obtíže s cestováním mohou mít i spřátelení agenti, v jejichž cestě se bude zrovna nacházet formace záškodníků. V takovém případě bude potřeba řešit, zdali se mají záškodníci rozestoupit (a tím přijít o drahocenné kroky svého původního záměru) aby mohl spojenec projít a nebo jestli bude muset spojenec formaci obejít (čímž si může výrazně prodloužit cestu k aktuálnímu cíli).
- **Vhodnost vzniku formace** záškodníků lze odhadovat velmi složitě. Je potřeba promyslet kritéria jako např. aktuální pozice záškodníků, zdali agenti stihnout formaci vytvořit dříve než nepřítel zmizí z jejich zorného pole, a další. Je také nutné odhadovat, zdali formace vůbec stojí na vytvoření podle toho, jací nepřátelští agenti jsou právě zpozorováni.

Kapitola 6

Testování a porovnání funkčnosti

Kapitola se věnuje zejména testování implementovaného řešení a jeho zhodnocení, jak dobře vyhovuje požadovanému multiagentnímu systému a s jakou úspěšností dokáže řešit problém definovaný soutěží *Multi-Agent Programming Contest*. V rámci hodnocení kvality řešení je podrobněji rozebráno hlavně jak realizovaná implementace odpovídá systému z návrhu v kapitole 3.

Dále jsou zde zmíněny také některé nedostatky, které mají značný vliv na úspěšnost vyhotoveného řešení. Jsou podrobněji rozebrány schopnosti agentů, jejich dovednost komunikace, jak dobře jsou schopni plnit požadované úkony a jaký to má vliv na výslednou úspěšnost implementovaného systému.

V kapitole je uveden i stručný popis týmů, které se v roce 2020 soutěže zúčastnili. Následně bylo provedeno srovnání úspěšnosti vlastní implementace s řešeními jednotlivých týmů.

6.1 Vlastní testování

V případě potřeby je uveden i návod, jak implementované řešení zprovoznit na vlastním zařízení. Jsou uvedeny i důležité náležitosti, které musejí být splněny pro úspěšné spuštění. Většina informací je převzata z řešení týmu *MLFC* [5].

Uvedený návod byl použit ke zprovoznění implementovaného řešení a následnému testování na vlastním zařízení s operačním systémem *Windows 10* a architekturou procesoru 64 bitů. Veškeré testování a srovnávání probíhalo ve vývojovém prostředí *Eclipse*.

6.1.1 Návod na zprovoznění

Jak již bylo zmíněno v kapitole 5, jelikož je vlastní implementace postavena na řešení týmu *MLFC*, lze ke zprovoznění postupovat dle jejich návodu na spuštění projektu, který je uveden v repozitáři na GitHubu [5]. Na zařízení, kde má být projekt spuštěn, musí být nejprve nainstalován *Java JDK* verze 13 a nebo vyšší. Spouštění probíhá ve vývojovém prostředí *Eclipse*, takže je potřeba jej mít také nainstalováno. Doporučená verze je *Eclipse IDE for Java EE Developers*.

Po instalaci vývojového prostředí *Eclipse* je potřeba přidat do něj ještě *JaCaMo plugin* dle návodu uvedeného na stránce `http://jacamo.sourceforge.net/eclipseplugin/tutorial/`. Je nutné postupovat pouze po krok 10, jelikož další kroky se věnují již vytváření nového projektu. Ve zmíněných 10 krocích návod podrobně popisuje postup krok po

kroku, jak plugin do vývojového prostředí úspěšně nainstalovat. Pro větší přehlednost a porozumění je návod doprovázen i obrázky přímo z vývojového prostředí *Eclipse*.

Po instalaci pluginu je potřeba získat řešení týmu *MLFC*. Lze využít možnost importování projektu přímo z GitHubu přes URL. V horní liště vývojového prostředí Eclipse lze zvolit možnosti:
`File -> Import -> Git -> Projects from Git -> Clone URI`
a následně vložit URL řešení týmu https://github.com/autonomy-and-verification-uol/mapc2020-lfc.git kde se pak na konci dialogu objeví možnost importování již existujícího projektu.

Nyní je potřeba nahradit realizaci řešení týmu *MLFC* vlastní implementací. Stačí přesunout zdrojové soubory této práce do složky `src` ve staženém řešení týmu *MLFC*. Kromě přidání vlastních souborů dojde také k nahrazení organizačního souboru, kde jsou uvedeny využité role agentů a hlavně k nahrazení souboru `src/agt/agent.asl`, který slouží jako základní soubor pro chování agenta. Tím dojde k nahrazení implementace chování agentů týmu *MLFC* za vlastní implementaci této práce.

Tým *MLFC* ve svém řešení využil plánovač. Návod na jeho instalaci je také zahrnut v jejich řešení na GitHubu. Vlastní implementace této práce však využívá pouze kostru projektu a hlavně již vytvořený komunikační protokol pro komunikaci mezi systémem JaCaMo a jádrem řízení simulace. Není tedy potřeba zabývat se plánovačem, který se ve vlastní implementaci této práce nepoužívá. Část návodu, kde tým *MLFC* popsal instalaci plánovače, lze tedy přeskočit.

Nyní už by měl být projekt zcela provozuschopný. Spuštění ukázkového scénáře lze provést výběrem souboru `test/lfc/agentcontest2020/ScenarioRunSample.java` ve vývojovém prostředí Eclipse a volbou spuštění jako JUnit test. Otevře se informativní okno ze strany klienta, kde budou všechny zprávy a informace o připojených agentech. Dále se v Eclipse konzoli budou vypisovat zprávy a stav serveru, kde pak stačí stisknout klávesu `Enter` pro spuštění simulace. Připojením na localhost 127.0.0.1:8000 lze poté sledovat průběh scénáře.

6.1.2 Zhodnocení funkčnosti implementovaného řešení

Soustavným testování implementovaného řešení byla zjišťována úspěšnost agentů ve vykonávání náležitých aktivit v prostředí. V reakci na výsledky testování bylo chování agentů opakovaně upravováno za účelem zvýšení efektivnosti odpovídajících aktivit. Ovšem i nyní má jejich chování k dokonalosti daleko. I přes to je implementované řešení dostatečně funkční, aby bylo schopné obstát v soutěži a úspěšně řešit související problémy.

Implementované řešení vyhovuje požadavkům [1] ke kvalifikaci do soutěže *Multi-Agent Programming Contest*, jak byly stanoveny pro rok 2020. Jelikož ale požadavky byly velmi mírné, že by je mohl splňovat i velmi primitivní systém, tak úspěšnost v kvalifikaci není příliš směrodatná pro zhodnocení agentního systému. Bylo potřeba realizovat agentní systém, který:

- Splní alespoň jeden úkol v každém ze dvou kvalifikačních kol. Jedno kvalifikační kolo trvalo 300 kroků, takže agentní systém musel odevzdat alespoň jeden úkol do 300 kroků od začátku simulace.
- Vykoná alespoň 70% akcí. V každém kroku systém požaduje od každého agenta akci, kterou bude chtít agent v tomto kroce vykonat. Pokud agent nevybere žádnou akci do určitého počtu sekund (pro kvalifikační kola byly nastaveny 4 sekundy), pochopí to systém tak, že agent neprovedl v daném kroce žádnou akci.

Testováním bylo ověřeno, že agenti dokážou zvládat potřebné aktivity s dostatečně uspokojivou úspěšností. Již minimálně úspěch v kvalifikaci dokazuje, že jsou agenti schopni plnit úkoly. Samotné splnění úkolu se skládá z mnoha dalších aktivit, které je potřeba provádět. Lze tedy vyvodit, že když agenti zvládnou plnit úkoly, musejí se také umět:

- spolupracovat na tvorbě sestavy
- nacházet potřebné dávkovače a cílové oblasti
- získávat kostky a skládat je do požadovaných formací
- najít úkolovou tabuli, přijmout úkol a následně jej i odevzdat

Agenti se v průběhu testování při pohybu občas zasekli v okolí skupin překážek. Avšak vždy byli schopni po různě dlouhé době nalézt alternativní cestu k aktuálnímu cíli a pokračovat tak v pohybu a plnění jejich aktivit.

Samozřejmě testování odhalilo také značné nedostatky implementovaného systému. Největším blokátorem pro účinnější plnění úkolů se stal problém shromažďování agentů v cílové oblasti, blíže zmíněný v sekci 5.2.1. Jelikož v implementovaném řešení pomocníci připojují kostky ke stavitelům až v cílové oblasti, občas dojde ke shromáždění tolika stavitelů okolo jedné cílové oblasti, že pomocníci již nemají dostatek místa k dokončení sestavy, čímž se zamezí splnění přijatých úkolů.

Dalším markantním nedostatkem systému je neschopnost efektivně používat akce čištění. Jednak při pohybu v rámci odstraňování menších překážek, ale zejména jako útočné chování sabotéra pro zasahování nepřátelských agentů a jejich kostek.

S aktuální verzí implementovaného řešení bylo provedeno celkem 10 testovacích běhů za účelem zhodnocení efektivnosti realizace agentního systému. Testování probíhalo v ukázkovém scénáři, o němž hovoří také sekce 6.1.1. Výsledné skóre na konci simulací bohužel v průměru nepřesáhlo 10 bodů. Ve čtyřech simulacích nastal právě zmíněný problém, kdy si agenti navzájem překáželi v cílových oblastech a nebyli schopni odevzdat jediný úkol. Během ostatních testovacích běhů však již byli agenti schopni plnit úkoly, avšak nebyli s jejich odevzdáváním dostatečně rychlí, aby měli nárok na plnohodnotnou odměnu. S časem se totiž odměna za úspěšné splnění úkolu zmenšuje. Tedy ve zmíněných 6 případech byl agentní systém schopen dosáhnout skóre průměrně 14 bodů, což je velmi chabý výsledek ve srovnání s dalšími účastníky soutěže.

6.2 Účastníci soutěže pro rok 2020

V roce 2020 se do soutěže *Multi-Agent Programming Contest* přihlásilo celkem 7 týmů. Jeden z nich však svou registraci brzy stáhl a další tým neprošel kvalifikací. Celkem tedy zbylo níže uvedených 5 týmů, které se spolu v soutěži vzájemně utkávaly. Úspěšně se kvalifikovaly týmy:

- **JaCaMo Builders** s řešením na platformě *JaCaMo*[1]
- **MLFC** s řešením na platformě *JaCaMo*[2]
- **FIT BUT** s řešením na platformě *Java*[3]
- **GOAL-DTU** s řešením na platformě *GOAL*[4]
- **LTI-USP** s řešením na platformě *Jason*[5]

[1] http://jacamo.sourceforge.net/
[2] http://jacamo.sourceforge.net/
[3] https://www.java.com/
[4] https://goalapl.atlassian.net/wiki/
[5] http://jason.sourceforge.net/wp/

Monday

Match	Sim 1	Sim 2	Sim 3	Score
FIT BUT vs. JaCaMo Builders	**38** : 0	**113** : 8	**98** : 0	9:0
GOAL-DTU vs. LTI-USP	**65** : 16	**100** : 14	0 : **38**	6:3
FIT BUT vs. MLFC	0 : **48**	**53** : 44	**90** : 18	6:3
JaCaMo Builders vs. LTI-USP	**8** : 0	10 : **20**	10 : **18**	3:6
GOAL-DTU vs. MLFC	**187** : 26	0 : 0	**105** : 48	7:1

Tuesday

Match	Sim 1	Sim 2	Sim 3	Score
FIT BUT vs. GOAL-DTU	62 : **181**	**104** : 0	**282** : 0	6:3
GOAL-DTU vs. JaCaMo Builders	**353** : 10	**235** : 12	0 : **8**	6:3
FIT BUT vs. LTI-USP	**185** : 16	**148** : 10	**314** : 0	9:0
JaCaMo Builders vs. MLFC	10 : **60**	10 : **26**	10 : **148**	0:9
LTI-USP vs. MLFC	12 : **160**	38 : **195**	8 : **235**	0:9

Obrázek 6.1: Tabulky naplánovaných zápasů *Multi-Agent Programming Contest* v roce 2021. Ve sloupcích označených jako *Sim 1/2/3* jsou uvedeny počty bodů, které týmy během simulace byli schopni získat za odevzdávání úkolů. V posledním sloupci jsou pak poměry bodů přepočteny na celkové skóre týmu. Zdroj [1]

Zápasy jednotlivých týmů proti sobě byly rozplánovány do dvou dnů. Výsledky jednotlivých klání lze vidět na obrázku 6.1, včetně přepočtu nasbíraných bodů za plnění úkolů v každém zápase na poměrné celkové skóre. Po součtu všech získaných poměrů pro každý tým dopadlo výsledné umístění týmů v soutěži takto:

- **1. místo:** 30 bodů – tým *FIT BUT*
- **2. místo:** 22 bodů – týmy *GOAL-DTU* a *MLFC*
- **4. místo:** 9 bodů – tým *LTI-USP*
- **5. místo:** 6 bodů – tým *JaCaMo Builders*

6.3 Porovnání s účastníky

Již z nízkého počtu bodů za splněné úkoly, které se zjistilo testováním realizovaného řešení této práce, v porovnání s objemy bodů ostatních týmů ukázaného na obrázku 6.1, lze vyvodit, že implementované řešení na tom z hlediska úspěšnosti není nejlépe. Téměř v každém zápase alespoň jeden z týmů získal znatelně větší množství bodů, než jsou schopni ziskat agenti v aktuálním stavu realizovaného řešení této práce.

Implementované řešení je možné srovnávat s ostatními účastníky soutěže minimálně z hlediska splnění kvalifikačních kritérií pro vstup do soutěže, ale i z dalších ohledů. Nejen, že se agentni v prostředí dokáží pohybovat relativně rychle, ale také si během pohybu dokáží zapamatovat významné objekty (tabule úkolů, dávkovače, cílové oblasti) a dokáží dokonce jejich pozice sdílet se všemi pozičně synchronizovanými agenty. Pro srovnání v realizovaném řešení zmíněnou funkci o zapamatování a sdílení významný objektů provádí každý agent kdežto u týmu *MLFC* tuto funkci zastává pouze jedinný agent, a to kartograf. Stejně jako ostatní kvalifikované týmy, je i implementovaný agentní systém schopen tvořit sestavy z kostek a odevzdávat úkoly. Tato funkcionalita je bohužel omezována neschopností přátelských agentů se vhodně seřazovat a neblokovat ostatní v cílových oblastech.

Kapitola 7

Závěr

V kapitole 2 byla popsána pravidla soutěže *Multi-Agent Programming Contest* tak, aby podle nich mohl být proveden návrh agentního systému, který bude schopen se soutěže účastnit. Akce, vjemy a další sounáležitosti scénáře byly popsány pevně a podrobně aby zcela vyhověly návrhu akcí a vjemů, které tím pádem mohly být přeskočeny. Informace byly čerpány přímo z pravidel scénáře tak, jak jej popisuje soutěž.

Byly uvedeny všechny diagramy vytvořené při návrhu nesoucí informační hodnotu, což přidalo práci další rozměr. Jedná se o krásné doplnění textu práce. Čtenář se nemusí věnovat pouze čtení textu, ale má možnost získat užitečné informace i z přehledných diagramů.

Cílem práce bylo seznámit se s pravidly soutěže, následně navrhnout a implementovat agentní systém na platformě *JaCaMo*, který bude úspěšně řešit problém specifikovaný soutěží, a nakonec porovnat a zhodnotit implementované řešení oproti dalším týmům, které se soutěže účastnily. Celý návrh postupoval po jednotlivých fázích dle definovaného postupu metodologií *Prometheus* až na vynechání některých částí z důvodu triviality. Návrh je založen na pravidlech soutěže a je vyhovující k implementaci agentního systému nejen na platformě *JaCaMo*, ale i na jiných jako např. *Jason*, *JADE*, aj. Řešení bylo implementováno dle zadání v systému *JaCaMo*, který je v textu také stručně popsán včetně náhledu na jeho pozitiva a negativa. Nakonec bylo provedeno zhodnocení a porovnání funkčnosti implementovaného řešení s týmy, které se tento rok soutěže zúčastnily.

Testování realizovaného řešení odhalilo významné nedostatky, způsobené zejména nekompletní implementací systému. Z časových důvodů se nepodařilo dokončit a vylepšit některé části implementace, o kterých hovoří sekce 5.2.1 a které byly následně přidány k plánům a návrhům na zlepšení a další vývoj práce.

I přes odhalené nedostatky je však realizovaný agentní systém schopen úspěšně řešit scénáře soutěže *Multi-Agent Programming Contest*. Systém zcela vyhovuje kladeným kvalifikačním požadavkům do soutěže. Navíc souhrnné chování agentů vede k úspěšnému plnění úkolů scénáře, takže lze prohlásit zadané cíle práce za splněné.

Literatura

[1] *Multi-Agent Programming Contest* [online]. Clausthal University of Technology [cit. 2021-01-12]. Dostupné z: https://multiagentcontest.org/.

[2] Boissier, O., Bordini, R. H., Hübner, J. a Ricci, A. *Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo*. 1. vyd. The MIT Press, 2020. ISBN 9780262044578.

[3] Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A. a Santi, A. *JaCaMo Project* [online]. [cit. 2021-05-10]. Dostupné z: http://jacamo.sourceforge.net/.

[4] Bordini, R. H., Hübner, J. F. a Wooldridge, M. *Programming Multi-Agent Systems in AgentSpeak using Jason*. 1. vyd. Wiley-Interscience, 2007. ISBN 978-0-470-02900-8.

[5] Cardoso, R. C., Ferrando, A. a Papacchini, F. *MLFC Team Readme* [online]. GitHub, Inc., 2020 [cit. 2021-05-12]. Dostupné z: https://github.com/autonomy-and-verification-uol/mapc2020-lfc.

[6] Fowler, M. *Distilované UML*. 1. vyd. Grada Publishing, a.s., 2009. ISBN 978-80-271-2846-4.

[7] Hessler, A. *MIAC: Methodology for Intelligent Agents Componentware*. Berlin, 2013. Doctoral Thesis. Technische Universität Berlin, Fakultät IV - Elektrotechnik und Informatik. Dostupné z: http://dx.doi.org/10.14279/depositonce-3492.

[8] Jomi, H. F., Boissier, O., Kitio, R. a Ricci, A. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*. 2010, s. 369–400. DOI: 10.1007/s10458-009-9084-y.

[9] Padgham, L. a Winikoff, M. *Developing Intelligent Agent Systems: A Practical Guide*. 1. vyd. John Wiley & Sons, 2004. ISBN 0-470-86120-7.

[10] Rao, A. S. a Georgeff, M. P. Modeling Rational Agents within a BDI-Architecture. In: Allen, J., Fikes, R. a Sandewall, E., ed. *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991, s. 473–484. Dostupné z: http://jmvidal.cse.sc.edu/library/rao91a.pdf.

[11] Ricci, A., Piunti, M., Viroli, M. a Omicini, A. Environment Programming in CArtAgO. In:. 2009, sv. 2, s. 259–288. ISBN 978-0-387-89298-6.

[12] Ricci, A., Santi, A. a Piunti, M. Action and Perception in Agent Programming Languages: From Exogenous to Endogenous Environments. In: Collier, R., Dix, J. a Novák, P., ed. *Programming Multi-Agent Systems*. Springer Berlin Heidelberg, 2012, s. 119–138. ISBN 978-3-642-28939-2.

[13] Wooldridge, M. *An Introduction to MultiAgent Systems*. 2. vyd. John Wiley & Sons, 2009. ISBN 978-0-470-51946-2.

[14] Zhu, H. a Zhou, M. Role-Based Multi-Agent Systems. *Personalized Information Retrieval and Access: Concepts, Methods and Practices*. 2008. DOI: 10.4018/978-1-59904-510-8.ch012.