



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**  
BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**  
FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**  
DEPARTMENT OF INTELLIGENT SYSTEMS

## **REALIZACE TESTOVACÍCH ÚLOH PRO MULTIA- GENTNÍ SYSTÉM**

IMPLEMENTATION OF TEST TASKS FOR MULTIAGENT SYSTEM

**BAKALÁŘSKÁ PRÁCE**  
BACHELOR'S THESIS

**AUTOR PRÁCE**  
AUTHOR

**DAVID KANÓCZ**

**VEDOUCÍ PRÁCE**  
SUPERVISOR

**doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**

**BRNO 2025**

## Zadání bakalářské práce



Ústav: Ústav inteligenčních systémů (UIT)

163439

Student: **Kanócz David**

Program: Informační technologie

Název: **Realizace testovacích úloh pro multiagentní systém**

Kategorie: Modelování a simulace

Akademický rok: 2024/25

### Zadání:

1. Seznamte se s architekturou multiagentního systému FRAg, zejména s jeho rozhraním pro interakci agentů s prostředím.
2. Vyhledejte a případně navrhněte testovací úlohy, které se v současnosti využívají pro vyhodnocování výkonu multiagentního systému. Pro zvolené úlohy definujte multiagentní prostředí včetně jeho rozhraní k systému FRAg.
3. Implementujte tyto úlohy jako prostředí multiagentního systému tak, aby je bylo možné vykonávat systémem FRAg. Otestujte jejich správné fungování, tedy platné reakce prostředí na zadáné akce.
4. Navrhněte a naprogramujte alespoň jeden multiagentní systém pro každou ze zvolených úloh a zhodnotěte schopnost takových systémů úlohy plnit.
5. Vytvořte plakát formátu A1, na kterém budou názorně demonstrovány dosažené výsledky.

### Literatura:

- Michael Wooldridge. 2009. An Introduction to MultiAgent Systems (2nd. ed.). Wiley Publishing.
- Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. 2007. Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology). John Wiley & Sons, Inc., Hoboken, NJ, USA.

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: Kočí Radek, Ing., Ph.D.

Datum zadání: 1.11.2024

Termín pro odevzdání: 14.5.2025

Datum schválení: 11.5.2025

## Abstrakt

Cílem této práce je realizace testovacích úloh pro multiagentní systém FRAg (**Flexibly Reasoning BDI Agent**). Výstupem práce jsou tři úlohy, které jsou podrobně popsány, klasifikovány a jejich výběr je odůvodněn na základě chování, které mají testovat. Na realizaci testovacích úloh byl vyvinut simulační systém prostředí v jazyce Python, pojmenovaný PES (**Python Environment System**). PES komunikuje s jazykem Prolog na straně systému FRAg, skrze knihovnu janus SWI-Prologu. Technická dokumentace vyvinutého systému popisuje komunikační stránku mezi systémy PES a FRAg, princip činnosti PES, odlišné možnosti konfigurace prostředí (mřížka, struktura) a konečně návod ke zprovoznění a vytváření nových prostředí. Na závěr je krátký popis limitací systému PES a jeho plánovaná rozšíření. Výsledný PES může být použit pro tvorbu testovacích prostředí typu mřížka a struktura, poskytující uživatelům systému FRAg novou funkcionality, kterou je rychlejší tvorba prostředí v programátorům často bližším jazyce Python a to s vestavěnou možností animace.

## Abstract

The aim of the thesis is realization of test tasks for multiagent environment FRAg (**Flexibly Reasoning BDI Agent**). The result of this effort are four test tasks, each described in detail, classified, and their choice is explained, based on different behaviour they are intended to test. For the purpose of implementing aforementioned tasks, an environment simulation system named PES (**Python Environment System**) was developed. PES communicates with FRAg via janus library of SWI-Prolog. Technical documentation of the developed system outlines the communication between the two systems PES and FRAg, how PES functions, different options of environment configuration (grid and structure), and finally setup instructions and a guide on creating new environments. To conclude this, there is a short list describing the limitations of PES and planned extensions. PES in its current form can be used to create testing environment of either the grid or structure type, providing users of FRAg with new functionality, which is a quicker way of creating new test environments in a Python language typically preferred by programmers, all with a built-in animation support.

## Klíčová slova

Multiagentní systém, Racionální agent, Agentní rozhraní, Testovací prostředí, FRAg, Prolog, Python

## Keywords

Multiagent system, Rational agent, Agent interface, Test environment, FRAg, Prolog, Python

## Citace

KANÓCZ, David. *Realizace testovacích úloh pro multiagentní systém*. Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

# Realizace testovacích úloh pro multiagentní systém

## Prohlášení

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

.....  
David Kanócz  
13. května 2025

## Poděkování

Touto cestou si dovoluji vyjádřit srdečné poděkování panu docentu Ing. Františku Zbořilovi, Ph.D., za jeho trpělivost, cenné rady a odborné vedení, které mi věnoval při zpracování této práce. Mé upřímné díky dále patří mé rodině a přítelkyni. Bez laskavé finanční podpory a povzbuzení rodičů bych své vysokoškolské studium nemohl zahájit a bez neochvějné psychické opory přítelkyně by bylo jen stěží možné jej úspěšně dokončit.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>4</b>
<b>2</b>	<b>Agent a agentní systémy</b>	<b>6</b>
2.1	Vlastnosti agentů . . . . .	6
2.2	Motivace pro používání agentů . . . . .	7
2.3	Druhy/Architektury agentů . . . . .	8
2.4	BDI architektura agenta . . . . .	14
<b>3</b>	<b>Multiagentní systémy</b>	<b>17</b>
3.1	Multiagentní systém - definice . . . . .	17
3.2	Příklady užití multiagentních systémů . . . . .	18
3.3	Příklady softwarových nástrojů pro vytváření a simulaci multiagentních systémů . . . . .	20
3.4	Způsoby evaluace výkonnosti multiagentních systémů . . . . .	20
<b>4</b>	<b>Návrh a parametry vhodných prostředí pro evaluaci výkonu agentů</b>	<b>23</b>
4.1	Zodpovědnosti prostředí . . . . .	23
4.2	Možné parametry, odlišnosti a specifika prostředí . . . . .	24
4.3	Komunikační standard EIS používaný systémem FRAg . . . . .	25
<b>5</b>	<b>Seznámení se systémem spravujícím prostředí (PES), návod na tvorbu nových prostředí v systému</b>	<b>28</b>
5.1	Prolog, knihovna janus a komunikace mezi FRAgem a PES . . . . .	28
5.2	Detailní popis systému PES . . . . .	30
<b>6</b>	<b>Podrobný popis implementovaných prostředí</b>	<b>41</b>
6.1	Výrobní linka . . . . .	41
6.2	Výtah Miconic 10 – pouze agentní systém . . . . .	44
6.3	Prostředí stalker . . . . .	47
6.4	Obchod – není implementovaný skrze PES . . . . .	50
<b>7</b>	<b>Závěr</b>	<b>52</b>
<b>Literatura</b>		<b>53</b>
<b>A</b>	<b>Obsah SD karty</b>	<b>57</b>

# Seznam obrázků

2.1	Přehled a vývoj programovacích paradigmat rozšířený agenty. . . . .	7
2.2	Jednoduchý reflexní agent – Russell, Norvig[20]. . . . .	9
2.3	Agent založený na modelu – Russell, Norvig[20]. . . . .	10
2.4	Agent zaměřený na cíl– Russell, Norvig[20]. . . . .	11
2.5	Užitkově zaměřený agent – Russell, Norvig[20]. . . . .	12
2.6	Učící se agent – Russell, Norvig[20]. . . . .	13
2.7	BDI agent architektura.[17] . . . . .	15
2.8	Plánovač – Wooldridge.[31] . . . . .	16
4.1	Komponenty meta-modelu.[30] . . . . .	25
4.2	Propojení agentů a entit prostředí.[30] . . . . .	26
4.3	Ukázková struktura stromu v jazyce IIL.[30] . . . . .	27
5.1	Posloupnost volání predikátu py_call při posílání akce entitě. . . . .	30
5.2	Struktura systému PES, jeho moduly a s čím FRAg komunikuje. . . . .	31
5.3	Tvarování pomocí neaktivních buněk. . . . .	33
5.4	Tvarování bludiště pomocí zdí. . . . .	33
5.5	Dvě identická pod-prostředí se stejnými parametry a rozdílnými agenty. . .	34
5.6	Dvě pod-prostředí s odlišnou strukturou mezi kterými agent může libovolně přeskakovat skrze koridory, které jsou označené fialovou barvou. . . . .	34
5.7	Heatmapa představující nebezpečí jednotlivých buněk prostředí. . . . .	35
5.8	Ukázka dohledu entity částečně obehnáné zdí. . . . .	35
5.9	Ukázka trasy neprocházející rohy. . . . .	36
5.10	Ukázka trasy procházející rohy. Obě oranžové buňky jsou na trase. . . . .	36
5.11	Ukázka vykreslení heatmapy nebezpečí. . . . .	37
5.12	Ukázka strukturálního vykreslení. . . . .	37
5.13	Ukázka vykreslení dohledového režimu. . . . .	37
5.14	Ukázka směrového propojení jednotlivých buněk. . . . .	38
6.1	Směrové propojení jednotlivých místností. . . . .	43
6.2	Porovnání agregace pasažérů podle jejich destinace v systému Miconic 10 oproti běžnému výtahu.[2] . . . . .	45
6.3	Trasy výtahu Miconic 10 oproti běžným výtahům.[2] . . . . .	45
6.4	Vizualizace prostředí miconic společně s výtahem (červená tečka) a pasažéry (celočíselné hodnoty). Zelené tečky jsou viditelné buňky, ve kterých není ani výtah, ani čekající pasažér. V zájmu úspory místa je obrázek otočen o 90° .	46

6.5	Snímek obrazovky ze hry S.T.A.L.K.E.R, konkrétně oblasti <i>Army Warehouses</i> . Jedna z několika desítek samostatných oblastí, která je součástí mapy. Ideální kandidát pro rozšíření funkčnosti díky bohaté přítomnosti frakcí a frekvenci mutantů. . . . .	49
6.6	Struktura prostředí společně s heatmapou nebezpečí. . . . .	50

# Kapitola 1

## Úvod

V posledních letech se v oblasti umělé inteligence a distribuovaných výpočetních systémů čím dál více prosazují agentní a multiagentní přístupy. Agent je samostatná jednotka, která dokáže vnímat své okolí, rozhodovat se a plnit předem stanovené úkoly. Když jich v prostředí spolupracuje či soupeří více, hovoříme o multiagentních systémech (MAS). Abychom mohli zhodnotit, jak dobře se „myšlení“ těchto agentů vyrovnaná skutečným situacím, je nepostradatelné pečlivě navrhnut testovací úlohy i samotné prostředí. Hledá se přitom zejména, jak efektivně agenti dokážou plánovat, přizpůsobovat se změnám a spolupracovat bez centrálního systému řízení.

Cílem této bakalářské práce je navrhnut a implementovat sadu testovacích úloh pro multiagentní systém FRAg (Flexibly Reasoning BDI Agent), který je vyvíjen na Ústavu inteligenčních systémů Fakulty informačních technologií VUT v Brně. V rámci práce byl vytvořen systém pro simulaci prostředí PES (Python Environment System), který umožňuje rychlé a pohodlné vytváření nových testovacích scénářů v jazyce Python, přičemž zachovává kompatibilitu se systémem FRAg. Používá k tomu knihovnu janus pro komunikaci mezi jazyky Prolog a Python. Výstupem práce jsou tři reprezentativní testovací úlohy – výrobní linka, výtah inspirovaný výtahem Miconic 10 a prostředí stalker – které jsou detailně klasifikovány a popsány z hlediska chování, které mají ověřovat.

Druhá kapitola této práce se věnuje základním pojmem v oblasti agentů a agentních systémů, včetně motivace k jejich nasazení, jaké výhody přináší a jakou formu nebezpečí mohou představovat. Popsány jsou jednotlivé architektury agentů a motivace pro jejich nasazení v reálných aplikacích, od jednoduchých, reflexních agentů, až po učící se a BDI agenty. Kapitola třetí pak krátce rozšiřuje agentní paradigma na multiagentní systémy. Objasňuje širší spektrum jejich využitelnosti v porovnání s pouze agentními systémy, možnosti organizace agentů v rámci MAS a poskytuje krátký přehled současných programovacích frameworků, platforem pro modelování a simulaci a knihoven pro multiagentní posilované učení (MARL – Multi-Agent Reinforcement Learning). V neposlední řadě popisuje způsob evaluace MAS, které používá tato bakalářská práce. Ve čtvrté kapitole je popsána problematika návrhu testovacích prostředí pro vyhodnocení výkonu agentů, včetně parametrů, zodpovědností prostředí a standardu EIS (Environment Interface Standard), který využívá systém FRAg a zajišťuje interoperabilitu mezi agentní platformou a prostředím. Pátá kapitola podrobně představuje systém PES – jeho komunikaci se systémem FRAg, možnosti konfigurace prostředí (mrázka, struktura), a obsahuje technickou dokumentaci systému včetně návodu na tvorbu a parametrizaci nových prostředí. Následující šestá kapitola pak popisuje jednotlivé implementované testovací úlohy, analyzuje jejich cíle, detaily realizace a způsob vyhodnocení

chování agentů. Práci uzavírá shrnutí dosažených výsledků, zhodnocení omezení navrženého řešení a návrh možných směrů dalšího rozvoje systému PES.

Výsledné prostředí PES rozšiřuje funkcionality systému FRAg o rychlou a intuitivní tvorbu testovacích scénářů v Pythonu, s možností simulace a animace chování agentů. Tím přispívá k usnadnění experimentů s různými parametry BDI agentů jako brzké versus pozdní vázání proměnných a k jejich důkladnějšímu testování v různorodých úlohách, což je nezbytný krok k posouvání a vylepšování systému FRAg.

# Kapitola 2

## Agent a agentní systémy

Co je to agent? Definice agenta jako entity může být složitá, zvláště kvůli tomu, co do definice zahrnout, a co ne. Různá odvětví používají jiné definice slova agent, například v oblasti umělé inteligence pojmenování agent většinou zahrnuje více vlastností. Proto se základní definice agenta nachází až o pár řádků později, poté co se nejdříve definují jeho nutné vlastnosti a požadavky, aby se této entitě dalo říkat agent. Tímto způsobem se tato sekce snaží o rychle pochopitelnou, ale dostatečnou definici, která poslouží k pochopení, jak obecně agent funguje a která bude rozšířená a doplněná podle potřeby později. Nejdůležitějšími vlastnostmi, co dělá agenta agentem, je tato čtverice vlastností, která je převzatá z [32], ačkoliv dílčí části definice pochází z jiných zdrojů.

### 2.1 Vlastnosti agentů

Na začátek upozornění. Agent nemusí splňovat všechny následující vlastnosti, aby to byl agent. Některé typy agentů například nekomunikují s ostatními prvky prostředí (agenty/lidmi). Obecně platí, že pouze autonomie a reaktivita jsou vlastnosti, které najdeme v každém typu agenta.

- **Autonomie:** Autonomie je vlastnost agenta, spočívající v samostatnosti rozhodování o svém chování v rámci daného systému, bez implicitní závislosti na jakýchkoliv jiných prvcích tohoto systému.[33]
- **Schopnost socializace:** Agenti mají schopnost komunikovat s ostatními agenty, případně s lidmi, aby dosáhli svých cílů.
- **Reaktivita:** Agenti vnímají prostředí ve kterém působí, které může nabývat různého charakteru (fyzický/reálný svět, internet, uživatel skrze uživatelské grafické rozhraní, simulované prostředí, ...). Zavčasu reagují na změny prostředí a přizpůsobují se jím.
- **Pro-aktivita:** Agenti nereagují na změnu prostředí, ale jsou proaktivní jednotkou v systému, **kde přebírají iniciativu**. Mají své cíle a vykazují chování, které vede k dosažení jejich cílů.

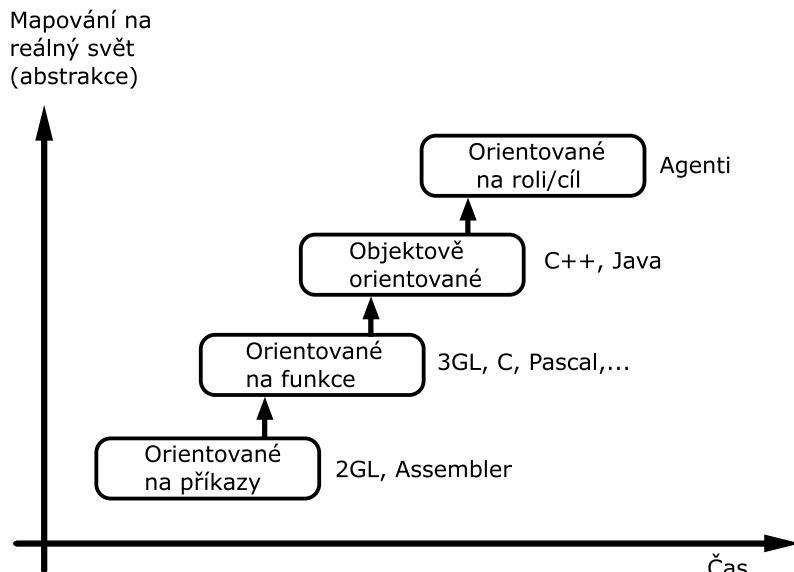
V této chvíli se může zdát, že objekt v kontextu objektově orientovaného programování může sám o sobě být agent. Vskutku, pokud dotyčný objekt obohatíme o vlastnosti, které normálně nemá, jako je autonomie, můžeme ho brát jako agenta. Objekt by ale bylo potřebné rozšířit například tak, že při zavolání metody objektu se nevždy metoda vyplní.

Objekt danou metodu provede pouze za podmínky, že ji vyhodnotí jako výhodnou pro splnění jeho plánu, jinak ji neprovede. Rozšíření objektů na agenty se však obecně nepoužívá, protože objekt a agent jsou z podstaty rozdílné entity. U agenta jako takového autonomii vyžadujeme, kdežto objekt často bereme pouze jako entitu, která obsahuje nějaká svá data a metody. Ve zkratce by se dalo říci, že objekt je pasivní entita, která nemyslí, nemá záměry či cíle a nemá svou vlastní vůli. Agent je pravý opak. Autonomie/světové je pro agenta podstatou. [31]

Agentem můžeme tedy nazývat nějakou entitu, splňující tyto vlastnosti. Z předchozí definice pro-aktivity – a z čistě logického pohledu – je patrné, že agent musí být umístěn v nějakém prostředí, které musí být schopen vnímat a nějakým způsobem s ním interagovat. Detailní rozebrání problematiky prostředí se nachází ve čtvrté kapitole.

### Agent jako programovací paradigma

Jak už bylo řečeno dva odstavce výše, koncepce agenta a objektu od sebe nejsou tak vzdálené. Můžeme agenta vnímat jako pokračování evoluce programovacího paradigmatu na obrázku 2.1, převzatého z [6].



Obrázek 2.1: Přehled a vývoj programovacích paradigmat rozšířený agenty.

### Definice pojmu agent

Agent je nějaký systém, který se nachází v nějakém prostředí, které vnímá, a je v tomto prostředí schopen nezávisle provádět akce svými efektory, kterými splní své cíle. [31][35]

## 2.2 Motivace pro používání agentů

V dnešní době informatika ovlivnila takřka všechna odvětví, ať už profesní, zábavní, nebo z osobního života. Ovšem existují oblasti, kde je lidský operátor stále vyžadován, protože problém je tak složitý, že prostý program založený na algoritmech nestačí. Například autopilot

nemůže být pouhou knihovnou příkazů. Občas bychom totiž jako uživatelé či programátoři chtěli, aby program vykazoval nějaké chytřejší chování, jako jsou touhy, cíle nebo schopnost se přizpůsobit konkrétnímu prostředí, ve kterém je nasazen. Vzhledem ke schopnosti agentních a multiagentních systémů řešit náročnější problémy, je důležité mít na paměti i potenciální problémy související s jeho selháním.

### Jak může vypadat selhání agenta

Jako příklad může být uvedena smrtelná nehoda testovacího autonomního vozidla firmy Uber. Řidič měl kontrolovat tento autonomní systém, agenta, ale kvůli nepozornosti nezasáhl včas. Agent selhal, nejprve klasifikoval chodce jako neznámý objekt, poté jako vozidlo a následně jako jízdní kolo, přičemž měl různá očekávání ohledně jeho budoucí trajektorie. 1,3 vteřiny před nárazem vyhodnotil, že se musí provést nouzové brzdění, aby se zabránilo kolizi. Tato funkce však byla vypnuta, aby se předešlo nevyzpytatelnému chování při testování. [16]

### Úspěšně nasazené agentní systémy

- **Chytré vysavače:** Vysavač Roomba od společnosti iRobot je velice populárním autonomním agentem. Jako své prostředí má místnost, kterou vysívá. V závislosti na modelu se jedná buď o jednoduchý reflexní agent, který reaguje na podněty z prostředí jako 'narazil jsem do zdi, změním směr', nebo u novějších modelů, například i7 [8], už využívají i učení, aby svoje vysávání zefektivnili.[26]
- **Perseverance rover:** Systém AutoNav roveru Perseverance od agentury NASA, který zajišťuje autonomní řízení byl v prvním roce na Marsu použit 88% z celkových 17.7 ujetých kilometrů. „AutoNav získal několik rekordů planetárních roverů, včetně nejdelší ujeté vzdálenosti bez lidského zásahu (699.9m)"[27]

Poznámka: Tato kapitola pojednává o agentech. O využití multiagentních systémů, které mají širší využití, pojednává následující kapitola.

## 2.3 Druhy/Architektury agentů

Jako u všech technologií, i agenty můžeme rozdělit podle mnoha vlastností.[25] Můžeme je rozdělovat například:

- Podle schopnosti učení:
  - učící se,
  - neučící se (fixní).
- Podle fyzikální podoby:
  - softwarový agent,
  - fyzický robot.
- Podle účelu:
  - organizace křížovatek,

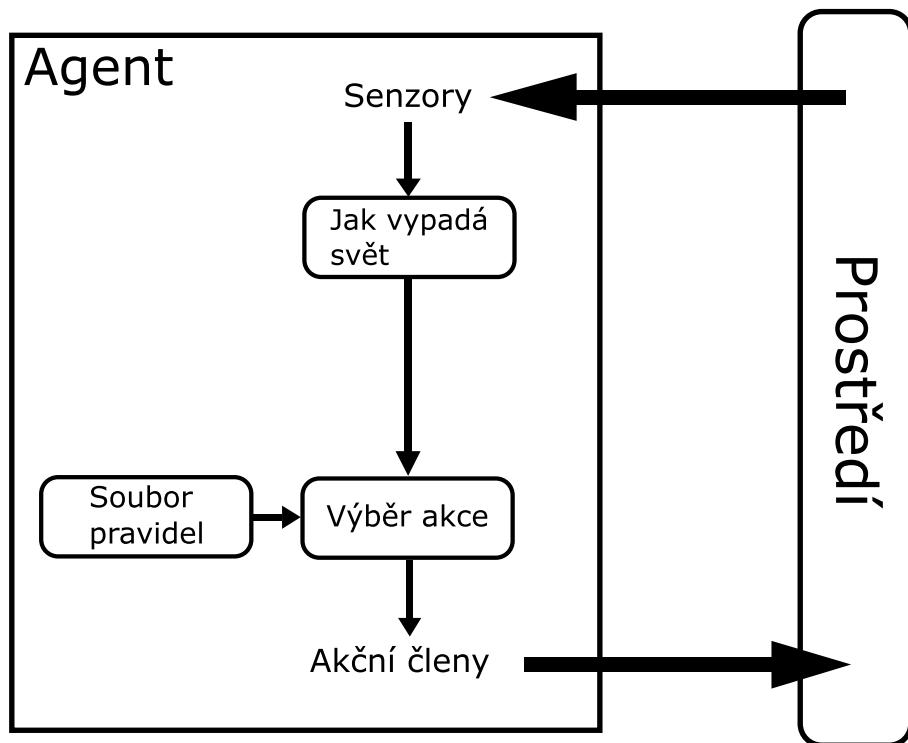
- autopilot,
- obchodní agent, ...

Rozdělení, kterému tato práce věnuje větší pozornost, je však rozdělení podle míry vnímané inteligence a schopnosti – zvolená architektura implikuje schopnosti a limity agenta.

## Rozdělení typů agentů podle architektury

### Jednoduchý reflexní agent

Asi nejjednodušší, ale i tak využitelná a často využívaná architektura agentního systému. Reflexní agent si neudržuje žádné vnitřní stavy, vlastní představu o prostředí, či historii. Místo toho reaguje (proto se tomuto typu agenta říká i *reaktivní*) čistě na podněty prostředí, kterými jsou výstupy senzorů. Z toho vyplývá, že nemá jakýkoliv prostředek pro tvorbu plánů.<sup>[7]</sup>



Obrázek 2.2: Jednoduchý reflexní agent – Russell, Norvig[20].

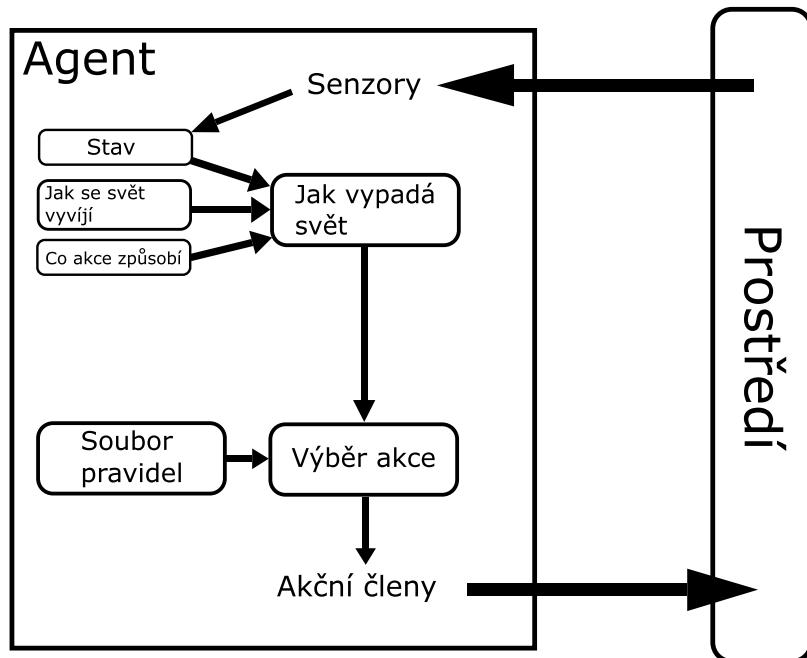
Jak je vidět z obrázku 2.2, reflexní agent pouze reaguje na momentální vstup senzorů. Příkladem asi nejvíce využívané aplikace této architektury jsou jednoduché intelligentní vysavače jako vysavače Roomba od společnosti iRobot, kterých se prodalo přes 50 milionů jednotek během 22 let od vydání prvního vysavače Roomba.<sup>[9]</sup> (Ačkoliv do técho 50 milionů spadají i ty modely, které by byly klasifikovány jako učící se, a tedy nebyly v kategorii pouhého reflexního agenta.)

## Agent založený na modelu

V nějakých situacích nám pouhé současné sledování světa k vybrání vhodné akce nestačí. K příkladu si můžeme uvést situaci, kdy máme auto s autonomním řízením – autopilotem. Vysvětlování příkladů na autopilotu/taxi v této kapitole pochází taktéž z knihy *Artificial Intelligence - A Modern Approach*[20]. Pokud autopilot nemá senzory pokrývající celou svoji zornou plochu, a tedy nemusí vždy ostatní auta vidět, musí si o ostatních autech a dopravních prostředcích na vozovce udržovat vlastní vnitřní stav. Aby se tento vnitřní stav dal aktualizovat, potřebujeme dva druhy znalostí. **První znalost** – jak se svět chová nezávisle na agentovi. V příkladu autopilota by to znamenalo vědět takové věci, jako:

- Auto které k nám zkracuje vzdálenost nás možná bude chtít předjet.
- Pokud nějaké auto zapne ukazatel směru (blinkr), s největší pravděpodobností změní směr jízdy.

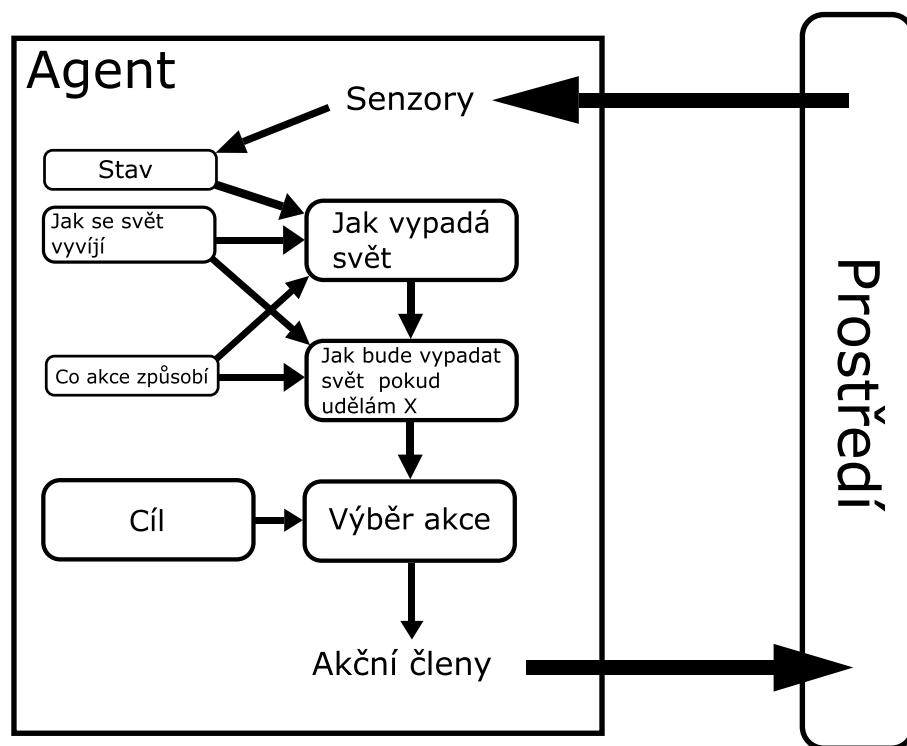
Je to tedy nějaká znalost o tom, jak se okolní svět chová. **Druhá znalost**, co agent potřebuje, je znalost, jak jeho akce ovlivní okolní svět. Například sešlápnutí brzdrového pedálu má za následek snížení rychlosti. Tyto dvě znalosti pro agenta tvoří **vnitřní stavový model**. Proto se tomuto typu agenta říká *založený na modelu*. Tento vnitřní stavový model však zřídka odpovídá současnému a úplnému stavu prostředí. Jednak to nemusí být vůbec možné u částečně pozorovatelných prostředí (senzory autopilota například nevidí skrz překážky). A za druhé, někdy kvůli množství dat, či nedokonalosti senzorů musí agent pracovat pouze s abstrakcí daného prostředí a tedy nějaké důležité informace nemá přístupné (senzory nemusí detektovat ledovku). I přes nedokonalý stav znalostí však musí nějakou akci vybrat. Struktura agenta je vidět na obrázku 2.3.



Obrázek 2.3: Agent založený na modelu – Russell, Norvig[20].

## Agent zaměřený na cíl

I dokonalý model světa by však agentům nemusel stačit. V případě autopilota by agent nedokázal pasažéry dopravit tam, kam oni chtejí, kdyby neznal cíl cesty. V armádním pojetí sebevražednému dronu, který by měl za úkol zasáhnout tank při ztrátě signálu, často způsobenou rušičkami, nestačí vědět, kde co ve světě je nebo jak pomocí ovládání rotorů pilotovat dron. I tu agent potřebuje cíl, tank. V případě, kdy žádný nedetekuje, tak je jeho možným cílem RTB (return to base – vrácení se na základnu). V tomto případě je tedy agent rozšířen i o cíl, jak je k vidění v jeho struktuře 2.4.

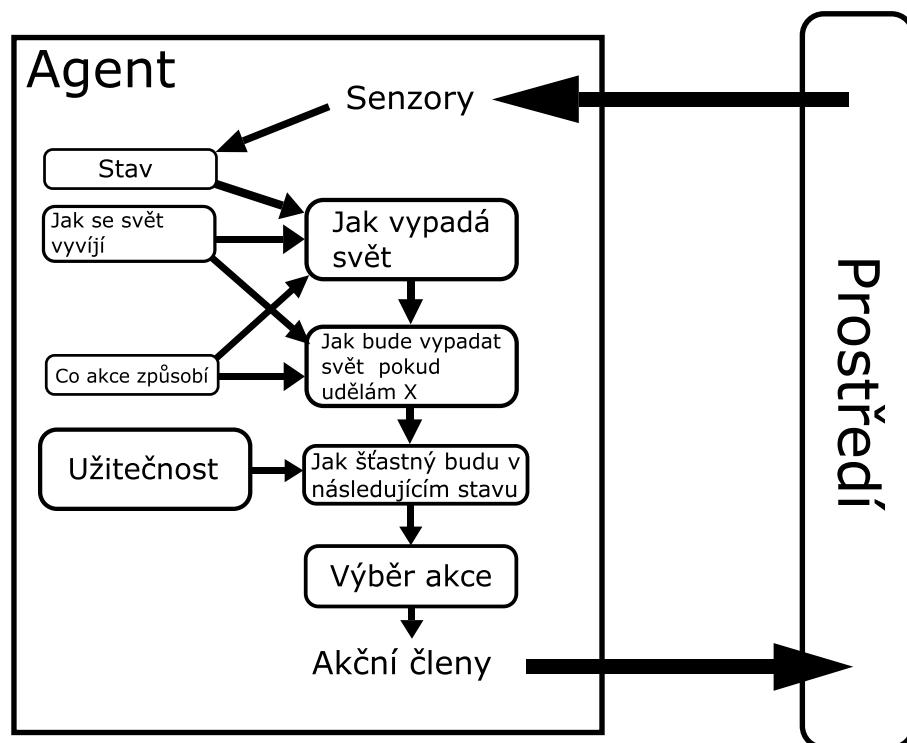


Obrázek 2.4: Agent zaměřený na cíl– Russell, Norvig[20].

## Užitkově zaměřený agent

Agenti zaměření na cíl mají jednu nevýhodu. Dosáhnutí cíle je pro ně binární, tedy buď ho dosáhnou, nebo ne. V duchu příkladů z minulých typů agentů se bude vysvětlovat i tento rozdíl na autopilotu, v tomto případě rozšířeném o vícero typů vozidel dle jejich funkce (civilní auto a policejní auto). Pokud by oba typy vozidel využívala jako autopilota agenta zaměřeného na cíl, obě vozidla by zvolila tu samou trasu. To je nežádoucí, protože ne každý dopravní prostředek chce jet tím samým způsobem. Autopilot civilního automobilu se musí striktně držet zákonů a předpisů jako přikázaný směr jízdy či maximální povolená rychlosť. Autopilot policejního či hasičského automobilu si však nemůže dovolit ztrácat čas, když jede k zásahu. Tedy ačkoliv by měly tyto automobily stejný cíl cesty, způsob dosažení této cesty by měl být jiný. Tomuto říkáme užitek. Užitek je tedy nějaké vnitřní výkonnostní ohodnocení – **užitková funkce**[20] – která říká, jak ohodnocené jsou různé plány akcí, které vedou k dosáhnutí toho samého výsledku. Tato vlastnost dává agentovi mnohem větší škálu přizpůsobivosti, kdy například u již zmíněného osobního automobilu si pasažér může vybrat, zda preferuje cestu, která bude šetrnější k životnímu prostředí, bude rychlejší, bezpečnější, nebo ve které nebude stát v koloně.

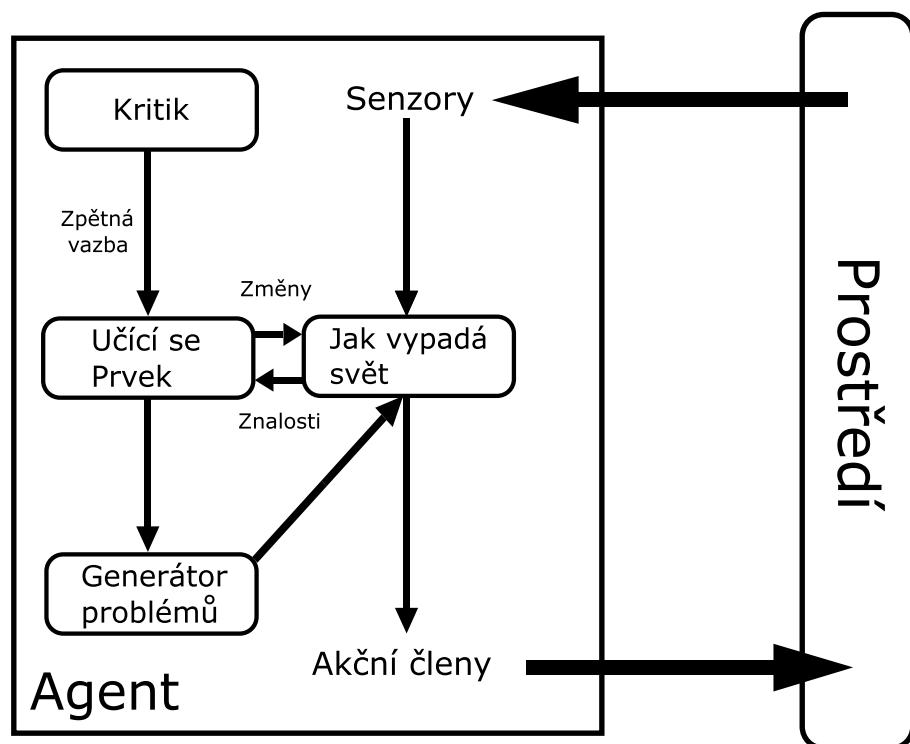
U agentů se zaměřením na cíl byl problém, pokud existovali cíle, které se vzájemně vylučovaly. Oproti tomuto typu mají užitkově zaměření agenti 2.5 právě onu užitkovou funkci, která je de facto návod na to, jaký kompromis by agent měl udělat.[20]



Obrázek 2.5: Užitkově zaměřený agent – Russell, Norvig[20].

## Učící se agent

Schopnost učení se poskytuje agentovi další, významný výkonnostní skok. Agenti se často nacházejí v prostředích, u kterých nemají možnost nabýt všechny informace. Například učící se vysávající roboti bývají vypouštěni do prostředí, které svými senzory typicky nedokáží mít v jeden moment celkový přehled o místnosti, protože se musí vyhýbat překážkám. Tyto překážky zabírají tomu, aby roboti – agenti – měli v jeden moment přehled o celé místnosti a tedy mohli vymyslet ideální plán. Schopnost učení u těchto vysavačů umožňuje vědět informace jako "pokud udělám akci A, v minulosti jsem za to dostal vyšší ohodnocení, než za akci B. Tedy provedu akci A." To u vysávacího robota může znamenat, že místnost bude vysávat kratší dobu.[20]



Obrázek 2.6: Učící se agent – Russell, Norvig[20].

Z obrázku 2.6 lze vidět, že jeho struktura je zcela jiná. Struktura je nyní rozdělena na 4 části, z nichž se jedna nazývá výkonnostní prvek. Výkonnostní prvek je původní celý agent, viz například obrázek 2.5. Tento prvek má své senzory, akční prvky a vybírání akce. Schopnost agenta učit se tedy není rozšířením jeho struktury jako takové, ale přidáním zcela nových prvků. **Nové prvky agenta:**

- **Kritik:** Sleduje a hodnotí, jak si agent počíná v porovnání s předem daným, pevně stanoveným, výkonnostním standardem. Na základě kritikové zpětné vazby se učícímu se prvku vrací informace o následcích jím provedených akcí.

- **Generátor problémů/úkolů:** Navrhuje experimentální akce, které agentovi mohou přinést nové zkušenosti, tedy potenciál lepšího plánování akcí v budoucnu, na úkor krátkodobě neoptimálních akcí.
- **Učící se prvek:** Na základě zpětné vazby od kritika upravuje rozhodovací algoritmus výkonnostního prvku. Díky tému úpravám v rozhodování se agent učí a postupem času dosahuje lepších výsledků.

## 2.4 BDI architektura agenta

BDI agent, neboli belief - desire - intention (přesvědčení - touha - záměr), je typ agentů zaměřených na cíl. K vysvětlení pojmu přesvědčení, touha a záměr je užitečné znát více specifické vlastnosti často přisuzované agentům v oblasti umělé inteligence. Podle práce Wooldridge a Jennings,[32] se pro popis agenta často používají jiné, spíše mentalisticky popisované vlastnosti.

### Další vlastnosti agentů

Sem samozřejmě spadají i BDI vlastnosti, které jsou však popsány o dvě stránky níže ve větším detailu.

- **Pravdivost:** Agent nebude vědomě šířit nepravé informace.
- **Shovívavost:** Agent nemá konfliktní cíle. Agent se tedy bude vždy snažit dosáhnout jemu daného cíle.
- **Racionalita:** Agent bude racionální. Bude konat tak, aby dosáhl svých cílů a nebude konat tak, že způsobí, že jeho cíle se stanou nedosáhnutelné.

Agent samozřejmě není vševedomý, bude tedy konat podle jeho vnitřního přesvědčení. K lepší ilustraci, člověk je z ekonomického hlediska racionální agent. Dělá rozhodnutí, o kterých je přesvědčen, že jsou správná, i přesto, že ve skutečnosti to správná rozhodnutí nejsou. Například lidé, kteří naletí podvodům **jsou stále racionální**. Dělali to, co je pro ně nejvhodnější akce, na základě jejich vnitřního přesvědčení. Problém tedy nebyl ve výběru akce, tu zvolili racionálně správnou. Problém byl v jejich modelu světa, který byl mylný. Stejně tak, i agent, který bude mít nesprávnou představu o světě, nebude dělat optimální akce.

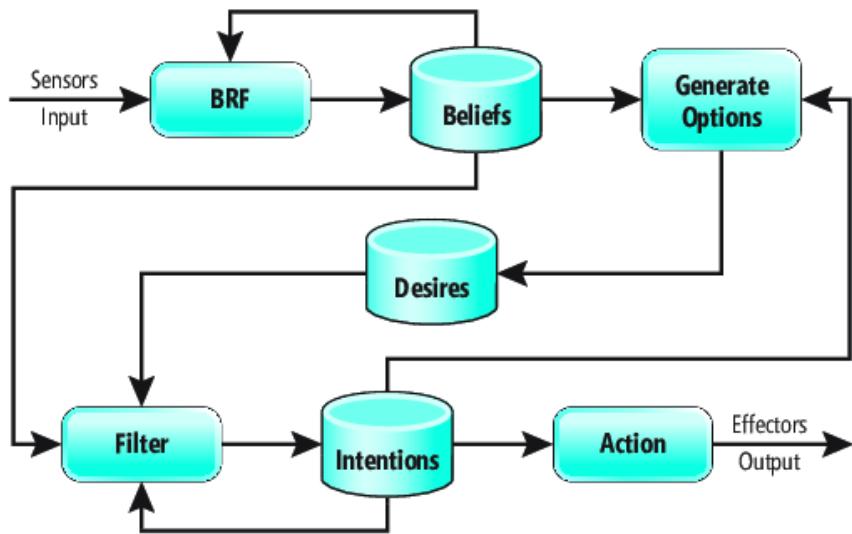
### Sumarizace vlastností BDI agenta

K summarizaci je potřeba ještě jeden termín, kterým je deliberativní agent. Podle disertační práce docenta Zbořila[33, s. 8] je definice následující.

*Na rozdíl od reaktivního agenta má deliberativní (rozvážný) agent schopnost plánovat postup svých akcí vedoucích k dosažení zvolených nebo zadaných záměrů/cílů. To znamená, že agent musí mít schopnost různých výpočtů, což bude později v textu chápáno jako druh vnitřní činnosti agenta. K dosažení svých záměrů pak agent ovlivňuje okolní prostředí tak, aby získal nějakou výhodu. Toto jednání je další z často uváděných vlastností agentů a nazývá se proaktivita.*

BDI agentem je tedy **deliberativní** (na rozdíl od reaktivních agentů si udržuje symbolickou představu o světě), **racionální**, **na cíl zaměřený** agent.

## Vysvětlení struktury BDI agenta



Obrázek 2.7: BDI agent architektura.[17]

BDI agent na obrázku 2.7 má odlišnou strukturu, než agenti před ním. Rozrostl se o několik částí, které budou teď popsány.[34][32]

### Beliefs - přesvědčení, víra

Množina znalostí, co o světě "ví", přesněji řečeno, v co věří, o stavu světa. Na základě této v některých případech neúplné a neaktuální báze znalostí se rozhoduje a vytváří plány.

### Desires - přání, touhy

Množina potenciálních přání agenta, neboli stavů světa, kterých by chtěl agent docílit. Těchto přání může být veliké množství a mohou se i navzájem využívat. Přání se zpravidla rozdělují na krátkodobá a dlouhodobá, přičemž dlouhodobá přání se označují jako cíle - Goals. BDI agent se tedy dá klasifikovat jako speciální typ agenta zaměřeného na cíl.

### Intentions - záměry

[TODO cite JENNINGS, str3] Záměr je přání nebo cíl z množiny Desires, které se agent rozhodl provést. Agent má omezené zdroje, tedy nemůže stále vyhodnocovat svoje mezi sebou soutěžící touhy. V nějakém bodu se musí rozhodnout, o které z těchto cílů bude usilovat. Tím si vytvoří závazek, že vybraný záměr bude dokonán do konce, nebo do bodu, ve kterém se jeho splnění ukáže jako neproveditelné. Na základě záměrů se plánují budoucí akce.

### Plánovač

Jednou z důležitých částí BDI agenta je plánovač. Plánovač je systém, který ze vstupu (ze záměrů), stavu světa a dostupných akcí, jako výstup sestaví plán.[31] Plánovač je k vidění

na obrázku 2.8. Prvním plánovačem byl systém Stanford Research Institute Problem Solver, neboli STRIPS, vyvinut dvěma autory, kterými byli Richard Fikes a Nils Nilsson.



Obrázek 2.8: Plánovač – Wooldridge.[31]

STRIPS si postupně rozebírá hlavní cíl na menší podcíle, které si ukládá na zásobník. Když je něco potřeba (predikát není v databázi), najde se akce, která tento predikát dokáže přidat. Všechny akce mají své podmínky, které je nutné splnit. Kvůli tomu se rekursivně řeší tyto podmínky, dokud nejsou v databázi. Tímto způsobem se vytvoří postup (sekvence operátorů) pro dosažení cíle. Ačkoliv se v dnešní době stejnojmenný algoritmus STRIPS již nevyužívá, jeho formalizmy akcí lze najít v téměř všech implementacích plánovačů.[31]

Na závěr této kapitoly bych rád upozornil, že problematika BDI agentů je rozhodně mnohem širší a komplikovanější. Například BDI logika, a s ní logiky, ze kterých vychází, v této práci popsány nejsou. Pro více informací a detailnější popis bych čtenáře rád odkázal na disertační práci[33] pana docenta Zbořila mladšího, konkrétně na třetí kapitolu disertační práce nazvanou *Nástroje pro návrh racionálních agentů*.

## Kapitola 3

# Multiagentní systémy

Doposud se tato práce zabývala pouze agenty samotnými, případně systémy, kde je umístěn jeden agent. Tato kapitola pojednává o multiagentních systémech (MAS), co multiagentní systémy poskytují za funkčnost oproti systémům s jedním agentem, jak se agenti mohou v prostředí chovat vůči sobě, reálné případy užití multiagentních systémů, prostředky k jejich programování a nakonec i samotný MAS FRAg vyvýjený na VUT.

### 3.1 Multiagentní systém - definice

Definice agenta v kapitole první byla složitá, nicméně to neplatí u definice multiagentního systému. V multiagentním systému musí platit, že v prostředí působí více než jeden agent. Každý agent je entita, která myslí samostatně a plní své vlastní přání, respektive přání jeho uživatele/majitele. Každý agent má nekompletní informace nebo schopnosti pro řešení problému. Není zde centralizovaný systém řízení, data jsou decentralizovaná a výpočty probíhají asynchronně.[10]

**Definice:** Multiagentní systém (MAS) je systém, ve kterém působí alespoň dva agenti, kteří mezi sebou mohou soupeřit nebo spolupracovat.[10]

Co se týče mezi-agentní komunikace, je zde opět spousta možností, jak agenty v prostředí propojit.[22]

- **Propojení každý s každým:** Nejjednodušší typ propojení, kdy každý agent má schopnost komunikace s každým jiným agentem v prostředí.
- **Propojení s vedoucím:** Každý agent komunikuje pouze s jedním agentem, vedoucím, který je společný pro všechny agenty v prostředí.
- **Hierarchické propojení:** Rozšíření propojení s vedoucím. Agenti komunikují se svým lokálním vedoucím, který sám má svého vedoucího. Těchto vedoucích může být několik a zároveň může být také několik hierarchických úrovní. Výsledný systém propojení tedy vypadá jako pyramida.
- **Propojení na míru:** Manuálně definovaná propojení mezi jednotlivými agenty pro specifické aplikace. Možnost vytvořit pouze jednostranné propojení mezi agenty.

## Chování agentů v rámci MAS

V MAS se agenti mohou chovat různě. To, jakým způsobem se chová jeden agent, ovlivňuje akce ostatních agentů, ať už přímo, nebo nepřímo. Proto je možné zaškatulkovat jejich chování vůči sobě do dvou skupin. [33]

- **Kompetitivní/Soupeřící:** Cíle jednoho agenta jsou přímo v rozporu s cíli jiného agenta. Nemohou uspět společně a každý z nich se snaží vyhrát na úkor druhého.
- **Spolupracující:** U spolupracujících agentů – se v extrémním případě – jejich cíle shodují. Tedy akci, kterou vykoná jeden agent, bude ve shodě s agentem druhým. Ve většině případů se však takovéto kompletní krytí cílů nestavá.

**Kooperativní agent** je agent, který v konkrétní situaci zvažuje, zda se vyplatí spolupracovat, nebo soutěžit, a to na základě shody jeho cílů s jiným agentem. Mějme hypotetický příklad dvou agentů, agenta A a agenta B, jejichž množiny cílů mají neprázdný průnik. Cíle v tomto průniku představují takové cíle, ve kterých je výhodné spolupracovat (neboť je chtějí dosáhnout oba agenti). Naopak cíle, které se v průniku nenacházejí, mohou vést ke konfliktu či konkurenci, pokud jsou vzájemně neslučitelné nebo pokud prosazování jednoho z nich poškozuje či omezuje druhého.

## 3.2 Příklady užití multiagentních systémů

Množství aplikací multiagentních systémů (MAS) je širší, než u pouhých systémů s jedním agentem. Pokud bychom měli rozdělit oblasti, ve kterých se MAS dají využít, mohlo by být například následující (a určitě existují i další oblasti využití).

- **Hraní her** Asi nejznámějším využitím je hraní her, které jsou často takzvané zero-sum, česky hra s nulovým součtem. Tento termín z teorie her znamená, že pokud si strana A zlepší svoji pozici o X, straně B se zhorší její pozice přesně o -X. Tedy, pokud se zisk jedné strany a ztráta strany druhé sečtou, výsledný součet vychází přesně nula. Typicky se jedná o hry, jako šachy, poker a jim podobné, kde nejsou *další ambice* těchto hráčů. V realitě tento typ her moc neexistuje, protože i ve válce obě strany pravděpodobně budou chtít zvítězit, aniž by přitom zničily celý svět, nebo si svým výběrem strategie poštvali zbytek světových zemí proti sobě.
- **Optimalizace a organizace** Prvním příkladem je organizace semaforů na křižovatkách ve městech, aby pokud možno město mělo co nejméně zatížené uzly a auta tedy měla nejrychlejší možné cesty. Největší překážkou u dopravy je, že hustota a konfigurace se neustále mění. Problém tedy není až tak doménou optimalizace, ale adaptace. [5] Využitím agentů se dá dosáhnout lepších výsledků, než pouze "chytřím algoritmem." Ze studie Gehersyna bruselské univerzity vykázala nejlepší metoda oproti metodám bez schopnosti adaptace průměrně 30% zrychlení průměrné rychlosti, snížení zastavených aut na polovinu a snížení času čekání zastavených aut na jednu sedminu.

Dalším příkladem je systém, který se využívá již od roku 1995. Systém OASIS[13] (Optimal Aircraft Sequencing using Intelligent Scheduling - Optimální řazení letadel pomocí inteligentního plánování) je systém, který zajišťuje co možná největší využití vzletových a přistávacích dráh. Využívá MAS implementovaný pomocí BDI agentů. Vytváří sekvensi přistání a odletů pro letadla tak, aby bylo jejich zpoždění co nejmenší.

Každé letadlo je v tomto systému agentem, kterému je přidělena runway, ze které má vzlétnout, nebo přistát a v jaký čas. Mezi zajímavé vlastnosti systému OASIS například patří:

- Schopnost reagovat na změny počasí a chování letadel v okamžiku detekce těchto změn.
  - Dokáže měnit plány v případě náhlého požadavku na přistání od neplánovaného letadla.
  - Rychlé přepočítání, aby se přizpůsobil změnám na vzletových a přistávacích plochách.
  - Zbytek vlastností je k dispozici na [13].
- **Krizové řízení** Po proběhlé katastrofě jde o minuty. Složky záchranných sborů se snaží využít všechny prostředky, které mají k dispozici, k záchraně co nejvíce životů a minimalizaci škod. Nasazení multiagentního systému k simulaci může státu poskytnout cenné informace ohledně strategií vedoucích k největšímu užitku – nejvíce zachráněných životů – o potenciálních nedostatečnostech týkajících se množství personálu či vybavení a řadě dalších. Simulace může být detailní, modelující různé stroje či lidi jako agenty, šíření ohně v průběhu času nebo kolabující či přetíženou infrastrukturu. Simulace jsou dlouhou dobu cenným nástrojem pro získávání informací v doménách, kde je experimentování v realitě nemožné nebo nevýhodné, ale takto komplexní systém bez multiagentních systémů simulovat nelze.[11]
  - **Bojové nasazení** V této oblasti se multiagentní systémy vyvíjejí hojně právě v této chvíli. Asi veřejnosti nejznámějšími jsou takzvané swarm (rojové) systémy dronů. Do jaké míry autonomie dokáží tyto roje svoji práci vykonávat, je teď samozřejmě utajované. Ovšem je snadné si představit, že například pokud jeden dron identifikuje zbraň proti dronům, či identifikuje cíl, schopnost poslat tuto informaci skupině, která na to společným způsobem autonomně zareaguje, je ohromným navýšením výkonnosti dronových systémů.[15]

Jako druhý známý příklad by mohl být uveden program amerického letectva NGAD (Next Generation Air Dominance - Nová generace vzdušné nadvlády). Tento program má nahradit stíhací letouny zaměřené na leteckou převahu, kterými teď jsou F-22 Raptor. Generačním skokem stíhacího letounu F-22 byla technologie stealth (obtížná zjistitelnost). Generačním skokem letounu z programu NGAD by měl mimo jiné být právě MAS. V tomto případě se jedná o to, že na piloty jsou kladený stále větší požadavky. Žádného člověka však nelze vycvičit tak, aby jeho tělo vydrželo přetížení 13 G. Tento problém, stejně jako spoustu dalších, jako sdílení detekovaných cílů, vyšší počet nesených střel a leteckých pum, aniž by došlo k navýšení radarového průřezu. To vše je možné díky MAS, který v tomto příkladu umožňuje změnu paradigmatu. Umožňuje mít skupinu letounů, kterou řídí jeden pilotovaný stíhací letoun, kolem kterého je skupina bezpilotních letounů, které se pilotovi podřizují a pomáhají mu. [24] Během psaní této práce vyhrála společnost Boeing kontrakt na zhotovení letounu, dnes již známého pod názvem F-47, který z programu NGAD vzešel.

### 3.3 Příklady softwarových nástrojů pro vytváření a simulaci multiagentních systémů

#### Agentní programovací frameworky

- **JADE (Java):** Napsán kompletně v jazyce Java, používající middleware splňující FIPA<sup>1</sup> specifikaci ke zjednodušení vývoje MAS. Možnost distribuce napříč počítači.[12]
- **Jason (Java):** BDI struktura, interpretace vylepšené, rozšířené verze jazyka Agent-Speak(L). Napsán kompletně celý v Javě. Lze snadno distribuovat po síti.[18]
- **SPADE (Python):** Narozdíl od JADE používá místo FIPA komunikační protokol XMPP. Umožňuje jak mezi-agentní komunikace, tak komunikaci mezi agentem a člověkem, takzvané human-in-the-loop (do procesu práce je zapojen člověk).[23]
- **FRAg (Prolog):** Interpretuje jazyk AgentSpeak(L) s možností pozdního vázání promenných. Aktivně vyvíjen fakultou informatiky VUT, momentálně v alfa verzi.[29]

#### Platformy pro agentní modelování a simulaci

Nástroje integrující jazyk, vykonávací jádro i vizualizaci. Umožňují spíše "klikací" tvorbu a běh velkých simulací bez nutnosti psaní nízkoúrovňového kódu.

- **GAMA Platform:** Grafické IDE, využití platformy i lidmi, kteří nejsou počítačový vědci za použití vysokoúrovňového jazyka GAML.[21]
- **MASON:** Navrženo jako základ pro rozsáhlé simulace v jazyce Java. Jednoduchost a rychlosť jsou hlavními přednostmi.[14]

#### Knihovny pro multiagentní posilované učení (MARL)

MARL knihovny poskytují standardizovaná prostředí a API pro trénink samoučících se agentů.

- **Pettingzoo:** Python rozhraní obsahující obecné MARL úlohy. Mnoho referenčních prostředí jako šachy nebo pong, s možností vytváření vlastních prostředí.
- Podobnými pak jsou například **Gymnasium**<sup>2</sup>, dříve OpenAI's Gym library, nebo **BenchMARL**<sup>3</sup>.

### 3.4 Způsoby evaluace výkonnosti multiagentních systémů

Stejně jako různé druhy algoritmů, například řadících, nám binární výsledek řazení – seřazeno, neseřazeno, nestačí. Zajímá nás, jak rychle dokáže řadit, na jakou nátuру dat se hodí, jaké jsou jeho paměťové požadavky, a tak dále. Stejně tak multiagentní systémy mají spoustu metrik, které udávají výkonnost, ať už MAS systému jako je FRAg, nebo schopnost agentů plnit jejich úkoly v prostředí. Metriky, které nás zajímají, závisí na schopnostech a účelech používání MAS.

Metriky se mohou rozdělit do následujících kategorií[19]:

<sup>1</sup><http://www.fipa.org/specifications/>

<sup>2</sup><https://gymnasium.farama.org/>

<sup>3</sup><https://arxiv.org/abs/2312.01472>

- funkční testování,
- behaviorální a scénářové testování,
- Výkonnostní testování,
- testování zaměřené na posilované učení,
- ověřování uvažování a validace rozhodnutí,
- odolnost, nepřátelské strojové učení,
- integrace v reálném čase a externí systémy,
- a další, které jsou k nalezení na<sup>4</sup>.

Pro tuto práci jsou podstatné hlavně kategorie ověřování uvažování a validace rozhodnutí, a behaviorální a scénářové testování.

### **Behaviorální a scénářové testování**

Pro behaviorální testování, neboli testování týkající se chování, je podstatné prozkoumávání nejen pouhé splnění cílů, ale i průběh chování, který ke splnění cílů vedl. Zde se pozoruje, často doslově – (pozorovatel) sleduje chování agentů v systému – a odvozuje z toho efektivitu spolupráce, způsob řešení konfliktů a objevuje případné emergentní strategie, které agentům nebyly explicitně naprogramovány.

K tomu pozorovateli poslouží záznamové výstupy, případně animace, pokud to daný systém umožňuje. U pozorovaného prováděného člověkem je jak testování, tak i výsledná analýza chování kvalitativní. Kvantitativní testování používá místo lidského pozorovatele vyhodnocovací skript, který dokáže vyhodnocovat z rádově více simulačních běhů. Na rozdíl od kvantitativní však sama nedokáže klasifikovat strategii jako *dobrou*, *špatnou* nebo snad *morální*.

### **Ověřování uvažování a validace rozhodnutí**

Halucinace ve světě generativní umělé inteligence způsobují u uživatelů zaslouženou nedůvěru vůči AI. To samé se děje i v případě multiagentního systému, kdy jde o komplexní systém s mnoha agenty. Pokud uživatel nerozumí rozhodnutím a vybraným strategiím agentů, vzbuzuje to nedůvěru v celý MAS.<sup>[4]</sup> V ideálním případě lze všechno agentovo chování logicky vysvětlit, s vědomím odměn za akce, seznamem vjemů a agentovou touhou maximizovat odměnu. Pokud však je nějaké chování nejasné, může pomoci například agentně-lidský dialog, který nabízí některé implementace BDI agentů, například SimpleBDI.<sup>[1]</sup>

Příklad dialogu v implementaci SimpleBDI.<sup>[1]</sup>

---

<sup>4</sup><https://www.linkedin.com/pulse/comprehensive-methodologies-metrics-testing-ai-agents-ramachandran-mkxne>

human: Why Not move2 at 17  
robot: Why move2 at 17  
human: Selected at\_waypoint,goal\_move\_to\_location,safe\_terrain, ->  
do(move2),-goal\_move\_to\_location,-at\_waypoint,+at\_location,+goal\_take\_sample, at 16  
robot: Why select at\_waypoint,goal\_move\_to\_location,safe\_terrain, ->  
do(move2),-goal\_move\_to\_location,-at\_waypoint,+at\_location,+goal\_take\_sample, at 16  
human: +at\_waypoint at time 14 and it remained so until at least 16  
robot: I agree +at\_waypoint between 14 and 16  
human: +goal\_move\_to\_location at time 11 and it remained so until at least 16  
robot: I agree +goal\_move\_to\_location between 11 and 16  
human: +safe\_terrain at time 6 and it remained so until at least 16  
robot: -safe\_terrain at time 15 and it remained so until at least 16  
human: Why -safe\_terrain at 15  
robot: I perceived -safe\_terrain at 15

Možnost dotazování se a komunikace s agentem je způsob ověření, případně k vysvětlení důvodů jeho chování.

## Kapitola 4

# Návrh a parametry vhodných prostředí pro evaluaci výkonu agentů

Prostředí jsou nedílnou součástí agentního či multiagentního systému (MAS). Zároveň je to jedna z jeho nejdůležitějších částí, protože právě podle prostředí, ve kterém se bude systém nacházet, se pro daného agenta budou jinak definovat cíle, plány a akce. Ať už se jedná o reálná prostředí, kde mají agenti fyzickou reprezentaci a interagují s hmatatelnými objekty, nebo o simulované prostředí, které umožňuje testovat a ladit chování systémů v kontrolovaných podmínkách, volba vhodného prostředí je nezbytná pro správnou evaluaci výkonu agentů. Evaluace agentů je úzce spjata s prostředím, ve kterém se agenti nacházejí – například je zřejmé, že metriky pro chytré křižovatky budou zcela odlišné od metrik pro autonomní drony.

Prostředí si lze představit jako kontejner, ve kterém jsou agenti situováni. Zároveň prostředí mají zodpovědnosti.

### 4.1 Zodpovědnosti prostředí

Výčet zodpovědností prostředí[30]:

- **Strukturování:** Prostředí vytváří sdílený rámec, který určuje, jak bude celý systém organizován.
- **Správa zdrojů a služeb:** Prostředí funguje jako kontejner, který umožňuje přístup k různým zdrojům (objekty s definovaným stavem) a službám (reaktivním entitám, jež zapouzdřují určitou funkcionality). Zajišťuje řízení a kontrolu přístupu k těmto zdrojům a službám.
- **Udržování dynamiky prostředí:** Kromě aktivit samotných agentů může prostředí iniciovat a spravovat vlastní procesy, jako je například agregace či difuze digitálních signálů, evoluce lokálních podmínek. Těmi mohou například být teplota, nebo obecněji odstraňování zastaralých dat.
- **Umožnění komunikace:** Prostředí definuje konkrétní kanály a metody, prostřednicitvím kterých mohou agenti komunikovat. Nejčastější formou je přímé posílání zpráv

mezi agenty. V některých případech však mohou agenti využívat nepřímou komunikaci, například vytvářením a konzumováním komunikačních objektů v prostředí.

- **Řízení interakcí v systému:** Prostředí může stanovovat pravidla a zákony, které omezují nebo usměrňují přístup k určitým zdrojům či službám, případně určují, jaké důsledky mají akce agentů.
- **Zajištění pozorovatelnosti:** Na rozdíl od agentů musí být prostředí vždy pozorovatelné. Agenti potřebují přístup k informacím o zdrojích a službách, a často i možnost sledovat činnost ostatních agentů.

## 4.2 Možné parametry, odlišnosti a specifika prostředí

Vlastnosti	Pátrání a záchrana	Fotbal	Šachy
Počet agentů	100 a více	11 v každém týmu	-
Agenti v týmu	Heterogenní	Homogenní	-
Logistika	Závažný problém	Ne	Ne
Dlouhodobé plánování	Závažný problém	Méně zdůrazněné	Zapojeno
Emergentní spolupráce	Závažný problém	Ne	Ne
Hostilita	Prostředí	Protější tým	Protihráč
Čas na rozhodnutí	Sekundy – Minuty	Milisekundy	-
Dostupnost informací	Velmi špatná	Přiměřeně dobrá	Kompletní
Reprezentace	Hybridní	Nesymbolické	Symbolické
Řízení	Distribuované / Polo-Centrální	Distribuované	Centrální

Tabulka 4.1: Vlastnosti pátrání a záchrany, fotbalu a šachu

Další parametry rozdělení prostředí[34], které v tabulce 4.1 nejsou uvedeny, jsou rozdělení dle:

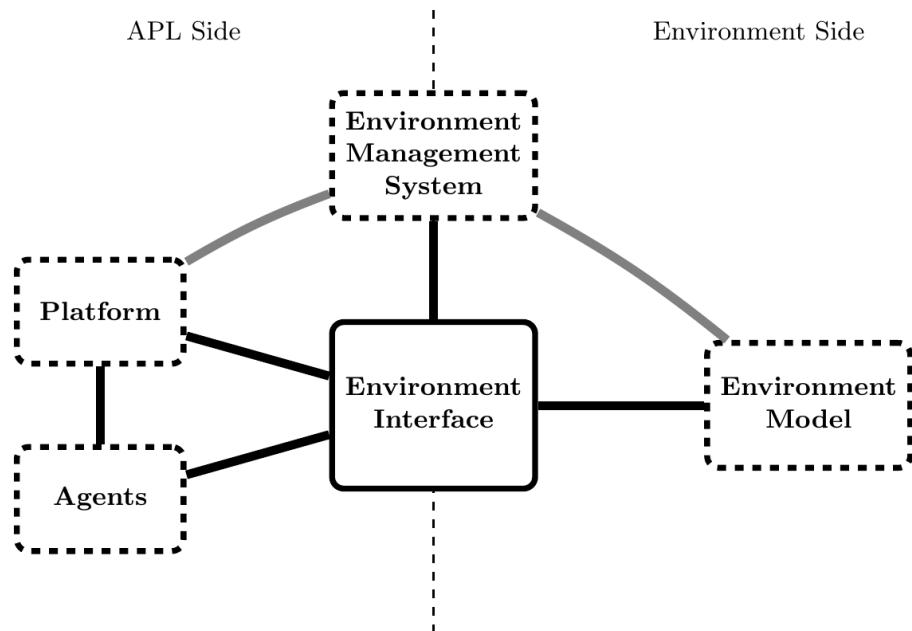
- **Určitosti:**
  - **stochastické**,
  - **deterministické**.
- **Spojitosti:**
  - **diskrétní**,
  - **spojité**.
- **Časové závislosti:**
  - **statické**,
  - **dynamické**.
- **Návaznosti:**
  - **episodické**,
  - **sekvenční**.

Z tabulky 4.1 [11], si lze všimnout, že prostředí se od sebe mohou lišit ve spoustě věcí. Ačkoliv jsou všechny tyto parametry důležité, tak jedny z nejdůležitějších jsou dostupnost informací a čas na rozhodnutí. Pokud každý agent má pouze znalost svého okolí a nikoliv celého prostředí, může být vhodné řízení centrální – více informací na jednom místě pro lepší plánování akcí. Avšak záleží na typu prostředí a akcí, které od agentů v prostředí chceme. Na některé události musí agenti reagovat co nejrychleji a centrální řízení by mohlo být příliš pomalé.

Zároveň každé prostředí může mít své vnitřní parametry, které se mohou nastavovat a měnit, ať už počáteční, nebo jejich vývoj. V příkladu s obchodem by to mohly být parametry jako kolik zákazníků přichází, zdražování produktů v čase a podobné.

### 4.3 Komunikační standard EIS používaný systémem FRAg

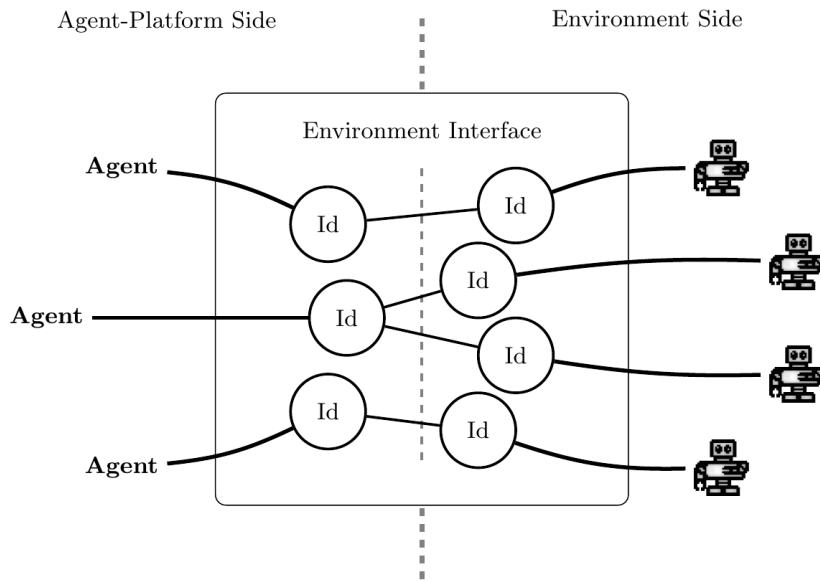
V technologických odvětvích razících cestu vpřed není výjimkou pozorovat, že každý autor implementuje něco po svém. To má řadu výhod, jako rychlejší prozkoumání možností implementací, zjištění, jaké cesty vývoje jsou slepé, a tak dále. Opačnou stranou mince je však častá nekompatibilita. Standard **EIS** (Environment Interface Standard) je tímto případem. Jak autoři standardu EIS sami uvádějí, mnoho prostředí je navrženo a je kompatibilní jen s jednou vývojovou platformou (2APL, Jason, Jadex, ...)[30]. Byla tedy vyvinuta prostředí založená na stejném principu – jako jsou různé 2D světy, ve kterých agenti pracují. Mezi sebou však nebyly kompatibilní. Cílem EIS tedy bylo navrhnout standard mezi **APLs** (agent programming language) a prostředím samotným. EIS tím usnadňuje vývoj, sdílení a znovupoužití prostředí napříč více agentovými platformami, testování výkonnosti agentních systémů ve stejném prostředí a usnadní tvorbu rozsáhlých multiagentních systémů. EIS je přitom pouhý prostředník, popisující způsob komunikace mezi agentní platformou a prostředím, jak je vidět na obrázku 4.1.



Obrázek 4.1: Komponenty meta-modelu.[30]

## Princip standardu EIS

Standard zavádí dva termíny, se kterými bude tato práce dále pracovat, nebude-li uvedeno jinak. Jsou jimi **entita** a **kontrolovatelná entita**. Entitou se rozumí "něco" v prostředí. Může to být auto, dům, předmět, cokoliv. Kontrolovatelná entita je pak taková entita, která **má možnost být agentem kontrolovaná**. Kontrolovatelné entity zajišťují spojení mezi agentem – který operuje v nějaké agentní platformě, a prostředím samotným skrze identifikátor, výčtem dostupných akcí a schopností vnímání – senzory.[30] Propojení kontrolovatelných entit a agentů je k vidění na obrázku 4.2.



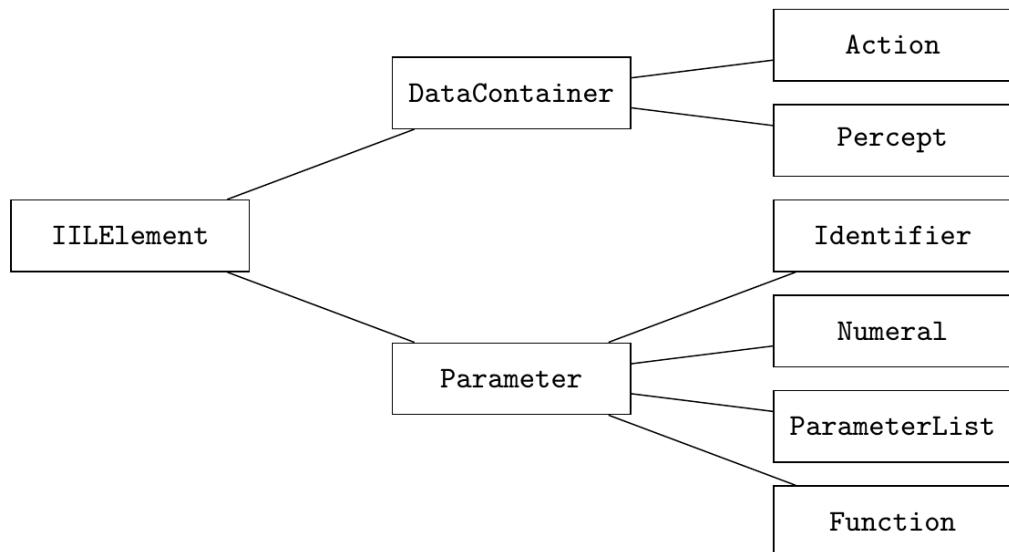
Obrázek 4.2: Propojení agentů a entit prostředí.[30]

## Jazyk III

**IIL (Interface Intermediate Language)** je jazyk, který EIS používá pro výměnu dat mezi agenty a prostředím. Reprezentuje akce i vjemy/percepty stromově strukturovaným způsobem a tato struktura může být vypsaná prologovským nebo XML stylem, k vidění na obrázku 4.3. Díky stejnemu rozhraní mohou agentní platformy zpracovat tytéž informace a napojit se na libovolné prostředí splňující standard EIS. Prvky vyskytující se v jazyce EIS jsou:

- *DataContainer* je buď *Action*, nebo *Percept*.
- *Action* se skládá z:
  - řetězce **name**, který udává název akce,
  - uspořádané kolekce **parameters** obsahující instance *Parameter* (tedy parametry akce),
  - celočíselného **timeStamp**, který zachycuje čas vzniku objektu akce.
- *Percept* se skládá z:

- řetězce `name`, který udává název perceptu,
  - uspořádané kolekce `parameters` obsahující instance *Parameter* (tedy parametry perceptu),
  - celočíselného `timeStamp`, který zachycuje čas vzniku objektu perceptu.
- *Parameter* může být typu *Numerical*, *Identifier*, *ParameterList* nebo *Function*.
  - *Numerical* zapouzdřuje numerickou (číselnou) hodnotu.
  - *Identifier* zapouzdřuje řetězcovou hodnotu.



Obrázek 4.3: Ukázková struktura stromu v jazyce IIL.[30]

## Kapitola 5

# Seznámení se systémem spravujícím prostředí (PES), návod na tvorbu nových prostředí v systému

V zájmu pokračujícího vývoje systému FRAg byl naprogramován systém, který umožňuje vývojáři naprogramovat prostředí nová, na míru, bez nutnosti zdlouhavého programování celého prostředí a všech jeho podpůrných funkcí. Jádro všech prostředí této práce se tedy neodehrává v prostředích napsaných v jazyce Prolog, ale v systému spravujícím prostředí v jazyce Python (dále jen **PES** – Python Environment System). Vyhnout se Prologu ze sta procent v systému, který je na něm postaven, nelze. Následující část proto představí komunikační mezivrstvu mezi systémy FRAg a PES, ve které se nachází veškerý Prolog této práce.

### 5.1 Prolog, knihovna janus a komunikace mezi FRAgem a PES

Komunikaci lze rozlišit do dvou částí. Té první, jednodušší, ve které se posílají z FRAgu akce, které má daný agent vykonat v prostředí, a samotné vytvoření prostředí. Druhé, ve které se z PES posílají vjemy prostředí zpět agentovi do FRAgu. Komunikaci mezi těmito systémy zajišťuje knihovna janus projektu SWI-Prolog, což je jedna z populárních implementací jazyka Prolog.

#### Knihovna janus

Knihovna janus implementuje rozhraní mezi jazyky Python a Prolog. Je to jedna z řady knihoven, které nabízí SWI-Prolog, implementujících rozhraní mezi programovacími jazyky. Existuje řada dalších knihoven, které realizují rozhraní mezi těmito jazyky, ale právě janus je jako jediná vestavěná a v rámci snížení bariér ke zprovoznění byla vybrána. **Verze SWI-Prologu, obsahující knihovnu janus jsou 9.2.X<sup>1</sup>**

---

<sup>1</sup><https://www.swi-prolog.org/versions.md>

K ověření funkčnosti knihovny je možné použít sadu příkazů:

- `swipl`,
- `use_module(library(janus)), janus:py_version()`

Pokud je vše v pořádku, vypíše se verze knihovny janus a jakou verzi vestavěného jazyka Python obsahuje.

Jsou dvě verze knihovny janus, jedna pro Python, která spouští SWI-Prolog, a druhá, která spouští vestavěnou instanci Pythonu z SWI-Prologu – případ této práce. Zodpovědností obou verzí však je datová konverze a obousměrná komunikace. Následující strany popíší využití knihovny k propojení systému, následně ke komunikaci ze strany agenta (FRAgu) k entitě v PES, a nakonec vracení vjemů, které agent, lépe řečeno entita, kterou v prostředí ovládá, obdržela.

## Kroky k propojení systémů FRAg a PES

1. `use_module(library(janus))`: Importování knihovny janus.
2. `prolog_load_context(directory, Dir)`: Získání absolutní cesty z relativní cesty `directory`.
3. `py_add_lib_dir(Dir)`: Načtení adresáře Dir, ve kterém se nachází soubor popisu-jící prostředí.
4. `py_import(module, [])`: Načtení modulu obsahující Python prostředí.

I přesto mohou nastat problémy, většinou jsou tvořeny na straně SWI-Prologu. Pokud nemůže SWI-Prolog (dále jen swipl) knihovnu janus vůbec najít, je pravděpodobné, že verze používaného swipl je neaktuální (verze swipl musí být alespoň 9.2.X). Pokud je janus nainstalovaný, ale při importování modulu janus dojde k chybě, pravděpodobné je buď chybějící Python jako takový, nebo cesta k němu, konkrétně cesta k `python3.dll`. Python se doporučuje instalovat z oficiální stránky<sup>2</sup> projektu Python. Pokud již Python v systému instalovaný je, ale janus stále nefunguje, pravděpodobně `python3.dll` není v cestě. K napravě v operačním systému Windows je možné provést následující příkaz přes PowerShell:

`setx PATH "%PATH%;C:\Users\<username>\AppData\Local\Programs\Python\<version>"`  
Poté je třeba mít nainstalovanou knihovnu `numpy`, instalovanou pomocí `pip install numpy`.

## Tvoření prostředí a posílání akcí agenta

Nejdříve k ujasnění pojmu v této kapitole. Ačkoli se ve zdrojovém kódu PES pro kontrolovatelnou entitu v prostředí používá označení agent, pro přehlednost slovem agent budeme označovat pouze agenty na straně FRAgu. Standard EIS definoval předměty a obecně všechny nekontrolovatelné entity jako *entity* a agentem kontrolovatelné entity jako *kontrolovatelné entity*. Pro účely této kapitoly, kontrolovatelnou entitou, dále jen entitou, nechť je považován Python objekt, který se nachází v PES. Entita je tedy v python prostředí PES ovládána agentem z FRAgu, kde figuruje jako jeho tělo. Všechny agentem nekontrolovatelné entity se budou nazývat předměty.

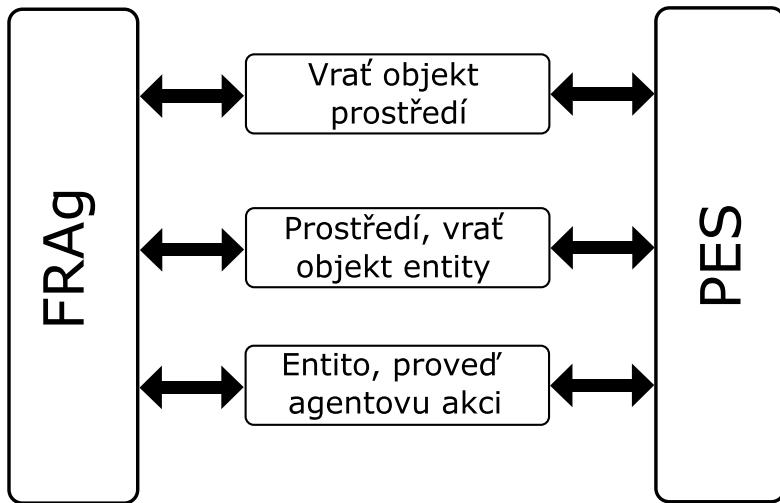
K zavolání modulu slouží predikát `py_call`. V závislosti na počtu parametrů dokáže zpracovat odpověď funkce nebo metody, kterou v Pythonu volal. Návratovou hodnotu bud-

---

<sup>2</sup><https://www.python.org/downloads/>

může použít jako hodnotu pro Term, nebo jako kontrolu, že akce v Pythonu proběhla v pořádku na základě unifikace s konstantou.

Pro účely PES se volání predikátu `py_call` dělí na dva typy. Volání modulu `py_call(module:function)` a volání objektu `py_call(object:method_name(args))`. Posloupnost volání, k tomu aby se získala z PES entita agenta, je na následujícím obrázku 5.1.



Obrázek 5.1: Posloupnost volání predikátu `py_call` při posílání akce entitě.

Počáteční tvoření prostředí je prosté, stačí jedno zavolání predikátu `py_call` na zavolání funkce `initialize_environment`.

### Posílání vjemů entity agentovi

Komplikace vzniká v této části, z důvodu reprezentace dat přicházejících z PES skrze knihovnu janus. Jako příklad jednoho z vjemů, který poslala entita do Prologu, byl `fact(danger--(10,9,8))`; nebo `fact(Episode-5)`;

S pomocí Prolog napsaného modulu `py2pl` se podařilo přetrasformovat tuto reprezentaci na žádoucí, tedy na `fact(danger(10,9,8))`; a `fact(Episode(5))`;

Stále by se však nemělo přehánět a je možné, že například při přílišném zanořování python dictionary/slovníků do jiných slovníků a listů, může konverze selhat. Proto je potřeba vždy kontrolovat, jaké vjemy a jak reprezentované agent dostává. Jedním ze způsobů je například prohlédnout si výstupní soubor agenta `.fap` (je nutno mít zapnutý debugging v `.mas2fp` v predikátu `load`).

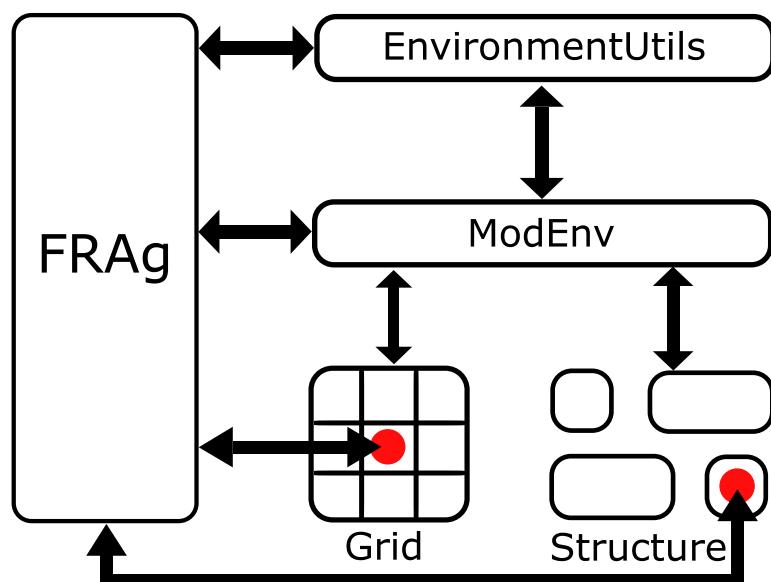
## 5.2 Detailní popis systému PES

PES využívá prostředí pracující s buňkami, které se rozlišuje na typ `Grid` (mřížka) a typ `Structure` (struktura). Mřížka i struktura mají vestavěnou časovou kontrolu, která jednou za globální cyklus vnímání posune čas dopředu o jednu časovou jednotku. Zároveň mají obě prostředí systém animace, který je však odlišný a bude popsán v jejich samostatných sekcích.

## Prostředí s buňkami – obecně

Z obrázku 5.2 lze vidět, jak FRAg komunikuje s prostředím s buňkami. Na pravé straně obrázku se nachází PES samotný. `EnvironmentUtils` je meta-prostředí zprostředkovávající vnější vstup do systému. Pokud chceme například posílat akce entitě, musíme nejprve skrze `EnvironmentUtils` získat objekt, respektive odkaz na objekt, samotného `ModEnv`. `ModEnv` (Modular Environment) je schránkou pro jeden objekt typu `Grid` – mřížka, nebo `Structure` – struktura. Zároveň si uchovává informace k hledání v prostředí, jako jsou různé předměty a jiné entity, které se v mřížce nebo struktuře nacházejí.

Předměty, entity i buňky jsou pak samy objekty, nacházející se v jednom ze dvou druhů prostředí. Pro adresaci se používají X a Y koordináty, pokud je prostředí typu mřížka. Pro typ struktury se používá název buňky.



Obrázek 5.2: Struktura systému PES, jeho moduly a s čím FRAg komunikuje.

### Předmět (Item)

Objekt, který reprezentuje předmět v prostředí. Vždy musí někomu náležet – bud ležet v buňce, nebo být nesen agentem/entitou. `Item/předmět` má atributy: název, hodnota, váha, popis a umístění. Předmět může agenta skrze entitu zvednout, za předpokladu, že ho unese. Také ho může položit do buňky, ve které se právě nachází. Pokud se předmět položí do speciální buňky (například do buňky s obchodníkem), agent dostane plnou odměnu (závislou na hodnotě předmětu) za položení. V takových případech se poruší pravidlo, že předmět musí někomu náležet, a tím efektivně z prostředí navždy zmizí.

### Buňka (Cell/NCell)

`Cell` a `NCell` jsou podobné, jejich rozdílem je jejich účel. `Cell` je určena pro typ prostředí `Grid`, zatímco `NCell` se používá na typ prostředí `Structure`. Hlavním programovacím roz-

dílem je adresování a počet sousedů. Tam, kde **Cell** využívá koordináty X a Y, využívá **NCell** název buňky – její UID. Co se týče počtu sousedů, **Cell** má čtyři hrany, tedy až čtyři přechody do sousedních buněk. **NCell** žádná omezení na počet sousedů nemá, může mít <0, N> přechodů v každé buňce. Buňka je základním stavebním prvkem obou druhů prostředí. Mezibuněčné přechody lze nastavovat jak jednosměrně, tak obousměrně. Sousední buňky, pokud je dobrý důvod, nemusí být vždy fyzicky vedle sebe. Například je možné z obdélníkové mřížky udělat válec. Stačí nastavení přechodů krajních sloupcových buněk na jejich protější krajní buňky a tím mřížku ohnout. Teoreticky lze vytvářet v mřížkovém prostředí něco jako červí díry, kdy jedním posunutím může efektivně skočit o mnohem více buněk jinam. Následuje přehled ostatních vlastností buněk:

- **Dostupnost:** Binární hodnota označující buňku jako aktivní/otevřenou, či neaktivní/uzavřenou. K neaktivním buňkám se systém staví tak, jak kdyby tam nebyly. Entita je nevnímá a i kdyby v nich byly cenné předměty, agent se o těch buňkách nikdy nedozví a do těchto buněk nelze za žádných okolností vstoupit. Primárně určené k formování přesnější podoby mřížkového prostředí, ale lze i použít například na dočasné odstavení cesty. Po reaktivaci buňky jsou její původní vlastnosti nezměněné.
- **Hodnota nebezpečí:** Pokud je prostředí samo o sobě nebezpečné, mohou mít buňky nastavenou binární hodnotu udávající, že buňka je nebezpečná. V každém takovém prostředí je pak nutné mít definovaný výpočet nebezpečí. Nebezpečnost buňky ovlivňuje penalizace agenta za pohyb.

## Generátor

Třída zajišťující dynamické generování předmětů v prostředí za běhu. Každý generátor má v této verzi PES následující vlastnosti.

## Entita

Entita, jak již bylo zmíněno, je objekt v prostředí, kterému agent posílá příkazy určující, co má daná entita vykonat. Na oplátku entita posílá agentovi seznam vjemů, které dostane od prostředí.

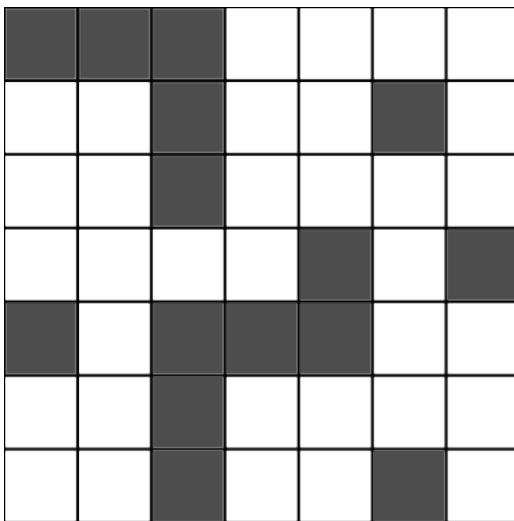
Přehled vlastností entity:

- **Váhový limit:** Entity musí mít nastavený váhový limit, určující kolik váhy unesou. Pokud je pro účely prostředí vhodné, je možné měnit váhový limit za běhu.
- **Umístění:** Každá entita musí mít za všech okolností zaregistrovanou buňku, ve které se nachází. Tato buňka navíc nesmí být neaktivní.
- **Nesené předměty:** Všechny entity, které nesou předměty, si udržují seznam všech nesených předmětů, společně s jejich celkovou hmotností. Agent vždy zná přesnou váhu, kterou nese, svůj váhový limit a všechny předměty, které právě nese.
- **Přesvědčení:** Každá entita si udržuje svá přesvědčení, která se posílají agentovi přes **Add** a **Delete** listy. Ony listy jsou pak tvořeny porovnáním vjemů z nového vnímacího cyklu se slovníkem přesvědčení, který je na konci cyklu aktualizovaný, aby reflektoval nové vjemy.

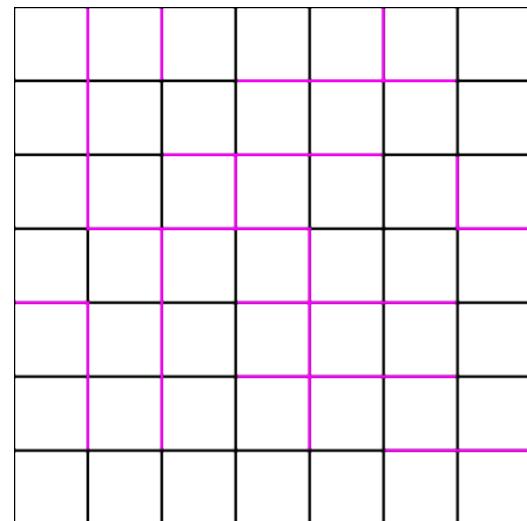
## Prostředí s buňkami typu mřížka (Grid)

Klasický typ prostředí typu mřížka. Rozšířením systému FRAg touto prací by typ prostředí **Grid** poskytl stejnou, nebo velmi podobnou funkcionality, jako **GridWorldModel** poskytuje systému Jason.<sup>3</sup> Mřížka jako taková může být jak čtvercového, tak obdélníkového tvaru. Maximálně však může nabývat velikosti  $1000 \times 1000$  políček.

Prostředí se dá dále tvarovat dle potřeb. Na tvarování jsou v prostředí dvě možnosti. První je nastavení buněk na neaktivní. Tím se z prostředí pro účely agenta/entity odstraní a nebudou o nich vědět. Druhou možností je tvorba zdí, které zruší přechody mezi sousedními buňkami. Další možností je oddělení různých lokálních prostředí.



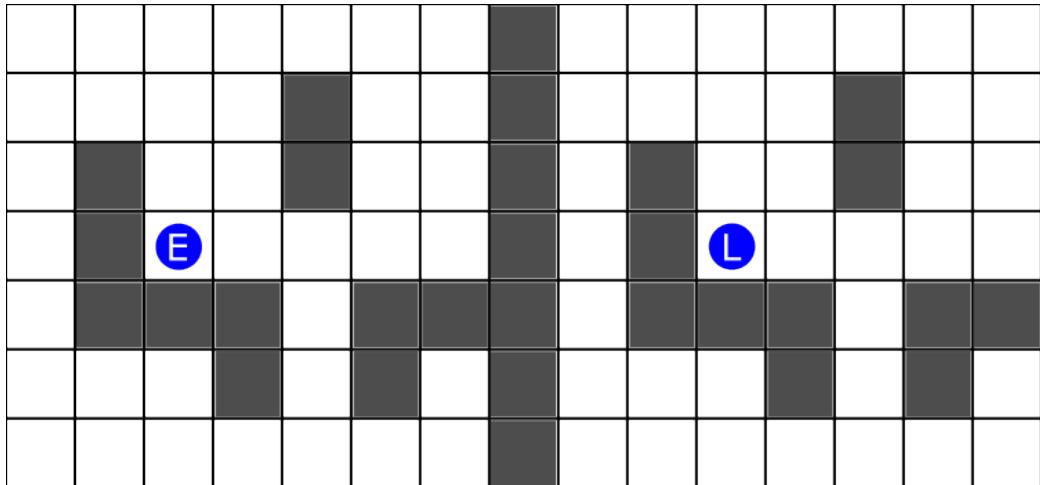
Obrázek 5.3: Tvarování pomocí neaktivních buněk.



Obrázek 5.4: Tvarování bludiště pomocí zdí.

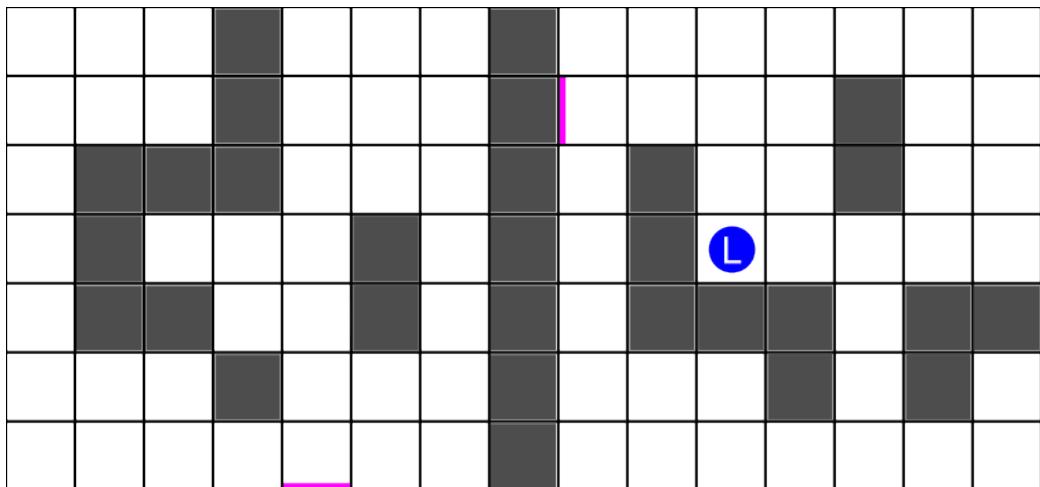
Možnost oddělení buněk dává návrháři prostředí další možnosti, pro které existují zajímavá využití. Například možnost současného testování více scénářů v jednom běhu 5.5 či porovnávání výkonnosti agenta s pozdním vázáním proměnných oproti agentovi s včasným vázáním. Stačí přes neaktivní buňky rozdělit prostředí na jednotlivá pod-prostředí, která budou mít identicky nastavené parametry.

<sup>3</sup><https://www.emse.fr/~boissier/enseignement/maop14/DOC/jason/api/jason/environment/grid/GridWorldModel.html>



Obrázek 5.5: Dvě identická pod-prostředí se stejnými parametry a rozdílnými agenty.

Další využití dělení prostředí neaktivními buňkami je například rozdělení herní plochy, které měly starší hry. Jako příklad poslouží hry ze série S.T.A.L.K.E.R. Ve hrách bylo několik menších map, mezi kterými se hráč mohl kdykoliv přesunout skrze koridory, kterými hráčova postava přeskocila během načítání z pozice  $(X_1, Y_1)$  koridoru Z mapy A, na pozici danou druhým koncem koridoru Z, ústícím v pozici  $(X_2, Y_2)$  na mapě B. Takto konfigurované prostředí je k vidění na obrázku 5.6.

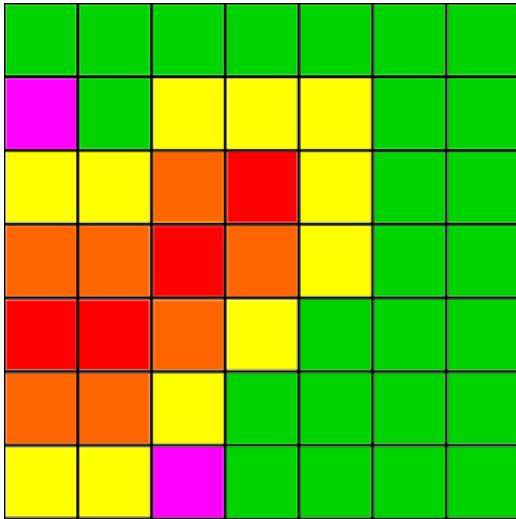


Obrázek 5.6: Dvě pod-prostředí s odlišnou strukturou mezi kterými agent může libovolně přeskakovat skrze koridory, které jsou označené fialovou barvou.

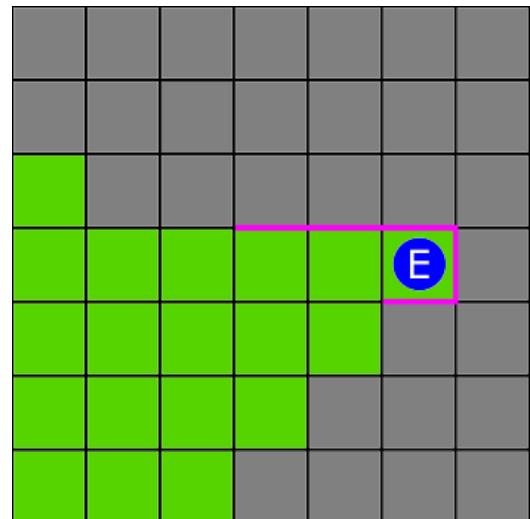
Dalšími vlastnostmi, které je vhodné zobrazit graficky, jsou zaprvé tepelná mapa, dále jen ‘heatmapa’, a buňky v dohledu entity. Heatmapy slouží svým způsobem také ke tvarování prostředí, ale ne skrze fyzické zábrany, ale jako penalizační, měkké zábrany. Agenti by si měli vybírat takové trasy, které se budou nebezpečí vyhýbat, nebo se nebezpečí snažit minimalizovat. Na obrázku 5.7 je ukázková heatmapa, kde má agent za úkol nosit z jednoho fialového políčka do druhého předměty. Agent by si měl vybrat nejkratší bezpečnou cestu.

Viditelné buňky entity v prostředí jsou dané jejími parametry, jako je dohled, tak prostředím samotným, jako jsou zdi, neaktivní buňky a hranice prostředí. K ilustraci slouží

obrázek 5.8. Vyhodnocení viditelnosti buněk má na starosti algoritmus, kterému bude nyní věnována pozornost.



Obrázek 5.7: Heatmapa představující nebezpečí jednotlivých buněk prostředí.



Obrázek 5.8: Ukázka dohledu entity čas-tečně obehnáné zdí.

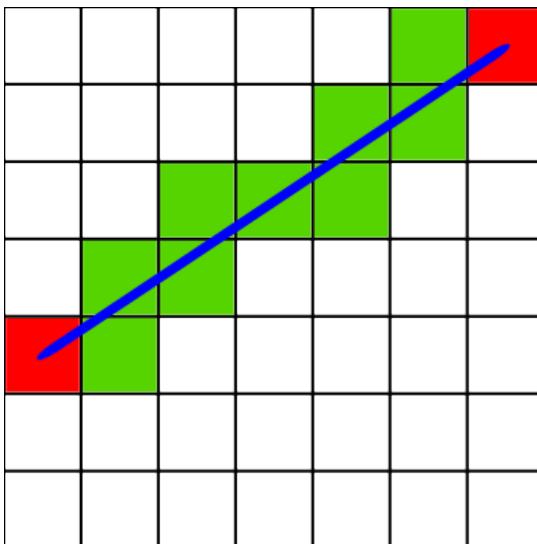
### Algoritmus na vyhodnocení viditelnosti buňky

Srdcem algoritmu je takzvaný *supercover line* algoritmus založený na Bresenhamově algoritmu, dále jen SLA. Klasický Bresenhamův algoritmus nestačí, kvůli přeskakování buňky X a Y v jednom kroku. PES používá modifikovaný SLA[3] napsaný Eugenem Dedu.

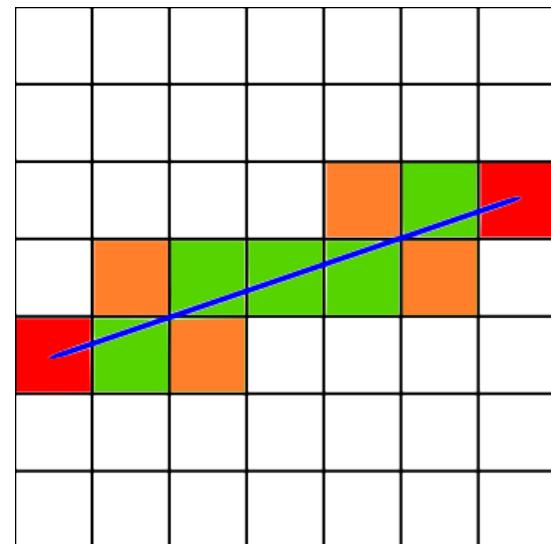
Implementace v PES vrací kompletní, nepřerušovanou cestu, tedy i s rohy, které Bresenhamův algoritmus nemá. Když má PES cestu od buňky A do buňky B, vyhodnotí se line of sight (dále jen LoS), česky čára viditelnosti. LoS se pokusí pro každou cestu vygenerovanou SLA ověřit, že buňky na cestě jsou aktivní a všechny přechody existují – zdi se nevyskytují v cestě. LoS rozlišuje dva případy.

- **Čára neprocházející rohy 5.9:** Jednoduchý případ, testující možný průchod cestou. Pokud kdykoliv dojde k jakémkoliv chybě, cílová buňka viditelná není.
- **Čára procházející rohy 5.10:** Složitější případ, který vyžaduje v případě selhání:
  1. ověření, jestli se z buňky bude procházet rohem,
  2. pokud selhal první přechod, pokusí se udělat krok na alternativní buňku,
  3. při selhání druhého přechodu, provede se návrat a následně se vyzkouší alternativní cesta.

Cílová buňka může být viditelná i s chybou, pokud proběhlo úspěšné nalezení alternativní cesty u čáry procházející rohy.



Obrázek 5.9: Ukázka trasy neprocházející rohy.



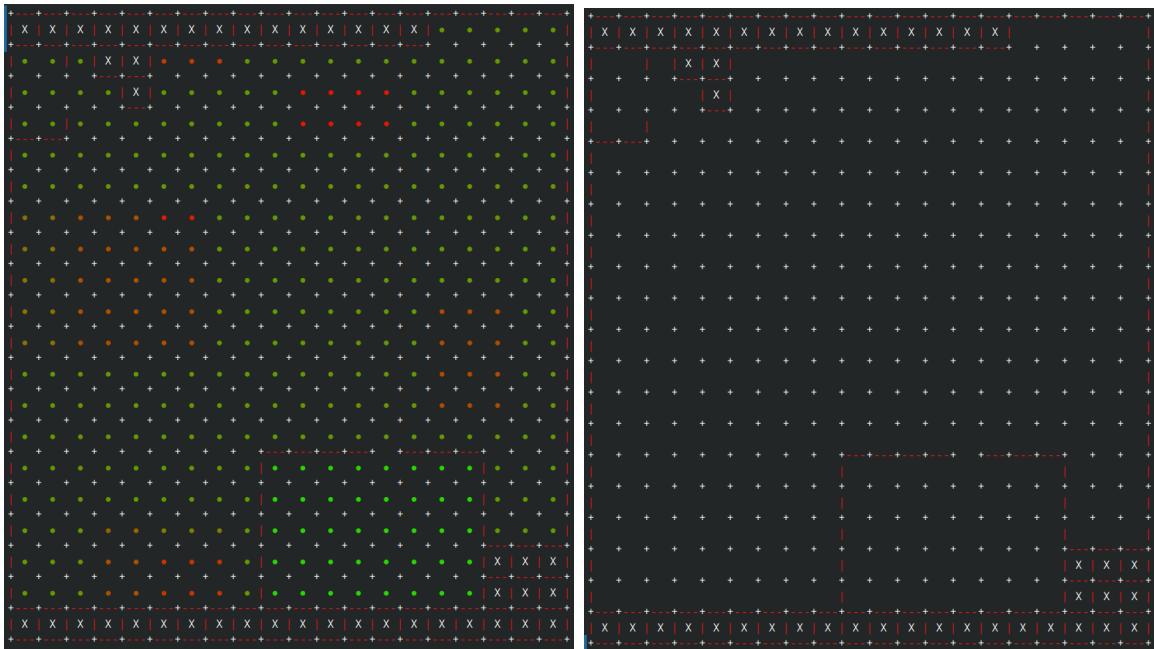
Obrázek 5.10: Ukázka trasy procházející rohy. Obě oranžové buňky jsou na trase.

### Animace v mřížkovém prostředí (Grid)

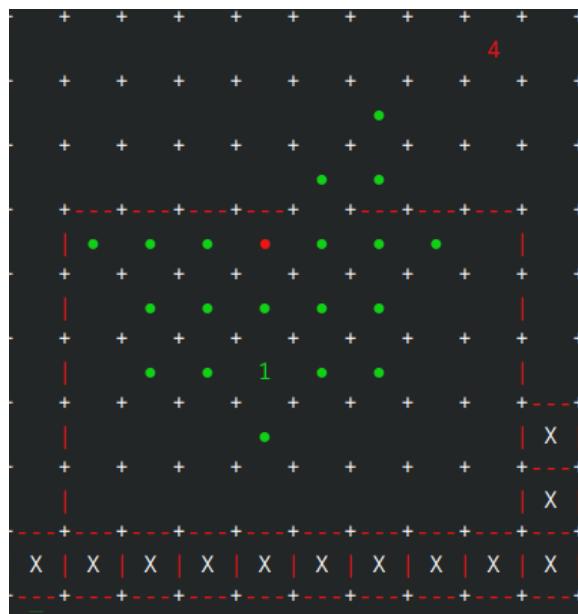
Animace prostředí jsou realizovány skrze metodu `draw` třídy `Grid`. Kreslící metoda na standardní výstup generuje ASCII art prostředí s možností různých režimů. Následuje krátký přehled režimů vykreslování, kterých má metoda v základu pět.

- **Strukturální 5.12:** Vykreslí pouze kompletní mřížku, se všemi zdmi a neaktivními buňkami. Vhodné zejména pro validaci, že prostředí je konfigurováno korektně.
- **Předmětové:** Vykreslí všechny předměty, nacházející se v prostředí. Předměty jsou reprezentovány v buňce celočíselnou hodnotou, dle jejich četnosti.
- **Heatmapové nebezpečí 5.11:** Vykreslí nebezpečí každé aktivní buňky v prostředí. Nebezpečnost je zobrazována kolečkem barvy. Kolečko nabývá barvy v rozmezí zelené (žádné nebezpečí), plynule až po barvu červenou (maximální nebezpečí v prostředí).
- **Entitní:** Vykreslí všechny entity, nacházející se v prostředí. Entity jsou reprezentovány v buňce celočíselnou hodnotou, dle jejich četnosti.
- **Dohledové 5.13:** Vykreslí všechny předměty v prostředí jako celočíselnou hodnotu červenou barvou. Pokud se však předmět nachází v entitou viditelné buňce, je číslo vypsáno zelenou barvou. U ostatních viditelných buněk, ve kterých nejsou žádné předměty, se vykreslí pouze zelené kolečko. Entita samotná se vykreslí jako kolečko červené. V případě, že by entita sama stála na buňce obsahující předmět, vypíše se místo kolečka opět číslo počtu předmětů v buňce, tentokrát však modrou barvou.

Pro ukázkou, jak některé takové režimy vypadají v praxi, následují snímky obrazovky režimů z prostředí **stalker**, které bude podrobně představené v další kapitole.



Obrázek 5.11: Ukázka vykreslení heatmapy Obrázek 5.12: Ukázka strukturálního vykreslení.  
nebezpečí.



Obrázek 5.13: Ukázka vykreslení dohledového režimu.

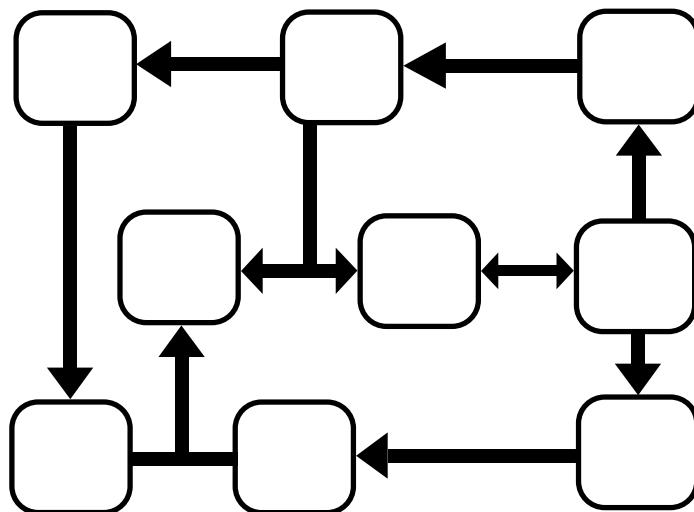
Animace pak využívá metodu `draw` následujícím způsobem. Podle následujících parametrů spouští některý z agentů animaci, a to po dokončení svého vnímacího cyklu.

- Délka prodlevy animace: Jak dlouhá mezera je mezi jednotlivými vykresleními.
- Režim vykreslení: Jeden z pěti výše uvedených režimů.
- Animátor: Agent, který po dokončení vnímacího cyklu volá kreslící metodu.

## Prostředí s buňkami typu struktura (Structure)

Jak bylo zmíněno dříve, na rozdíl od mřížky má struktura neomezeně mnoho možných sousedních buněk. Ukázka struktury se nachází na obrázku 5.14. To je nesporná výhoda, která je však vykoupena absencí dohledu entity. Entita může dostávat vjemy pouze z té buňky, ve které je právě v době vnímacího cyklu situována. Dalším rozdílem a ústupkem je systém animací. Animace jako takové by byly problematické, ačkoli ne nemožné. PES v současném stavu řeší animace struktur jako opakování volání metody `debug`, náležející entitě. Ta přehledně vypisuje veškeré informace, které by mohly výzkumníky zajímat:

- **Název sama sebe – entity.**
- **Název buňky ve které se nachází.**
- **Seznam předmětů, které nese.**
- **Seznam přesvědčení, který zahrnuje:**
  - váhovou kapacitu a současně nesenou váhu,
  - seznam předmětů v buňce,
  - seznam ostatních agentů v buňce a
  - jeho vlastní pozici.



Obrázek 5.14: Ukázka směrového propojení jednotlivých buněk.

K verifikaci správnosti struktury se dá po vytvoření zavolat metoda `debug` třídy `ModEnv`, která pro všechny buňky zavolá jejich vlastní metodu `debug`. Ta vypisuje předměty a entity, které se v ní nacházejí, stejně jako seznam přechodů do ostatních buněk (buňky, ze kterých se dá pouze přicházet, se v seznamu neobjeví).

## Bonus – Tvorba abstraktních prostředí

Zastupitelem abstraktního prostředí je například shop (jedna z ukázkových úloh systému FRAg, které budou vysvětleny v další kapitole), u kterého nedává smysl, aby kupující, prodávající či zprostředkovatel měli jakoukoli lokalitu. Není podstatné, aby v prostředí existovaly buňky, které postrádají smysl. Takováto prostředí lze v současné implementaci PES tvořit jako jednu buňku, ve které se nacházejí všechny dostupné informace. Kvůli animacím se doporučuje využívat prostředí typu struktura. Celkově je systém PES v současné době orientovaný spíše na práci s prostředím obsahujícím buňky.

## Současné limitace a plánovaná rozšíření PES

Zde následuje výčet současných limitací PES, které existují, seřazené číselně dle priority jejich budoucího rozšiřování.

1. **Klonování prostředí:** Umožňuje lepší plánování.
2. **Animace struktur:** Vytvoření systému animace pro struktury místo animování skrze zaznamenávání debugovacích funkcí.
3. **Robustnější systém dynamického tvoření předmětů:** V současné verzi se jedná o vlastnost buňek, které mohou generovat předměty podle manuálně nastavených parametrů.

## Tvoření prostředí s využitím PES

Tato sekce pokrývá spíše teoretický postup, pro více praktický, detailní a programátorský postup odkazuje tato práce na soubor `guide.md`, který se mimo jiné zabývá už samotnými třídami, metodami a celkově programovacími aspekty systému PES.

Při vytváření nového prostředí je nutné vytvořit minimálně 4 nové soubory. První dva se starají o prostředí PES a jeho komunikaci se systémem FRAg. Zbývající řeší nastavení prostředí z agentního hlediska. Soubor `.mas2fp`, který určuje, kolik agentů v prostředí bude, jaké budou používat metody uvažování, jejich maximální doba běhu a podobné, a jak se daný agent bude v prostředí chovat (jaké jsou jeho záměry, plány, ...) nacházející se v souboru `.fap`.

Prvními dvěma soubory zmíněnými v předešlém odstavci se rozumí `prostredi.pl` a `prostredi.py`. Je nutné, aby nově vytvořená prostředí zachovávala FRAg strukturování, tedy oba soubory by měly být ve složce `prostredi`, která se nachází v cestě `FRAg/core/environments/`. Dále, `prostredi.pl` spoléhá na to, že jeho sesterský soubor `prostredi.py` se nachází na jeho úrovni a jeho pomocný soubor `py2pl.pl` se nachází na úrovni pod ním. Soubor `.pl` slouží na přemostění ze systému FRAg do PES a zpátky. V každém `.pl` takového typu je nutné mít:

- pojmenovaný modul s exportovaným predikátem názvu prostředí, které může agent volat,
- Importovanou knihovnu `janus`,
- Importovaný modul `py2pl`, který transformuje výstup PES pro FRAg interpretovateľný výstup,

- predikátová pravidla, která musí v souboru existovat. Mezi ně patří `perceive`, který z PES vrátí `add_list` a `delete_list` a pravidla popisující možné agentovy akce, které musí mít definici u třídy `Agent` v `prostredi.py`,
- Direktivu nastavující janus a volání funkce na inicializaci prostředí `janus`.

Soubor `prostredi.py` zajišťuje prostředí samotné. Jeho zodpovědnosti, které musí být zajištěny, jsou:

- funkce inicializace prostředí, která musí prostředí vytvořit (vytvoření `Grid`, nebo `Structure`, případně tvarování prostředí, počáteční předměty,...), a to předtím, než se do něho FRAg pokusí zavést agenty,
- načtení si všech podpůrných tříd z `FRAg/env`,
- pomocí dědění tříd přepsat metody tříd `Agent`, `ModEnv`, `Cell`, `Item` pokud je třeba, aby byly v souladu s parametry tvořeného prostředí,
- definované hodnoty proměnných, které byly ve třídě `ModEnv` pouze deklarované,
- vytvořený slovník koordinátů či názvů buněk, spojujících podle jména agenta s jeho počátečním umístěním.

Mezi nejčastěji přepisované metody patří `percept_g` a `percept_s`, třídy `Agent`, tvořící `add_listy` a `delete_listy` z prostředí `Grid`, nebo `Structure`. Dalšími často přepisovanými metodami jsou obecně metody:

- `debug`, u kterých můžeme chtít vypisovat informace specificky přidané prostředí a
- akce agenta jako `move_to_cell`, `drop_item`, `pick_item`, u kterých můžeme chtít jiné odměny, nebo jako v případu prostředí miconic zabalit metodu `move_to_cell` do jiné, která se bude posouvat místo jedné buňku za tah a libovolný počet opakovaným voláním metody `move_to_cell`.

# Kapitola 6

## Podrobný popis implementovaných prostředí

### 6.1 Výrobní linka

#### Obecný popis prostředí:

Prostředí má za úkol simuloval výrobní linku[28]. Agenti mají za úkol dopravovat materiály ze skladů A a B do dílny, kde následně jeden či více strojů vyrobí produkt. Tento produkt pak ti samí agenti, kteří mají na starost přepravu materiálů, musí dopravit do místnosti pro odbavení. Prostředí je převzato z článku[28] a lehce modifikováno. Jeho struktura je k vidění na obrázku 6.1.

#### Cíle prostředí

- Spolupráce:** Agenti by si měli vybírat vhodné role a tím zvyšovat efektivitu. Kámen má vyšší hodnotu, než dřevo. Ke zhotovení výrobku je však třeba jak dřevo, tak kámen. Některý agent by se tedy měl "obětovat," pro dobro celku, aby mohl stroj vyrábět.
- Vhodné upřednostňování akcí:** Jsou-li všechny stroje na výrobu vyřazeny z provozu a opravují se, agenti by měli tento čas využít produktivně k naskladnění materiálů a doručení produktů do místnosti pro odbavení.

V tomto prostředí se tedy hodnocení vztahuje na agenty jako na celek. Čím více agentů bude sebestrádných a odmítat práce, které mají nižší ohodnocení, tím méně efektivní celá výrobní linka bude.

#### Klasifikace

- Počet agentů:** Dva a více.
- Agenti v týmu:** Homogenní.
- Logistika:** Závažný problém.
- Dlouhodobé plánování:** Závažný problém.
- Emergentní spolupráce:** Závažný problém.

- **Hostilita:** Ne.
- **Čas na rozhodnutí:** Sekundy – Minuty.
- **Dostupnost informací:** Přiměřeně dobrá.
- **Reprezentace:** Symbolické.
- **Řízení:** Distribuované.

Prostředí výrobní linky je zhotoveno s pomocí třídy **Structure**. Jedná se tedy o prostředí, které není abstraktní – entita/agent musí mít za všech okolností přiřazenou buňku.

### **Prostředí má pět buněk/místnosti:**

- **Shipment (Místo pro odbavení):** Všechny výrobky, které agenti vyrobili a donesli do této oblasti je vyvezeno ven – přestane v prostředí existovat.
- **Workshop (Dílna):** Agent má za úkol nosit materiály ze skladu A, kámen, a ze skladu B, dřevo. Pokud je v místnosti dostatek materiálů, agent může zapnout stroj, který produkt vyrobí. Jeho výroba spotřebuje daný materiál.
- **Warehouse (Sklady):** Ve dvou skladech se nachází materiály nutné k sestavení výrobku.
- **Hall (Hlavní hala):** Propojovací místnost, ze které je možné jít do skladů, či do místnosti pro odbavení.

### **Předměty nacházející se v prostředí:**

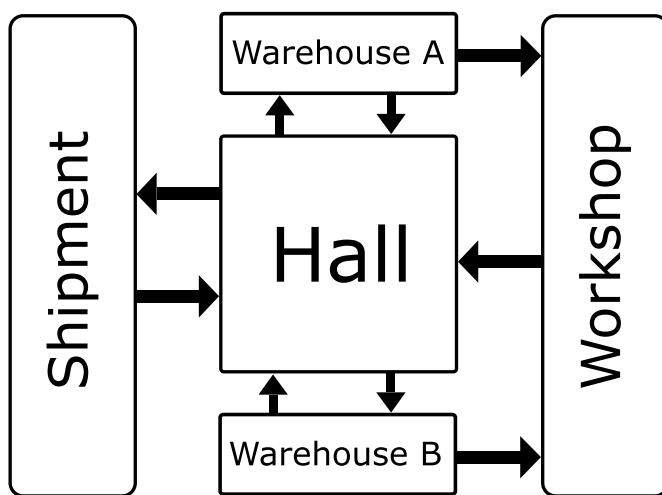
- **Kámen:** Váha 10, hodnota 5.
- **Dřevo:** Váha 10, hodnota 3.
- **Produkt:** Váha 10, hodnota 30.

### **Vjemy prostředí:**

- **fact(weight\_limit(10)):** Agent unese váhu (X), včetně.
- **fact(carrying\_weight(5)):** Teď nese (X).
- **fact(closed(@(true))):** Prostředí se zavírá.
- **fact(self(robot\_b,"Workshop")):** On sám se nachází v buňce s názvem "Workshop".
- **fact(agent(robot\_a,"Workshop")):** Agent paul se nachází v buňce s názvem "Workshop".
- **fact(item('Stone',5,10,"Warehouse A", <uuid>)):** předmět(Název, hodnota, váha, název buňky, nějaké uuid).
- **fact(carrying(stone,5,10)):** nesený předmět(Název, hodnota, váha).
- **fact(available\_machines(3)):** Momentálně fungující stroje, které čekají na práci.

## Akce a jejich odměny

- **Pohyb:** Jakýkoliv pohyb je penalizován ztrátou jednoho bodu.
- **Výroba:** Zapnutí stroje na výrobu je odměněno získáním patnácti bodů.
- **Zvednutí předmětu:** Jakékoliv zvednutí předmětu je odměněno získáním jednoho bodu.
- **Položení předmětu:** Pokud předmět položí do místnosti, do které daný předmět patří, dostane za něho plnou odměnu danou hodnotou předmětu. V opačném případě za položení nedostane odměnu žádnou, přesněji dostane odměnu nula.



Obrázek 6.1: Směrové propojení jednotlivých místností.

## Zvolené parametry prostředí:

Uvedené hodnoty parametrů se dají měnit, následující výpis jsou hodnoty výchozí.

- **Míra poruchovosti strojů:** Stroj se porouchá průměrně jednou za čtyřicet jednotek času a zůstává nefunkčním 10 časových jednotek.
- **Počet strojů v prostředí:** V základním nastavení se v prostředí nachází 3 stroje.
- **Počet časových jednotek na zhřivení produktu:** 2.
- **Počet agentů v prostředí:** 4.
- **Váhový limit agentů:** 10.
- **Generace:** Jakmile je možné naplánovat vytvoření materiálu Wood nebo Stone, plánují se pro generaci za průměrně 3 časové jednotky.

## Výsledky

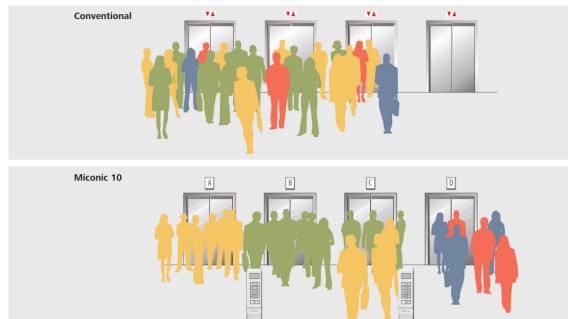
Test	Počet předmětů	Počet agentů	Early/Late	Omezení	Splněno za
1	5	4	Early	2	49.1
2	5	4	Early	5	43.7
3	5	4	Late	2	48.5
4	5	4	Late	5	42.8
5	100	4	Early	2	838.2
6	100	4	Early	5	709.4
7	100	4	Late	2	807.1
8	100	4	Late	5	693.2
9	100	4	Early	5	558.5
10	100	4	Late	5	534.2

Tabulka 6.1: Časy dokončení výrobní linky před a po optimalizaci .fap-souboru. Sloupec *Omezení* udává maximální počet materiálů ve skladu (při vypnuté generaci). Sloupec *Splněno za* uvádí čas v epizodách (1 epizoda = jedno vnímací kolo všech agentů).

Z tabulky 6.1 je vidět, že pozdní vázání proměnných dosahuje ve všech případech lepších výsledků. Mnohem markantnější zlepšení však bylo úpravou souboru .fap. Úprava spočívala například v tom, že agent nosící produkt k odvezení se nejdříve podíval, jestli nějaký produkt v dílně chybí. Ten posléze donesl zpátky, našel-li ho ve skladu.

## 6.2 Výtah Miconic 10 – pouze agentní systém

Obecný popis prostředí: Schindler Miconic 10 je komerčně nasazený *systém odesílání do místa určení*. Tento systém (výtah) využívající agenty od firmy *Schindler*, je používán v mnoha zemích světa. U běžných výtahů si pasažér pouze zvolí, zda chce z patra jet směrem nahoru či dolů, a nastoupí do prvního výtahu, který do patra přijede. Výtahy Miconic 10 pracují na jiném principu. Pasažér si zvolí konkrétní patro, do kterého by chtěl jet, a je mu sděleno, ke kterému konkrétnímu výtahu se má dostavit. Tímto způsobem systém agreguje pasažéry a minimalizuje počet zastávek. Podle výrobce je zkrácení doby jízdy cestujících během špičkového provozu až o 30 % oproti běžné skupinové kontrole.[2] To je především zásluha tří věcí pracujících ruku v ruce. Znalost, kolik pasažérů stojí za požadavkem, znalost cílového podlaží každého cestujícího a systém plánování. Porovnání aggregace pasažérů na jejich požadavcích je patrné z obrázku 6.2 pocházejícího od výrobce, stejně jako samotné cesty výtahu Miconic 10 v porovnání s typickými nacházejícími se na obrázku 6.3.



Obrázek 6.2: Porovnání agregace pasažérů podle jejich destinace v systému Miconic 10 oproti běžnému výtahu.[2]



Obrázek 6.3: Trasy výtahu Miconic 10 oproti běžným výtahům.[2]

Agent, výtah, umístěný v tomto prostředí, má za cíl přemístit všechny pasažéry do jimi zvoleného místa určení. Úloha byla zvolena jako jedno z nejjednodušších, ale přesto v realitě úspěšně nasazených využití MAS, ačkoli v implementaci této úlohy se jedná pouze o agentní systém. Implementované prostředí má takovouto podobu 6.4.

## Cíle prostředí

- Chytré plánování tras:** Agent by si měl vybírat vhodné trasy, k uspokojení všech pasažérů. Měl by minimalizovat celkový čas potřebný k převezení všech pasažérů.

## Klasifikace

- Počet agentů:** 1.
- Agenti v týmu:** Ne.
- Logistika:** Závažný problém.

- **Dlouhodobé plánování:** Závažný problém.
- **Emergentní spolupráce:** Ne.
- **Hostilita:** Ne.
- **Čas na rozhodnutí:** Sekundy.
- **Dostupnost informací:** Kompletní.
- **Reprezentace:** Symbolické.
- **Řízení:** -

Prostředí miconic je zhotoveno s pomocí třídy `Grid`. Jedná se tedy o prostředí, které není abstraktní – entita/agent musí mít za všech okolností přiřazenou buňku. Prostředí je zhotoveno jako mřížka  $1 \times 20$ . Aby entita byla schopna vidět všechny pasažéry, musí být její dohled nastaven alespoň na výšku mřížky.

#### Předměty nacházející se v prostředí:

- **Pasažér:** Jméno, destinace Y1, váha X(desítky), popis, současná lokace Y2.

#### Vjemy prostředí:

- **fact(weight\_limit(200)):** Agent unese váhu (X), včetně.
- **fact(carrying\_weight(50)):** Ted je zatížený (X).
- **fact(all\_served(@(true))):** Všichni pasažéri obsluženi.
- **fact(self(lift,0,10)):** Agentova pozice (X,Y).
- **fact(item('Martin', 3, 50, 0, 9, <uuid>)):** Pasažér(Jméno pasažéra, destinace, váha, 0, Y, nějaké uuid (v tomto případu by konkrétní shoda nastat neměla)).
- **fact(carrying('Martin', 3, 50)):** Přepravuje(Jméno pasažéra, destinace, váha).

#### Akce a jejich odměny

- **Pohyb:** Pohyb výtah provádí skokově, penalizuje se ztrátou absolutní vzdálenost, kterou agent skočil.
- **Stop:** Nejdříve nechá vystoupit všechny své pasažéry, které mají patro ve kterém stojí, jako svůj cíl. Za každého pasažéra dostane 41 bodů ( $2 \times$  výšku prostředí + 1). Poté nechá nastoupit všechny pasažéry, dokud bud nenaštoupí všichni čekající v patře, nebo není překročen váhový limit. Za nastoupené pasažéry odměnu nedostává.



## Zvolené parametry prostředí:

Uvedené hodnoty parametrů se dají měnit, následující výpis jsou hodnoty výchozí.

- **Počet pater budovy:** 20.
- **Počáteční pasažéři při startu prostředí:** 26.
- **Počet agentů v prostředí:** 1.
- **Váhový limit agenta:** 200.

## Výsledky

Výtah, ačkoliv neoptimální, řeší úlohu správně. Respektuje svoji váhu, nesnaží se nabrat cestující, které neunesete. Všechny cestující doveze na správné místo. Vždy, když nemůže udělat optimální plán, vybere si patro s náhodným cestujícím, kterého dokáže unést. Přepracování strategie by bylo jednodušší na straně prostředí. Při předělání metody

### 6.3 Prostředí stalker

Obecný popis prostředí: Prostředí stalker je inspirováno sérií her S.T.A.L.K.E.R, kde hráč a NPC (počítacem ovládané postavy) čelí nebezpečnému světu, na který jsou zpravidla sami. Nebezpečí v herním světě představuje nejen prostředí samotné skrze anomální zóny, ale i smrtná fauna, která se v něm pohybuje. Aby stalker – člověk, který žije v černobylské uzavřené zóně – přežil, musí hledat zásoby, přijímat nebezpečné práce a minimalizovat přitom vystavení se nebezpečí.

Prostředí stalker v této práci mnoho aspektů přežití zjednodušuje. Nebezpečnost prostředí je dána v každém políčku nebezpečí vyskytujícími se anomáliemi a frekventovaností výskytu zmutované fauny. Momentálně se jedná o nebezpečné prostředí, kde je hledání artefaktů na prvním místě a agenti se předhání o získání artefaktů. Ty agent sbírá a odnáší obchodníkovi, který mu za to dá peníze (odměnu). Je to však prostředí, které vybízí k mnoha rozšířením.

- Implementace více akcí, jako **střel**, kterou by agent mohl zranit, případně zabít jiného agenta a tím mu sebrat věci.
- S implementací střelby by se daly zavést frakce, které by znemožňovaly palbu do vlastních řad, ale umožňovaly agentům cestovat ve větší skupině, které by mohli věřit.
- Zavedení jiného druhu agentů, takzvaných mutantů, jejichž jediným úkolem je zabíjet stalkery a ostatní mutanty. Ačkoliv by nemohli střílet, pohybovali by se dvakrát tak rychleji a stalkerům, kteří preferují prozkoumávat prostředí o samotě, by velmi ohrožovali.
- Zavedení takzvaných emisí (smrtně nebezpečných, krátkých bouří, které pochází z černobylské jaderné elektrárny). Tyto emise by nutili stalkery přestat dělat cokoliv, co momentálně dělají a okamžitě najít úkryt, jimiž by byla předem daná políčka v prostředí. To, že probíhá emise by samozřejmě neznamenalo, že mezi stalkery existuje příměří, ba naopak je celkem pravděpodobné, že ke střelbě by docházelo v bezpečných oblastech, kde by najednou byla veliká koncentrace stalkerů. Implementovaná mapa ze hry stalker je zobrazena na obrázku 6.5, její struktura a heatmapa nebezpečí je poté zobrazena na obrázku 6.6.

## Cíle prostředí

1. **Chytré plánování tras:** Agent by si měl vybírat vhodné trasy, aby sesbíral co nejvíce artefaktů, které donese obchodníkovi. Artefakty se nachází v buňkách se zvýšenou anomální aktivitou. Cesty ke sbírání artefaktů by si tedy měl volit takové, aby vkročil na co nejméně velmi nebezpečných políček. Stejně tak při cestování za obchodníkem by si měl vybírat cestu, která má nejmenší celkovou penalizaci.
2. **Sbírání podpůrných předmětů:** Agent v prostředí může nalézt předměty, které samy o sobě nemají vysokou hodnotu při prodeji, ale pomohou mu jiným způsobem. Jsou jimi binokuláry, které mu zvýší jeho dohled o 1 políčko.

## Klasifikace

- **Počet agentů:** 2 a více.
- **Agenti v týmu:** Ne.
- **Logistika:** Ne.
- **Dlouhodobé plánování:** Mírný problém.
- **Emergentní spolupráce:** Ne.
- **Hostilita:** Ano – prostředí.
- **Čas na rozhodnutí:** Sekundy – minuty.
- **Dostupnost informací:** Špatná.
- **Reprezentace:** Symbolické.
- **Řízení:** Distribuované.

## Předměty nacházející se v prostředí:

- **Artefakt:** Název, hodnota, váha, popis, lokace (X, Y).

## Příklady vyskytujících se vjemů:

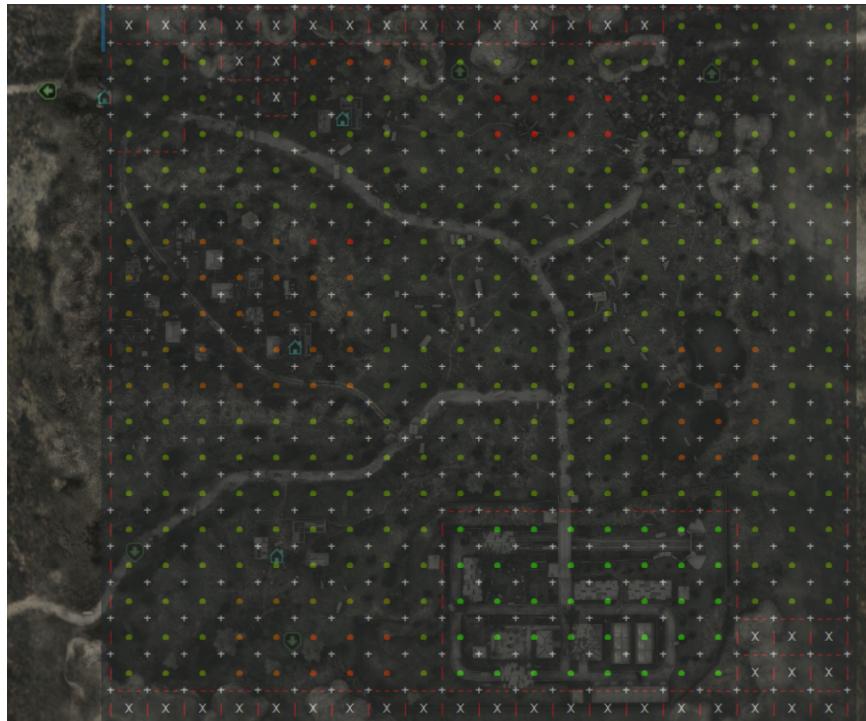
- **fact(danger(10,9,10)):** políčko(negativní odměna, X, Y).
- **fact('TRADER'(10,15)):** Obchodník(X,Y).
- **fact(weight\_limit(10)):** Agent unese váhu (X), včetně.
- **fact(carrying\_weight(5)):** Ted nese (X).
- **fact(closed(@(true))):** Prostředí se zavírá.
- **fact(self(peter,11,10)):** Agentova pozice (X,Y).
- **fact(agent(paul,11,10)):** Agent paul se nachází na políčku (X,Y).
- **fact(item('Ball',300,5,7,9, <uuid>)):** předmět(Název, hodnota, váha, X, Y, nějaké uuid).
- **fact(carrying(stone,5,10)):** nesený předmět(Název, hodnota, váha).

## Akce a jejich odměny

- **Pohyb:** Agent se pohybuje buňku po buňce, při každém pohnutí dostává penalizaci rovnající se nebezpečnosti buňky.
- **Prozkoumání předmětu:** Agent dostane malou odměnu za prozkoumání předmětu.
- **Sebrání předmětu:** Agent může dostat pouze sekundární odměnu, vyplývající z vylepšených vlastností agenta.
- **Položení předmětu:** Agent dostane odměnu v hodnotě neseného předmětu, pouze pokud položí daný předmět na poličko obchodníka. V opačném případě nedostane odměnu žádnou (0).



Obrázek 6.5: Snímek obrazovky ze hry S.T.A.L.K.E.R., konkrétně oblasti *Army Warehouses*. Jedna z několika desítek samostatných oblastí, která je součástí mapy. Ideální kandidát pro rozšíření funkčnosti díky bohaté přítomnosti frakcí a frekvenci mutantů.



Obrázek 6.6: Struktura prostředí společně s heatmapou nebezpečí.

### Zvolené parametry prostředí:

Uvedené hodnoty parametrů se dají měnit, následující výpis jsou hodnoty výchozí.

- **Poloměr dohledu agenta:** 3 (4, pokud nese dalekohled).
- **Počet agentů v prostředí:** 4.
- **Váhový limit co agent unese:** 10.
- **Velikost prostředí:**  $20 \times 20$ .

### Výsledky

Samotný MAS naprogramovaný vůči tomuto prostředí nebyl. Funkčnost komunikace je však ověřena pomocí předpřipravené cesty a akcí, které mají agenti v prostředí dělat.

## 6.4 Obchod – není implementovaný skrze PES

Vzhledem ke skutečnosti, že prostředí už implementované ve FRAgu je, není implementováno v PES a tedy v této práci. Je však popsáno, alespoň obecně, nejen prostředí **shop**, ale jak by se případně dalo vytvořit v PES. Obecný popis prostředí: Základní testovací prostředí dodávané se systémem FRAg. Slouží jak k testování, že systém FRAg na daném zařízení funguje, tak k vyhodnocení pozdní versus brzké strategie vazby proměnných v BDI agentech. Toto virtuální tržní prostředí simuluje obchodování se CDčky, kde kupující i prodávající postupně přicházejí a odcházejí podle stochastických rozdělení. Každé CD má

dynamickou cenu, kterou stanovují kupující a prodávající, přičemž platí, že kupující i prodávající zůstanou na trhu jen po omezenou dobu. Agenti zde vystupují jako zprostředkovatelé (resellers), kteří párují kupující a prodávající, pokud je cena pro obě strany přijatelná.[28]

### Kroky k implementaci prostředí v PES

Prostředí **shop** by mohlo být upraveno s ohledem na fakt, že se nachází v buňkovém prostředí. Původní fakta jako CDčka, prodávající a obchodníci by se stala předměty v jediném existujícím políčku. Bylo by tudíž nezbytné upravit **trader.fap** soubor, aby reflektoval změny vnímání.

### Předměty nacházející se v prostředí ve formě vjemů:

- **Prodávající:** `fact(Item(seller54, 0, 50, "CD5", "shop"))` – (název předmětu, bez významu, minimální cena za kterou předmět prodá, jaké CDčko, název buňky).
- **Kupující:** `fact(Item(buyer42, 0, 55, "CD5", "shop"))` – (název předmětu, bez významu, maximální cena za kterou předmět koupí, jaké CDčko, název buňky).
- **CD:** `fact(Item(CD5, 0, 0, "popis CD5", "shop"))` – (název CDčka, bez významu, bez významu, případný popis CDčka, název buňky).

# Kapitola 7

## Závěr

Tato práce podrobně představila agenty jako nástroj pro řešení složitých, jednoduchými algoritmy často neřešitelných problémů. Dále pak rozšířila škálu řešitelných problémů stručným vysvětlením multiagentních systémů. Před samotným návrhem byla rozebrána problematika prostředí, včetně jeho zodpovědností a vlastností.

Na samotnou volbu prostředí byl navržen a implementován systém PES (**P**ython **E**nvironment **S**ystem) pro rychlou tvorbu testovacích prostředí kompatibilních s multiagentním systémem FRAg. Se systémem PES jsou vytvořena tři prostředí (dvě multiagentní a jedno agentní). Prostředími jsou výrobní linka (multiagentní), model výtahu Miconic 10 (agentní) a prostředí inspirované hrou S.T.A.L.K.E.R. (multiagentní). Na kooperativním prostředí výrobní linky se ukázala pozdní vazba proměnných jako efektivnější než brzká, zároveň se však ukázalo, že optimalizovaný agentův program je o mnoho důležitější z hlediska efektivity.

Přínos práce více než v testovacích úlohách spočívá v modularitě a snadné rozšiřitelnosti prostředí díky systému PES. Systému FRAg poskytuje prostředí typu mřížka a typu struktura. To s vestavěným systémem animací, včetně různých režimů v případě mřížky. Dále je možné díky prostředí typu mřížka simulovat více agentů ve stejné kopii prostředí současně díky rozčlenění a izolaci celého prostředí na identické, menší části.

Realizované prostředí PES tak představuje užitečný nástroj pro další experimenty s BDI a multiagentními systémy a otevírá cestu k rychlejším iteracím při vývoji i evaluaci. Zároveň to otevírá dveře těm, kteří by si předtím kvůli nutnosti programování v Prologu systém FRAg ani nezkusili.

# Literatura

- [1] CAILLOU, P.; GAUDOU, B.; GRIGNARD, A.; TRUONG, C. Q. a TAILLANDIER, P. A Simple-to-use BDI Architecture for Agent-based Modeling and Simulation. In: *Proceedings of the Eleventh Conference of the European Social Simulation Association (ESSA 2015)* online. Groningen, Netherlands: [b.n.], Září 2015. Dostupné z: <http://www.essa2015.org/>.
- [2] CORPORATION, S. E. *Schindler Miconic 10®: The Original Destination Dispatch System*. Product brochure. Morristown, New Jersey, USA: Schindler Elevator Corporation, 2007. Dostupné z: [https://sweets.construction.com/swts\\_content\\_files/620/241912.pdf](https://sweets.construction.com/swts_content_files/620/241912.pdf). Brochure No. BRN-1004 A0710.
- [3] DEDU, E. *Bresenham-based Supercover Line Algorithm* online. Červen 2001. Dostupné z: <https://dedu.fr/projects/bresenham/>. [cit. 2025-05-12].
- [4] FAISAL, B. a TUNKEL, D. *Explainable AI in Multi-Agent Systems: Advancing Transparency with Layered Prompting* online. 2025. Dostupné z: <https://doi.org/10.13140/RG.2.2.11455.42400>. [cit. 2025-04-14].
- [5] GERSHENSON, C. Self-Organizing Traffic Lights. *Complex Systems* online, 2005, sv. 16, č. 1, s. 29–53. Dostupné z: [https://www.researchgate.net/publication/236896064\\_Self-Organizing\\_Traffic\\_Lights](https://www.researchgate.net/publication/236896064_Self-Organizing_Traffic_Lights). [cit. 2025-05-12].
- [6] HAMMOUD, M.; TANG, A. Y. C. a AHMAD, A. A Norms Decay Framework in Open Normative Multi-agent Systems. *British Journal of Applied Science & Technology* online, Leden 2016, sv. 12, č. 4, s. 1–15. Dostupné z: <https://doi.org/10.9734/BJAST/2016/21653>. [cit. 2025-04-10].
- [7] HÁJKOVÁ, V. *BDI Agenti: Úvod do umělé inteligence* online. Brno: [b.n.], 2012. Dostupné z: [https://nlp.fi.muni.cz/uui/referaty2012/veronika\\_hajkova/referat.pdf](https://nlp.fi.muni.cz/uui/referaty2012/veronika_hajkova/referat.pdf). [cit. 2024-11-12].
- [8] iROBOT. *iRobot Roomba i7: Robotický vysavač* online. 2018. Dostupné z: <https://www.irobot.cz/produkt/irobot-roomba-i7/>. [cit. 2024-12-11].
- [9] IVANOVA, I. *Roomba creator iRobot was once courted by Amazon for a vacuum revolution. Now, it might go out of business* online. 13. března 2025. Dostupné z: <https://fortune.com/2025/03/13/roomba-creator-irobot-amazon-going-concern-business-review/>.

- [10] JENNINGS, N. R.; SYCARA, K. a WOOLDRIDGE, M. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems* online, Březen 1998, sv. 1, č. 1, s. 7–38. Dostupné z: <https://doi.org/10.1023/A:1010090405266>.
- [11] KITANO, H.; TADOKORO, S.; NODA, I.; MATSUBARA, H.; TAKAHASHI, T. et al. RoboCup Rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research. In: *IEEE SMC'99 Conference Proceedings – 1999 IEEE International Conference on Systems, Man, and Cybernetics* online. Tokio, Japonsko: IEEE, říjen 1999, sv. 6, s. 739–743. ISBN 0-7803-5731-0. Dostupné z: <https://doi.org/10.1109/ICSMC.1999.816643>.
- [12] LAB, T. I. *JADE – Java Agent DEvelopment Framework* online. Květen 2022. Dostupné z: <https://jade.tilab.com/>. [cit. 2025-05-12].
- [13] LJUNGBERG, M. a LUCAS, A. The OASIS Air Traffic Management System. In: *Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence (PRICAI '92)* online. Seoul, Korea: [b.n.], Srpen 1992. ISBN 89-85368-00-093560. Dostupné z: <https://web-static.stern.nyu.edu/om/faculty/pinedo/book2/downloads/CMU-Salman/Reference%20Articles/oasis%20aircraft%20sequencing.html>.
- [14] LUKE, S.; CIOFFI, C.; PANAIT, L. a SULLIVAN, K. M. *MASON – A Fast Discrete-Event Multi-Agent Simulation Library* online. 2023. Dostupné z: <https://cs.gmu.edu/~eclab/projects/mason/>. [cit. 2025-05-02].
- [15] MITTAL, V. *Swarming Drones Will Be on the Russian–Ukrainian Battlefield in 2025* online. New York, USA: [b.n.], 2. ledna 2025. Dostupné z: <https://www.forbes.com/sites/vikrammittal/2025/01/02/swarming-drones-will-be-on-the-russian-ukrainian-battlefield-in-2025/>. [cit. 2025-02-14].
- [16] NATIONAL TRANSPORTATION SAFETY BOARD. *Preliminary Report – Highway HWY18MH010: Collision Between Vehicle Controlled by Developmental Automated Driving System and Pedestrian, Tempe, Arizona, March 18, 2018* online. Accident Investigation Preliminary Report. Washington, DC: National Transportation Safety Board, květen 2018. Dostupné z: <https://web.archive.org/web/20190831200841/https://www.ntsb.gov/investigations/AccidentReports/Reports/HWY18MH010-prelim.pdf>. [cit. 2025-01-10].
- [17] NO, A. P. C. Leveraging the Beliefs–Desires–Intentions Agent Architecture. *MSDN Magazine*, Leden 2019, sv. 34, č. 1. Dostupné z: <https://learn.microsoft.com/en-us/archive/msdn-magazine/2019/january/machine-learning-leveraging-the-beliefs-desires-intentions-agent-architecture>.
- [18] PROJECT, J. *Jason – A Development Platform for Multi-Agent Systems in AgentSpeak* online. 2025. Dostupné z: <https://jason-lang.github.io/>. [cit. 2025-05-12].
- [19] RAMACHANDRAN, A. *Comprehensive Methodologies and Metrics for Testing AI Agents* online. Duben 2024. Dostupné z: <https://www.linkedin.com/pulse/comprehensive-methodologies-metrics-testing-ai-agents-ramachandran-mkxne>. [cit. 2025-05-02].

- [20] RUSSELL, S. J. a NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3. vyd. Upper Saddle River, NJ: Pearson Education (Prentice Hall), prosinec 2010. ISBN 978-0-13-604259-4.
- [21] TEAM, G. D. *GAMA Platform Wiki* online. 2024. Dostupné z: <https://gama-platform.org/wiki/Home>. [cit. 2025-05-12].
- [22] TEAM, L. *Multi-agent architectures* online. 2025. Dostupné z: [https://langchain-ai.github.io/langgraph/concepts/multi\\_agent/#multi-agent-architectures](https://langchain-ai.github.io/langgraph/concepts/multi_agent/#multi-agent-architectures). [cit. 2025-02-02].
- [23] TEAM, S. D. *SPADE – Smart Python multi-Agent Development Environment* online. 2025. Dostupné z: <https://spade-mas.readthedocs.io/en/latest/readme.html>. [cit. 2025-05-12].
- [24] TECHNOLOGY, A. *Next Generation Air Dominance Programme, US* online. 08. března 2024. Dostupné z: <https://www.airforce-technology.com/projects/next-generation-air-dominance-programme-us/>. [cit. 2025-03-12].
- [25] TPOINT TECH. *Intelligent Agent in AI — Types of AI Agents*. Únor 2024. Dostupné z: <https://www.tpointtech.com/types-of-ai-agents>.
- [26] TRIBELHORN, B. a DODDS, Z. Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation* online. 2007, s. 1393–1399. ISBN 9781424406012. Dostupné z: <https://doi.org/10.1109/ROBOT.2007.363179>.
- [27] VERMA, V.; MAIMONE, M. W.; GAINES, D. M.; FRANCIS, R.; ESTLIN, T. A. et al. Autonomous robotics is driving Perseverance rover's progress on Mars. *Science Robotics* online, 2023, sv. 8, č. 80. Dostupné z: <https://doi.org/10.1126/scirobotics.adl3099>.
- [28] VIDENSKY, F.; ZBORIL, F.; KOCI, R. a ZBORIL, F. V. Advanced Evaluation of Variable Binding Strategies in BDI Agents with Integrated Failure Handling. In: ROCHA, A. P.; STEELS, L. a HERIK, J. van den, ed. *Agents and Artificial Intelligence* online. Cham: Springer Nature Switzerland, 2025, s. 3–29. ISBN 978-3-031-87327-0. Dostupné z: [https://doi.org/10.1007/978-3-031-87327-0\\_1](https://doi.org/10.1007/978-3-031-87327-0_1).
- [29] VUT-FIT-INTSYS. *FRAg – Flexibly Reasoning BDI Agent* online. 2025. Dostupné z: <https://github.com/VUT-FIT-INTSYS/FRAg/>. [cit. 2025-05-12]. Zdrojový kód projektu FRAg.
- [30] WEYN, D.; SCHUMACHER, M.; RICCI, A.; VIROLI, M. a HOLVOET, T. Environments in Multiagent Systems. *The Knowledge Engineering Review* online, Červen 2005, sv. 20, č. 2, s. 127–141. Dostupné z: <https://doi.org/10.1017/S0269888905000457>.
- [31] WOOLDRIDGE, M. *An Introduction to MultiAgent Systems*. 2. vyd. Chichester: John Wiley & Sons, 2009. ISBN 978-0-470-51946-2. Dostupné z: <https://www.wiley.com/en-us/An+Introduction+to+MultiAgent+Systems%2C+2nd+Edition-p-9780470519462>.
- [32] WOOLDRIDGE, M. a JENNINGS, N. R. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review* online, 1995, sv. 10, č. 2, s. 115–152. Dostupné z: <https://doi.org/10.1017/S0269888900008122>.

- [33] ZBOŘIL, F. *Plánování a komunikace v multiagentních systémech*. Brno, 2004.  
Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií.  
Dostupné z: <https://web.archive.org/web/20060310062154/http://www.fit.vutbr.cz/~zborilf/PhD/thesis.pdf>.
- [34] ZBOŘIL, F. *IZU – Přednáška 13: Úvod do agentních systémů*. Brno, Czech Republic: [b.n.], 2019.
- [35] ŽIŽKA, J. *Inteligentní agenti: Přednáška 1* online. Brno: [b.n.], 2004. Dostupné z: [https://is.muni.cz/el/1433/podzim2004/PA161/um/01\\_Inteligentni\\_agenti.pdf](https://is.muni.cz/el/1433/podzim2004/PA161/um/01_Inteligentni_agenti.pdf). [cit. 2025-05-12]. Slidy k předmětu PA161 Vybrané kapitoly z umělé inteligence.

## Příloha A

### Obsah SD karty

Přiložené médium obsahuje následující:

- thesis-latex/ – zdrojový kód této práce v L<sup>A</sup>T<sub>E</sub>X.
- thesis.pdf – pdf soubor práce
- src/ – adresář obsahující pouze soubory vzniklé touto prací.
- srcfrag/ – adresář obsahující soubory vzniklé touto prací, integrované do systému FRAg (starší verze, na které byl PES vyvýjen – pokud nefunguje PES na nové).
- src/documentation.md – soubor dokumentující systém PES.