VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INTELIGENTNÍ BOJOVÉ JEDNOTKY

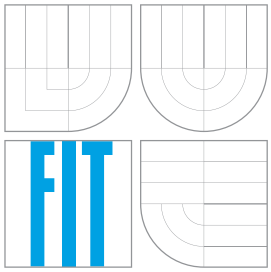
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN KUŽELA

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INTELIGENTNÍ BOJOVÉ JEDNOTKY

INTELLIGENT FIGHTING UNITS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN KUŽELA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARTIN DRAHANSKÝ, Ph.D.

BRNO 2008

Abstrakt

Výcvik vojenských jednotek v terénu je spojen s velkými náklady, ať už se jedná o finance, materiální nebo lidské zdroje. Proto se čím dál tím více klade důraz na výcvik bojových jednotek prostřednictvím simulátoru. Pro řádný výcvik je pak potřeba, aby se inteligence simulovaných jednotek co nejvíce podobala inteligenci lidské, aby mohla úspěšně nahradit lidského protivníka. Tato práce se zabývá návrhem postupu realizace inteligentního chování bojové jednotky, který bude aplikovatelný na prostředí simulátoru firmy E-COM s.r.o. Je zde obecně popsána problematika inteligentních agentů a způsobu dosažení jejich racionálního chování a autonomie. V této práci je také popsán a rozebrán návrh realizace inteligentní jednotky a její komunikace s okolním prostředím. Dále se zabývá základní implementací vytvořeného návrhu a nad ní provedenými experimenty.

Klíčová slova

Bojová jednotka, inteligentní agent, struktura agenta, racionalita, autonomie, užitek, agent orientovaný na cíle, plánování, báze znalostí, prostředí, učení

Abstract

The field training of army units includes high financial, material and human resource investments. From this reason, an emphasis on the simulator training of these units arised recently. But the training in simulator needs to have the simulated units as intelligent as a human beings are, so the field training with real human opponents can be successfully replaced with the simulator training. This work deals with the design of fighting unit's intelligent behaviour, that will be applicable in the E-COM simulator environment. Work covers the description of intelligent agents and ways how to achieve their rational and autonomous behaviour. The proposal and the analysis of intelligent fighting unit's implementation and unit's communication with surrounding environment, basic implementation of this proposal and experiments with created implementation are also described in this work.

Keywords

Fighting unit, intelligent agent, agent structure, rationality, autonomy, utility, goal-based agent, planning, knowledge base, environment, learning

Citace

Martin Kužela: Inteligentní bojové jednotky, diplomová práce, Brno, FIT VUT v Brně, 2008

Inteligentní bojové jednotky

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Ing. Martina Drahanského, Ph.D. Další informace mi poskytl Ing. Martin Hrubý, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Kužela
14. května 2008

Poděkování

Rád bych touto cestou poděkoval Ing. Martinu Drahanskému za vedení a podporu při psaní této práce a také Ing. Petru Janečkovi, ze společnosti E-COM s.r.o., za poskytnutý čas a rady, které mi věnoval při konzultacích.

Obsah

Kapitola 1

Úvod

Vysoké náklady na výcvik a trénink bojových jednotek si žádají nalezení levnější alternativy. Možnost simulace bojových situací a trénink bojových jednotek prostřednictvím počítače je jednou z těchto alternativ, které se v dnešní době, a také v dobách minulých, objevují. Výcvik s využitím počítačové simulace má oproti výcviku v reálném prostředí nemálo výhod. Představuje úsporu nejenom financí, ale také přírodních a lidských zdrojů, nemá negativní ekologický dopad, bojové situace v simulátoru jsou opakovatelné (čehož v reálném prostředí těžko docílíme) a lze je provést prakticky kdykoliv. Například pro výcvik jednoho pilota stíhačky je zapotřebí několika dalších pilotů, stíhaček a paliva, aby byl tento výcvik efektivní. Počítačová simulace toto umožňuje bez potřeby zaměstnat tak velké množství lidí a také bez výrazné materiální spotřeby.

Tvorba kvalitního simulátoru sebou nese potřebu docílit co nejrealističtější simulace bojových situací. Jedna z oblastí na které je kladen důraz, je inteligence jednotek, které se v prostředí simulátoru vyskytují. Chování těchto jednotek by mělo být podobné chování člověka.

1.1 Cíl práce

Cílem práce je nastudovat literaturu zaměřenou na umělou inteligenci bojových jednotek a vytvořit návrh modulu umělé inteligence, který bude využitelný v simulátorech společnosti E-COM s.r.o.

Simulátor společnosti E-COM s.r.o. je využíván k vojenským účelům, k tréninku bojových jednotek. V prostředí simulátoru se pohybují vojenské jednotky a člověk dovnitř vstupuje podobně jako u akčních her, avšak s tím rozdílem, že místo tomuto účelu typických periferních zařízení (klávesnice, myš, joystick, gamepad) ovlivňuje prostředí simulátoru pomocí speciálních zařízení, která umožňují efektivnější výcvik. Těmito speciálními zařízeními jsou přitom myšleny nejrůznější makety zbraní, kokpitů tanků a jiných zařízení.

Když k simulátoru přistoupí jedinec, aby byl proveden trénink, je vybrán a spuštěn skript (tzv. "scénář") ovlivňující chování jednotek. Tento skript přímo řídí jednotlivé entity v prostředí. Nemají tedy žádnou možnost přemýšlet o svých akcích, měnit množinu jim přiřazených příkazů, maximálně může skript definovat reakci na nějakou událost (pokud někdo střílí, schovej se a braň se). Toto je ve svém principu pro samotný trénink vojenských jednotek výhodou, protože se jednotky v prostředí vždy chovají jednoznačně a trenér může mnohem objektivněji posuzovat jednání vojáků v dané situaci (všichni mají stejné podmínky). V simulátoru by však bylo také žádoucí, aby bylo možné jednotky

trénovat i v prostředí, které neznají, a v rozsahu větším než je prakticky možné pomocí skriptu postihnout (psaní skriptu pro jednu tréninkovou situaci je náročné a ne vždy se dá pamatovat na definici všech správných akcí pro všechny entity).

Hlavním cílem práce je navrhnout modul umělé inteligence určující chování bojových jednotek v něm. Jednotky by měly být schopny vyhodnotit současnou situaci a podle jim přiděleného úkolu se samy rozhodnout, jakou akci podniknout, aby dosáhly stanoveného cíle. Navržený modul by měl zároveň zjednodušit práci při vytváření scénářů tak, aby tvůrce scénáře nemusel jednotce zadávat pokyny do nejzazších podrobností, ale mohl se spolehnout, že již sama bude schopna splnit obecněji zadané úkoly.

Dalším cílem je provést základní implementaci vytvořeného návrhu, nad kterou bude možné provádět experimenty a ověřit tak jeho správnost. Posledním cílem je provedení experimentů nad touto implementací a zhodnocení dosažených výsledků.

1.2 Struktura práce

Úvodní kapitola byla konzultována se zastupiteli společnosti E-COM s.r.o. Další kapitola (2) této práce tvoří stručný úvod do umělé inteligence. V této kapitole jsem čerpal z bakalářské práce [3] a knihy [9]. Kapitola 3 obsahuje teoretická východiska důležitá pro správný návrh inteligentního agenta, kapitola 4 pak již analyzuje konkrétní téma této práce a rozebírá skutečnosti důležité při samotném návrhu zadané inteligentní jednotky. V těchto kapitolách jsem čerpal z děl [9, 7, 5, 10, 11]. V kapitole 4 jsem se navíc inspiroval díly [1, 4, 6, 8]. Kapitola 5 se zabývá popisem vytvořené implementace a experimenty nad ní provedené jsou sepsány v kapitole 6. Poslední kapitola 7 popisuje zhodnocení diplomové práce a možnosti jejího rozšíření do budoucna.

Tato práce navazuje na semestrální projekt, v rámci kterého byla nastudována zmíněná literatura a byl proveden základní návrh modulu umělé inteligence. V rámci diplomové práce byl tento návrh rozšířen, implementován a ověřen pomocí experimentů. Všechny tyto úkony odpovídají změnám v kapitole 4 a přidáním kapitol 5 a 6. Kapitoly 2 a 3 byly převzaty beze změny.

Kapitola 2

Umělá inteligence

> *"Inteligence je vlastností některých živých organismů, která jim dává v přírodě mimořádné postavení. Vznikla a vyvíjela se v průběhu dlouhého časového intervalu a dnes umožňuje některým živým organismům efektivně reagovat na složité projevy prostředí a aktivně je využívat ve svůj prospěch a k dosažení svých cílů." (Mikulecký, Ponce, 1996)*

S neustálým rozvojem techniky vzniká otázka, zda se člověku podaří uměle vytvořit systém, který by se choval a reagoval inteligentně, a byl tedy srovnatelný s živými a inteligentními organismy. Neustále se objevují a experimentálně se ověřují algoritmy, metody a postupy, které se toto inteligentní chování snaží napodobovat, alespoň v určitých jeho aspektech. Jsou k tomu využívány nejrůznější techniky vycházející z analýzy činnosti živých organismů (neuronové sítě, genetické algoritmy), z matematické abstrakce mentálních procesů lidského mozku (metody reprezentace a využívání znalostí ve stavovém prostoru, metody učení založené na modelech) a dalších oblastí. Všechny tyto postupy a algoritmy, které vedou k určitému napodobení projevů inteligentního člověka, jsou předmětem zkoumání poměrně mladé vědní disciplíny – **umělé inteligence**.

Oblast umělé inteligence se pokouší pochopit inteligentní entity a snaží se je napodobit. Jedním z jejích cílů je vytvořit takovou umělou entitu, která by projevovala známky inteligence. Pojem "inteligence" u živých organismů nebyl nikdy přesně vymezen, a proto ani pojem "umělá inteligence" nebyl doposud přesně definován. **Inteligenci** můžeme chápat například jako míru podobnosti **chování** dané entity k tomu, které bychom očekávali od člověka, nebo schopnost jednat **racionálně** – pak entita provádí takové akce, které jsou z jejího pohledu správné a vedou k dosažení jejích cílů. Definic umělé inteligence je nespočetně. Zde jsou některé z nich:

> *"Umělá inteligence je věda o vytváření strojů nebo systémů, které budou při řešení určitého úkolu užívat takového postupu, který – kdyby ho dělal člověk – bychom považovali za projev jeho inteligence." (Minsky, 1967)*

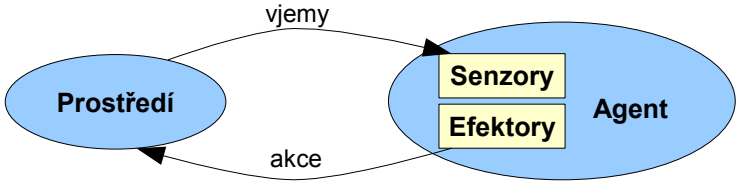
> *"Umělá inteligence je vlastnost člověkem uměle vytvořených systémů vyznačujících se schopností rozpoznávat předměty, jevy a situace, analyzovat vztahy mezi nimi a tak vytvářet **vnitřní modely světa**, ve kterých tyto systémy existují, a na tomto základě pak přijímat účelná rozhodnutí, za pomoci schopností předvídat důsledky těchto rozhodnutí a objevovat nové zákonitosti mezi různými modely nebo jejich skupinami" (Kotek a kol., 1983)*

Kapitola 3

Inteligentní agent

Jedním z cílů umělé inteligence je popsat a sestavit agenta, který bude z okolního prostředí získávat vjemy a bude toto prostředí ovlivňovat pomocí svých akcí, přičemž bude projevovat známky inteligence.

Agenta [9, 10] můžeme obecně chápat jako objekt zasazený do určitého prostředí, který je schopen vnímat své okolí pomocí senzorů a zasahovat do okolního prostředí pomocí svých efektorů (viz obrázek 3.1). Od ostatních objektů jej přitom odlišuje schopnost v tomto prostředí jednat samostatně. Schopnost samostatného jednání v prostředí nazýváme **autonomií agenta** [9, 10].



Obrázek 3.1: K interakci agenta s prostředím dochází skrze senzory a efektory agenta [9].

Racionální agent [9, 10] je takový agent, který dělá věci, které jsou z jeho pohledu správné. Správnou věc můžeme definovat jako **akci**, která umožní agentovi být co nejúspěšnějším – je mu nejvíce prospěšná.

Úspěch agenta v daném prostředí se odvíjí na základě námi stanovených kritérií. Na jejich základě pak můžeme zjišťovat **míru úspěšnosti** daného agenta podle toho, jak moc je schopen tato kritéria splňovat. Kritéria musí definovat nejen situace, kdy považovat agenta za úspěšného, ale také takové situace, kdy chování agenta skončilo neúspěchem. Musí být dostatečně objektivní, aby nepostihovala agenta za situace, které sám nemohl ovlivnit a zároveň dost obecná na to, aby agenta svou definicí nevedla k přílišné pasivitě. Jinak může dojít k situaci, kdy agent neprovádí žádnou akci, jen aby se vyhnul hrozbě snížení jeho dosavadní úspěšnosti. Vyhneme se tomu tak, že zavedeme trest za nedokončení úkolu, který bude výraznější než dosavadní agentův zisk. Agent se potom snaží vyhnout potrestání a nebude stagnovat. Případně můžeme hodnotit agentovu aktivitu v čase (zvýhodníme tak stále pracující agenty oproti nárazově pracujícím).

Racionalita [9, 10] agenta bere v úvahu *očekávaný* úspěch na základě *současných* znalostí a současných možností. Iracionalita nevyplývá z případného neúspěchu, pokud nastala událost, kterou agent nemohl tušit a proto nemohl neúspěchu zabránit. Racionalitu

ovlivňuje **míra úspěšnosti agenta**, **vjemy** které doposud získal, jeho **znalosti** o daném prostředí a **akce**, které může provést.

Ideální racionální agent [9] je pro každou množinu vjemů schopen provést takovou množinu akcí, která maximalizuje jeho míru úspěšnosti a to na základě informací, které jsou mu touto množinou vjemů poskytnuty a na základě získaných a zabudovaných znalostí, které agent má.

Takový agent se snaží **získávat užitečné informace** ze svého prostředí, aby tak maximalizoval očekávanou míru užitku a snížil riziko vykonávané akce.

Mapování [9] získaných vjemů na provedené akce popisuje agenta pomocí definice jeho chování. **Ideální mapování** popisuje ideálního agenta. Specifikace akcí, které by měl agent vykonat jako odpověď na danou sekvenci vjemů, poskytují návrh ideálního agenta.

Ideální racionální agent by neměl postrádat **autonomii**. Jeho chování by proto nemělo být založeno pouze na zabudovaných znalostech, ale měl by být schopen jednat i na základě svých vlastních zkušeností. Systém je autonomní do takové míry, do jaké je jeho chování určeno jeho vlastními zkušenostmi. Důležité je přitom nalézt rozumnou míru mezi počátečními znalostmi, které jsou agentovi vloženy a požadovanou mírou autonomie.

3.1 Struktura agenta

Úkolem umělé inteligence je navrhnout **program agenta** – funkci implementující mapování agentových vjemů na akce.

Algoritmus 1 Základní kostra agenta. Při každém volání je agentova paměť upravena tak, aby odrážela nově získaný vjem z prostředí. Je zvolena nejlepší akce a její volba, a provedení je promítnuto do paměti. Paměť setrvává do dalšího volání [9].

```
function KostraAgenta(vjem)
    static: paměť                          ▷ Agentova paměť o stavu světa
    paměť ← AktualizujPaměť(paměť,vjem)
    akce ← ZvolNejlepšíAkci(paměť)
    paměť ← AktualizujPaměť(paměť,akce)
    return akce
end function
```

Podle struktury programu rozlišujeme několik typů agentů:

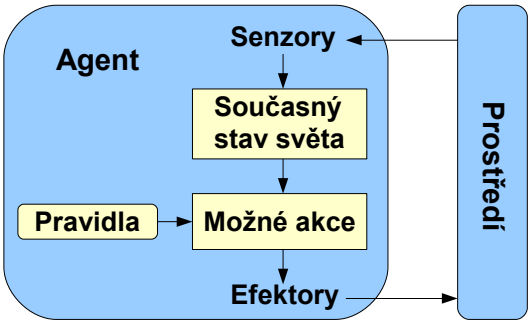
3.1.1 Agent založený na vyhledávací tabulce

Takový agent není autonomní, mapování je zcela definováno vyhledávací tabulkou, která určuje jakou akci má agent podniknout v závislosti na dané vstupní množině vjemů. Reakce na vjemy jsou tedy tímto mapováním pevně určeny. Tabulku pro tento typ agenta je náročné vybudovat, jelikož musí postihnout všechny případy, které mohou v prostředí nastat, v opačném případě nebude agent schopen správně fungovat. Učení takového agenta může být nekonečné, protože trvá dlouho zjistit správnou hodnotu pro všechny záznamy tabulky.

3.1.2 Jednoduchý reflexní agent

Tato struktura vychází z poznání, že lidé nemají všechny akce předem definované, ale přesto mají podmíněné a nepodmíněné reflexy, které definují reakce na vybrané události. U agenta

je vybrána množina nejběžnějších vstupně/výstupních asociací. K těmto asociacím jsou definována pravidla, která mapují vjemy na akce, podobným způsobem jako tomu bylo u tabulkou řízeného agenta. Rozdíl spočívá ve vytvoření pravidel na vyšší úrovni než tomu bylo u agenta s vyhledávací tabulkou. Takový agent je efektivněji programovatelný, ale možnosti jeho použití jsou stále velmi úzké.



Obrázek 3.2: Schéma jednoduchého reflexního agenta [9].

Algoritmus 2 Jednoduchý reflexní agent. Na základě vjemů určí současný stav okolního prostředí a vybere první pravidlo, které tomuto stavu odpovídá. Následně provede akci asociovanou s tímto pravidlem [9].

```
function ReflexníAgent(vjem)
    static: pravidla                          ▷ Množina pravidel vjemů na akce
    stav ← VstupZeSenzorů(vjem)
    pravidlo ← VyhovujícíPravidlo(stav,pravidla)
    akce ← VyberAkciZPravidla(pravidlo)
    return akce
end function
```

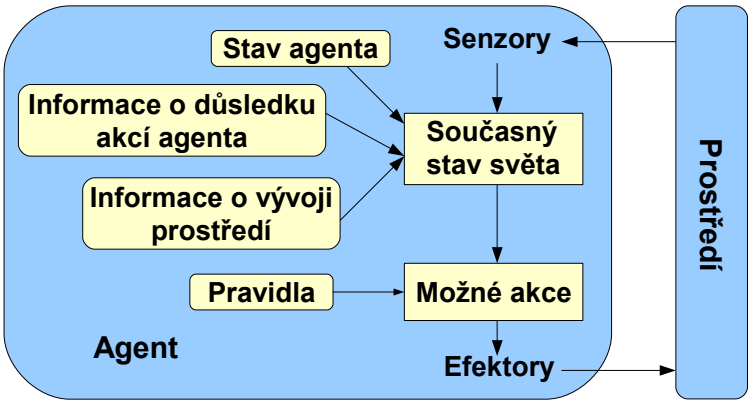
3.1.3 Agent udržující si informace o světě

Jednoduchý reflexní agent je využitelný jen v případech, kdy se dokáže správně rozhodovat pouze na základě získaného vjemu. Přidáme-li však takovému agentu vnitřní stavy a informaci o tom, ve kterém stavu se agent právě nachází, pomůže nám tato informace lépe zvolit správnou akci k vykonání. Pokud získáme stejný vjem ve dvou různých situacích, pro něž existuje různé pravidlo mapující vjem na akci, pomůže nám informace o vnitřním stavu zvolit tu správnou akci v současném stavu světa. Kromě této informace je navíc potřeba mít informace o způsobu, jakým se svět kolem agenta vyvíjí, nezávisle na jeho chování (např. jakým způsobem se v něm pohybují další jednotky), a také informace o tom, jak agentovy akce tento svět ovlivňují.

Stav takového agenta je závislý nejenom na vjemu z prostředí, ale také na předchozím stavu a takto získaný nový stav určuje zvolenou akci k vykonání.

3.1.4 Agent orientovaný na cíle

Ani informace o současném stavu prostředí nám vždy nemusí dostatečně pomoci správně rozhodnout, jakou akci právě vykonat. Někdy totiž můžeme mít v současném stavu s danými



Obrázek 3.3: Reflexní agent s vnitřním stavem [9].

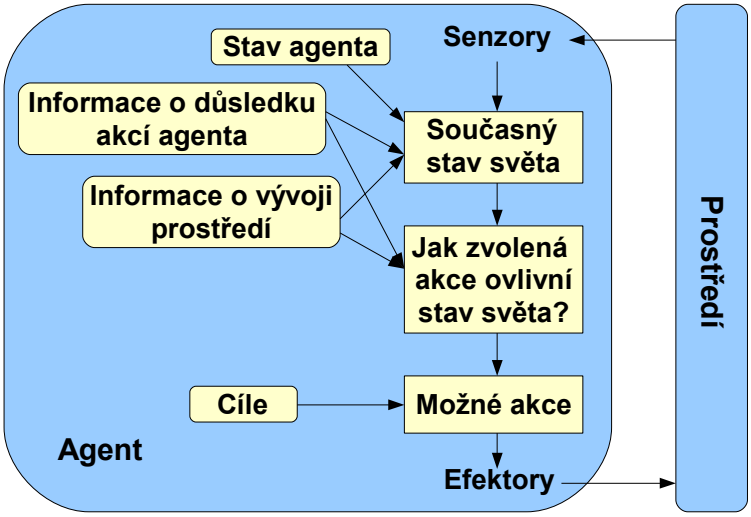
Algoritmus 3 Reflexní agent s vnitřním stavem. Nalezne pravidlo, jehož podmínka odpovídá současné situaci (definované stavem a vjemem) a potom vykoná akci asociovanou se zvoleným pravidlem [9].

function ReflexníAgentSVnitřnímStavem(*vjem*)
  static: *stav* ▷ Popis současného stavu světa
  static: *pravidla* ▷ Množina pravidel vjemů na akce
  stav ← AktualizujStav(*stav, vjem*)
  pravidlo ← VyhovujícíPravidlo(*stav,pravidla*)
  akce ← VyberAkciZPravidla(*pravidlo*)
  stav ← AktualizujStav(*stav, akce*)
  return *akce*
end function

vjemy několik zdánlivě správných akcí k vykonání a jejich správnost je určena **cílem** agenta (například můžeme odbočit doleva nebo doprava, ale správný směr nám určuje cíl, kam se chceme dostat). Chceme-li zvýšit rozhodovací schopnost agenta, musíme mu určit **cíl**, kterého má dosáhnout, a který může kombinovat s informacemi o výsledcích možných akcí, aby zvolil správné akce, které mu pomůžou dosáhnout stanoveného cíle. Někdy je volba akce k dosažení cíle jednoduchá, jindy zahrnuje spoustu **hledání** a **plánování**.

Tento způsob rozhodování již nutí agenta přemýšlet do budoucnosti – co se stane, když vykoná tuto akci? Reflexní agent jenom využíval zabudovaných zkušeností návrháře (např. když se rozsvítí brzdová světla auta, tak zpomal), agent orientovaný na cíle předpokládá, že daný vjem něco znamená a má dopad na jeho akce (např. se dovtípí, že rozsvícená brzdová světla auta před ním znamenají, že toto auto zpomalí, proto musí na toto zpomalení zareagovat, aby nedošlo ke srážce. Protože jedinou akci, která mu umožní tomu zabránit je šlápnout na brzdu, tak začne brzdit). Přestože přemýšlení o tom jakou akci vykonat, namísto jejího okamžitého vykonání, se zdá být méně efektivním, je na druhou stranu značně flexibilnějším. Pokud se změní stav prostředí (např. bude mokrá vozovka), u reflexního agenta bychom museli přepsat spoustu pravidel (aby začal dřív brzdit, udržoval větší bezpečnou vzdálenost, apod.), zatímco agent orientovaný na cíle se sám přizpůsobí (ví, že je schopnost jeho brzdění zhoršená, proto bude udržovat větší vzdálenosti od před ním jedoucího vozidla). Také bude flexibilní při změně cíle, protože je sám schopen naplánovat, jak to-

hoto nového cíle dosáhnout, zatímco pravidla jednoduchého reflexního agenta mu umožňují dosáhnout jediného, předem definovaného cíle a musí být upravena, pokud chceme tento cíl změnit.



Obrázek 3.4: Agent s explicitně uvedenými cíli [9].

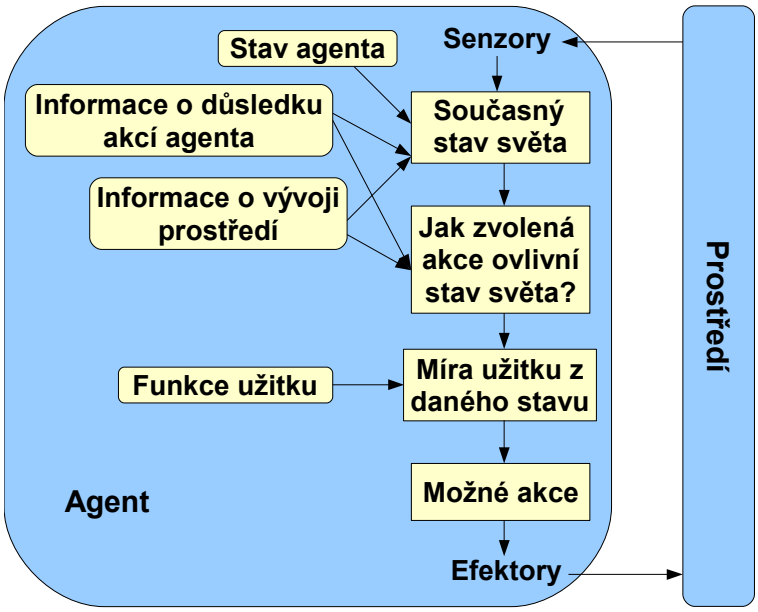
3.1.5 Agent orientovaný na užitek

Ani definování cíle není samo o sobě postačující pro generování vysoce kvalitního chování. Může totiž existovat několik sekvencí akcí, které agentovi umožní dosáhnout stanoveného cíle, ale některé z nich mohou být výhodnější, méně namáhavější, či rychlejší než jiné. Cíle nám pouze rozdělují stavy agenta na ty, které nám umožní dosáhnout stanoveného cíle a na ty, které nám ho dosáhnout neumožní. Proto ještě u jednotlivých sekvencí stavů, které agent naplánuje určujeme, jaký **užitek** pro něj bude mít vykonání právě této sekvence akcí a procházení touto sekvencí stavů.

Užitek je funkce mapující stav nebo sekvenci stavů na reálné číslo. Toto číslo určuje zisk agenta z provedení takové sekvence stavů. Užitek nám pomáhá vyřešit situaci, kdy máme více konfliktních cílů, kde nemůže být všech dosaženo (např. nestihnu během 10 minut nakoupit a zároveň přijít včas do práce) tím, že nám pomůže vybrat takové, které nám přinesou největší užitek. Také můžeme mít několik cílů, kterých může chtít agent dosáhnout, ale žádný z nich nemůže být zaručeně dosažen. V takovém případě nám užitek pomáhá vážit předpoklad splnitelnosti daných cílů s důležitostí jejich dosažení.

3.2 Prostředí

Agentní systém [9, 10] je tvořen nejenom agenty samotnými, ale také prostředím, v němž se tito agenti nacházejí. Agenti se v tomto prostředí pohybují, komunikují s ním a zasahují do něj, prostředí je tedy důležitou součástí tohoto systému. Proto je při návrhu agenta důležité znát, v jakém prostředí se bude agent přesně pohybovat. Různé typy prostředí vyžadují různé struktury programu agenta, aby se s nimi dokázal efektivně vypořádat. Správný návrh



Obrázek 3.5: Agent orientovaný na užitek [9].

agenta by však měl být proveden tak, aby byl agent použitelný na co největším množství prostředí. Podle následujících vlastností můžeme rozlišit několik druhů prostředí [9, 10]:

Přístupné vs. nepřístupné – Prostředí se nazývá **přístupné** agentovi, pokud mu jeho senzory poskytují kompletní informaci o stavu tohoto prostředí. Prostředí se nazývá **efektivně přístupné**, pokud senzory zachycují všechny aspekty relevantní pro volbu správné akce. Přístupné prostředí je výhodné, protože si agent nemusí uchovávat vnitřní informaci o stavu prostředí.

Deterministické vs. nedeterministické – Prostředí se nazývá **deterministické**, pokud je jeho následující stav vždy určen stavem současným a akcí, kterou na něm agent provádí.

Epizodické vs. neepizodické – Prostředí nazveme **epizodickým**, pokud je agentova zkušenost rozdělena do tzv. **epizod**. **Epizoda** sestává z agentova získání vjemů a následného konání akcí. Tyto akce přitom ovlivňují pouze současnou epizodu, takže předchozí epizody nejsou již dále závislé na akcích, které se vyskytnou v epizodě současné. V takovém prostředí mizí potřeba agenta myslet dopředu.

Statické vs. dynamické – Prostředí nazveme **dynamickým**, pokud se může změnit v průběhu agentova rozhodování. V opačném případě řekneme, že prostředí je **statické**. Prostředí nazveme **semidynamickým**, pokud se v čase nemění, ale mění se jeho **funkce užitku** pro agenta.

Diskrétní vs. spojité – Pokud existuje konečné množství oddělených, jasně definovaných vjemů a akcí, říkáme, že je prostředí **diskrétní**. Jinak je prostředí **spojité**.

Nejobtížnějším je nepřístupné, neepizodické, dynamické a zároveň spojité prostředí.

3.2.1 Simulátor prostředí

Simulátor prostředí má jako vstup jednoho nebo více agentů. Simulátor těmto agentům poskytuje správné vjemy a zpětně od nich získává akce, které se agenti rozhodli provést.

V závislosti na získaných akcích následně aktualizuje svůj stav. Prostředí je tedy definováno svým **počátečním stavem** a **aktualizační funkcí**.

Algoritmus 4 Jednoduchý simulátor prostředí, který každému agentu poskytne požadované vjemy, poté vyčká jaké akce se na něm agent rozhodne vykonat, a podle těchto akcí aktualizuje stav prostředí [9].

```
function SimulátorProstředí(vjem)
    static: stav                                  ▷ Počáteční stav prostředí
    static: agenti                                ▷ Množina agentů
    static: konec                                 ▷ Predikát určující, kdy ukončit činnost
    repeat
        for all agent in agenti do
            PoleVjemů[agent] ← ZískejVjem(agent, stav)
        end for
        for all agent in agenti do
            PoleAkcí[agent] ← Mapování[agent](PoleVjemů[agent])
        end for
        stav ← AktualizujProstředí(akce, agenti, stav)
    until konec
end function
```

Při tvorbě simulátoru prostředí nesmí dojít k tomu, aby byl stav prostředí přístupný agentovi přímo! Vnitřní stav agenta musí být vytvořen pouze z jeho vlastních vjemů, bez možnosti přístupu ke kompletní informaci o stavu prostředí, protože v reálné aplikaci mu není možné tento stav prostředí prakticky zpřístupnit.

Kapitola 4

Návrh modulu umělé inteligence

4.1 Základní požadavky

Základním požadavkem je návrh implementace modulu umělé inteligence v simulátoru společnosti E-COM s.r.o. Entitám v simulátoru by mělo být umožněno inteligentní chování, jednotky by měly být schopny samy od sebe rozhodovat, jakou akci provedou a to primárně v těchto oblastech:

- **Výběr cíle** – jednotka by měla být schopna vyhodnotit, jestli je v oblasti jejího působení přítomen protivník. Pokud takovou informaci má, měla by být schopna vybrat nejvhodnější cíl pro vykonání jí definovaných akcí (eliminace protivníka, jeho pozorování apod.)
- **Výběr munice** – jednotka by měla být schopna řešit otázku: "Jak bude bojovat proti vybranému protivníkovi?". Tento problém zahrnuje volbu vhodného typu munice, účinného na daný typ jednotky (například průrazná munice na opancéřované jednotky)
- **Výběr akce** – jednotka by vždy měla být schopna určit, jakou akci v dané situaci vykonat. Výběr akce může být ovlivněn jak funkcí jednotky, tak rozkazy, které jí byly uděleny. Přitom si její funkce a udělené rozkazy mohou protiřečit (obranné jednotce bude rozkázáno opustit své stanoviště a zaútočit na protivníka, dělostřelecký pluk bude situací nucen k nasazení jako pěchota).
- **Autonomní pohyb terénem** – jednotka by měla být schopna přemístit se inteligentně na požadovanou pozici (například by neměla probíhat místem, které je pod palbou, nýbrž se přesunovat krytá podél zdí).

Umělou inteligenci v tomto systému přitom můžeme vnímat ze dvou pohledů:

První z těchto pohledů vnímá implementaci umělé inteligence jako tvorbu autonomní jednotky, která vpuštěna do prostředí simulátoru, s předem definovaným cílem, dokáže samostatně fungovat a tento svůj cíl plnit. Jednotka se bude snažit svého cíle dosáhnout, případně se v prostředí pohybovat se známkami inteligence. Taková jednotka se v prostředí bude pohybovat zcela autonomně bez potřeby toho, aby byla v průběhu své činnosti jakkoliv ovlivňována.

Další z těchto pohledů vnímá tento úkol jako tvorbu autonomní jednotky, která však bude částečně řízena člověkem (její cíle budou upravovány během jejího působení v prostředí). V tomto případě může být její inteligence uměle potlačována člověkem. Můžeme

například tanku přikázat, aby se přesunul na novou pozici a během své cesty nebojoval. Když se tak dostane do nepřátelské palby, nebude ji opětovat, přestože by to bylo jeho přirozeností, nýbrž bude pokračovat ve splnění jemu přiděleného cíle. Po splnění tohoto cíle může být jednotce řečeno, že má v nově dosaženém stavu prostředí působit autonomně za účelem splnění nějakého obecnějšího cíle (např. ubránit pevnost).

Dalším požadavkem na výsledný systém je, aby v konečném důsledku umožnil jednodušší práci při vytváření scénářů. Scénář je skript, který v současné době určuje chování jednotek v simulátoru (viz kapitola 1.1). Zjednodušit by se měla práce tvůrce scénáře, který by mohl spoléhat na určité schopnosti jednotky a nemusel by tak její chování definovat do nejmenších detailů. Například u simulátoru pro střelbu z ručních zbraní by scénář definoval pouze několik bodů trasy jednotlivým jednotkám, které se přibližují směrem ke studentovi, ale už by jednotkám nemusel definovat jejich chování při ohrožení střelbou a mohl se spolehnout na to, že každá jednotka se už sama ze své podstaty bude schopna rozhodnout, jestli bude opětovat palbu, nebo se ukryje (případně také jak se toto ukrytí provede – přikrčeným během, plížením apod.).

4.2 Analýza problému

Cílem je vytvořit inteligentní bojovou jednotku. Tato jednotka se nebude vyskytovat v prostředí sama, nýbrž se zde budou pohybovat i ostatní jednotky. Některé z těchto jednotek budou na její straně, může s nimi kooperovat. Jiné budou jejími nepřáteli, tyto musí eliminovat, případně se jim vyhýbat, podle konkrétního cíle, který jí bude zadán. Mimoto se v prostředí ještě bude vyskytovat speciální jednotka a tou bude člověk sám, který do tohoto prostředí vstupuje.

Tím před námi vyvstává několik problémů, které je potřeba vzít v úvahu – jak budou na sebe jednotky vzájemně reagovat, jak bude probíhat jejich spolupráce, a jak vlastně budeme samotné jednotky v prostředí reprezentovat, aby spolu mohly komunikovat.

Začneme ale prvotní jednoduchou úvahou: Co kdyby se jednotka vyskytovala v prostředí sama a její okolní prostředí by bylo statické – měnilo by se pouze v závislosti na akcích této jednotky? V tomto případě by bylo postačující, aby jednotka byla schopna zjistit současnou situaci, ve které se nachází, zvážit své možnosti vzhledem k jí definovaným cílům a začít tyto cíle plnit.

Jak ale rozhodnout o tom, který z těchto cílů splnit jako první? A jak vytvořit postup, kterým může tohoto cíle efektivně dosáhnout? Člověk je schopen plán dosažení svého cíle jednoduše vytvořit, ale plány různých lidí se od sebe dostatečně liší na to, abychom byli schopni prohlásit, že takové řešení není jednoznačné. Jak tedy u počítačem ovládané entity dosáhnout vytvoření plánu, který by pomohl dosáhnout zvolení toho nejvýhodnějšího cíle a jak tento cíl vůbec zvolit? V první řadě je tedy potřeba se zabývat návrhem **plánování** jednotky.

Pokud má být jednotka schopna sestavit úspěšný plán, potřebuje k tomu znalosti o okolním prostředí. Teprve s pomocí těchto znalostí může zjistit svoji současnou situaci a z ní při sestavování plánu vycházet. Je tedy třeba také vyřešit jak tyto **znalosti získat, ukládat** a **pracovat** s nimi.

Předpokládejme, že jednotka již má znalosti o svém prostředí a je schopna na jejich základě vygenerovat plán. Zbývá nám tento plán krok po kroku provést, případně obecný krok plánu rozložit do více kroků konkrétního provedení.

Rozplánování obecného kroku do kroků jednoduchých může obhospodařit buď plánovací centrum samotné, nebo může být od definice těchto jednoduchých kroků oproštěno a tyto

mohou být na pokyn plánovacího centra vytvořeny až v příslušných centrech, s nimiž je plánovací centrum provázáno. Například může plánovací centrum vytvořit plán, v němž se vyskytuje příkaz "jdi 50 metrů na severovýchod". Buďto přímo plánovací centrum může blíže specifikovat tento příkaz pomocí příkazů "posuň se o krok na východ", "posuň se o krok na sever",... nebo může obecný příkaz předat podsystému pro řízení pohybu a ten už se postará o jeho konkrétní provedení. Za tímto účelem je třeba navrhnout konkrétní řešení **zpracování** a **provedení vytvořeného plánu**, případně definovat moduly, které se na zpracování budou podílet.

Nyní již můžeme předpokládat, že jednotka umí získat znalosti o svém prostředí. Ze současného stavu prostředí je také schopna zjistit nejvýhodnější proveditelný cíl, naplánovat jeho dosažení a dosáhnout provedení celého plánu. Je tedy prakticky schopna dosahovat svých cílů v daném statickém prostředí, kde statickým myslíme takové prostředí, které se mění pouze v závislosti na zásahu jednotky do tohoto prostředí.

Co když ale bude prostředí ovlivňováno i akcemi jiných jednotek? Potom se nám návrh jednotky zkomplikuje, protože již musí počítat s tím, že se prostředí může změnit i bez toho, že by jednotka sama do tohoto prostředí jakkoliv zasáhla. Může tak dojít ke znemožnění vytvořeného plánu ve chvíli, kdy se prostředí změní natolik, že jej již není možné provést, a také jednotka může počítat ve svém plánu se skutečností, která, dříve než je vykonání plánu dokončeno, zanikne a tím naruší proveditelnost tohoto plánu. Proto je také potřeba vyřešit schopnost jednotky **plánovat v dynamicky se měnícím prostředí**, kde pojmem "dynamické prostředí" myslíme právě takové prostředí, jehož stav se mění, aniž bychom do něj sami zasahovali.

Co když mají ostatní jednotky působící v okolním prostředí jiné zájmy než námi sledovaná jednotka? Co když je jejich úkolem dosahovat takových cílů, které cíle naší jednotky přímo nebo nepřímo poškozují? Potom by měla být námi sledovaná jednotka schopna "přechytračit" tyto jednotky, a dosáhnout tak největšího zisku za daných možností. A to i kdyby měla některé ze svých cílů obětovat (samozřejmě jen pokud jí to umožní dosáhnout cíle pro ni výhodnějšího). Je potřeba strategicky přemýšlet nad akcemi ostatních jednotek, volit akce, které budou v dané situaci nejvýhodnější i v případě, že budou ostatní jednotky nekontrolovaně zasahovat do okolního prostředí. Za tímto účelem je potřeba navrhnout také **strategické myšlení** jednotky.

Je téměř jasné, že spousta akcí, které jednotka bude volit, jejich výhodnost pro ni a informace o tom, které akce budou vést ke splnění jejích cílů a které ne, budou z počátku hodně ovlivněny člověkem, který jí je buď zadá, nebo alespoň naznačí, kterým směrem se ubírat, stejně jako nás v dětství vedly jiné osoby, abychom dokázali dosahovat cílů našich. Když ještě chvíli zůstaneme v této paralele, také my jsme se v průběhu dospívání dostali do situací, v nichž jsme se dříve neocitli, a byli jsme nuceni sami nalézt řešení a případně se učit z našich chyb, pokud námi zvolené řešení nemělo takový dopad, v jaký jsme původně doufali. Stejně tak, jako se člověk v životě učí, by bylo velmi žádané, aby se navržený systém dokázal učit z toho, co se v něm děje a upravovat jednání jednotek v něm tak, aby v ideálním případě spělo jejich chování v daných situacích k co největší dokonalosti. Bylo by tedy vhodné implementovat i **učení** takového systému, případně také **učení** jednotek samotných.

V neposlední řadě, z důvodu robustnosti systému společnosti E-COM s.r.o., je potřeba také vytvořit vlastní prostředí, které bude simulovat v něm se pohybující a jednající jednotky. Na tomto prostředí by mělo být dobře viditelné, jaké akce jednotky volí, jak probíhá jejich plánování a jak dobře vlastně doposud implementovaný postup funguje. Návrh tohoto prostředí bude rozebrán v kapitole 4.5.

4.3 Inteligentní bojová jednotka

Inteligentní bojovou jednotku můžeme vnímat jednoduše jako inteligentního agenta, jehož cíle se vztahují k působení na bitevním poli, v našem případě v simulátoru. Inteligentní bojová jednotka v tomto simulátoru bude mít určen cíl, popřípadě více cílů, kterých má dosáhnout. Aby se dokázala rozhodovat mezi více cíli, a také aby dokázala zvolit, jaká strategie dosažení daného cíle bude pro ni nejvýhodnější, je důležité, aby měla definovánu také funkci ohodnocující jednotlivé strategie, tedy **míru užitku**.

Zároveň chceme, aby se jednotka chovala inteligentně ve svém prostředí, tedy jednala racionálně. Prakticky se tedy bude v našem případě jednat o návrh racionálního bojového agenta orientovaného na užitek.

4.3.1 Užitek bojové jednotky

Má-li být agent racionální, bude provádět takové akce, které mu přinesou co největší užitek. Chceme-li, aby agent prováděl takové akce, musíme správně definovat **kritéria užitku** jednotlivých akcí, tedy určit, kdy můžeme agenta považovat za úspěšného (viz podkapitola 3.1.5). Také musíme určit **užitek agenta** za úspěšné provedení dané akce, případně musíme specifikovat **hodnocení dosaženého stavu prostředí**. Musíme tedy určit, jak moc velký bude užitek agenta z úspěšného provedení dané akce, případně do jak moc pro něj užitečného stavu prostředí se tímto dostane. Přitom nám postačí určit pouze jednu z těchto dvou uvedených variant. Můžeme se rozhodnout, zda agenta odměníme například za to, že zabil nepřátelského vojáka, nebo stanovíme odměnu pro každého agenta tohoto typu pro případ, že bude dosaženo takového stavu prostředí, v němž je o jednoho nepřátelského vojáka méně.

Kritéria užitku i samotná míra užitku se mohou s vývojem prostředí měnit. Nebudou v něm definována pouze pravidla typu – "pokud zabiješ nepřátelského vojáka, budeš odměněn 1000 body hodnocení" – ale také pravidla s proměnlivou mírou užitku. Pokud má například jednotka za úkol dohnat prchajícího vojáka, je vhodná míra užitku taková, kdy poroste ohodnocení daného stavu prostředí s blížící se vzdáleností jednotky k prchajícímu vojákovi. Nejenom, že jednotka bude takto motivována k jeho pronásledování, ale zároveň si bude, podle rostoucí míry užitku, vědoma, že její řešení dané situace je správné (jinak by mohla běžet na druhou stranu s pošetilou nadějí, že se s tímto vojákem někde setkají). Pokud bude úkolem jednotky ubránit určené stanoviště, může být jeho ohodnocením počet nepřátelských jednotek v okolí hlídaného stanoviště s tím, čím více nepřátelských jednotek je v okolí tohoto stanoviště, tím nižší je ohodnocení takového stavu prostředí pro jednotku, která jej má hlídat. Jednotka se tak pokusí nepřátelské jednotky eliminovat, aby dosáhla příznivějšího stavu okolního prostředí a tím vyšší míry užitku z takto dosaženého stavu.

Jednotka by měla být zároveň negativně hodnocena za akce neracionální. My můžeme pro zjednodušení celého hodnocení tímto negativním hodnocením vnímat samotnou skutečnost toho, že pokud se jednotka nebude chovat racionálně, nedosáhne takového stavu prostředí, který by ji odměnil. Tím, že o tuto odměnu přijde (nezíská ji), sama sebe potrestá. Kdyby měla jednotka za úkol hlídat daný cíl, ke kterému by se blížil nepřátelský voják a jednotka by se zabývala jinou činností než eliminací tohoto vojáka, bude cíl nepřátelskou jednotkou obsazen, což samo o sobě představuje negativní hodnocení pro vojáka, který měl stanoviště střežit.

Ve speciálních případech bychom ovšem měli provést dodatečné potrestaní (disciplinární trest). Proč? Měli bychom totiž zohlednit vojáky, kteří se snažili v rámci svých možností toto stanoviště ubránit, ale nepodařilo se jim to, oproti těm, kteří vůbec nic nepodnikli

(třeba se ani neobtěžovali získávat nové znalosti o svém prostředí, aby tak zjistili, že se blíží nepřítel). Takto rozlišíme mezi správně navrženými agenty, kteří, i když neúspěšně, dělají vše co je v jejich silách oproti těm, kteří by dosahovali podobného hodnocení, avšak ve skutečnosti by nic nedělali. Správně navržený racionální agent se snaží získávat užitečné informace o svém prostředí, aby tak maximalizoval svůj zisk a minimalizoval rizika. Nemůže sice vědět vše, ale správně se má alespoň snažit o to, aby získal co nejvíce informací, které by mu mohly být užitečné. Pokud se takto agent nechová, je špatně navrhnut.

4.3.2 Chování agenta

Chování agenta bude principiálně jednoduché. Agent bude umístěn do svého prostředí s předem definovanými cíli. Sám v sobě bude mít definovánu míru užitku pro dané stavy prostředí. Ta bude určovat, jaký užitek má agent z příslušného stavu prostředí. Agent v každém kroku získá informace o okolním prostředí pomocí svých senzorů. Na základě svého současného stavu, zjištěného stavu okolního prostředí, definované míry užitku a předpokladu o akcích ostatních jednotek zvolí pro něj nejvýhodnější akci (nebo-li tu, která mu přinese nejvyšší užitek) a tuto provede pomocí svých efektorů. Poté zjistí, jak se stav okolního prostředí změnil po provedení této akce, zda bylo dosaženo požadovaného výsledku a do jak moc dobře ohodnoceného stavu se touto akcí dostal. Tento proces se bude opakovat, dokud nebude činnost agenta ukončena.

V celém procesu je nezanedbatelně důležité zvážení toho, jaké akce budou provádět ostatní jednotky, protože tyto mohou znatelně narušit současný plán zkoumané jednotky, případně znatelně ovlivnit okolní prostředí v průběhu provádění akce agenta. Agent by tak mohl vše provést správně, ale tím, že by zanedbal předpoklad o činnosti ostatních jednotek, by nedosáhl ani požadovaného stavu prostředí, ani chtěného užitku z provedené akce.

4.3.3 Senzory

Agent bude získávat podněty ze svého okolí pomocí svých senzorů. Senzory budou napodobovat jak lidské senzory (oči, uši, nos, hmat ...), tak také různá speciální zařízení (radar, noční vidění apod). V našem případě si v návrhu vystačíme s obecným pojmem senzoru, protože se tímto tématem podrobněji zabývá paralelně vznikající bakalářská práce [2].

Každý použitý senzor bude mít svá specifika, jakými jsou např. **záběr** a **dosah** senzoru, kde **záběr senzoru** udává ve stupních rozsah oblasti, kterou senzor pokrývá a **dosah senzoru** udává vzdálenost, kterou senzor spolehlivě pokrývá do dálky. Dále senzor ponese informaci o svých schopnostech – **míře vlivu** stavu prostředí na daný senzor. Zde můžeme zařadit schopnost senzoru vidět v mlze, v noci, za tmy, za špatného počasí a další. **Míra vlivu** určuje, jak velký vliv má stav prostředí na daný senzor. Pokud specifikujeme např. senzor, na který má tma 100% vliv, určíme tím, že tento senzor není schopen získávat podněty, pokud se ocitne ve tmě. Jiný senzor, který bude mít tuto míru vlivu menší než 100% bude mít naopak schopnost získávat podněty ve tmě, některý samozřejmě lepší než jiný (podle velikosti míry vlivu). Nakonec bude mít také vlastnosti, které budou přímo ovlivnitelné agentem. Mezi tyto vlastnosti patří např. **směr senzoru**, který určuje, kterým směrem má daný senzor snímat. Tato vlastnost je přímo ovlivnitelná agentem, protože právě on zadává požadavek, ve kterém směru má být provedeno snímání.

Výsledkem funkce senzoru pak bude informace o objektech v jeho poli snímání, které je schopen rozeznat a také procentuální míra určující, z jak velké procentuální části tento objekt senzor zachytil. Agent již podle schopnosti rozeznat částečně viditelné objekty buď takto zjištěný objekt rozpozná, nebo o něm bude muset zjistit více informací. Jednotka

například vidí ruku vojáka. Zatímco jedna jednotka si všimne vlajky vyšité na jejím rukávě, jiná nemusí být tak všímavá a musí blíže prozkoumat tohoto vojáka, aby zjistila, zda se jedná o nepřítele nebo spojence. Toto už záleží na individuálním nastavení jednotlivých jednotek.

4.3.4 Efektory

Kromě získávání podnětů bude agent schopen své okolí také ovlivňovat, provádět nad ním akce. Tyto akce bude volit na základě vytvořeného plánu. Plán bude vytvořen na základě podnětů získaných ze senzorů agenta, jemu definovaného cíle a míry užitku, a také na základě jeho současného stavu. Akce jsou prováděny tzv. **efektory**. Za efektory můžeme považovat jakékoliv prvky agenta, které jsou schopny ovlivňovat okolní prostředí, a to ať už se jedná o lidské efektory (ruce, nohy), nebo o efektory mechanické (zbraně, vozidla). Akcemi, které se týkají našeho návrhu agenta, jsou například **pohyb**, **střelba**, a další.

K **pohybu** slouží **pohybová centra**, či-li u člověka by se těmito centry nazvaly jeho nohy. Agent se bude pohybovat tak, že informuje prostředí o zamýšleném pohybu – např. mu dodá informaci "přesunuji se 100 metrů na východ rychlostí 3 metry za sekundu." Prostředí tuto skutečnost akceptuje a vnitřně si spolu s pohybem agenta upravuje informaci o jeho současné poloze, přičemž jej o této poloze zpětně informuje. Agent tak pomocí senzorů dostává zpětnou vazbu o tom, jestli se mu jeho akci podařilo provést. Zároveň je tato informace vizuálně promítnuta do prostředí. Akce se také nemusí zdařit, agent může narazit na překážku (skálu, stěnu budovy), protože se špatně informoval na stav svého okolního prostředí, nebo se v jeho prostředí od posledního dotazu něco změnilo (zastavilo před ním vozidlo, vběhla mu do cesty jiná jednotka). Potom je nutné, aby mu prostředí pohyb znemožnilo. Agent pak již sám může svými senzory zjistit, že mu byl jeho pohyb znemožněn, a se vzniklým problémem se vypořádat.

Podobně budou řešeny i ostatní akce agenta s tím, že různým akcím budou přidělena různá centra. Efektorům, které budou sloužit k pohybu, bude náležet pohybové centrum, efektorům sloužícím ke střelbě bude náležet centrum střelby apod. Jednotlivé efektory přidělené stejnému centru se přitom od sebe mohou lišit, například efektorů pro pohyb může být několik (nohy, kola) a také mohou mít rozdílné vlastnosti (rychlost pohybu, schopnost pohybu v rozličném terénu, ve vodě).

4.3.5 Mapování z vjemů na akce

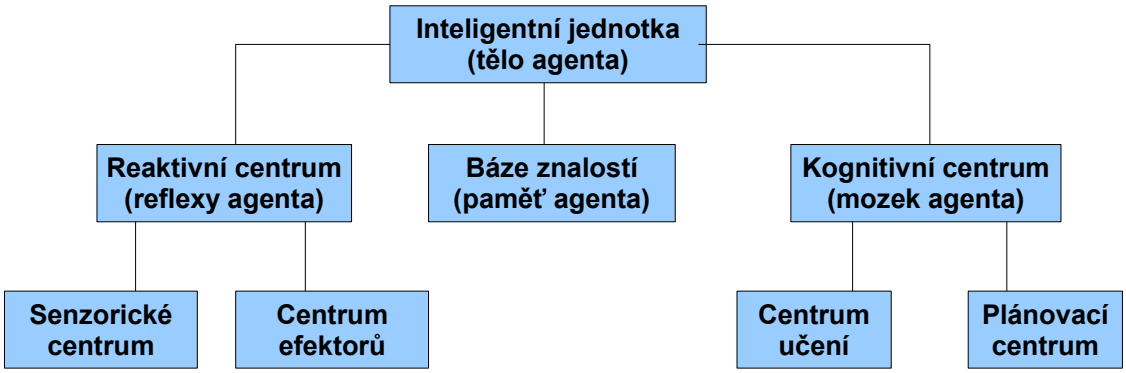
Je potřeba navrhnout mapování z množiny vjemů na množinu akcí. Jednotka by měla umět řešit problémy typu: "Když se objeví nepřátelský voják, co mám dělat?" Toto mapování by mělo zohlednit cíle jednotky – pokud má jednotka za cíl ubránit stanoviště za každou cenu, tak v případě výskytu nepřátelského vojáka bude mít za úkol na něj zaútočit. Pokud bude jejím cílem jenom skrytě pozorovat nepřátelské jednotky, jejím úkolem bude nahlásit tuto jednotku veliteli. Tyto reakce na vjemy tvoří charakteristiku jednotlivých jednotek, čili jednotka bude charakterizována její funkcí mapování z množiny vjemů na množinu akcí.

Mapování bude tvořeno množinou pravidel převádějící vjemy na akce, návrh mapování přitom bude proveden co nejvíce obecně. Nebudeme se zabývat pravidly na tak konkrétní úrovni, kde bychom definovali úkony typu "Pokud je jednotka 5 metrů na sever, postup 5 kroků na sever", tím se budou zabývat centra navržená speciálně pro tento účel (například pohybové centrum). My budeme definovat pravidla na obecnější úrovni – např. "Pokud je v okolí nepřátelská jednotka, přesuň se do její blízkosti tak, aby tě neviděla."

Zároveň se však budeme snažit o to, aby jednotka zůstala co nejvíce autonomní. Pokusíme se najít střed mezi tím, co by jednotka měla znát od svého tvůrce a tím, co by se měla naučit sama. Pokud bychom jí nadefinovali veškeré jednání, byla by tato jednotka neschopná řešit jí neznámé situace, tedy ty, pro které by neměla definována pravidla. Dost významně by tak ztrácela flexibilitu, protože by sama v sobě nesla nějaké předpoklady o dané situaci, které, kdyby byly nečekaně porušeny, způsobily její neschopnost zareagovat na takto vzniklou situaci. Pokud jí naopak nadefinujeme málo pravidel, nebude schopná jednat inteligentně. Když se podaří nalézt správný střed, jednotka dokáže využít správně definované báze znalostí k tomu, aby sama nacházela řešení na určité situace a pomocí různých pokusů reakce na danou situaci časem zjistila, která reakce je na tuto situaci ta nejlepší.

Přirovnat si to můžeme přímo k nám a našemu intelektuálnímu vývoji. Také nás rodiče naučili některým základním pravidlům, avšak v životě se vyskytla spousta situací, s nimiž jsme si museli poradit sami, někdy s dodatečnou znalostí jiné osoby. Z toho lze usoudit, že ani agent nemůže být připraven na všechny situace a pak již záleží na jeho schopnosti nalézt řešení v dané situaci. V hledání řešení nám přitom pomáhají doposud získané znalosti o problému.

4.4 Návrh agenta



Obrázek 4.1: Návrh základních center agenta.

Inteligentní bojová jednotka:

- sdružuje v sobě všechna ostatní centra, můžeme si ji představit jako lidské tělo
- všechna důležitá komunikace prochází skrze toto centrum
- obsahuje základní informaci o své identitě (identifikátor jednotky v prostředí, typ jednotky, stav jednotky – v pořádku, poškozená (nemůže střílet, nemůže se pohybovat, je mrtvá), kouří, hoří, nebo se za ní práší.

Senzorické centrum:

- zastřešuje všechny podsystémy sloužící k získávání znalostí z prostředí
- komunikuje s prostředím – zadává mu požadavky a získává z něj znalosti

- výsledky komunikace putují do reaktivního centra, kde jsou zpracovány

Centrum efektorů:

- zastřešuje všechna centra sloužící k provádění akcí na prostředí
- zadává požadavky na prostředí (provádí na něm definované akce)
- nemá z prostředí přímou zpětnou vazbu o úspěchu akce

Reaktivní centrum:

- je zodpovědné za rychlé reakce agenta na kritické události
- je neustále v kontaktu se senzorickým centrem a kontroluje své okolí na výskyt kritických událostí. Stejně tak je schopno kdykoliv provést potřebnou akci na prostředí pomocí efektorů
- znalosti získané ze senzorického centra ukládá do báze znalostí

Báze znalostí:

- tvoří model agenta o okolním prostředí
- ukládá v sobě znalosti, které se agentovi podařilo zjistit o okolním prostředí
- toto centrum je při každém zjišťování stavu prostředí aktualizováno, přitom jsou však zachovávány důležité informace (např. kde se naposledy nacházela jednotka, kterou jsme ztratili z dohledu)
- je potřeba ukládat různorodé typy znalostí (pozice jiných agentů, jejich odhadované typy, akce, které provádějí na prostředí apod.). Je potřeba definovat jednoznačný formát znalostí, které budou moci být v této bázi znalostí ukládány.

Kognitivní centrum:

- představuje "mozek" agenta – zajišťuje jeho inteligentní chování
- sdružuje v sobě podsystémy, které se podílí na racionalitě agenta

Plánovací centrum:

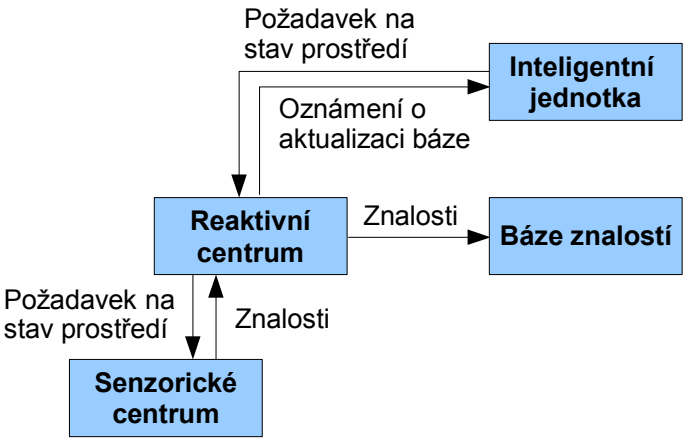
- je zodpovědné za tvorbu plánu agenta na základě jeho cíle, míry užitku a současných znalostí o prostředí
- vytvořený plán je předán do těla agenta k provedení (to jej rozešle příslušným efektorům a senzorům k provedení)

Centrum učení:

- centrum, které se snaží z informací o změnách okolního prostředí vyčíst informace o chování ostatních jednotek, účinnosti jednotkou prováděných akcí apod.
- získané informace využívá k definování nových pravidel, kterými by mohla jednotka zvýšit svoji úspěšnost.

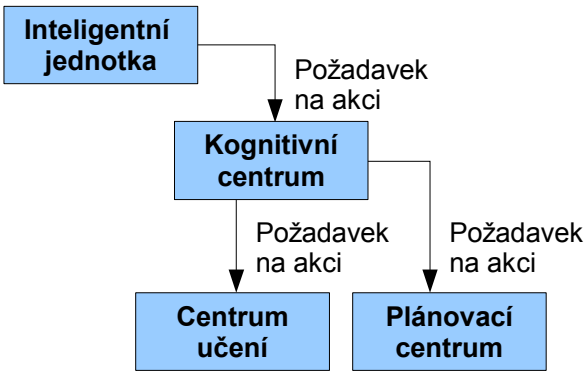
4.4.1 Komunikace agenta

Agent od svého umístění do prostředí až do svého odebrání z tohoto prostředí, případně svého zániku, prochází neustále se opakujícím komunikačním cyklem. Tento cyklus je neměnný a různorodé chování agenta je ovlivněno pouze současným stavem prostředí a cíli, které si agent předsevzal dosáhnout, případně mu byly určeny.



Obrázek 4.2: Agent získává znalosti o současném stavu prostředí.

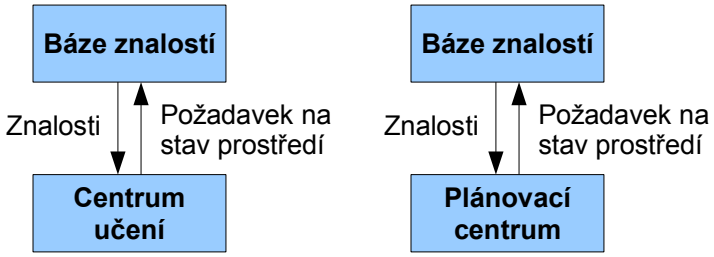
V první fázi komunikačního cyklu agent nemá informaci o současném stavu prostředí, v němž se právě nachází. Potřebuje tedy získat informace o svém prostředí, aby mohl pracovat s aktuálními znalostmi. Tímto požadavkem adresuje své reaktivní centrum, které tento požadavek předá centru senzorickému. Senzorické centrum následně z okolního prostředí extrahuje význačné rysy a vrátí reaktivnímu centru důležité znalosti o tomto prostředí. Příkladem takové znalosti může být informace, že se 100 metrů před ním nachází nepřátelská pěší jednotka. Tuto informaci mu vrátí již ve formě, která se dá přímo uložit do agentovy báze znalostí. Závěrečným krokem této fáze je právě zmíněné uložení získaných znalostí do báze znalostí a informování těla agenta o tom, že báze znalostí byla aktualizována.



Obrázek 4.3: Žádost agenta o akci, kterou je vhodné v současné chvíli provést.

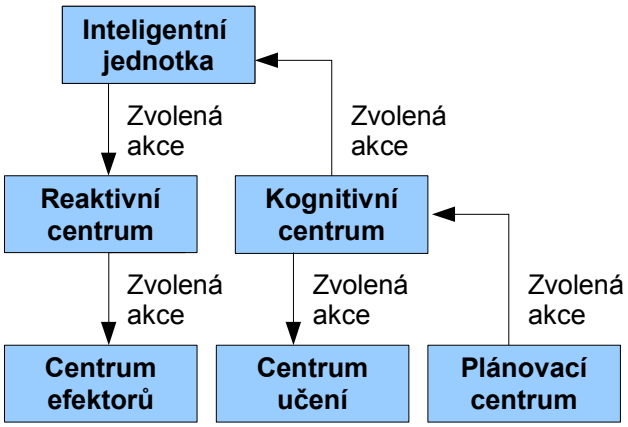
Poté, co agent aktualizoval svou bázi znalostí, potřebuje zjistit jakou akci má v následujícím kroku provést. Tímto úkolem pověří kognitivní centrum, jemuž zadá požadavek na akci,

kterou je vhodné provést v aktuálním stavu prostředí. Kognitivní centrum tento požadavek předá plánovacímu centru, které je za výběr akce zodpovědné. Kopii tohoto požadavku předá také centru učení, čímž jej informuje o tom, že byl právě podán požadavek na akci.



Obrázek 4.4: Adresovaná centra získávají informace o současném stavu prostředí.

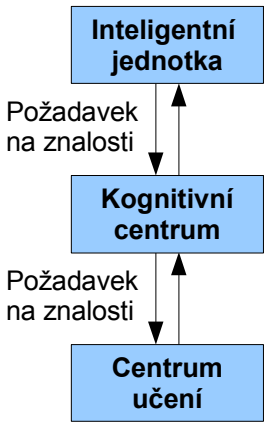
Po obdržení požadavku na naplánovaní akce, která by měla být provedena, si plánovací centrum vyžádá informaci o současném stavu prostředí, nad kterým bude provedeno plánování. Stejně provede i centrum učení, protože potřebuje znát stav světa, nad nímž bylo plánování provedeno. Čeká totiž, jak plánovací centrum při současném stavu světa odpoví na obdržený požadavek a jaký vliv bude mít zvolená akce na okolní prostředí.



Obrázek 4.5: Naplánovaná akce je zaslána efektorům k provedení.

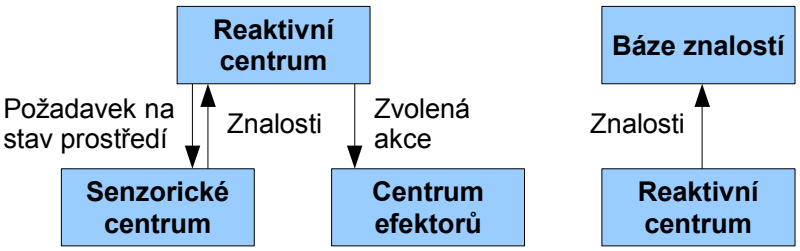
Na základě získaných informací plánovací centrum rozhodne o akci, kterou je vhodné provést. Tuto informaci zašle kognitivnímu centru, které ji dále přepošle nadřazené inteligentní jednotce a také centru učení. Inteligentní jednotka zvolenou akci zašle reaktivnímu centru, které ji již předá centru efektorů k vykonání. Centrum efektorů se pokusí zvolenou akci vykonat, ale její úspěch není zaručen. Akce může být sama o sobě neúspěšná, případně se jejím vykonáním nemusí podařit dosáhnout zamýšleného cíle, ale také se mohlo stát, že se stav prostředí v průběhu plánování změnil natolik, že již není možno akci provést vůbec. Příkladem takové akce může být rozhodnutí zaútočit na nepřítele, který byl mezitím zastřelen jinou jednotkou.

V závěrečné fázi se agent ptá centra učení, zda se mu podařilo z proběhlé komunikace zjistit nějaké zajímavé vzory, případně zda se mu podařilo odvodit nějaké znalosti. Centrum učení by mělo být schopno zjistit takové znalosti na základě obdržené informace



Obrázek 4.6: Dotázání se centra učení zda z proběhlé komunikace zjistila zajímavé znalosti.

o stavu prostředí a zvolené akci, navíc také díky znalostem o minulosti běhu agenta a o volených akcích v dřívějších komunikačních cyklech. Odvozená znalost pak může být využita k urychlení plánování, nebo k ustanovení nového pravidla pro reaktivní centrum, protože agent již má informaci o tom, v jakém stavu prostředí je potřeba podniknout jakou akci. Například příště, až uvidí nepřátelského vojáka, rovnou na něj vystřelí na úrovni reaktivního centra bez potřeby složitějšího plánování. Stejně tak může dojít k posílení určitého pravidla tam, kde je více možností volby akce. Pokud byla zvolená akce úspěšná, bude mít i příště větší šanci na to, aby byla vybrána. Naopak v případě neúspěchu bude pravděpodobnost její volby snížena a tím bude v příštím cyklu umožněno vybrat akci jinou. Reakce jednotky na získané znalosti může být různá. Buď tyto znalosti přímo uloží do báze znalostí, pokud se jedná o odvozené znalosti o světě, nebo upraví pravidla, která jsou k plánování použita, pokud se jedná o informaci, která pomůže agentovo plánování zefektivnit.



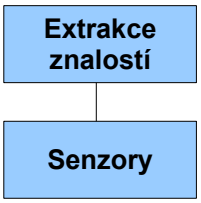
Obrázek 4.7: Reaktivní smyčka agenta. Reakce na výskyt kritické události v prostředí.

Kromě komunikačního cyklu v agentovi nezávisle na tomto cyklu probíhá tzv. **reaktivní smyčka**. Tato bez ohledu na plánování agenta žádá senzorické centrum o současný stav prostředí a v navrácených znalostech se snaží nalézt **kritické události** a případně na ně zareagovat (viz 4.4.4). V případě, že nalezne zajímavou znalost, která by mohla ovlivnit současné plánování agenta, vloží ji do báze znalostí, aby plánovací algoritmus měl tuto informaci k dispozici a snížilo se tak riziko vytváření neplatných plánů. Stejně tak pokud je podniknuta nějaká akce v reakci na kritické události, je informace o podniknuté akci promítnuta do báze znalostí. Agent může například plánovat, jakou akci podniknout, když

poblíž není nepřátelská jednotka, která se však v průběhu plánování objeví. Reaktivní smyčka zamezí plánování na základě neplatného předpokladu upozorněním na to, že se situace závažně změnila.

4.4.2 Senzorické centrum

Senzorické centrum slouží k získávání znalostí o současném stavu okolního prostředí. Jedná se o jediný prvek agenta, který mu umožňuje takové znalosti získávat a tak jej udržovat informovaného o všem, co se mimo něj děje. Centrum čeká na požadavek od vyšší vrstvy a na obdržení tohoto požadavku odpovídá aktuálním stavem okolního prostředí, který se podařilo zachytit pomocí senzorů. Aktuální stav vrací v podobě znalostí, které je možné rovnou uložit do báze znalostí.



Obrázek 4.8: Prvky senzorického centra.

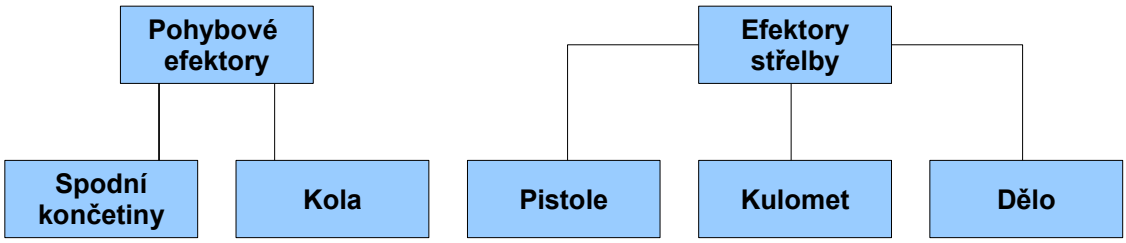
Senzorické centrum sestává ze dvou prvků. Prvním z jich jsou **senzory**. Senzory přímo komunikují s okolním prostředím a extrahují z něj rysy, které jsou pro agenta důležité. Může se jednat jak o objekty, které agent vidí, tak o informace o rozmístění a typech okolních jednotek, nebo také zachycený zvuk připomínající vzdálený pohyb tanku, který senzory zaregistrují. Senzorů může existovat celá řada a blíže o nich a o jejich smyslu bylo pojednáno v kapitole 4.3.3.

Druhým prvkem je **extrakce znalostí**. Jedná se o mezičlánek mezi vstupem senzorů a výstupem senzorického centra. Jedním z požadavků, které jsou kladeny na senzorické centrum je ten, že má vracet takové znalosti, které je možno rovnou uložit do báze znalostí bez nutnosti dalších úprav. Výstupem senzoru však takové znalosti nejsou a proto je potřeba je právě v tomto kroku do požadované podoby transformovat.

4.4.3 Centrum efektorů

Centrum efektorů je schopno provádět akce na okolním prostředí. Stává se tak centrem, díky kterému agent navenek vykazuje určité chování. Jeho role je jednoduchá a částečně je také popsána v kapitole 4.3.4. Centrum dostane na vstupu akci, o níž bylo rozhodnuto, že je v současné chvíli vhodná k vykonání. Tato akce je centrem přebrána a následně vykonána na okolním prostředí.

Toto centrum se skládá ze dvou úrovní - na úrovni obecnější se nachází **moduly efektorů**, které pod sebou sdružují jednotlivé **efektory** a určují rozhraní, které musí dané efektory splňovat, aby mohly být zařazeny pod tento modul. Přitom již nemusí znát konkrétní implementaci, kterou je splnění tohoto rozhraní dosaženo - to je řešeno na úrovni konkrétních efektorů. Příkladem takového modulu může být modul pohybových efektorů, který vyžaduje schopnost efektoru přemístit agenta na určené místo. Konkrétními implementacemi efektorů pro tento modul pak mohou být například nohy nebo kola, které obě dvě splňují požadavek schopnosti přemísťovat agenta.

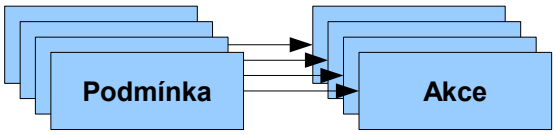


Obrázek 4.9: Příklad prvků centra efektorů. Dva moduly (pohybové efektory a efektory střelby) v sobě zahrnují konkrétní efektory.

Po obdržení akce je potřeba rozhodnout, který modul se má postarat o její vykonání. Toto lze jednoduše provést rozesláním obdržené akce všem modulům efektorů s předpokladem, že příslušný modul, jemuž je tato akce určena, ji bude schopen zpracovat. Ostatní moduly budou požadavek na vykonání takové akce ignorovat. Příslušný modul musí ještě rozhodnout o tom, který konkrétní efektor bude vybrán k provedení této akce na prostředí. V této otázce předpokládá, že mu bude v rámci akce předán konkrétní efektor, jeho výběr bude předmětem některé z předchozích akcí, případně použije ten efektor, který v současné chvíli považuje za aktuálně používaný. Příkladem může být zaslání sekvence akcí "Zvol kulomet" a "Vystřel na protivníka A", případně akce "Vystřel na protivníka A kulometem", nebo modul sám určí při příchodu akce "Vystřel na protivníka A" kulomet, protože jej právě drží v ruce.

4.4.4 Reaktivní centrum

Toto centrum uzavírá **reaktivní smyčku**. Jeho hlavním úkolem je vyhledávání **kritických událostí** v prostředí a reakce na ně, je však také prostředníkem mezi senzorickým centrem a centrem efektorů. Vyhledávání kritických událostí je prováděno neustálým dotazováním se senzorického centra na aktuální stav okolního prostředí a hledáním těchto událostí v poskytnutých znalostech. Pokud se taková událost objeví, reaktivní centrum na ni okamžitě reaguje provedením příslušné akce. V případě, že v průběhu své činnosti centrum narazí na důležitou znalost, informuje o ní bázi znalostí. Stejně tak v případě, kdy samo podnikne kroky k vyřešení nastalé události, informuje bázi znalostí o akcích, které provedlo.



Obrázek 4.10: Množina pravidel určuje na jaké události a jakým způsobem má reaktivní centrum reagovat.

Hlavní část tohoto centra tvoří uživatelem určená **množina pravidel**. Tato pravidla sestávají z definice událostí, na které by mělo centrum reagovat, a akcí, které by mělo podniknout, pokud se definovaná událost vyskytne. Jedná se o rychlou smyčku, která je provedena okamžitě a je vyhrazena pro události, na které je potřeba zareagovat v reálném

čase. V případě, že na agenta někdo útočí, nemá čas přemýšlet nad tím, co je nyní potřeba podniknout, ale může mít stanovenu prioritu schovat se, nebo naopak opětovat palbu. To již závisí na konkrétní množině pravidel, které jsou mu přiřazeny.

Komunikace s bází znalostí je v souvislosti s tímto centrem důležitá ze dvou důvodů, které však mají oba stejný základ. Tím je fakt, že reaktivní smyčka funguje neustále a to i ve chvíli, kdy plánovací centrum právě provádí plánování na okolním prostředí. V případě, že se v okolí vyskytne důležitá a zároveň nová znalost, reaktivní centrum ji vloží do báze znalostí. Plánovací centrum jenom ověří, zda daná znalost nezneplatní doposud vytvářený plán a pokud ne, pokračuje dál v plánování. V opačném případě by se zbytečně zaobíralo plánem, který by v konečném důsledku stejně nebyl proveditelný. Druhým případem, kdy je potřeba o takovéto změně informovat, je při výskytu kritické události, na kterou reaktivní centrum přímo zareaguje. To je už závažnější situace vzhledem k plánování, protože se vyskytly změny týkající se nejenom okolního prostředí, ale také agenta samotného. Provedením definované akce agent změnil výchozí předpoklady o své poloze, stavu nebo o bezpečném okolí, které mohly být klíčovým aspektem pro vytvořený plán.

Množina pravidel

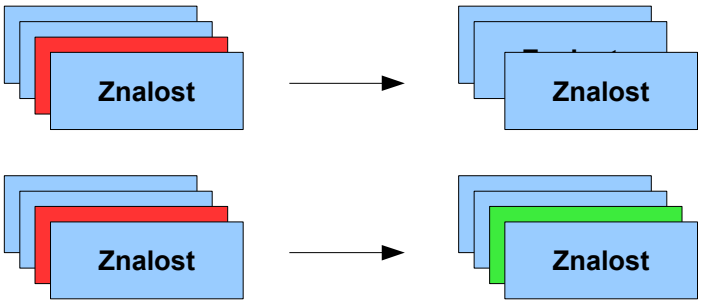
Množina pravidel tvoří základ funkcionality reaktivního centra. Bez této množiny by bylo reaktivní centrum pouze prostředníkem v komunikaci mezi senzorickým centrem, centrem efektorů a tělem agenta. Volba množiny pravidel proto tvoří důležitou součást při určování chování agenta a je potřeba ji volit rozumně. Při špatném návrhu množiny pravidel může agent projevovat zmatené a nelogické chování (pokud toto není účelem) a také může uváznout v nekonečné smyčce. Důležité je si také uvědomit, že množinou reaktivních pravidel nemůžeme postihnout všechny vyžadované úkony. Všechny akce musí proběhnout co nejrychleji a tudíž zde není čas na dlouhé přemýšlení a dlouho trvající akce.

Základními prvky množiny pravidel jsou **pravidla**. Tyto určují, kterými akcemi je potřeba zareagovat na výskyt jakých událostí. Pravidla se skládají z **množiny podmínek** a **množiny akcí**. Množina podmínek je tvořena podmínkami, které musí být splněny, aby bylo dané pravidlo označeno za platné k provedení. Takovou množinou může být například množina podmínek "Vidím vojáka ve vzdálenosti 5 metrů ode mě" a "Spatřený voják je nepřátelský". Množina akcí pak určuje, jaké akce je potřeba okamžitě podniknout, pokud je příslušná množina podmínek splněna. Tyto akce simulují reflexy agenta a proto je musí být možné provést okamžitě, bez nutnosti jejich dlouhého plánování. Příkladem takové akce může být "Vystřel po vojákovi" a "Ustup stranou".

Definovaná množina pravidel určuje chování agenta tím, jak reaguje na současný stav okolního světa. Při každém stavu světa centrum zjišťuje, která pravidla jsou právě platná k provedení. Změna stavu okolního světa mohla způsobit splnění množiny podmínek některého pravidla a agent v reakci na tento stav podniká akce definované množinou akcí splněného pravidla.

4.4.5 Báze znalostí

Báze znalostí nemá sama o sobě moc velkou funkcionalitu, přesto je však důležitou součástí agenta. Slouží jako úložiště znalostí, které agent uchovává o okolním prostředí, a vytváří tak agentův model okolního světa. Agent pak nemusí vnímat okolní prostředí a přesto se v něm dokáže na základě znalostí pohybovat a působit až do chvíle než se prostředí změní. Bohužel v realitě se prostředí často mění a proto potřeba agenta neustále obnovovat svou bázi znalostí přetrvává.



Obrázek 4.11: Báze znalostí je skladem všech znalostí agenta o okolním světě. Jelikož se svět neustále vyvíjí, mohou být některé znalosti vymazány nebo aktualizovány.

Ve chvíli, kdy je do báze znalostí ukládána znalost nová, může dojít k tomu, že je zneplatněna znalost, která se již v bázi znalostí vyskytuje. V tomto případě může dojít ke dvěma důsledkům - buďto je nová znalost aktuálnější verzí znalosti staré a potom je stará znalost přepsána znalostí novou, nebo je nová znalost v konfliktu se starou znalostí a potom je stará znalost z báze vymazána. Díky neustálé proměnlivosti světa se totiž předpokládá, že novější znalosti nesou "správnější" informaci o okolním světě než ty, které se v ní současně nacházejí. Za tím účelem je také aktualizace báze znalostí prováděna.

Znalosti

Jak již bylo výše naznačeno, báze znalostí je důležitá, protože umožňuje agentovi provádět smysluplnější akce. Pokud chceme, aby měl agent schopnost plánovat své akce, nebo aby se byl schopen rozhodovat na úrovni vyšší než reaktivní, musíme mu poskytnout znalosti o okolním prostředí, z nichž může vycházet a s nimiž může pracovat.

Aby s těmito znalostmi mohl agent pracovat, musí mít definován způsob, jakým v něm budou znalosti uloženy (nebo-li **syntaxi** znalostí) a co bude takto uložená znalost vypovídat o okolním světě (**sémantika** znalostí). Báze znalostí může být také schopna odvozovat ze současných znalostí znalosti nové, tzv. **odvozené znalosti**, k tomu však potřebuje mít ještě definován **odvozovací mechanismus**. Na takto definovanou bázi znalostí pak můžeme pokládat dotazy, o jejichž pravdivosti by měla být schopna rozhodnout a poskytnout tak agentovi informaci, kterou využije jak při plánování, tak při svém rozhodování.

Typy znalostí

Přestože se náš agent pohybuje v uměle vytvořeném světě, vyskytuje se v něm spousta informací, které by měl být schopen ukládat a pracovat s nimi. Základními z nich jsou: **pozice jednotek**, **typ jednotek** (**přátelské/nepřátelské**, ale i **voják/tank**, případně **raketometčík/tank Panther**), **stav jednotek**, **současný směr pohybu jednotek v dosahu**, ale také **poslední zaznamenaný směr pohybu okolních jednotek mimo dosah** a mnoho dalších.

S každou znalostí je potřeba uložit také **časové razítko** znalosti, které určuje okamžik, v němž byla tato znalost pořízena. Toto je důležité pro zamezení práce agenta s neaktuálními informacemi. Díky časovému razítku může agent při plánování zvažovat také informaci, kdy byla tato znalost pořízena a zda má smysl se s ní ještě vůbec zaobírat. Doba od

pořízení znalosti totiž už dávno mohla překročit dobu užitečné platnosti poskytnuté informace. Například informace o tom, že jsem před hodinou viděl nepřátelskou jednotku pohybovat se směrem ode mě může být neaktuální, případně může být považována za aktuální, ale méně důležitou než je informace o tom, že jsem na opačné straně zahlédl jinou nepřátelskou jednotku před 5 minutami. Zde je časové razítko užitečné k určení novosti informace, která může být v takovýchto situacích kritická pro rozhodnutí agenta, jakou akci je vhodné provést.

Mimo znalosti informující agenta o stavu světa je také vhodné ukládat i informace o **náročnosti** jednotlivých akcí, které může agent provádět. Pokud máme informaci o tom, jak budou jednotlivé akce, které agent volí, náročné na čas, suroviny, spotřebu munice, jak moc je bezpečné je provést apod., může se agent pokusit nalézt nejméně náročné řešení.

Syntaxe znalostí a sémantika znalostí

Syntaxe znalostí určuje, v jaké formě budou znalosti v bázi znalostí uloženy. Forem může být libovolně mnoho, proto záleží jenom na tom, jakou formu uložení a reprezentace daných znalosti zvolíme. Jednou z možností je vnímat okolní prostředí jako množinu objektů a proces získávání znalostí o prostředí jako proces získávání znalostí o objektech v tomto prostředí. Potom můžeme jednotlivé prvky prostředí ukládat jako objekty a příslušné znalosti, o nich zjištěné, jako vlastnosti těchto objektů. Sémantika znalostí potom představuje mapování z uložené vlastnosti o daném objektu na skutečnou vlastnost daného objektu, kterou v bázi znalostí daná vlastnost reprezentuje. Zvoleným odvozovacím mechanismem můžeme ze získaných vlastností odvodit další vlastnosti, které nejsme schopni zjistit přímo.

4.4.6 Kognitivní centrum

Kognitivní centrum je ve své podstatě pouze prostředníkem mezi zadavatelem požadavků a jejich adresáty. Stará se o správnou komunikaci mezi příslušnými prvky systému a dohlíží na to, aby se všechny potřebné informace dostaly na správné místo. Například kopíruje žádost o akci i centru učení, které sice není adresátem, avšak tato informace je k jeho činnosti nezbytná. Stejně tak mu předává i informaci o zvolené akci.

4.4.7 Centrum učení

Centrum učení se snaží vylepšovat plánování a reaktivní chování agenta, a také obohacovat jeho bázi znalostí o nové znalosti, které mohou být klíčové pro jeho strategii. Jeho hlavní činností je monitorování akcí agenta a změn v prostředí, zjišťování toho, jak moc se prostředí vyvíjí samo od sebe a do jaké míry v závislosti na agentových akcích, a snaží se v těchto informacích nalézt opakující se vzory, případně se snaží poučit z agentových chyb.

Jednou ze zmíněných činností bylo monitorování akcí agenta. V tomto případě centrum učení vyčkává na chvíli, kdy bude plánovacímu centru agenta zaslán požadavek na naplánování akce. Centrum učení se podívá do báze znalostí, aby zjistilo z jakých znalostí plánovací centrum vychází a vyčká na naplánovanou akci. Pokud dochází k opakované volbě stejných akcí a centrum učení dokáže nalézt vzor výskytu určitých znalostí v bázi znalostí, lze vytvořit pravidlo, na jehož základě může být specifikována odezva již v reaktivním centru - pokud se objeví znalosti vedoucí k naplánování této akce, proveď tuto akci bez potřeby složitého plánování.

Mimo vytváření nových pravidel by mělo být centrum učení schopno sledovat úspěšnost zamýšlené akce v prostředí. V případě, že byla z množiny více akcí, mezi nimiž nebylo jinak

možno rozhodnout, vybrána náhodně jedna akce a její odezva v prostředí nebyla úspěšná, centrum učení pro příští plánování sníží pravděpodobnost volby takovéto akce ve stejné nebo podobné situaci. Naopak v případě, že tato akce vedla k dosažení předsevzatého cíle, je pravděpodobnost opětovného zvolení této akce posílena.

Monitorováním prostředí by centrum učení mělo být schopno nalézt pravidla změny prostředí bez zásahu agenta, ale také s jeho zásahem do prostředí. V prostředí se mohou v těchto situacích objevovat zajímavé vzory, které mohou být agentovi užitečné - například informace, že nepřítel vždy útočí ze severu nezávisle na volené akci agenta, nebo že při každém výstřelu agenta uhýbá nepřátelský agent doleva.

4.4.8 Plánovací centrum

Jedná se o strategické centrum agenta, zabývající se tvorbou dlouhodobějšího plánu. Právě zde se odehrává veškeré zvažování, logické usuzování a plánování. Toto centrum je tedy zodpovědné za logické chování agenta.

Pro tvorbu plánu je potřeba, aby měl agent definován dlouhodobější cíl a míru užitku. Stejně tak potřebuje mít přístup k bázi znalostí, znát teoretické dopady svých akcí a priority (tyto jsou dány mírou užitku) při dosahování většího počtu definovaných cílů.

Plánování můžeme chápat jako způsob řešení vzniklého problému tím, že se snažíme nalézt sekvenci takových akcí, které budou maximalizovat užitek, kterého provedením této sekvence akcí dosáhneme. Množství volitelných akcí však může být obrovské a proto je potřeba omezit tento prostor na co nejmenší. V tom nám pomůže správná **formulace cíle** s ohledem na současný stav světa. Pak je úkol agenta omezen na nalezení takové množiny stavů okolního prostředí, v níž bude formulovaný cíl splněn a nalezení sekvence takových akcí, které těchto stavů dosáhnou. Řešením plánování je pak právě tato sekvence akcí, kterou agent nalezl a která umožní transformovat současný stav světa na cílový. K hledání lze použít správně definované strategie prohledávání stavového prostoru.

Dost významným prvkem může být omezení plánování agenta časovou nebo hloubkovou funkcí. Tzn. umožnit agentu plánovat jen po dobu jistého časového intervalu, nebo prohledat stavový prostor jenom do určité hloubky. Po uplynutí určeného intervalu, resp. po dosažení určené hloubky prohledávání, agent vrátí nejlepší zjištěnou akci, kterou se mu v rámci tohoto omezení podařilo nalézt. Toto omezení jej tlačí dosahovat efektivních rozhodnutí v co nejkratším čase. Vhodné je použití heuristiky v kombinaci s prohledáváním stavového prostoru.

Při plánování také nesmíme zapomenout zavést **ohodnocovací funkci**, která udává jak je stav, jehož dosažení volenou akcí zvažujeme, výhodný ke zvolení. Tato se podstatně liší od míry užitku. Míra užitku nám udává, jaký užitek budeme mít z dosažení cílového stavu. Naopak ohodnocovací funkce předpokládá, že jsme si nejvíce užitečný cíl již vybrali a hodnotově udává správnost s jakou se nám daří přibližovat ke zvolenému cíli. Přitom by tato funkce měla lépe hodnotit cestu vedoucí k dosažení cíle než-li cesty ostatní.

4.5 Prostředí

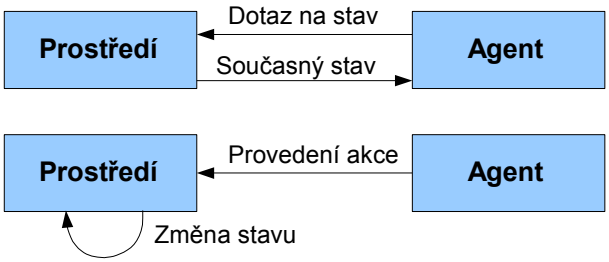
Jedním z úkolů je také navrhnout jednoduché testovací prostředí, v němž bude testováno chování inteligentních bojových jednotek. Simulátor společnosti E-COM s.r.o. je totiž vytvořen dost robustně a testování přímo na tomto simulátoru by nebylo při návrhu efektivní.

Prostředí simulátoru společnosti E-COM s.r.o. poskytuje nejenom informace o jednotkách v něm, ale také o počasí v každém bodě, světelných podmínkách, množství mlhy,

profilu terénu a dalších skutečnostech, které můžeme při našem návrhu zanedbat.

Testovací prostředí však bude co nejblíže simulovat prostředí samotného simulátoru v aspektech, které jsou důležité pro návrh inteligentní jednotky. Když jednotka zadá dotaz na toto prostředí, prostředí jej vyhodnotí a vrátí jednotce výsledek. Když na něm jednotka bude chtít provádět akce, prostředí se pokusí jednotce vyhovět a upravit svůj stav, v rámci definovaných omezení, tak aby jednotce vyhověla. Jednotka však nemá informaci o tom, jak dokázala prostředí ovlivnit, protože nemá přístup k jeho stavu. Musí tedy opět použít senzory, kterými se zeptá, jak bylo okolní prostředí ovlivněno. To je jediná možnost, jak může zjistit stav okolního prostředí. Prostředí se přitom bude měnit v závislosti na všech akcích všech jednotek, takže bude dynamické.

4.5.1 Návrh prostředí



Obrázek 4.12: Návrh prostředí.

Stav prostředí bude určen:

- jednotkami a jejich rozmístěním v prostředí
- pasivními objekty a jejich rozmístěním v prostředí
- akcemi, které na něm lze v současné chvíli provádět
- vjemy, které je schopno jednotce poskytnout

Funkce prostředí:

- poskytování vjemů jednotkám (podmnožina současného stavu prostředí)
- změna prostředí v závislosti na požadované akci agenta
- změna prostředí při požadavku na přidání nové jednotky nebo objektu do prostředí

Kapitola 5

Popis implementace

5.1 Cíle implementace

Jedním z požadavků zadavatele bylo vytvořit základní implementaci navrženého modulu tak, aby bylo možné ověřit správnost návrhu a provádět s touto implementací experimenty. Přitom nemusí být implementována všechna centra, nýbrž pouze ta, která umožní bojové jednotce autonomně se chovat ve svém prostředí. Jak již bylo zmíněno v podkapitole 4.5, diky robustnosti simulátoru společnosti E-COM s.r.o. vzniká také požadavek implementace vlastního prostředí, ve kterém se tato jednotka bude pohybovat.

Cílem je tedy vytvořit základní implementaci inteligentní bojové jednotky, která bude vnímat své okolí pomocí senzorů, ovlivňovat jej svými efektory, bude reagovat na kritické události v okolí pomocí pravidel reaktivního centra a zároveň bude schopna dlouhodobějšího plánování v rámci centra plánování. V průběhu svého pohybu v daném prostředí si bude vytvářet vnitřní model okolního světa v bázi znalostí. Jediné z center, jehož základní implementace nebyla vybrána k provedení, je centrum učení, které není pro autonomní chování jednotky bezpodmínečně nutné. Jeho funkce umožňuje zlepšování dosavadního chování jednotky a tím větší míru projevů inteligence. Vypuštění centra učení sebou zároveň nese vypuštění centra kognitivního, které by jinak zastávalo funkci nadbytečného prvku implementace. Jeho úkolem by bylo pouze přeposílání požadavků centru plánování, což je možné za těchto podmínek uskutečnít přímo z těla jednotky bez potřeby prostředníka.

5.2 Implementace prostředí

Základem pro lepší porozumění fungování jednotky je pochopení prostředí, ve kterém se tato jednotka pohybuje. Prostředí bylo navrženo a implementováno tak, aby jej bylo možné nahradit libovolným jiným prostředím, které však poskytuje základní rozhraní potřebné pro fungování inteligentní bojové jednotky.

5.2.1 Rozhraní IEnvironment

Aby byla zaručena možnost správného fungování jednotky pro různé typy prostředí, bylo nutné specifikovat co nejjednodušší rozhraní, které musí dané prostředí splňovat, aby v něm mohla jednotka působit. Toho bylo docíleno právě rozhraním `IEnvironment`. Toto rozhraní vyžaduje od prostředí implementaci nejzákladnějších metod potřebných k správnému fungování jednotky:

Metoda GetObjects

```
List<EnvironmentObject^>^ GetObjects(String^ a_unitId, int a_radius);
```

Tato metoda umožňuje jednotce získat informace o okolním prostředí. Jednotka předá prostředí svou jednoznačnou identifikaci, aby bylo možné rozlišit, od které jednotky byl vznesen požadavek. Dále jednotka prostředí informuje o oblasti, kterou je schopna pokrýt současně zvoleným senzorem. Prostředí na tento požadavek zareaguje navrácením objektů vyskytujících se v prostředí, které je daná jednotka schopna zaznamenat.

Metody Move, ChangeDirection, Shoot

```
void Move(String^ a_unitId, double distance);
void ChangeDirection(String^ a_unitId, Direction^ a_direction);
void Shoot(String^ a_unitId, int a_shotSpeed, Direction^ a_direction,
  int a_strength);
```

Jedná se o metody provádějící akce na daném prostředí. Představují tak akce, které je jednotka schopna na zvoleném prostředí vykonávat. Současná implementace jednotky je schopna provádět tyto základní akce: **pohyb vpřed**, **změna směru pohybu**, **změna směru střelby** a **výstřel**. Aby těmito akcemi mohla jednotka ovlivňovat své okolí, musí také toto okolí umožňovat, aby na něm tyto akce mohly být prováděny. Jednotlivý dopad těchto akcí již záleží na konkrétní implementaci daného prostředí.

5.2.2 Třída WorldEnvironment

Tato třída představuje konkrétní implementaci zvoleného prostředí. To bylo implementováno jako prostředí s externím časováním (v určitých časových intervalech je mu dána možnost podniknout změnu svého stavu) a potřebou grafické nadstavby, které není samo o sobě konfigurovatelné - objekty jsou mu předávány externě. Prostředí si udržuje informaci o všech jednotkách a jiných objektech, které se v něm nacházejí, poskytuje implementaci metod z rozhraní `IEnvironment` a umožňuje časově diskrétní změnu svého stavu. Reakce na akce zjištění stavu okolního prostředí, pohyb a změnu směru implementuje intuitivně. V reakci na akci výstřel prostředí vygeneruje střelu, která se v něm následně pohybuje jako nový objekt. Tento objekt není možné zaznamenat senzory. Prostředí umožňuje existenci 3 typů objektů a to: **bojová jednotka**, **zeď** a **střela**.

Kromě metod definovaných rozhraním poskytuje prostředí několik dalších metod, které umožňují zjistit jeho stav pro vykreslení grafickou nadstavbou. Dále obsahuje následující podstatné metody:

Metoda AddObject

```
void AddObject(EnvironmentObject^ a_object);
```

Jelikož prostředí není samo o sobě konfigurovatelné, je potřeba vytvořit možnost přidání nových objektů do prostředí externě. Toto je zajištěno zmíněnou metodou. Veškeré objekty, které se v prostředí následně vyskytují, jsou přidány pomocí této metody.

Metoda ChangeState

```
void ChangeState();
```

Vyvoláním této metody je prostředí sděleno, že nastal okamžik, kdy může změnit svůj vnitřní stav. Změna tohoto stavu probíhá následovně: Nejprve je každému objektu v prostředí zaslána informace, že mu byl přidělen určený časový interval, v rámci kterého může žádat prostředí o provedení některé z akcí. Prostředí všechny požadavky na akce obslouží. Jakmile všechny objekty zažádaly o provedení naplánované akce, prostředí aktualizuje informace o střelách, které se v něm vyskytují. V případě, že došlo ke kolizi střely s některým z objektů, je střela odebrána z prostředí a danému objektu je snížen počet "životů". Pokud počet životů daného objektu klesl pod určenou hranici, je objekt považován za neaktivní a je mu odebrána možnost provádět akce na prostředí. Tento objekt již nikdy nedostane povolení k provedení akce na prostředí.

5.2.3 Třída EnvironmentObject

Stejně tak, jako musí být definováno rozhraní, s kterým může jednotka komunikovat, musí být také definována forma, v jaké jednotka získá informace o okolním prostředí. V budoucnu se může jednat o obraz složený z pixelů, záznam nějakého zvuku, či podobná informace, ze které se bude jednotka pokoušet sestavit si představu o tom, v jakém prostředí se vlastně pohybuje, a co se kolem ní nachází. V této implementaci byl tento aspekt zjednodušen na fakt, že v prostředí se nachází objekty se společnými vlastnostmi a jednotka je schopna tyto objekty vnímat jako celek. Každý objekt v prostředí je proto potomkem právě této třídy specifikující společné vlastnosti o objektu vždy zjistitelné.

Zároveň tato třída reprezentuje objekty vyskytující se v prostředí, jejichž metody jsou vyvolatelné při změně stavu prostředí (např. zjištění, jestli byl daný objekt zasáhnut střelou, umožnění objektu vykonat nějakou akci apod.). V širším záběru se tedy jedná o třídu definující takové objekty v prostředí, které je jednotka schopna zaznamenat jako celek (výjimkou je např. střela, která není zaznamenatelná a proto není implementací této třídy).

Vlastnosti třídy EnvironmentObject

Mezi zjistitelné vlastnosti každého objektu v prostředí patří **pozice**, **směr natočení**, **výška**, **šířka**, **identifikátor**, **typ**, **současný stav**, **přiřazený tým**, **právě vykonávaná akce**, **počet životů**, **informace o průchodnosti** a **schopnost střílet**.

Pozice udává umístění objektu v prostředí. **Směr natočení** určuje, kterým směrem je jednotka natočena. **Šířka** a **výška** objektu udává rozměry objektu. Když jednotka zaznamená nový objekt, může si jej vnitřně libovolně označit. Když jej spatří příště, musí však znovu zjišťovat, jestli se jedná už o dříve spatřený objekt a najít jemu odpovídající označení a informace, s nimiž by mohla pracovat, nebo se jedná o objekt neznámý a tudíž je mu nutno přiřadit identifikátor nový. Jelikož se implementace nezabývá zpracováním senzorických údajů, je využito faktu, že každý objekt musí mít v prosředí jednoznačnou identifikaci. Zjištěním jeho **identifikátoru** a jeho uložením je vždy možno rozpoznat, jestli jsme tento objekt již dříve zaznamenali.

Typ objektu rozlišuje mezi jednotlivými typy objektů v prostředí. V implementovaném prostředí jsou přítomny pouze dva typy objektů a proto může tato vlastnost nabývat pouze odpovídajících dvou hodnot SOLDIER (voják) nebo WALL (zeď).

Současný stav objektu udává jeho míru poškození. Stav může nabývat hodnot HEALTHY (objekt je nepoškozen), DAMAGED (objekt je poškozen), DESTROYED (objekt

je zničen) nebo NOT_DESTROYED (objekt je nepoškozen nebo poškozen, ale ještě není zničen).

Přiřazený tým sjednocuje objekty do skupin, v rámci kterých se snaží spolupracovat. K výběru jsou tři týmy a to TEAM1, TEAM2 a NEUTRAL (objekty neřadící se do žádného týmu, např. civilisté, civilní budovy apod.).

Právě vykonávaná akce pojednává o tom, co daný objekt v současném stavu prostředí vykonává za akci. Tato vlastnost může nabývat hodnot STANDBY (objekt neprovádí žádnou akci), MOVING (objekt se pohybuje), SHOOTING (objekt střílí) a TURNING (objekt se otáčí). Možná je také libovolná kombinace těchto hodnot.

Počet životů představuje životaschopnost objektu. Ve chvíli, kdy hodnota této vlastnosti klesne pod definovanou hodnotu, je objekt považován za zničený (ve stavu DESTROYED). **Informace o průchodnosti** objektu udává, jestli je možné daným objektem projít, nebo zda dojde k zablokování pohybu jednotky při kontaktu s tímto objektem. **Schopnost střílet** představuje informaci o možném ohrožení daným objektem.

Metoda PerformAction

```
void PerformAction();
```

Tato metoda představuje důležitý prvek ve funkcionalitě implementovaného systému. Aby prostředí nebylo statické a objekty v něm mohly podnikat zvolené akce, je nutno zajistit mechanismus, kterým jim přidělíme časový interval, v němž tyto akce mohou provést. To je zaručeno implementací právě této metody v každém z nich. Prostředí tuto metodu vyvolává v každém běhu změny svého stavu pro všechny aktivní prvky v prostředí a tím jim dává šanci zažádat v daném kroku systému o provedení jimi naplánovaných akcí.

5.2.4 Třídy UnitEntry a Wall

Tyto třídy představují implementaci abstraktní třídy `EnvironmentObject` pro 2 konkrétní typy objektů, které je jednotka schopna v prostředí vnímat. Jedná se o reprezentaci inteligentních bojových jednotek (`UnitEntry`) a zdí (`Wall`). Jak již bylo napovězeno, jednotka není schopna vnímat ostatní jednotky do jejich úplných detailů, ale pouze tu část, která je v prostředí reprezentována v rámci třídy `UnitEntry`. Ostatní vlastnosti jednotek, které touto třídou reprezentovány nejsou, jsou jednotce skryty. Je tak dosaženo konceptu, kdy o ostatních jednotkách víme pouze to, co je o nich zjistitelné a nemůžeme pracovat s detaily, které by měly být skryty (záměry a plány jednotky, dosah používaných senzorů apod.). O těchto detailech můžeme pouze usuzovat z jejich vnějších projevů.

5.3 Implementace jednotky

V rámci základní implementace jednotky byla vybrána implementace všech navržených center, kromě centra učení a kognitivního centra (viz podkapitola 5.1). Jednotka je externě synchronizována prostředím, ve kterém se nachází, a získává tak od něj časové intervaly, v rámci kterých může plánovat své akce, zjišťovat stav okolního prostředí, nebo na tomto prostředí akce provádět.

5.3.1 Třída FightingUnit

Jedná se o třídu představující tělo celé inteligentní bojové jednotky. Sdružuje v sobě všechna navržená centra, přičemž si ukládá přímý odkaz na reaktivní centrum, centrum plánování a bázi znalostí, kterým přeposílá příslušné požadavky. Veškerá externí synchronizace z okolního prostředí prochází touto třídou, tudíž je právě ona zodpovědná za aktivaci příslušných center a jejich vzájemnou komunikaci. Stejně tak v případě, že jsou jednotce externě zadány některé akce k provedení nebo cíle k naplánování, jsou předány této třídě, která je již přepošle efektorům k vykonání (pokud se jedná o přímo specifikované akce) nebo plánovacímu centru k naplánování (pokud se jedná o obecněji zadaný cíl).

Metoda ScheduleAction

```
void ScheduleAction(ActionBase^ a_action);
```

Jednotka je schopná v prostředí autonomně působit a pohybovat se vzhledem k naplánovanému cíli. Může však vzniknout požadavek od uživatele, aby jednotka napříč svému současnému plánu provedla akci, která naplánována nebyla. Využitím této metody může uživatel zadat jednotce takovou akci k provedení. Požadavek je následně přeposlán centru efektorů, aby naplánovalo její provedení.

Metoda PlanAction

```
void PlanAction(PlanCommand^ a_planCommand);
```

V případě, že uživatel nechce předat jednotce pouze jedinou akci k provedení, ale chce jednotce specifikovat další cíl, kterého by se měla snažit dosáhnout, může pomocí této metody přidat do množiny cílů k dosažení další specifikovaný cíl. Jednotka požadavek na dosažení specifikovaného cíle přepošle plánovacímu centru, které, jakmile jsou dosaženy všechny předchozí cíle, naplánuje jeho dosažení.

Metoda PerformAction

```
void PerformAction();
```

Ve chvíli, kdy se prostředí chystá přidělit jednotce daný časový interval, vyvolá tuto metodu. Jednotka tak dostává možnost zjistit nové informace o svém prostředí, případně jej ovlivnit vykonáním vybraných akcí. Konkrétní implementace této metody tedy určuje, jak jednotka naloží s přiděleným časovým intervalem.

V současné implementaci jednotka nejprve zjistí, zda jsou naplánovány nějaké akce k provedení. V případě, že v současné chvíli nejsou naplánovány žádné akce, požádá plánovací centrum o vytvoření plánu. Plánovací centrum podle aktuálního cíle vytvoří seznam akcí, jejichž provedením je možné daného cíle dosáhnout, a jednotka si jej uloží jako posloupnost akcí k provedení v následujících krocích.

V případě, že jsou naplánovány nějaké akce k provedení, jednotka vezme posloupnost těchto akcí a postupně zadává akce z této posloupnosti reaktivnímu centru k provedení. To se snaží předat tyto akce k provedení centru efektorů, příp. senzorickému centru, které je přeposílá jednotlivým efektorům, příp. senzorům. Pokud se podařilo danou akci předat až na nejnižší úroveň, je odebrána z posloupnosti akcí k provedení a jednotka se snaží předat

akci další. Tak pokračuje až do chvíle, kdy není možné některou akci předat, nebo není posloupnost naplánovaných akcí prázdná.

Jakmile jsou všechny potřebné akce předány příslušným efektorům, příp. senzorům, předá jednotka reaktivnímu centru povolení k provedení všech právě předaných akcí. Pokud z nějakého důvodu nebylo možné tyto akce provést (prostředí se významně změnilo, na nižší úrovni došlo k reakci na kritickou událost apod.), jednotka zneplatní celou množinu akcí naplánovaných k provedení. Plánovacímu centru je následně oznámeno, že došlo k zneplatnění plánu a je proto potřeba znovu naplánovat dosažení současného cíle.

5.3.2 Třída ReactiveCenter

Tato třída představuje implementaci navrženého reaktivního centra inteligentní bojové jednotky (viz 4.4.4). Sdružuje v sobě senzorické centrum a centrum efektorů, a ukládá si přímé odkazy jak na tato centra, tak také na bázi znalostí. Reaktivní centrum je zodpovědné zejména za přeposílání akcí k provedení senzorickému centru a centru efektorů, vyhledávání kritických událostí ve znalostech získaných od senzorů a reakce na tyto události, čímž uzavírá reaktivní smyčku agenta. Specifikace kritických událostí i reakcí na ně jsou mu určeny danou množinou reaktivních pravidel.

Metoda ScheduleAction

```
bool ScheduleAction(ActionBase^ a_action);
```

Tato metoda přeposílá předanou akci odpovídajícímu centru. Rozhoduje, zda je daná akce určena pro centrum efektorů nebo centrum senzorické a snaží se naplánovat tuto akci k provedení určeným centrem. Návratovou hodnotou jednotce oznámí, zda se danou akci podařilo naplánovat k provedení nebo naopak nepodařilo.

Metoda PerformAction

```
bool PerformAction();
```

Centrum vyvoláním této metody dostává pokyn k uskutečnění dříve naplánovaných akcí senzorického centra a centra efektorů. Reakce na tento pokyn byla naimplementována tak, že centrum dá nejprve možnost k provedení naplánovaných akcí centru senzorickému. Tento krok je zvolen pro případ, kdyby senzory zaznamenaly kritickou událost v prostředí, na kterou by bylo nutné okamžitě zareagovat.

Senzorické centrum na pokyn k uskutečnění naplánovaných akcí odpovídá navrácením znalostí získaných o okolním prostředí. Reaktivní centrum se v těchto znalostech snaží nalézt vzory, které by znamenaly výskyt kritické události v prostředí. Takový vzor je nalezen, pokud jsou některými znalostmi splněny podmínky některého z reaktivních pravidel, které byly pro toto centrum specifikovány. V reaktivních pravidlech jsou zároveň specifikovány i akce, jejichž provedením se má na výskyt těchto událostí zareagovat.

Při nalezení kritické události jsou z pravidel, jejichž podmínky byly splněny, získány příslušné akce a doposud plánované akce v centrech efektorů jsou těmito akcemi nahrazeny. Tělo jednotky je o tomto informováno sdělením, že došlo ke zneplatnění jeho plánu a akce pro efektory byly přeplánovány. Do doby, než je obslužná rutina vzniklá v reakci na kritickou událost dokončena, není možné naplánovat novou akci (taková akce vždy skončí zneplatněním). Proto je doporučeno volit obslužné rutiny co nejkratší, aby nedocházelo

k příliš dlouhým intervalům, kdy jednotka není schopna jakkoliv reagovat, protože právě bezmyšlenkovitě plní příslušné reakce.

V případě, že nejsou pomocí senzorů zaznamenány takové znalosti v prostředí, které by představovaly výskyt kritické události, je centru efektorů dán souhlas k provedení dříve naplánovaných akcí a tyto jsou následně provedeny.

Metoda SpecifyActions

```
List<ActionBase^>^ SpecifyActions(List<ActionBase^>^ a_actions);
```

Jelikož jsou senzory a efektory schopny vykonávat pouze atomické akce (takové, které není možné rozgenerovat na akce více specifické), je nutno mít mechanismus, který je schopen obecnější akce transformovat na akce atomické. Každá obecnější akce sebou nese informaci o tom, jak může být na takové akce rozgenerována, a právě tato metoda reaktivního centra vyvolává rozgenerování těchto akcí na akce atomické. Všechny naplánované akce by měly být nejprve rozgenerovány na akce atomické.

5.3.3 Třída ReactiveRule

Tato třída představuje reaktivní pravidla určená pro konfiguraci chování reaktivního centra (4.4.4). Každé pravidlo obsahuje množinu podmínek a množinu akcí, a definuje tak kritickou událost a akce, kterými je na její výskyt potřeba zareagovat.

Metoda TestPrecondition

```
bool TestPrecondition(List<RepresentationObject^>^ a_representations);
```

Testuje, zda předaná množina znalostí o objektech splňuje množinu podmínek pravidla. Množina podmínek je splněna tehdy, když v předané množině znalostí existuje množina znalostí o některém z objektů taková, že splňuje každou podmínku z dané množiny podmínek. Pokud znalosti o některém objektu splňují všechny podmínky, znamená to, že se v předaných znalostech vyskytla kritická událost, na kterou je potřeba zareagovat.

Metoda GetActions

```
List<ActionBase^>^ GetActions();
```

Metoda vrací seznam akcí, které jsou asociovány k reaktivnímu pravidlu. Posloupnost těchto akcí určuje, jaká by měla být reakce na výskyt události určené množinou podmínek.

Metoda GetPassedIdList

```
List<String^>^ GetPassedIdList();
```

V případě, že se v předaných znalostech vyskytla kritická událost, je možné zjistit identifikátor objektu, který výskyt této události způsobil. Metoda vrací seznam všech objektů, které způsobily splnění všech podmínek reaktivního pravidla.

5.3.4 Třída SensorCenter

Třída představuje implementaci senzorického centra (viz 4.4.2). Jedná se o jediný prvek systému, který je schopen získávat informace o okolním prostředí. Sdružuje v sobě jednotlivé senzory, z nichž vždy právě jeden zvolený senzor je aktivní a vykonává předané akce.

Metoda ScheduleAction

```
bool ScheduleAction(ActionBase^ a_action);
```

V současné chvíli senzorické centrum umožňuje provést v daném časovém intervalu neomezený počet akcí. Metoda uloží každou předanou akci k vykonání právě aktivním senzorem.

Metoda ClearScheduledActions

```
void ClearScheduledActions();
```

Přestože byly naplánovány akce k vykonání, mohlo ve vyšších vrstvách dojít ke zneplatnění stanoveného plánu a naplánované akce již mohou být neplatné (nyní je prioritou provést např. akce reagující na kritickou událost). Metoda zruší všechny naplánované akce jejich smazáním ze seznamu akcí k vykonání.

Metoda PerformAction

```
List<RepresentationObject^>^ PerformAction();
```

Tato metoda provádí naplánované akce na okolním prostředí. Jelikož je v současnosti možné provést na úrovni senzorů jedinou akci, a to **akci zjištění stavu okolního prostředí** (akce **scan**), je v konečném důsledku vždy vyvolán právě aktivní senzor, který má za úkol vrátit současný stav okolního prostředí.

Senzor vrací stav okolního prostředí jako seznam objektů třídy `EnvironmentObject`. Tyto samotné by již bylo možné vrátit jako nalezené znalosti, ale vznikly by tím dva problémy. Jelikož bychom si uložili přímý odkaz na tyto objekty, získali bychom tak přístup ke vždy aktuálním informacím o daných objektech, bez nutnosti zjišťovat stav okolního prostředí. To by bylo jistě užitečné, ale odporující realitě. Druhý problém je ten, že v budoucnu neočekáváme návrat celých objektů, ale pouze některých příznaků, zvuků nebo částí obrazu zjištěných na okolním prostředí, ze kterých teprve musíme zjistit co se kolem nás vlastně nachází. Vznikne tak potřeba provést zpracování obrazu, zvuku apod., abychom vůbec z navrácených informací zjistili nějaké znalosti o okolním prostředí. Proto je ještě samotnému předání zjištěných znalostí z prostředí předřazena fáze extrakce těchto znalostí z informací navrácených od senzorů.

Fáze extrakce znalostí je v současnosti implementována jako pouhé překopírování důležitých vlastností objektu do struktury, která jej bude v bázi znalostí reprezentovat. Přitom je podle typu zjištěného objektu specifikováno, jakého tvaru tento objekt nabývá, aby mohlo být s tímto tvarem plánováno pro případ že bychom jej chtěli obejít, schovat se za něj apod.

5.3.5 Třída SensorBase

Abstraktní třída určující vlastnosti, které má mít každý senzor definovány, a metody, které má implementovat. Zděděním této třídy a specifikací vlastností je možné vytvořit senzor komunikující s okolním prostředím.

Vlastnosti senzorů

Jedinou v současnosti implementovanou vlastností senzorů je **dosah senzoru**.

Metoda Scan

```
List<EnvironmentObject^>^ Scan();
```

Tato metoda komunikuje s okolním prostředím a žádá jej o poskytnutí všech objektů, které se nacházejí v dosahu tohoto senzoru. Jejím provedením je možné zjišťovat stav okolního světa.

5.3.6 Třída EffectorCenter

Třída představuje implementaci centra efektorů (4.4.3). Jedná se o jediný prvek systému, který umožňuje svým působením ovlivňovat okolní prostředí. Sdružuje v sobě podcentra zodpovědná za provádění příslušných typů akcí (zde konkrétně pohybové centrum a centrum střelby).

Metoda ScheduleAction

```
bool ScheduleAction(ActionBase^ a_action);
```

Centrum rozesílá předanou akci všem centrům, které jsou v něm registrovány. Pokud alespoň jedno z nich umožnilo naplánování předané akce, informuje, že byla akce naplánována k provedení. Jinak oznámí, že nebylo možné naplánovat předanou akci.

Metoda ClearScheduledActions

```
void ClearScheduledActions();
```

Metoda ruší všechny doposud naplánované akce k provedení (viz stejná metoda v podkapitole 5.3.4).

Metoda PerformAction

```
void PerformAction();
```

Vyvoláním této metody je centru oznámeno, že mu byl přidělen předem specifikovaný časový interval, který může využít k provedení naplánovaných akcí. Centrum dává registrovaným centrům postupně povolení k provedení jedné naplánované akce v pořadí, ve kterém byly akce naplánovány. Tak činí až do chvíle, kdy již nejsou žádné naplánované akce k provedení.

5.3.7 Třídy ActorCenterBase, MotionCenter a ShootCenter

Třída `ActorCenterBase` je abstraktní třídou udávající, jaké metody musí implementovat dané centrum, aby mohlo být registrováno v centru efektorů. Třídy `MotionCenter` a `ShootCenter` jsou takovými implementacemi, představujícími centrum pohybu a centrum střelby. Všechna tato centra mají informaci o délce časového intervalu, který jim je v jednom kroku systému přidělen a v rámci kterého mohou plánovat své akce. Každá implementace si také definuje, jaké akce je schopna provádět a jak moc náročné na čas je provedení příslušné

akce. Podle toho se odvíjí, kolik akcí je schopno dané centrum v rámci přiděleného časového intervalu naplánovat a následně provést. Navíc v sobě jednotlivá centra sdružují efektory jim příslušející, z nichž vždy právě jeden zvolený efektor je aktivní a vykonává akce předané příslušnému centru.

Metoda ScheduleAction

```
bool ScheduleAction(ActionBase^ a_action);
```

Tato metoda předává implementovanému centru akci k naplánování. Centrum ověří, jestli se jedná o akci, kterou je schopno provést. Pokud se jedná o typ akce, který toto centrum nemá ve svém poli působnosti (např. akce pohyb pro centrum střelby), vrátí informaci, že danou akci nebylo možné naplánovat.

Pokud se jedná o akci, kterou je dané centrum schopno provést, naplánuje ji a odečte si ze specifikovaného časového intervalu, který má k dispozici, čas potřebný k provedení dané akce. Pokud je časová náročnost akce větší než zbývající část přiděleného časového intervalu, akce naplánována není a je vrácena informace o tom, že se ji naplánovat nepodařilo.

Prakticky ve chvíli, kdy žádné centrum nemohlo z časových důvodů naplánovat předanou akci, končí úkolování jednotlivých center vyššími vrstvami a přichází fáze, kdy vyšší vrstvy vyvolají požadavek na provedení daných akcí.

Metoda PerformAction

```
void PerformAction();
```

Tato metoda provede jednu akci naplánovanou v příslušném centru. Nejprve je rozhodnuto, o kterou akci se pojedná, a poté je provedením této akce zaúkolován právě aktivní efektor. Přitom je uvolňována využitelná časová kapacita pro plánování dalších akcí.

5.3.8 Třídy MotionBase a ShootBase

Abstraktní třídy určující vlastnosti, které má mít každý efektor definovány, a metody, které má implementovat, aby mohl být zařazen jako efektor pohybového centra, resp. centra střelby. Zděděním této třídy a specifikací vlastností je možné vytvořit efektory různých vlastností a možností ovlivňovat okolní prostředí.

Vlastnosti efektorů

Efektory pohybu mají specifikovánu **rychlost**, s jakou se může jednotka pohybovat. Efektory střelby pak mají určenu **rychlost vystřeleného projektilu** a **sílu projektilu**. Tyto údaje udávají, jak rychle se vystřelený projektil bude v prostředí pohybovat a jak velkou škodu způsobí zasaženému objektu.

Metoda ChangeDirection

```
void ChangeDirection(Direction^ a_newDirection);
```

Jedná se o metodu implementovanou oběma centry. Její vyvolání se specifikovaným směrem natočení způsobuje změnu směru pohybu jednotky, příp. změnu jejího směru střílení (podle toho, pro které centrum je metoda vyvolána).

Metoda Step

```
void Step();
```

Metoda pohybového centra způsobující posun jednotky v prostředí o jeden krok v současném směru natočení efektoru. Velikost kroku je závislá na rychlosti právě aktivního efektoru.

Metoda Shoot

```
void Shoot();
```

Metoda centra střelby způsobující výstřel jednotky ve směru současného natočení efektoru. Rychlost a síla výstřelu je určena nastavením právě používaného efektoru.

5.3.9 Třída ActionBase

Jedná se o abstraktní třídu udávající předpis pro definici nové akce. Každá akce musí mít určenu **časovou náročnost** svého vykonání, **jméno** (pojmenování daného typu akce) a **typ** (pro rozlišení akcí se stejným jménem, ale určených pro různá centra). Na úrovni efektorů vyvolává jméno akce provedení odpovídající akce (např. jméno shoot akce výstřel vyvolává metodu `Shoot()` centra střelby, jméno step akce krok metodu `Step()` pohybového centra) a typ akce rozlišuje, kterému centru je daná akce určena (typ **motion** pro akce pohybového centra, typ **shoot** pro akce centra střelby a typ **sensor** pro akce senzorického centra).

Některé konkrétní implementace akcí sebou navíc nesou **hodnotu** dané akce (např. akce změny směru natočení sebou nese informaci, kterým směrem se natočit).

Typy akcí

V současné implementaci je používáno 5 druhů akcí:

Akce **krok** se jménem **step**, typem **motion**, bez dodatečné hodnoty. Akce vyvolává metodu `Step()` pohybového efektoru, způsobující posun o jeden krok ve směru natočení pohybového centra. Náročnost této akce byla stanovena na 90% přiděleného časového intervalu.

Akce **pohyb** se jménem **move**, typem **motion** a hodnotou udávající vzdálenost, o kterou se má jednotka posunout. Tato akce je schopna se rozgenerovat na odpovídající množinu akcí krok. Náročnost této akce je nulová, protože se nejedná o akci atomickou. Její náročnost je určena množinou z ní vygenerovaných atomických akcí.

Akce **změna směru** se jménem **direction**, typem **motion** nebo **shoot** a hodnotou udávající nový směr natočení centra specifikovaného typem akce. Akce vyvolává metodu `ChangeDirection()` efektoru příslušného centra, způsobující změnu směru natočení tohoto centra. Náročnost této akce byla stanovena na 10% přiděleného časového intervalu.

Akce **výstřel** se jménem **shoot**, typem **shoot**, bez dodatečné hodnoty. Akce vyvolává metodu `Shoot()` na úrovni efektoru centra střelby. Náročnost této akce byla stanovena na 90% přiděleného časového intervalu.

Akce **zjištění stavu** okolního prostředí se jménem **scan**, typem **sensor**, bez dodatečné hodnoty. Akce vyvolává metodu `Scan()` aktivního senzoru, zjišťující stav okolního prostředí. Náročnost této akce byla stanovena na 100% přiděleného časového intervalu.

Jelikož jsou různá centra na sobě nezávislá co se týká přiděleného časového intervalu pro každé centrum, je vhodné se zamyslet nad řazením akcí. Pokud bychom zadali jednotce sekvenci akcí <move, move, direction>, je schopna tuto sekvenci provést ve dvou

krocích (spotřebuje na její provedení 2 časové intervaly). Stejně časově náročná by byla také sekvence akcí <move, shoot, move, shoot, direction>. To z toho důvodu, že centrum střelby má přidělen vlastní časový interval, jelikož jsou jeho akce nezávislé na akcích centra pohybu. Pokud bychom však zvolili sekvenci akcí <move, move, shoot, shoot, direction>, již bychom potřebovali časové intervaly tři. Přestože by bylo možné provést akce výstřelu paralelně s akcemi pohybu, a tak tuto sekvenci vykonat jen ve dvou krocích, nebyla by pak zachována posloupnost akcí, což je mnohem podstatnější.

Metoda IsAtomic()

```
bool IsAtomic();
```

Efektory jsou schopny vykonávat pouze atomické akce. Atomické akce jsou takové akce, které již nelze rozložit na množinu více specifických akcí. Atomická akce musí být dokončitelná v časovém intervalu, který je určen pro provádění akcí na prostředí. Metoda vrací informaci, zda se o takovou akci jedná.

Metoda GenerateActions()

```
List<ActionBase^>^ GenerateActions();
```

Ne každá akce je atomická (např. akce pohyb). Neatomické akce jsou však schopny samy sebe rozgenerovat na množinu akcí atomických a to vyvoláním právě této metody.

5.3.10 Třídy KnowledgeBase, KnowledgeTable a KnowledgeTableId

Třída `KnowledgeBase` představuje bázi znalostí jednotky (viz podkapitola 4.4.5). Znalosti v ní uložené představují současný model světa jednotky, a proto je báze znalostí důležitým prvkem inteligentního chování jednotky. Kdyby jednotka nebyla schopna uchovávat znalosti o okolním prostředí a vytvářet si jeho model, těžce by se bez těchto znalostí plánovalo jiné než-li reaktivní chování.

Veškeré znalosti jsou ukládány do tabulek, které odpovídají předpisu definovanému abstraktní třídou `KnowledgeTable`. Tyto tabulky mohou umožňovat nejrůznější druhy ukládání znalostí, případně se každá z nich může specializovat na jiný typ znalostí.

Třída `KnowledgeTableId` je implementací této abstraktní třídy, která ukládá znalosti jako seznam reprezentací objektů (reprezentace objektu představuje naše znalosti o něm), které byly v prostředí zaznamenány. Znalosti o těchto objektech jsou v seznamu uloženy pod jednoznačným identifikátorem, který danému objektu přísluší (což ulehčuje aktualizaci znalostí o daném objektu).

Metody AddKnowledge a AddKnowledges

```
void AddKnowledge(RepresentationObject^ a_knowledge);
void AddKnowledges(List<RepresentationObject^>^ a_knowledges);
```

Obě dvě metody slouží pro uložení znalostí do tabulek báze znalostí. Liší se pouze počtem znalostí, které ukládají. V případě, že již v bázi znalostí máme uloženy znalosti o daném objektu, a tyto znalosti se liší od znalostí právě získaných, pak provedeme jejich aktualizaci. V bázi znalostí totiž chceme vždy uchovávat co nejaktuálnější znalosti.

Metody GetRepresentation a GetKnowledges

```
RepresentationObject^ GetRepresentation(String^ a_identification);
List<RepresentationObject^>^ GetKnowledges();
```

Tyto metody jsou si hodně podobné. Obě dvě vracejí znalosti uložené v bázi znalostí. Metoda `GetRepresentation()` však vrací znalosti pouze o jednom specifikovaném objektu. Pokud tedy chceme zjistit informace o jednom konkrétním objektu, je vhodné použít tuto metodu. Metoda `GetKnowledges()` naopak vrací všechny znalosti uložené v bázi znalostí.

Metoda GetNewKnowledges

```
List<RepresentationObject^>^ GetNewKnowledges();
```

Metoda navrací nově zjištěné znalosti. Jedná se buď o znalosti o objektech, o nichž jsme získali znalosti poprvé, nebo o znalosti o objektech, o nichž jsme již informace dříve měli, ale získali jsme informace aktuálnější. Takové znalosti jsou považovány za nové do chvíle, než jsou označeny za staré.

Metoda ClearNewKnowledges

```
void ClearNewKnowledges();
```

Všechny nově zjištěné znalosti jsou označeny za staré, vyvoláním právě této metody.

5.3.11 Třída RepresentationObject, RepresentationCircle a RepresentationRectangle

Stejně tak, jako je definována forma, v jaké jednotka získá informace o okolním prostředí, musí být také definována forma, v níž si bude získané znalosti ukládat. Tuto formu definujeme třídou `RepresentationObject`. Prakticky se jedná pouze o kopii třídy `EnvironmentObject`, která má v sobě uloženy dodatečné informace a definovány metody umožňující lehčí práci s reprezentací znalostí o jednotlivých objektech. Jedná se například o hraniční body a body ukrytí. Třída poskytuje také metody umožňující výpočet těchto bodů v libovolné vzdálenosti od příslušného objektu.

Třída `RepresentationCircle` představuje implementaci třídy `RepresentationObject` pro objekty kruhového tvaru (v současné implementaci jsou jimi pouze vojáci). Třída `RepresentationRectangle` pak implementaci pro objekty tvaru obdélníkového (zdi). Výpočet jednotlivých vlastností se odvíjí od tvaru těchto reprezentovaných objektů.

Vlastnosti třídy RepresentationObject

Jak již bylo zmíněno, vlastnosti této třídy odpovídají vlastnostem třídy `EnvironmentObject`. Navíc však obsahuje informace o **hraničních bodech** a **bodech ukrytí**.

Hraniční body jsou body, které tvoří význačné hranice neprůchozích objektů. V případě, že se chce jednotka pohybovat v prostředí, je pomocí senzorů schopna zjistit, kde se nachází volný prostor, který k tomuto pohybu může využít. Vzhledem k výpočetní náročnosti a velkému počtu takových bodů byl zvolen přístup, kdy každá reprezentace objektu v sobě nese informaci o ohraničujících bodech. Získáním těchto bodů, a se znalostí šířky pohybujícího se objektu, je pak možné vypočítat, kudy se pohybovat, aby nedošlo ke kolizi s neprůchozím objektem.

Body ukrytí jsou body, které určují pozice kolem příslušného objektu, které jsou vhodné k ukrytí před zrakem a střelbou nepřítele. V implementovaném programu jsou za tyto body považovány závětrné strany zdí, u nichž se předpokládá, že alespoň jeden z bodů ukrytí je vhodný pro ukrytí se před palbou nepřítele. Pak již zbývá jenom vybrat takový bod ukrytí, který je nepřítelem nejméně ohrožen. Pro jednotky nebyly body ukrytí implementovány, proto je tato vlastnost využita pouze u zmíněných zdí.

5.3.12 Třída PlanningCenter

Třída představuje implementaci plánovacího centra jednotky (viz podkapitola 4.4.8). Toto centrum sebou nese informaci o zadaných cílech a informaci o cíli, který byl v současnosti vybrán k dosažení. Navíc si uchovává přímý odkaz na bázi znalostí a množinu plánovacích pravidel, která jsou sice podobná jako pravidla reaktivní (viz podkapitola 5.3.3), ale logicky se od nich liší. Akce pravidel centra plánování, které vznikají v reakci na splnění specifikovaných podmínek, si žádají delší čas k naplánování a nejsou proveditelné okamžitě. Nejedná se tedy o pravidla, která určují okamžité reakce na nastalé události v prostředí, ale spíše udávají, za jakého stavu prostředí se máme zabývat tvorbou jakého plánu. Splnění jejich podmínek také nenarušuje dříve vytvořený plán, spíše určuje chování jednotky ve chvíli, kdy volí cíl, kterého by měla v současné chvíli dosáhnout, a dává jí tak možnost výběru z více cílů k dosažení (každé splněné pravidlo poskytuje nový cíl). Jedná se třeba o naplánování akce "schovej se" v případě, kdy je v blízkosti spatřena nepřátelská jednotka. Tato akce nejdříve vyžaduje nalézt bezpečná místa, kde je možné se uschovat, následně vybrat místo k tomuto účelu nejvhodnější a potom ještě naplánovat cestu k jeho dosažení.

Aby nebylo veškeré plánování soustředěno do jednoho centra, jsou vytvořena centra další, kterým jsou rozděleny skupiny plánovacích úkolů. Tato se u centra plánování zaregistrují, a v případě příchozího požadavku na naplánování určitého cíle je tento požadavek přesměrován příslušnému centru. Příkladem takového centra je implementované centrum pro plánování cesty, které má na starosti veškeré plánování týkající se dosažení určité pozice v prostředí.

Metoda CreatePlan

```
List<ActionBase^>^ CreatePlan();
```

Metoda je vyvolána ve chvíli, kdy od plánovacího centra požadujeme vytvoření plánu. Metoda vrací plán jako seznam akcí, jejichž provedením by mělo být možné dosáhnout stanoveného cíle.

Centrum nejprve zjistí, jestli má k dispozici znalosti o okolním světě. Pokud tyto znalosti má, aktualizuje svou množinu cílů (zjistí, které z cílů již byly v současném stavu prostředí dosaženy, a tyto z množiny cílů odebere). Pokud byl v minulosti vytvořen plán k dosažení, jehož cíl doposud nebyl dosažen, je zváženo, zda je nutné provést přeplánování tohoto již hotového plánu, nebo můžeme pokračovat v jeho provádění. Pokud je možné pokračovat v jeho provádění, centrum vrátí akce k dosažení dalšího kroku tohoto plánu, a prozatím nevytváří plán nový.

Pokud byly cíle dříve vytvořeného plánu dosaženy, nebo doposud nebyl žádný plán vytvořen, je vybrán první z cílů, jejichž dosažení bylo plánovacímu centru zadáno a naplánuje se jeho dosažení. Přitom se berou v potaz také cíle plánovacích pravidel, jejichž podmínky byly splněny. Dosažení cílů splněných plánovacích pravidel je v současné době upřednostněno před cíli zadanými metodou `PlanAction()`.

Metoda PlanAction

```
void PlanAction(PlanCommand^ a_planCommand);
```

Chceme-li plánovacímu centru specifikovat cíl, jehož dosažení by mělo v budoucnu naplánovat, učiníme tak vyvoláním této metody.

Metoda InvalidatePlan

```
void InvalidatePlan();
```

Tuto metodu je potřeba vyvolat v případě, kdy došlo k události, která zneplatňuje doposud prováděný plán (byla provedena akce na prostředí, která nebyla předem plánována, případně se toto prostředí významně změnilo). Centrum tak ví, že je potřeba znovu vytvořit plán dosažení právě dosahovaného cíle, protože mohlo dojít k jeho narušení.

5.3.13 Třída PlanCenterBase

Tato abstraktní třída představuje předpis, který musí každé centrum implementovat, aby mohlo být registrováno v plánovacím centru. Jediným požadavkem na toto centrum je schopnost vytvářet kroky plánu v reakci na zadaný cíl.

Metoda CreatePlanSteps

```
List<PlanStepBase^>^ CreatePlanSteps(PlanCommand^ a_planStep);
```

Tato metoda dostává na vstupu zadaný cíl, jehož dosažení má naplánovat. Na výstupu je očekáván seznam takových akcí plánovacího centra, které je již plánovací centrum schopno rozgenerovat na akce určené efektorům.

5.3.14 Třída PathPlanCenter

Třída je implementací třídy `PlanCenterBase`. Je tedy zaregistrovatelná v plánovacím centru. Třída je schopna naplánovat dosažení stanoveného cíle cesty pomocí algoritmu A*. Přitom využívá hraničních bodů jednotlivých objektů v prostředí. Tuto cestu se snaží vždy hledat tak, aby byla co nejkratší, avšak centrum může být konfigurováno, aby při plánování byly zohledněny i jiné vlivy (například když je nejkratší cesta ohrožena nepřítelem, je vhodnější zvolit cestu delší, ale neohroženou).

Vlastnosti třídy PathPlanCenter

Vlastnosti této třídy zároveň udávají konfiguraci tohoto centra. Jedná se o vlastnosti **tolerance** (tolerance, s jak moc velkou přesností musíme dosáhnout dané pozice), **hide_distance** (jak daleko od okraje objektu se má jednotka ukrýt při plánování akce "schovej se"), **max_hide_distance** (umožňuje rozlišit body ukrytí, které ještě může jednotka k ukrytí zvažovat a které jsou pro ni již moc daleko), **safe_enemy_distance** (vzdálenost protivníka od naplánované cesty, která je již považována za bezpečnou) a **enemy_penalty** (míra penalizace úseku zvolené cesty, který je v nebezpečné vzdálenosti od protivníka). Všechny tyto vlastnosti ovlivňují volené akce a plánovanou cestu, a možnost jejich nastavení poskytuje další možnosti rozlišení chování různých jednotek.

5.3.15 Třídy PlanCommand, PlanCommandReach

Třída `PlanCommand` je pouhým předpisem cíle, kterého má jednotka dosáhnout. Nese v sobě **jméno** cíle, jehož má být dosaženo, a některé třídy od ní zděděné nesou také **hodnotu** specifikující předaný cíl.

Třída `PlanCommandReach` je příkladem zděděné třídy od třídy `PlanCommand`, která kromě toho, že nese informaci, že cílem je dosáhnout specifikované pozice, nese v sobě zároveň i hodnotu pozice, které má být dosaženo.

Typy cílů

V současné chvíli jsou implementovány pouze 2 cíle, které je možné zadat plánovacímu centru:

Cíl **dosáhnout požadované pozice**, se jménem **reach** a hodnotou specifikující pozici, které má být dosaženo. Dosažení tohoto cíle je plánováno centrem pro plánování cesty, které se snaží nalézt takovou cestu a sekvenci akcí, jejichž provedením by byl změněn stav okolního prostředí natolik, že by se jednotka nacházela na určené pozici.

Cíl **schovat se**, se jménem **hide** a bez specifikované další hodnoty. Dosažení tohoto cíle je plánováno centrem pro plánování cesty, které se snaží nalézt takovou pozici v prostředí a dosažení této pozice, aby jednotka v případě, že by se na tuto pozici zvládla přesunout, byla ukryta před zraky a střelbou nepřátel.

5.3.16 Třídy PlanStepBase a PlanReach

Třída `PlanStepBase` je abstraktní třídou určující jak má vypadat jeden krok plánu. Každý definovaný cíl lze dosáhnout pomocí plánu, který je rozdělen na několik kroků, z nichž každý má svůj vlastní podcíl. Postupným dosažením podcílů jednotlivých kroků je možné dosáhnout cíle celého plánu, který tyto kroky vygeneroval. Každý z těchto kroků přitom umí zjistit, jestli byl jeho podcíl splněn, jestli jej některá ze zjištěných znalostí nezneplatnila, a také je schopen vygenerovat takové akce, které umožní dosažení tohoto podcíle.

Třída `PlanReach` je konkrétní implementací této třídy, která představuje krok plánu pro dosažení stanovené pozice.

Metoda GenerateActions

```
List<ActionBase^>^ GenerateActions();
```

Tato metoda je schopna vygenerovat akce, jejichž provedením by se měl změnit stav světa tak, aby byl ve výsledném stavu světa splněn podcíl kroku současného plánu.

Metoda TestSuccess

```
bool TestSuccess();
```

Tato metoda zjišťuje, zda bylo dosaženo podcíle, který má tento krok plánu stanoven.

Metoda IsValid

```
bool IsValid(RepresentationObject^ a_representation);
```

Vyvoláním této metody můžeme zjistit, zda předaná znalost nezneplatnila provedení tohoto kroku plánu. Pokud jej zneplatnila, bude potřeba vytvořit novou posloupnost kroků plánu, které budou validní.

5.3.17 Konfigurační soubory

Všechny jednotky a jejich centra jsou plně konfigurovatelné pomocí konfiguračních XML souborů. Stejnou měrou je konfigurovatelné i jejich chování, a to na úrovni definovaných reaktivních a plánovacích pravidel.

Konfigurace jednotky

Každé jednotce můžeme konfigurovat její vlastnosti a také vlastnosti jejích center. Navíc můžeme specifikovat, které senzory a efektory má mít daná jednotka k dispozici. Konfigurace jednotky pak probíhá předáním konfiguračního souboru metodě `ConfigurateFromXml()` příslušné jednotky . Příklad takového konfiguračního souboru vidíme zde:

```
<SOLDIER>
  <PROPERTY name="width" value="20"/>
  <PROPERTY name="hitpoints" value="100"/>
  <PLANNING_CONFIG>
    <PATHPLAN_CONFIG>
      <PROPERTY name="tolerance" value="2"/>
      <PROPERTY name="hide_distance" value="40"/>
      <PROPERTY name="max_hide_distance" value="300"/>
      <PROPERTY name="safe_enemy_distance" value="20"/>
      <PROPERTY name="enemy_penalty" value="300"/>
    </PATHPLAN_CONFIG>
  </PLANNING_CONFIG>
  <SENSOR_CONFIG>
    <SENSOR type="eye" radius="50"/>
  </SENSOR_CONFIG>
  <EFFECTOR_CONFIG>
    <MOTION_CONFIG>
      <EFFECTOR type="leg" speed="2"/>
    </MOTION_CONFIG>
    <SHOOT_CONFIG>
      <EFFECTOR type="gun" speed="3" strength="25"/>
    </SHOOT_CONFIG>
  </EFFECTOR_CONFIG>
</SOLDIER>
```

XML soubor, který začíná tagem <SOLDIER> představuje soubor konfigurace jednotky. Této jednotce můžeme konfigurovat vlastnosti pomocí XML tagů <PROPERTY>, kde atribut **name** udává jméno konfigurované vlastnosti a atribut **value** udává hodnotu, na kterou tuto vlastnost nastavujeme. Vlastnosti, které nejsou zvláště zanořeny, představují konfiguraci těla jednotky. Jedná se o atributy **width** (šířka jednotky) a **hitpoints** (počáteční počet životů). To jsou zároveň všechny v současnosti konfigurovatelné vlastnosti

na úrovni těla jednotky. Atributy jako tým jednotky na této úrovni konfigurovány nejsou, jelikož volba hodnoty této vlastnosti již patří ke konkrétnímu nastavení světa.

Úroveň uvozená XML tagem <PLANNING_CONFIG> představuje konfiguraci plánovacího centra. Na této úrovni není možné v současnosti provádět žádné konfigurace. Můžeme však nakonfigurovat centrum plánování cesty v rámci úrovně <PATHPLAN_CONFIG>. Vlastnosti zde konfigurované korespondují s vlastnostmi zmíněnými v podkapitole 5.3.14.

Úroveň uvozená XML tagem <SENSOR_CONFIG> představuje konfiguraci senzorického centra. Zde můžeme pomocí tagu <SENSOR> jednotce definovat, jaké senzory má k dispozici. Atribut **type** tohoto tagu udává typ senzoru a atribut **radius** udává dosah tohoto senzoru. V současné implementaci je dostupný pouze senzor **eye**.

Úroveň uvozená XML tagem <EFFECTOR_CONFIG> představuje konfiguraci centra efektorů. Zde můžeme jednotce definovat, jaké efektory má k dispozici. To můžeme učinit pomocí tagu <EFFECTOR>, jehož atribut **type** udává typ daného efektoru a ostatní atributy jsou závislé na typu příslušného podcentra. Podúroveň <MOTION_CONFIG> konfiguruje centrum pohybu a je možné konfigurovat rychlost přidaného pohybového efektoru nastavením atributu **speed**. V současné implementaci je k dispozici jediný pohybový efektor a to efektor **leg**. Podúroveň <SHOOT_CONFIG> pak konfiguruje centrum střelby a je možné konfigurovat rychlost a sílu střely přidaného efektoru střelby pomocí atributů **speed** a **strength**. V současnosti je k dispozici jediný efektor střelby a tím je efektor **gun**.

Konfigurace pravidel

Chování jednotky je konfigurováno pomocí reaktivních a plánovacích pravidel. Tato mají specifikovány podmínky, za kterých je dané pravidlo platné, a akce, které určují co má být provedeno v případě, že jsou všechny podmínky pravidla splněny. Konfigurace pravidel se provádí předáním konfiguračního souboru metodě `ConfigureRules()` jednotky. Příklad konfiguračního souboru vidíme zde:

```
<RULES_CONFIG>
  <PLANNING_RULES>
    <RULE>
      <PRECONDITION type="distance" argument="50" />
      <PRECONDITION type="team" argument="TEAM2" />
      <PRECONDITION type="state" argument="NOT_DESTROYED" />

      <ACTION type="hide"/>
    </RULE>
  </PLANNING_RULES>
  <REACTIVE_RULES>
    <RULE>
      <PRECONDITION type="distance" argument="50"/>
      <PRECONDITION type="team" argument="TEAM2"/>
      <PRECONDITION type="state" argument="NOT_DESTROYED" />

      <ACTION type="shoot" argument="TOWARD"/>
      <ACTION type="move" argument="AWAY" argument2="20"/>
    </RULE>
  </REACTIVE_RULES>
</RULES_CONFIG>
```

XML soubor začínající tagem <RULES_CONFIG> představuje soubor konfigurace pravidel jednotky. Jednotlivá pravidla přitom můžeme rozdělit na dvě skupiny a to na pravidla **reaktivní** (specifikovaná v sekci <REACTIVE_RULES>) a pravidla **plánovací** (specifikovaná v sekci <PLANNING_RULES>). V rámci obou kategorií definujeme nové pravidlo pomocí XML tagu <RULE>. Každé pravidlo se skládá ze dvou typů položek. Jedná se o **podmínky** definované XML tagem <PRECONDITION> a **akce** definované XML tagem <ACTION>.

Podmínky jsou pro oba dva typy pravidel shodné a jsou specifikovány atributem **type**, který udává typ dané podmínky (vlastnost objektů, na kterou se daná podmínka zaměřuje) a atributem **argument**, který udává hodnotu této podmínky (hodnota vlastnosti, které má být dosaženo, aby byla daná podmínka splněna). V současné době jsou implementovány tyto typy podmínek: **distance** (tato podmínka je splněna, pokud je vzdálenost objektu menší než je hodnota této podmínky), **team** (tato podmínka je splněna, pokud je objekt v týmu specifikovaném hodnotou podmínky), **state** (tato podmínka je splněna, pokud stav objektu odpovídá hodnotě podmínky), **type** (tato podmínka je splněna, pokud je objekt daného typu) a **action** (podmínka je splněna, pokud objekt podniká specifikovanou akci). Možnosti hodnot výčtových podmínek je možné nalézt v podkapitole 5.2.3.

Akce jsou specifikovány atributem **type**, který udává typ dané akce a ostatní atributy jsou určeny právě typem této akce. Pro reaktivní pravidla je možné volit mezi dvěmi akcemi. První akcí je akce **move**, které můžeme atributem **argument** definovat směr, jímž se má jednotka pohybovat, a atributem **argument2** určit vzdálenost, o kterou se má daná jednotka tímto směrem posunout. Druhou akcí je akce **shoot**, které můžeme atributem argument definovat směr, kterým má jednotka vystřelit. Směr akce je možné volit z hodnot AWAY (pryč od nepřátelské jednotky), TOWARD (k nepřátelské jednotce), RIGHTTO (doprava ve směru kolmém k nepřátelské jednotce), LEFTTO (doleva ve směru kolmém k nepřátelské jednotce), HEADING (ve směru, kterým je jednotka natočena), BACK (opačným směrem než je jednotka natočena), LEFT (doleva od současného natočení jednotky) a RIGHT (doprava od současného natočení jednotky).

Pro plánovací pravidla je možné volit mezi akcemi **hide** (schovej se tak, aby na tebe objekt splňující podmínky nemohl střílet) a **reach** (dosáhni stanovené pozice). Akci **reach** přitom musíme atributem **argument** specifikovat X-ovou složku pozice, jíž má jednotka dosáhnout, a atributem **argument2** pak složku Y-ovou.

5.4 Implementace GUI

Grafické rozhraní bylo navrženo tak, aby odpovídalo celkovému konceptu. Obsahuje v sobě objekty třídy `Soldier`, které jsou spjaty s příslušnou jednotkou v prostředí a umožňují tak vykreslovat nejenom tuto jednotku na její aktuální pozici v prostředí, ale také dosah jejího aktuálního senzoru. U jedné zvolené jednotky jsou také vykreslovány její cíle a postup dosažení zvoleného cíle. Zdi jsou reprezentovány objekty třídy `SolidWall` a střely objekty třídy `Bullet`. Tyto třídy umožňují vykreslení těchto objektů na aktuálních pozicích.

Nastavení světa, které zahrnuje umístění a konfiguraci zdí a jednotek, je možné načíst z konfiguračního XML souboru pomocí nabídky **File -> Load World**. Po jeho načtení je možné spustit simulaci pomocí nabídky **Run -> Run Simulator**. Tato volba spustí časovač, který v pravidelných okamžicích vyvolává metodu `ChangeState()` objektu třídy `WorldEnvironment` a každé jednotce plánuje akci scan. Tento časovač a běh celé simulace je možné pozastavit pomocí nabídky **Run -> Stop Simulator**.

5.4.1 Konfigurace GUI

Jak již bylo řečeno výše, prostředí zobrazené v aplikaci je možné načíst z konfiguračního XML souboru. Příklad takového souboru vidíme zde:

```
<WORLD>
  <PROPERTY name="width" value="522"/>
  <PROPERTY name="height" value="370"/>
  <SOLDIER>
    <PROPERTY name="position_x" value="20"/>
    <PROPERTY name="position_y" value="20"/>
    <PROPERTY name="direction_x" value="0"/>
    <PROPERTY name="direction_y" value="1"/>
    <PROPERTY name="team" value="TEAM1"/>
    <PROPERTY name="name" value="Raven"/>
    <PROPERTY name="config_file" value="raven.xml"/>
    <PROPERTY name="rules_file" value="rules2.xml"/>
  </SOLDIER>
  <!-- Borders -->
  <WALL>
    <PROPERTY name="position_x" value="0"/>
    <PROPERTY name="position_y" value="0"/>
    <PROPERTY name="width" value="522"/>
    <PROPERTY name="height" value="5"/>
    <PROPERTY name="name" value="wall_up"/>
  </WALL>
</WORLD>
```

XML soubor začínající tagem <WORLD> představuje soubor konfigurace prostředí. Tomuto prostředí můžeme konfigurovat vlastnosti pomocí XML tagů <PROPERTY>, kde atribut **name** představuje název specifikované vlastnosti a atribut **value** pak hodnotu této vlastnosti. Vlastnosti, které nejsou více zanořeny, představují vlastnosti týkající se nastavení prostředí. Jedná se o vlastnosti **height** (výška plochy prostředí) a **width** (šířka plochy prostředí).

Do prostředí můžeme libovolně vkládat nové jednotky (pomocí tagu <SOLDIER>) a zdi (pomocí tagu <WALL>). Zdem i jednotkám přitom můžeme definovat vlastnosti typu **position_x** (X-ová souřadnice pozice objektu), **position_y** (Y-ová souřadnice pozice objektu) a **name** (jednoznačný identifikátor objektu v prostředí). Zdem navíc můžeme specifikovat atributy **width** (šířka objektu) a **height** (výška objektu). Jednotkám pak můžeme definovat atributy **direction_x** a **direction_y** (jejich kombinace udává počáteční natočení jednotky), **team** (tým, do něhož je jednotka zařazena), **config_file** (konfigurační soubor jednotky) a **rules_file** (konfigurační soubor pravidel pro jednotku).

Kapitola 6

Provedené experimenty

Posledním bodem zadání bylo provedení experimentů nad vytvořenou implementací a zhodnocení dosažených výsledků. Jedná se hlavně o ověření, zda bylo implementací části vytvořeného návrhu dosaženo příslušných cílů návrhu. Přitom se hlavně zaměřujeme na ověření, zda je jednotka schopna plnit obecněji zadané cíle (nemusí se jí v každém časovém okamžiku ručně určovat akce, kterou má právě provést), vytvářet vlastní plán k dosažení těchto cílů a okamžitě reagovat na kritické události, které nastanou v jejím prostředí. Vykazování těchto skutečností je pro nás dostatečným měřítkem toho, že zvolená implementace odpovídá zvolené části návrhu a splňuje tak požadavky od zadavatele. Jednotka by totiž měla jevit známky autonomnosti.

6.1 Experiment 1 - Dosažení zadaného cíle

První experiment byl proveden nad prostředím, v němž je jediným dynamickým prvkem námi zvolená jednotka, jejíž chování je v hlavní oblasti našeho zájmu. Mimo ni jsou v prostředí umístěny zdi a jedna nepřátelská jednotka. Pozice nepřátelské jednotky je neměnná a pomocí reaktivních pravidel je jí definována jediná reakce určující, aby vystřelila po každé jí nepřátelské jednotce, kterou zaznamená.

Námi sledovaná jednotka má definovány následující reakční pravidla: V případě, že spatříš nepřátelskou jednotku, vystřel po ní, couvni a znovu po ní vystřel. V případě, že jsi se dostala moc blízko zdi, couvni. Na úrovni plánovacích pravidel je jí definováno pravidlo: Pokud spatříš nepřátelskou jednotku, schovej se před ní. Této jednotce je možné zadat nový cíl stisknutím levého tlačítka myši na část jejího prostředí zobrazeného v GUI. Tím jednotce vytvoříme cíl dosáhnout tímto určené pozice. Centrum plánování jednotky je zároveň konfigurováno tak, aby se jednotka snažila volit cesty, které nejsou ohroženy výskytem nepřátelských jednotek.

Na obrázku 6.1 vidíme průběh daného experimentu. Nejdříve je jednotka umístěna do (pro ni nového) prostředí, o kterém nemá nejmenší představu, co ji v něm čeká. Jediné, co po prvním rozhlédnutí zjistí je, že se nad ní a vlevo od ní nachází zdi. Přestože vidí pouze část těchto zdí, vnímá tyto objekty jako celek (jak již bylo popsáno v kapitole 5.2.3) a má proto povědomí o celé jejich šířce a výšce, a tedy i o oblasti, kterou pokrývají. Tyto zdi jí nicméně nebrání v tom, aby se vydala přímo za svým cílem, protože nemá informaci, že by mezi ní a tímto cílem cokoliv stálo.

Při vykonávání svého plánu však jednotka narazí na nepřítele. Tím se pro ni vyskytla v prostředí kritická událost (a stejně tak pro nepřítele) a musí ji vyřešit. Reakce na kri-

tické události jsou oběma jednotkám definovány na úrovni reaktivních pravidel a proto po sobě obě jednotky vystřelí. Sledovaná jednotka navíc couvne a vystřelí ještě jednou. Nyní se jednotka nachází v takovém stavu prostředí, který neobsahuje kritické události. Proto se snaží naplánovat dosažení stanoveného cíle. Při snaze o tvorbu plánu však zjistí, že v získaných znalostech se objevily takové skutečnosti, které způsobily aktivaci pravidla plánovacího (tyto skutečnosti již nejsou platné, ale nebylo je možné na této úrovni dříve obsloužit, kvůli okamžité obsluze na úrovni reaktivní). Jedná se o pravidlo udávající, že pokud jsme zaznamenali v dosahu nepřítele, máme se před ním schovat. Je proto nalezen takový bod, kde je jednotka ukryta před střelbou nepřítele, následně je naplánováno jeho dosažení a také je tento bod dosažen.

Jakmile je jednotka ukryta, již nemá žádné pravidlo, kterému by musela vyhovět a tudíž naplánuje vykonání prvního cíle, který se nachází v množině jednotce definovaných cílů. Nyní vytvořený plán se liší od plánu prvotního, protože jednotka již zná některé další zdi nacházející se v prostředí a hlavně již ví o výskytu nepřátelské jednotky. Protože je sledovaná jednotka konfigurována tak, aby se snažila hledat cesty v bezpečné vzdálenosti od nepřátelských jednotek, zvolila cestu, která je pro ni sice teoreticky delší, ale zato bezpečnější. Opět ale nemá tušení, že se na této zvolené trase nachází překážka ve formě zdi. Ve chvíli, kdy se jednotka dostane k této zdi tak blízko, že ji zaznamená svými senzory, zjistí, že tudy její trasa vést nemůže. Zneplatní tedy svůj současný plán a vytvoří plán nový. To se opakuje až do chvíle, kdy jednotka své okolí zná natolik dobře, že může dosáhnout svého cíle. Na předposlední situaci také můžeme vidět, že je tohoto cíle schopna dosáhnout. Nyní zná jednotka téměř celé své prostředí. Pokud bychom jí zadali cíl jiný, již je schopna naplánovat jeho dosažení na základě svého modelu okolního prostředí tak, že nedojde k zneplatňování takto vytvořeného plánu a jednotka je tak hned napoprvé schopna tohoto cíle dosáhnout.

6.2 Experiment 2 - Konfigurace opatrnosti jednotky

Tento experiment byl zaměřen na nastavování opatrnosti jednotky, co se týče plánování cesty nepřátelským územím, a na pozorování, jak se podle tohoto nastavení chová. Bylo zvoleno prostředí s třemi statickými nepřátelskými jednotkami a dostatkem prostoru pro plánování cesty mimo působení těchto jednotek. Žádné z jednotek nebyla specifikována pravidla, takže se celý test zaměřil pouze na plánování cesty v tomto prostředí.

Na obrázku 6.2 vidíme průběh experimentu s jednotkou, která je nastavena, ať volí svoji cestu opatrně. Segmenty trasy, které vedou kolem nepřátelských jednotek mají horší ohodnocení než-li segmenty trasy, které jsou od nepřátelských jednotek dostatečně vzdáleny. Nejlepším řešením pro jednotku je tedy nalézt takové segmenty cesty, jejichž poskládáním bude ohodnocení cesty co nejmenší. Podle velikosti postihu jsou pak tyto cesty buď vždy (pokud je to možno) voleny v bezpečné vzdálenosti od nepřátelských jednotek, nebo může dojít k situaci, kdy i opatrný agent zvolí segment trasy vedoucí podél nepřátelské jednotky (pokud není postih dost velký). V našem experimentu můžeme vidět, že jsme zvolili postih dostatečně velký na to, aby jednotka plánovala svoji trasu mimo dosah nepřátelských jednotek.

V další části experimentu (viz obrázek 6.3) jsme nakonfigurovali jednotku tak, aby se nesnažila brát ohled na to, zda její trasa vede v nebezpečné vzdálenosti od nepřátelských jednotek, a vždy volila co nejkratší trasu. Na prvních situacích vidíme, že bez ohledu na to, že jednotka při vykonávání svého plánu potkala nepřátelskou jednotku, pokračuje v plánování své trasy přes její pozici. Na závěr také vidíme, že i když jednotka již zná své prostředí, volí

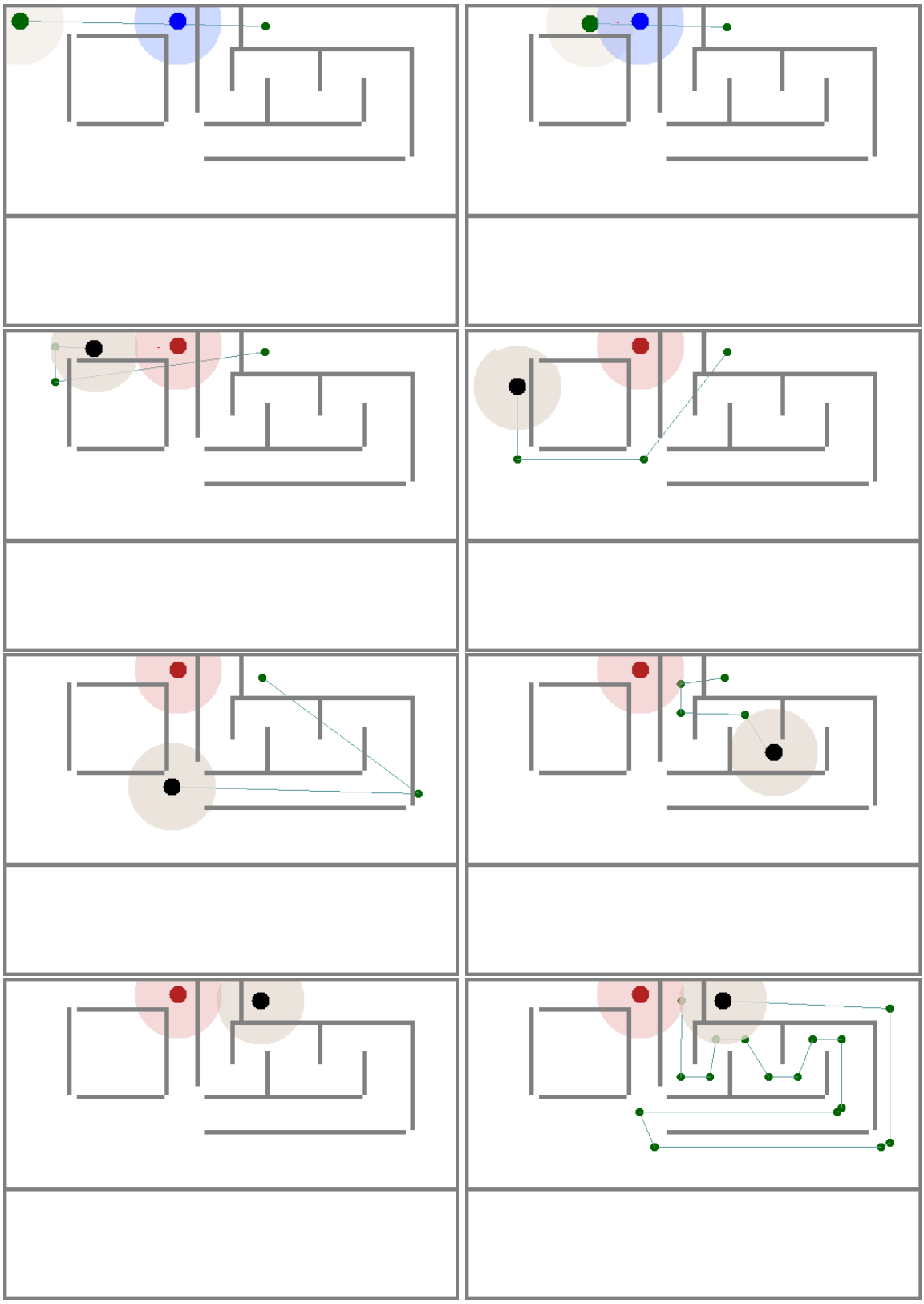
svoji trasu napříč nepřátelskými jednotkami. Konfigurace opatrnosti tedy funguje podle očekávání.

6.3 Experiment 3 - Pohybující se protivník

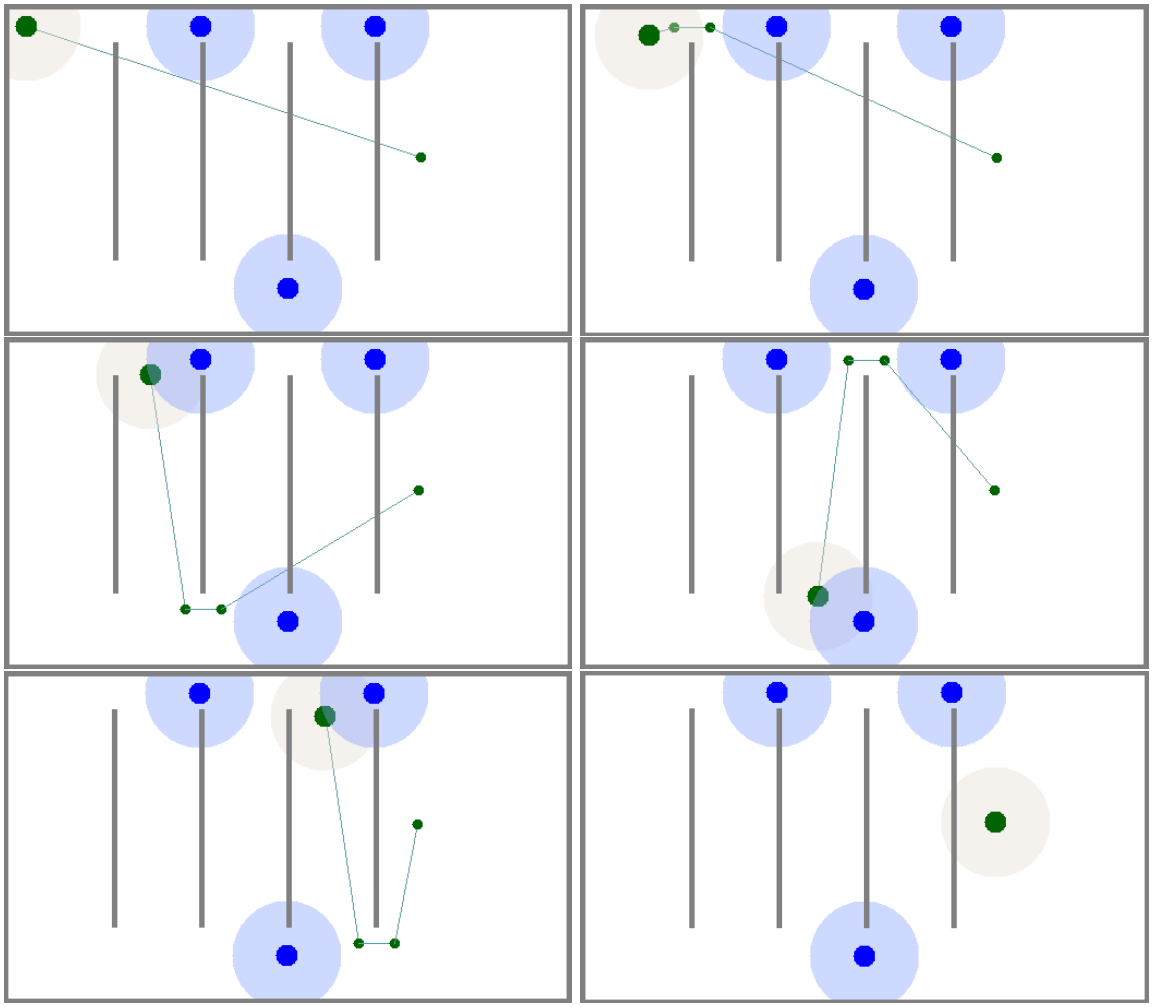
Všechny experimenty byly doposud prováděny se statickým protivníkem. Proto bylo vybráno k experimentu také prostředí, do něhož jsme umístili jednoho pohybujícího se protivníka. Tento protivník se pohybuje pomocí reaktivních pravidel, s explicitní znalostí výšky svého prostředí. Když uvidí zeď, otočí se a přesune se přes celou výšku svého prostředí na opačnou stranu. V případě, že spatří nepřítele, vystřelí po něm a pokračuje ve svém pohybu.

Jednotka, kterou budeme sledovat má definována pravidla, kdy v případě, že spatří protivníka, tak po něm vystřelí a ukročí dozadu a doleva. Jiná pravidla specifikována nemá a řídí se jen svým plánovacím centrem.

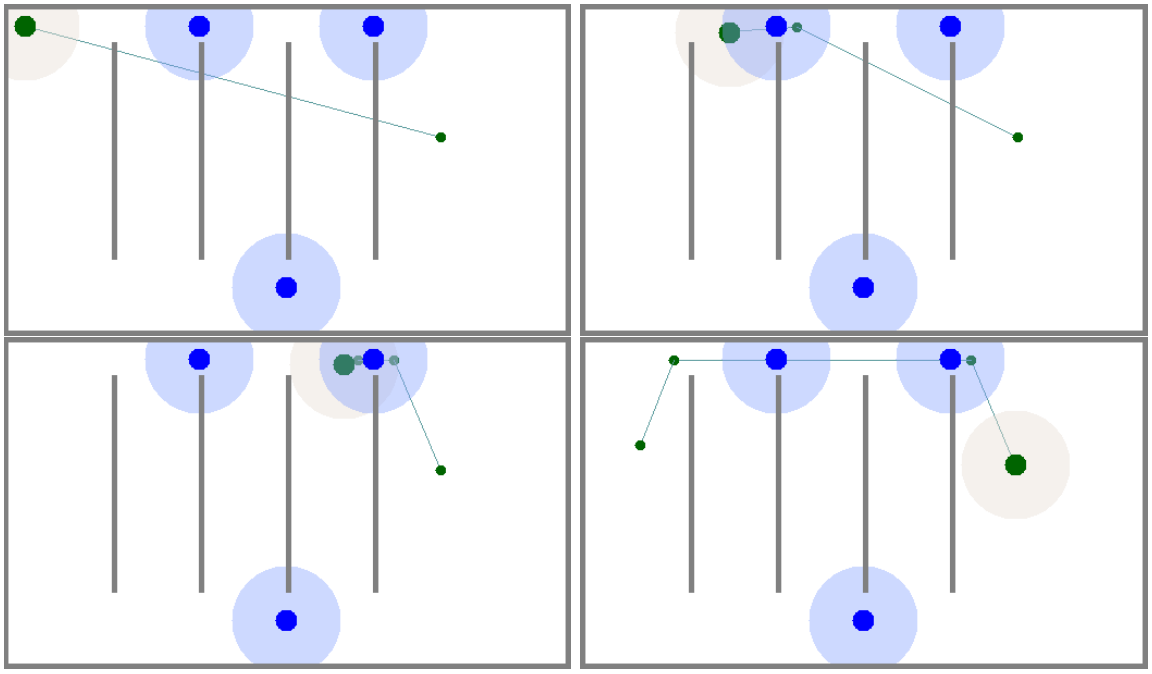
Na obrázku 6.4 vidíme průběh celého experimentu. Sledované jednotce byl zadán cíl, jehož dosáhnutí se snažila naplánovat a poté tento plán vykonat. V průběhu vykonávání plánu však narazila na nepřátelskou jednotku, proto patřičně zareagovala výstřelem a úhybem. Poté přeplánovala svoji trasu jinudy. Bohužel pro ni se však nepřátelská jednotka přesunula tak, že ani nově naplánovaná trasa nebyla k průchodu vhodná. Jelikož nepřátelská jednotka změnila svou pozici, rozhodla se námi sledovaná jednotka vyzkoušet předchozí průchod, který by mohl být nyní nehlídaný. Toto se jednotce osvědčilo a podařilo se jí tak dosáhnout určené pozice.



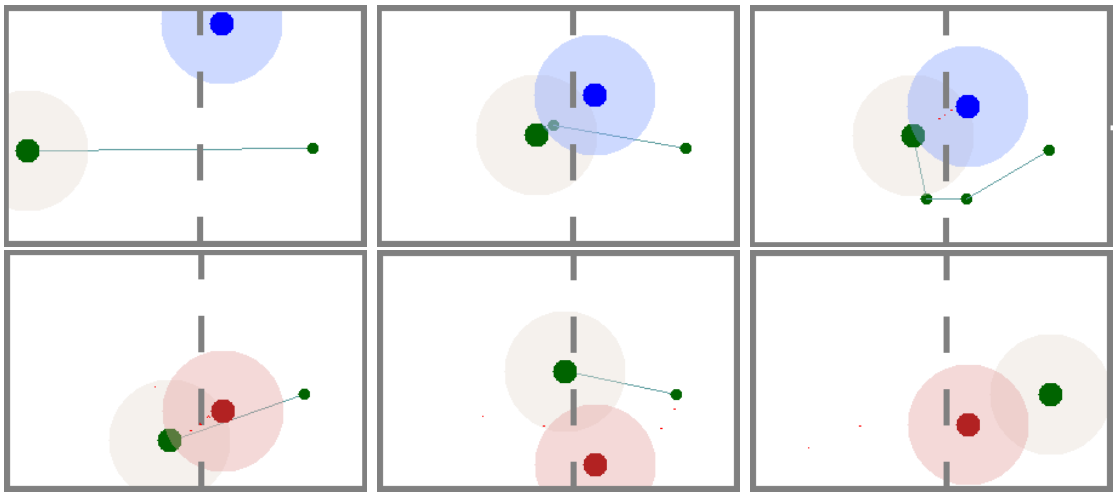
Obrázek 6.1: Průběh prvního experimentu.



Obrázek 6.2: Plánování cesty opatrnou jednotkou.



Obrázek 6.3: Plánování cesty neopatrnou jednotkou.



Obrázek 6.4: Jednotka v prostředí s dynamickým protivníkem.

Kapitola 7

Závěr

Cílem diplomové práce bylo nastudovat literaturu zaměřenou na umělou inteligenci bojových jednotek a reálné chování diskutovat s odborníkem z praxe. Dalším úkolem bylo navrhnout postup realizace inteligentního chování bojové jednotky, který bude aplikovatelný na databázi (3D prostředí) firmy E-COM s.r.o. a provést jeho základní implementaci za účelem otestování návrhu a možností provádění jednoduchých experimentů. Na závěr mělo být na této implementaci provedeno několik experimentů, které by ukázaly správnost návrhu. Všechny tyto cíle byly splněny. Byla nastudována literatura týkající se umělé inteligence bojových jednotek a byl vytvořen návrh, který byl diskutován s odborníky z praxe. Vybraná část návrhu byla implementována, čímž byla vytvořena jedna z možností implementace jednotlivých center. Na této implementaci byly také provedeny experimenty.

Do budoucna tato práce skýtá spoustu možností k rozšíření. Na každé z navrhnutých center je možné se zaměřit konkrétněji a rozebrat tak jednotlivé části návrhu na úrovni jejich konkrétní aplikace. Také vzniká možnost různých implementací jednotlivých center a jejich zodpovědností, a s tím také velké možnosti experimentování s vytvořenými implementacemi.

Přínos této práce spočívá ve shromáždění všech aspektů, které nesmí být v návrhu inteligentní bojové jednotky opomenuty, a hlavně v samotném návrhu rozdělujícím zodpovědnosti jednotlivých center jednotky a popisujícím jejich funkci, vzájemnou komunikaci ad. Částečná implementace a experimenty sloužily jako ověřující ukazatel, že byl návrh proveden správně a je použitelný a implementovatelný.

Literatura

[1] BIRUKOU, M.: Knowledge representation. 2002, [online], [cit. 2008-05-11].
URL http://arxiv.org/pdf/cs/0208019

[2] ILIEV, M.: *Model senzorů objektu pohybujícího se virtuálním prostředím.* Bakalářská práce, FIT VUT v Brně, 2008.

[3] KUŽELA, M.: *Vytváření map okolního prostředí robota.* Bakalářská práce, FIT VUT v Brně, 2006.

[4] LAIRD, J. E.; LENT, M. V.: Human-level AI's Killer Application: Interactive Computer Games. In *AAAI*, 2000, [online], [cit. 2008-05-11].
URL http://ai.eecs.umich.edu/people/laird/papers/AAAI-00.pdf

[5] ORSÁG, F.: Robotika, 2006, studijní opora k předmětu ROB.

[6] POOLE, D.; MACKWORTH, A.: Dimensions of Complexity of Intelligent Agents. In *ACM*, 2006, ISBN 1-74052-130-7, [online], [cit. 2008-05-11].
URL
http://www.cs.ubc.ca/labs/lci/papers/docs2006/poole_dimensions2006.pdf

[7] POOLE, D.; MACKWORTH, A.; GOEBEL, R.: *Computational intelligence: logical approach.* New York: Oxford University Press, 1998, ISBN 0-19-510270-3, 558 s.

[8] RAJIV S. DESAI, D. P. M.: A Simple Reactive Architecture for Robust Robots. In *ICRA*, 1998, [online], [cit. 2008-05-11].
URL http://www.kipr.org/papers/icra92.pdf

[9] RUSSELL, J. S.; NORVIG, P.: *Artificial intelligence: A Modern Approach.* New Jersey: Prentice-Hall, 2003, ISBN 0-13-790395-2, 1081 s.

[10] ZBOŘIL, F.: Agentní a multiagentní systémy, 2006, studijní opora k předmětu AGS.

[11] ZBOŘIL, F.: Základy umělé inteligence, 2006, studijní opora k předmětu IZU.