Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

**BAKALÁŘSKÁ PRÁCE**


Michal Štolba
Rozhodovací pravidla pro projekt Pogamut 2
Kabinet software a výuky informatiky
Vedoucí bakalářské práce: Mgr. Cyril Brom, Ph.D.
Studijní program: IOI
2008

Především bych chtěl poděkovat Cyrilu Bromovi za vedení práce, Jakubu Gemrotovi za mnoho podnětných konzultací a společně s Rudolfem Kadlecem, Michalem Bídou a dalšími tvůrci za projekt Pogamut 2 a za umožnění mi podílet se na jeho dalším rozvoji. Také děkuji Martině Koníčkové za poskytnutí klidného zázemí na psaní práce. Díky Bohu.

Prohlašuji, že jsem svou bakalářskou práci napsal(a) samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne Michal Štolba

Obsah

Název práce: Rozhodovací pravidla pro projekt Pogamut 2

Autor: Michal Štolba

Katedra (ústav): Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Cyril Brom, Ph.D.

e-mail vedoucího: Cyril.Brom@mff.cuni.cz

Abstrakt: Projekt Pogamut 2 umožňuje rychlou tvorbu chování agentů v komplexním prostředí hry Unreal Tournament 2004. Původní systém pro řízení rozhodovacími pravidly (POSH) byl nyní doplněn o systém pro řízení fuzzy rozhodovacími pravidly. Cílem práce bylo zvolit vhodný fuzzy systém, zjistit jakým způsobem je možné takový systém do projektu připojit a také zjistit jaké možnosti a jaké obtíže použití fuzzy pravidel přináší. Tato práce seznamuje s teoretickými východisky, jejich aplikací, navrženou architekturou, její částečnou implementací a s příkladem agenta řízeného implementovaným fuzzy systémem, jehož funkčnost byla ověřena několika experimenty. Tato práce má rovněž sloužit jako podklad pro další rozšíření, zejména plnou implementaci navržené architektury, vytvoření uživatelského rozhraní jako součásti IDE projektu Pogamut 2, a pro širší možnosti experimentů s agenty řízenými fuzzy rozhodovacími pravidly.

Klíčová slova: fuzzy rozhodovací pravidla bot Pogamut

Title: Rule based system for project Pogamut 2

Author: Michal Štolba

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Cyril Brom, Ph.D.

Supervisor's e-mail address: Cyril.Brom@mff.cuni.cz

Abstract: The project Pogamut 2 provides a possibility of fast prototyping of agent behaviours in a complex envirorment of the Unreal Tournament 2004 computer game. A fuzzy rule based system was introduced to be used beside the POSH rule based system, which was already a part of the project. Aiming to find out how exactly is it possible to add such system and what possibilities or complications it brings, this thesis presents the theoretical presumptions, their application, a design of an architecture, it‘s partial implementation and an example of agent controlled by the implemented fuzzy system. The agent’s functionality was proved by several experiments. This thesis should also be a basement for furher work, such as full implementation of presented architecture, adding an user interface integrated within Pogamut IDE, and for wider possibilities of experimenting with the fuzzy agents.

Keywords: fuzzy rule based systems bot Pogamut

# 1. Úvod

Anglické slovíčko „fuzzy“ (znamenající rozmazané, nejasné, neostré) se vžilo pro označení matematických disciplín pracujících s přesně definovaným pojmem nepřesnosti, vágnosti. Zejména jde o teorii fuzzy množin a z ní vycházející fuzzy logiku.

Fuzzy přístup našel své uplatnění v řízení a regulaci, kde se přes prvotní odmítání brzy stal zcela standardní technologií. Fuzzy logiku formuloval v 70. letech L.A.Zadeh (např. v [7]), v 80. a 90. letech nastal boom jejího používání v průmyslu, masivního nasazení se dočkala nejdříve v Japonsku a později i v dalších zemích [2] str. 5.

Fuzzy logika umožnila v mnoha oblastech řešit dříve neřešitelné problémy, řešení dalších problémů značně zjednodušila, zejména v oblastech řízení a regulace. Vyvstává tedy otázka, zda je možné a vhodné využít fuzzy logiku k řešení problému výběru akce, jinými slovy k ovládání autonomních agentů v komplexním prostředí. Existují různé návrhy takové aplikace, např. v [1] navrhuje A. J. Champandard použití fuzzy rozhodovacích pravidel k ovládání pohybu ve složitějších situacích (použití dveří, výtahu apod., str. 409-419), a rovněž představuje myšlenku fuzzy stavového automatu (str. 541-545). Využití podobných myšlenek v komerčních hrách je však spíše poskrovnu, často se jedná pouze o využití konceptu fuzzy logiky nebo o řešení nějakého specifického problému.

Pogamut 2 [3] je prostředí pro rychlou tvorbu a testování chování agentů ve virtuálním prostředí hry Unreal Tournament 2004 (v tomto kontextu se místo „agent“ často používá termín „bot“ a i já budu v dalším textu oba termíny volně zaměňovat), jedná se tedy o vhodnou platformu pro testování možnosti řízení agentů pomocí fuzzy rozhodovacích pravidel.

Cílem práce tedy bylo:

- Najít a vybrat vhodnou existující aplikaci fuzzy logiky.
- Navrhnout způsob jak ji zapojit do projektu Pogamut 2.
- Navrhnout a vytvořit agenta řízeného fuzzy rozhodovacími pravidly.
- Ověřit funkčnost a vhodnost několika experimenty.

V první části práce seznamuji čtenáře s jejími východisky - se základy fuzzy logiky a s projektem Pogamut 2 (kapitola 2). V teoretické části práce (kapitola 3) popisuji konkrétní aplikaci fuzzy logiky, kterou jsem použil a problémy, které jsem v průběhu práce řešil. V praktické části (kapitola 4) uvádím návrh architektury fuzzy agenta a příklad agenta, vytvořeného v tomto systému, včetně popisu s ním provedených experimentů.

# 2. Východiska práce

Tato práce vychází ze dvou oblastí - fuzzy rozhodovacích pravidel a prostředí projektu Pogamut 2. V úvodní kapitole nejdříve shrnuji teorii potřebnou k porozumění aplikaci fuzzy rozhodovacích pravidel na řízení autonomních agentů, poté seznamuji se základy platformy Pogamut 2.

## *2.1. Fuzzy rozhodovací pravidla*

Pro porozumění fuzzy rozhodovacím pravidlům (též nazývaným fuzzy IF-THEN pravidla) a jejich aplikace je nutné nejdříve se seznámit s teorií, na které jsou založena. V úvodu této kapitoly je nastíněn základní princip fuzzy modelování a jeho vymezení vůči teorii pravděpodobnosti. Následuje seznámení se základy teorie fuzzy množin a s pojmy sloužícími k modelování přirozeného jazyka pomocí této teorie. Další podkapitola se zabývá použitím rozhodovacích pravidel na řízení autonomních agentů a aplikací fuzzy logiky na tato pravidla. Nakonec jsou ještě detailně popsány zásadní metody používané při zpracování fuzzy rozhodovacích pravidel.

### 2.1.1. Modelování neurčitosti

První otázka zní: Proč bychom vůbec měli potřebovat zabývat se v matematice neurčitostí, vágností, když nám dnes měřící přístroje poskytují informace, které je ve většině aplikací možné považovat za přesné? Navíc matematika jako taková je založena na přesných definicích a formulacích, není to tedy celé nesmysl? Odpověď leží někde trochu jinde. V mnoha aplikacích totiž nepotřebujeme takovou přesnost – při parkování je dostatečná informace „popojeď kousek dozadu“, naopak informace „popojeď o 23,485321 cm dozadu“ by celou situaci neúměrně zkomplikovala. Jedná se o rozhodnutí mezi přesnou, avšak málo relevantní informací a informací relevantní, avšak málo přesnou (L. A. Zadeh tento princip nazval **principem inkompatibility** v [7], str. 28-44, citováno z [2]). Mimo to některé expertní znalosti ani nejsme schopni přesně zformulovat nebo by jejich formulace vedla na příliš komplexní systémy. Co se týče přesné matematické formulace pojmu vágnosti, dozvíte se více v následujících kapitolách nebo v knize [2].

Druhá otázka zní: Není již neurčitost dostatečně formalizována v teorii pravděpodobnosti a nejedná se tudíž o jiný název téhož? Není tomu tak. Rozdíl spočívá v tom, že v teorii pravděpodobnosti se zaměřujeme na to, s jakou pravděpodobností ten který jev může nastat, zatímco v teorii fuzzy modelování nás zajímá kvalita nastalého jevu. Neurčitost zde tedy neleží v tom, že nevíme jaký jev nastane, ale v tom, že neumíme (nebo nepotřebujeme) nastalý jev přesně popsat.

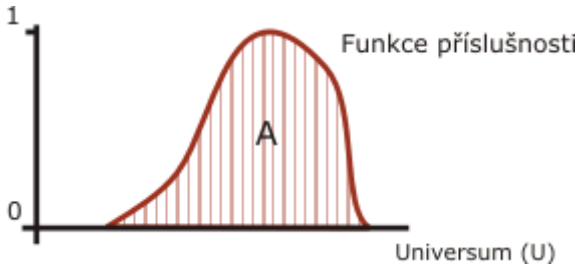
### 2.1.2. Fuzzy množiny

Fuzzy množina je zobecněním klasické množiny v tom smyslu, že neplatí pouze jedno z tvrzení „prvek v množině je“, nebo „prvek v množině není“, ale platí „prvek je v množině v určitém stupni“, což nám v některých situacích umožňuje přirozenější přístup. Často uváděným příkladem jsou výšky lidí. Definovat přesně hranici, kdy je člověk ještě vysoký a kdy už ne je nejen obtížné (a často subjektivní), ale většinou také nevhodné. Mnohem přirozenější je definovat pro každou výšku její stupeň příslušnosti do množiny „výšek vysokých lidí“ (např. 0 pro 150 cm a nižší, 1 pro 185 cm a vyšší a mezi 0 a 1 pro hodnoty mezi).

Přesněji **fuzzy množinou** rozumíme funkci:

$$A:U \rightarrow [0,1] \tag{2.1}$$

Kde U je nosná množina, tak zvané **universum**, a [0,1] je spojitý interval. Fuzzy množina A je tedy definována tzv. **funkcí příslušnosti** A(x), která každému prvku z universa U přiřadí jeho **stupeň příslušnosti** do množiny A. Funkce příslušnosti zároveň odpovídá přirozenému zobecnění charakteristické funkce množiny.




**Obrázek 2.1: Fuzzy množina**

Stejně jako v klasické teorii množin, definujeme i v teorii fuzzy množin různé operace. Podle [2] str. 24-27 významu logické spojky AND přirozeně odpovídá průnik definovaný takto:

$$C = A \cap B \Leftrightarrow C(x) = \min(A(x), B(x)) \tag{2.2}$$

významu logické spojky OR odpovídá sjednocení definované takto:

$$C = A \cup B \Leftrightarrow C(x) = \max(A(x), B(x)) \tag{2.3}$$

a doplněk (negaci) obvykle definujeme takto:

$$B = \text{non } A \Leftrightarrow B(x) = 1 - A(x) \tag{2.4}$$

Při bližším prozkoumání výše uvedených definic si můžete povšimnout zajímavé skutečnosti - $A \cap \text{non } A \neq \emptyset$ - tedy neplatí zákon o vyloučení třetího! A skutečně, vezmeme-li množinu „malých a ne-malých“ lidí, zjistíme, že v ní jsou ti lidé, o kterých nemůžeme s jistotou tvrdit, že jsou malí, ale ani bychom neřekli, že určitě malí nejsou (pozor, ne-malý a velký není to samé!). V teorii fuzzy množin jsou však na rozdíl od té klasické definovány i další operace (např. Łukasiewiczova konjunkce a disjunkce – viz [2], str. 26), které zákon o vyloučení třetího zachovávají. Obecně lze tedy říci, že je tento zákon v teorii fuzzy množin a ve fuzzy logice zachován, jen je potřeba dát si pozor při použití výše definovaných „přirozených“ operací.

### 2.1.3. Jazykové proměnné a výrazy

Jednou z výhod fuzzy přístupu je přiblížení se lidskému myšlení a způsobu vyjadřování, díky tomu lze pomocí teorie fuzzy množin modelovat význam slov a některých jednoduchých výrazů. Za tímto účelem si zavedeme pojem **jazyková proměnná**. Hodnotami jazykové proměnné mohou být některá slova a výrazy (už jsme se s nimi v textu setkali), jako „malý“,

„středně vysoký“ atp. Souhrnně se hodnoty jazykové proměnné nazývají **jazykové výrazy** (lingvisticky syntagmata). Matematicky je jazyková proměnná definována jako pětice:

$$< \chi, T(\chi), U, G, M>, \tag{2.5}$$

kde:

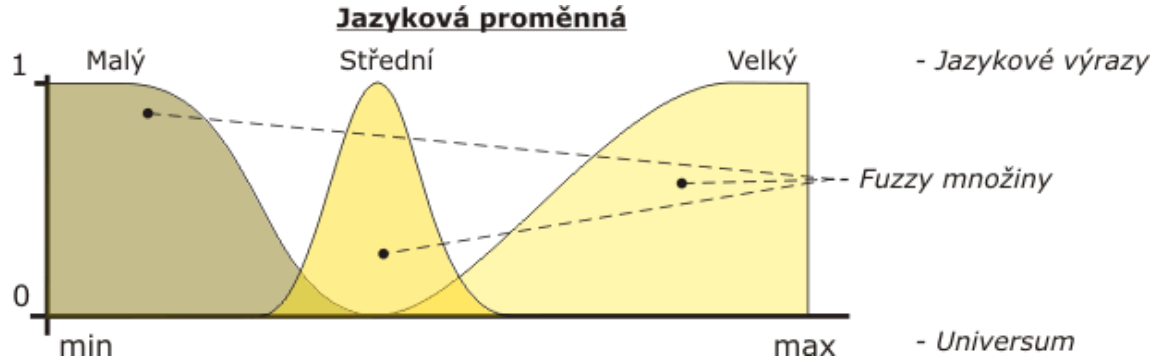
$\chi$ - jméno proměnné

$T(\chi)$ - množina možných hodnot (jazykových výrazů)

U - univerzum

G - syntaktické pravidlo, pomocí něhož jsou tvořeny jazykové výrazy z množiny $T(\chi)$

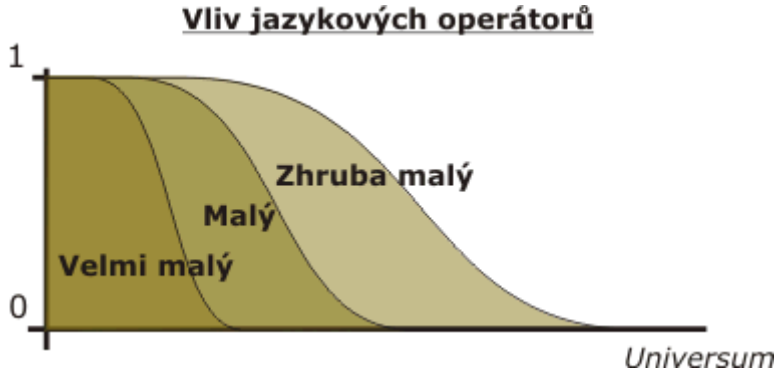
M - sémantické pravidlo, které každému jazykovému výrazu z $T(\chi)$ přiřadí jeho význam ve formě fuzzy množiny definované nad U




**Obrázek 2.2: : Přirozené uspořádání a tvary fuzzy množin odpovídajících výrazům „malý“, „střední“ a „velký“ podle [2] str. 48**

Příklad jazykové proměnné na obr. 2.2 znázorňuje člověku přirozené uspořádání výrazů „malý“, „střední“ a „velký“ při popisu jakékoliv uspořádané škály, včetně tvaru funkcí příslušnosti a mírného vychýlení výrazu „střední“ (podle [2], str.48). V praxi se však tvary funkcí často zjednodušují (viz kapitolu **3.1. Aplikace teorie**).

Výše uvedené výrazy „malý“, „střední“, „velký“ a jim podobné se nazývají **atomické jazykové výrazy** a jsou jakýmsi základním stavebním kamenem. Dále se tyto výrazy mohou modifikovat tzv. **jazykovými operátory**, jako jsou „velmi“, „více méně atd., jejichž významem mohou být funkce z $\{f \mid f : [0,1] \rightarrow [0,1]\}$ (někdy se používá i dvojice funkcí – modifikace a posunutí – [2], str. 54). Výslednou fuzzy množinu získáme složením její funkce příslušnosti s funkcí daného operátoru.




**Obrázek 2.3: Žádoucí vliv operátorů „velmi“ a „zhruba“ na atomický jazykový výraz „malý“ podle [2], 53**

Na obr. 2.3. je naznačen efekt operátorů, který odpovídá interpretaci přirozeného jazyka. Je vidět, že existují v zásadě dva typy operátorů – zužující a rozšiřující. Efekt zužujícího operátoru je jak strmější část fuzzy množiny, kde jsou prvky se stupněm příslušnosti z intervalu [0,1], tak užší část fuzzy množiny s prvky jejichž stupeň příslušnosti je právě 1. Při tom je právě zúžení množiny prvků s pravdivostní hodnotou 1 zcela zásadní. Vezměme si například myš a mravence. O myši bychom řekli, že je určitě malá (st. přísl. 1), ale o tom, jestli je velmi malá už by se dalo pochybovat (st. přísl. < 1), oproti tomu o mravenci to s klidným svědomím říct můžeme (st. přísl. 1). Je tedy vidět, že množina prvků se stupněm příslušnosti 1 se zmenšila (samozřejmě vždy záleží na konkrétním kontextu, ale tento princip je platný obecně). Analogicky by se dal popsat rozšiřující vliv.

Nejčastěji se pro operátory „velmi“ a „zhruba“ používají následující operace zavedené L. A. Zadehem v [8], citováno z [2], modifikující funkce příslušnosti:

$$CON(a) = a^2, \quad a \in [0,1] \tag{2.6}$$

$$DIL(a) = a^{0,5}, \quad a \in [0,1] \tag{2.7}$$

Oba takto definované operátory však nemají výše požadovanou vlastnost a nezužují (resp. nerozšiřují) množinu prvků se stupněm příslušnosti 1. Proto je vhodné jako operátor definovat kromě funkce, která mění strmost fuzzy množiny, také funkci posunutí, která mění velikost množiny prvků se stupněm příslušnosti 1.

Poslední termín, o kterém se zmíním, je **evaluační predikace**. Jedná se o výrazy typu „X je A“, kde X je podstatné jméno a A je jazykový výraz (např. „Petr je velmi vysoký“). Význam evaluační predikace nezávisí na konkrétním objektu X a je tedy možné jeho hodnotu ztotožnit s hodnotou jazykového výrazu A, tedy s konkrétní fuzzy množinou.

### 2.1.4. Rozhodovací pravidla jako mechanismus výběru akce

Problémem výběru akce (action selection problem) se nazývá situace, kdy se autonomní agent musí rozhodnout, co udělá v následujícím okamžiku. Aby se mohl správně rozhodnout, musí nějakým způsobem vnímat prostředí, ve kterém se pohybuje. K tomu slouží sada tzv. **senzorů**, které poskytují informace o prostředí, i o stavu agentova virtuálního těla (pokud ho má). Protějškem senzorů jsou tzv. **efektory** (také nazývané akce), kterými může agent virtuální prostředí ovlivňovat. Problém výběru akce spočívá v hledání odpovědi na otázku, kterou akci zvolit na základě informací přijatých ze senzorů.

Existují různé mechanismy řešící tento problém a jedním z nich jsou právě rozhodovací pravidla (známá také jako IF-THEN pravidla) [13] str. 4, která byla původně používána zejména v expertních systémech. Rozhodovací pravidlo je pravidlo typu

**IF** *podmínka* **THEN** *akce*

Jinými slovy: Pokud platí podmínka, proveď akci. Ta může přímo ovlivnit prostředí ve kterém se agent nachází nebo může modifikovat vnitřní stav a paměť agenta či může spustit vyhodnocování další sady podmínek. Nahrazením akce další sadou podmínek vzniká z ploché struktury hierarchie, která umožňuje snáze řešit komplexnější situace.

Rozhodovací pravidla jsou vedle konečných automatů jedním z nejpřirozenějších a nejrozšířenějších způsobů řízení autonomních agentů v počítačových hrách i jiných aplikacích. Chceme-li aplikovat řízení agenta pomocí fuzzy logiky, zdá se výhodné použít některý z již ověřených klasických přístupů. Kolem fuzzy IF-THEN pravidel je navíc vybudována rozsáhlá teorie, prakticky ověřená aplikací v různých oborech (např. regulace, expertní systémy apod., viz také [14]), a jedná se v podstatě o nejpoužívanější způsob aplikace fuzzy logiky. Fuzzy konečné automaty jsou představeny např. v [1], str. 541-545, kde je ale zároveň diskutována implementace těchto automatů pomocí fuzzy IF-THEN pravidel. Jedná se tedy spíše o jiný způsob návrhu a zaměříme-li se pouze na fuzzy rozhodovací pravidla, neměli bychom se připravit o žádnou z výhod fuzzy systémů.

Jak ale IF-THEN pravidla souvisí s výše uvedenou teorií? Rozhodovací pravidla bývají většinou chápána jako logické implikace, fuzzy rozhodovací pravidla tedy snadno získáme tak, že běžnou implikaci nahradíme **fuzzy implikací**. Stejně jako u ostatních operací existuje více různých definic, které ale všechny splňují obecná pravidla fuzzy implikace, která jsou detailně popsána v [2], str. 71-75. Nejběžnější fuzzy implikace pro $a,b \in [0,1]$ jsou tyto:

**Łukasiewiczova**: $a \Rightarrow b = \min(1, 1 - a + b)$ (2.8)

**Gödelova**:
$$a \Rightarrow b = 1 \quad \text{pro } a \leq b$$
$$a \Rightarrow b = b \quad \text{jinak} \tag{2.9}$$

**Produktová**:
$$a \Rightarrow b = 1 \quad \text{pro } a \leq b$$
$$a \Rightarrow b = b/a \quad \text{jinak} \tag{2.10}$$

Takto definovaná pravidla mohou poměrně snadno nahradit v procesu řízení agenta pravidla klasická, uvedená na začátku této kapitoly.

### 2.1.5. Inference a defuzzifikace

**Inferenční mechanismus** je postup, jakým se ze sady IF-THEN pravidel a daných vstupů odvodí hodnoty výstupů. Z teoretického hlediska existují na celý proces dva pohledy podle toho, jak se díváme na sadu rozhodovacích pravidel. První možnost je dívat se na pravidla jako na fuzzy implikace nějakého z výše uvedených typů a provádět logickou dedukci pomocí odvozovacího pravidla modus ponens (vše je samozřejmě zobecněno pro fuzzy logiku). Druhá možnost je představit si sadu pravidel jako předpis tzv. **fuzzy funkce** (funkce jejímž vstupem i výstupem jsou fuzzy množiny), tedy zjednodušeně řečeno, pravidlo „if X is A then Y is B“ znamená, že fuzzy množině A je přiřazena fuzzy množina B. Obě možnosti jsou podrobně popsány ve [2] v kapitole 3 (str. 67-113), zde se dalším rozborem uvedené problematiky nebudu zabývat, neboť to přesahuje rámec bakalářské práce a hlubší porozumění není pro tuto konkrétní aplikaci potřebné.

V případě řízení agenta se zpravidla uplatňuje pohled na pravidla jako na fuzzy implikace a pro výpočet hodnoty pravé strany pravidla z hodnot jeho levé strany je možné použít některou z fuzzy implikací uvedených v kapitole 2.1.4., nejčastěji se při tom používá produktová implikace (2.10), ze které dostaneme pro pravou stranu B a levou stranu A vzorec:

$$R(x) = A(x) \cdot B(x), \tag{2.11}$$

kde x jsou prvky univerza a R je fuzzy množina vzniklá inferencí. Někdy se pro inferenci používá metoda, při které v podstatě „ořízne“ množina na pravé straně pravidla podle hodnoty na levé straně. Přesněji se jedná o výběr minima:

$$R(x) = \min(A(x), B(x)) \tag{2.12}$$

Další fází zpracování sady pravidel je složení hodnot pravidel, na jejichž pravé straně je tatáž fuzzy množina. Výsledná hodnota se obvykle určí pomocí funkcí minima, což odpovídá složení pravidel spojkou AND (2.2), maxima odpovídající spojce OR (2.3), součtu (kde je potřeba ošetřit „přetečení“ intervalu [0,1]) a nebo jako průměrná hodnota. Nejčastěji se uvažují pravidla ve vztahu *a zároveň* (spojená spojkou AND) a *nebo* (spojená spojkou OR).

**Defuzzifikace** je proces, při kterém se z vágní (tedy fuzzy) hodnoty vypočítá přesné číslo. Tento proces je pro řízení nezbytný, neboť stroji (v našem případě agentovi) je stejně nakonec potřeba předat přesný signál, co má dělat. Tento proces sestává ze dvou kroků – defuzzifikace jednotlivých fuzzy množin odpovídajících jazykovým výrazům a defuzzifikaci jazykových proměnných, kdy je zpravidla potřeba „sečíst“ několik fuzzy množin, které odpovídají jazykovým výrazům nabývajícím nenulové hodnoty.

Ze svého principu se při defuzzifikaci ztrácí informace, a to v obou krocích. Při defuzzifikaci fuzzy množiny musíme z universa vybrat jeden prvek jako výsledek defuzzifikace, nenulový stupeň příslušnosti jich přitom má až na triviální případy víc. Nás sice budou zřejmě zajímat pouze prvky s *maximálním* stupněm příslušnosti, těch je ale většinou stále víc (v případě spojitého universa dokonce nekonečně mnoho). Pro defuzzifikaci se nejčastěji využívají následující metody:

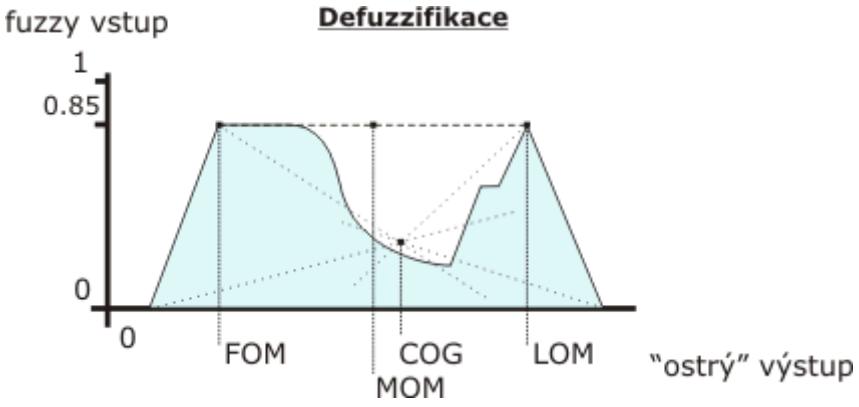
- **COG** (Center of Gravity) – Prvek odpovídající těžišti fuzzy množin, obdobná je metoda **COS** (Center of Sums). Tato metoda bere v úvahu tvar celé množiny, ne pouze prvků nabývajících maximálních hodnot. Pro diskrétní universum je definována takto (Pro spojité universum je potřeba nahradit sumu integrálem, z čehož pramení výpočetní náročnost této metody.):

$$COG(A) = \frac{\sum_{i=1}^{n} A(u_i) \cdot u_i}{\sum_{i=1}^{n} A(u_i)} \tag{2.13}$$

- **MOM** (Mean of Maxima) – Střední prvek (medián) univerza mezi prvky s maximálním stupněm příslušnosti. Pro spojité universum se jedná o průměrnou hodnotu. Tato metoda je definována následující rovnicí:

$$MOM(A) \frac{1}{n_{\max}} \sum_{i=1}^{n_{\max}} u_i^{\max} \tag{2.14}$$

- **FOM** (First of Maxima), **LOM** (Last of Maxima) – Vybere první (poslední) prvek jehož stupeň příslušnosti je maximální. Jedná se o nejjednodušší metody defuzzifikace, v praxi se však příliš neuplatňují.
- **DEE** (Defuzzification of Evaluating Expressions) – Tato metoda je složena z výše uvedených metod tak, že bere v úvahu tvar funkcí příslušnosti – rozlišuje fuzzy množiny na typy odpovídající jazykovým výrazům „malý“, „střední“ a „velký“ (Obr. 2.2), tyto jsou následně defuzzifikovány metodami LOM, COG a FOM v uvedeném pořadí (podle [2] str. 98, [6] str. 6).




**Obrázek 2.4: Grafický přehled základních metod defuzzifikace**

Každá z těchto metod je vhodná pro nějaký druh aplikace, ale žádná nedokáže zabránit výše zmíněné ztrátě informace.

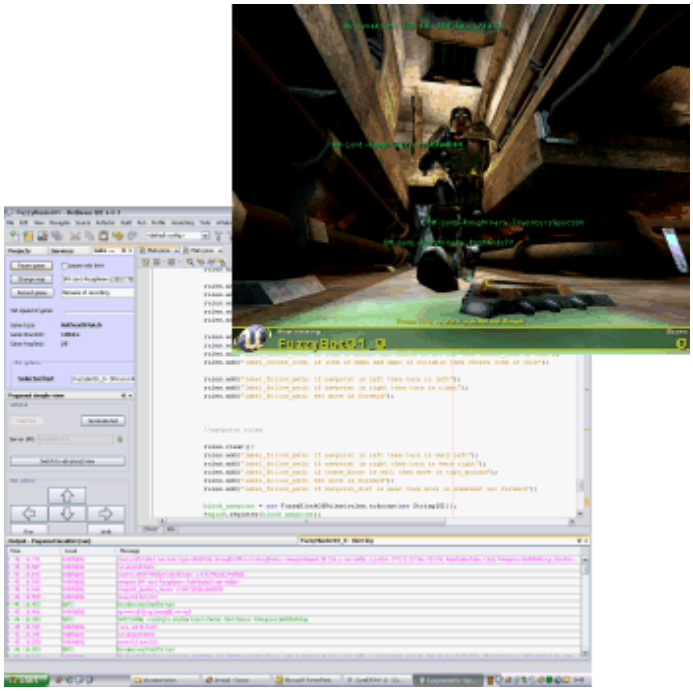
Další fází je defuzzifikace celých jazykových proměnných, neboť zpravidla převádíme stupně příslušnosti několika fuzzy množin, definujících význam jazykových výrazů, které mohou být hodnotou defuzzifikované jazykové proměnné. K tomuto problému můžeme přistupovat dvěma způsoby. První způsob je brát celou jazykovou proměnnou jako jednu fuzzy množinu (udělat v podstatě sjednocení fuzzy množin odpovídajících jednotlivým jazykovým výrazům) a defuzzifikovat ji najednou, tím nám ale mohou vzniknout fuzzy množiny složitých tvarů (jako např. na obr. 2.4), čemuž se většinou snažíme vyhnout. Druhý způsob je defuzzifikovat fuzzy množiny zvlášť. Opět lze použít různé metody:

- **MAX (MIN)** – defuzzifikujeme pouze tu fuzzy množinu, v níž nabývá (v rámci dané jazkové proměnné) nějaký prvek universa největší (nejmenší) stupeň příslušnosti. V kombinaci s defuzzifikací pomocí MOM, FOM a LOM je metoda MAX ekvivalentní k defuzzifikaci celé jazykové proměnné najednou, ale to platí pouze v případě, existuje-li právě jedna taková fuzzy množina.
- **COG** – těžiště výsledků defuzzifikace všech fuzzy množin. V kombinaci s COG defuzzifikací se výsledek blíží defuzzifikaci celé jazykové proměnné, rozdíl je pouze v tom, že „překrývající“ se části množin započítáme dvakrát.
- **AVG** – průměrná hodnota výsledků defuzzifikace všech fuzzy množin. V kombinaci s MOM defuzzifikací se jedná o ekvivalentní přístup k defuzzifikaci celé jazykové proměnné najednou.

## 2.2. Projekt Pogamut 2

Chceme-li ověřit funkčnost metody řízení autonomních agentů ve virtuálním prostředí, potřebujeme nějaké takové virtuální prostředí mít. Nabízejí se dvě možnosti – vyvinout vlastní prostředí, nebo použít prostředí už existující. Navíc, chceme-li ověřit funkčnost takového přístupu v prostředí počítačové hry, je druhá možnost prakticky nevyhnutelná.

**Pogamut 2** je vývojové prostředí určené k rychlé tvorbě virtuálních bytostí (agentů) a experimentování s nimi, které umožňuje návrh, vývoj a ladění logiky agenta. Součástí projektu je i integrované vývojové prostředí. Vytvořené agenty je možné spouštět ve virtuálním 3D prostředí počítačové hry Unreal Tournament 2004, které dovoluje jak vizuální sledování agenta, tak interakci s ním. Mimo to obsahuje Pogamut 2 rozsáhlou knihovnu funkcí pro přístup ke stavu světa, do paměti agenta a pro ovládání jeho chování. Projekt byl vytvořen s ohledem na možnosti dalšího rozšíření. Další informace a podrobnou dokumentaci k projektu naleznete v [3].

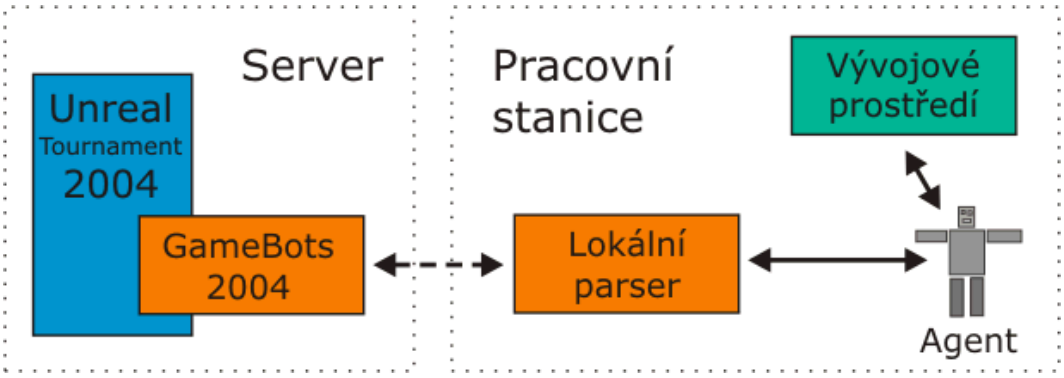


**Obrázek 2.5: Screenshot z vývojového prostředí projektu Pogamut 2 a ze hry Unreal Tournament 2004**

Architektura platformy Pogamut 2 sestává z několika částí, hlavní jsou:

- Unreal Tournament 2004 s rozšířením GameBots 2004
- Parser
- Klient
- IDE

Propojení jednotlivých částí je přehledně vidět na obr. 2.6.




**Obrázek 2.6: Jednotlivé části architektury platformy Pogamut 2 a jejich propojení**

Unreal Tournament je virtuální prostředí (jedná se o komerční počítačovou hru), které díky rozšíření GameBots, fungujícím jako TCP/IP server, umožňuje ovládat agenty z vnějšku. Ovládání je zajišťováno pomocí textového protokolu. To umožňuje, aby bylo virtuální prostředí s agentem na jiném počítači, než je vývojové prostředí a logika agenta (ale není to nutné). Komunikaci na klientské straně zajišťuje tzv. parser, který překládá zprávy z GameBots do javovských objektů, přičemž může běžet jak na stroji s GameBots, tak na stroji s logikou agenta.

Klient je zodpovědný za propojení agentovy logiky a parseru (ať už je na stejném, nebo jiném počítači) a poskytuje programátorovi logiky agenta reprezentaci mapy, agentovy paměti a jeho inventáře. Integrované vývojové prostředí (IDE) pak poskytuje pohodlný přístup ke všem těmto možnostem a umožňuje i sledování a ovládání agenta během jeho testování. Jako platforma pro toto prostředí bylo zvoleno prostředí Netbeans, což je vývojový nástroj pro jazyk Java (nicméně možnosti vývoje agentů se nutně neomezují na jazyk Java) a IDE Pogamutu je do něj zapojeno jakožto plug-in.

V rámci mé práce jsem tento projekt rozšířil o knihovny podporující použití fuzzy logiky pro řízení agenta. Také jsem vytvořil dva příklady takto řízených agentů, jeden jednoduchý, seznamující se základy fuzzy přístupu, a jeden složitější, představující většinu možností využití přidaných knihoven. Oba příklady jsem dal k dispozici jako vzor (template) pro uživatele. Podpora fuzzy přístupu bude k dispozici v nové verzi platformy Pogamut 2.

# 3. Teoretický rozbor

V teoretické části mé práce seznamuji s konkrétní aplikací výše popsané teorie a s problémy, které jsem v průběhu práce řešil a které se týkají použití fuzzy logiky na řízení agentů v komplexním virtuálním prostředí jako je prostředí hry Unreal Tournament 2004.

## *3.1. Aplikace teorie*

Cílem mé práce nebylo vyvinout zcela novou aplikaci fuzzy modelování (to by značně přesáhlo rozsah bakalářské práce), ale najít vhodnou existující knihovnu (případně ji upravit) a tu následně zapojit do projektu Pogamut 2. V první části této kapitoly se zabývám právě výběrem vhodné knihovny pro použití fuzzy logiky, následuje detailnější popis rozdílů mezi aplikovanou fuzzy logikou a teorií uvedenou v kapitole 2.1. Zvláštní kapitola je ještě věnována použitým metodám inference a defuzzifikace.

### 3.1.1. Výběr vhodné existující aplikace teorie

Při výběru jsem kladl důraz zejména na tato kritéria:

- Dostupnost (kompatibilní licence s projektem Pogamut 2).
- Možnost provázání s platformou Pogamut 2.
- Věrnost aplikace fuzzy logiky.
- Přehlednost kódu a možnost dalších úprav a rozšíření.

Z možných kandidátů byl zvolen **Fuzzy Engine for Java** od E. Sazonova [4], který má následující výhody:

- Jedná se o kompletní sadu javovských tříd pro zpracování fuzzy IF-THEN pravidel.
- Je k dispozici zdarma včetně zdrojových kódů (GNU-GPL licence).
- Kód je přehledný a umožňuje jednoduché rozšiřování a úpravy.
- Pravidla jsou zadávána v textové podobě.
- Umožňuje předzpracování pravidel za účelem zvýšení efektivity.

A následující nevýhody:

- Aplikovaná fuzzy logika je značně zjednodušená.
- Není zaručena funkčnost implementace.
- Nejsou podporovány některé funkce specifické pro řízení botů.

Zjednodušení fuzzy logiky a jeho dopady jsou diskutovány v následující kapitole, stejně tak specifické problémy řízení botů (agentů v UT2004) v kapitole 3.2. Oběma těmto problémům se při použití již existujícího software prakticky nedá vyhnout, problém nezaručené spolehlivosti je u volně dostupných nekomerčních knihoven rovněž běžný.

Další možnou volbou byla **Free Fuzzy Logic Library** od Louder Than A Bomb! Software [5], distribuovaná pod BSD open source licencí. Mezi její hlavní výhody patří:

- Komplexnější implementace fuzzy logiky (zejména tvar fuzzy množin).
- Je k ní k dispozici grafický editor a debugger, který je však samostatným programem a není ho tedy možné integrovat do IDE Pogamutu.

A zásadní nevýhody jsou:

- Neobsahuje podporu jazykových operátorů.
- Neumožňuje nastavovat váhu pravidel nebo jejich bloků.
- Jedná se o samostatnou knihovnu psanou v C++, její zapojení do projektu Pogamut 2 (psaného v Javě) by nebylo tak těsné.
- Kód je složitější, případné úpravy a rozšíření by byly pracnější.

Fuzzy logika implementovaná ve FFLL je rovněž zjednodušená, umožňuje však zadávat složitější tvary funkcí příslušnosti fuzzy množin, také umožňuje volbu inferenčního mechanismu (MIN, MAX) a metody defuzzifikace (COG, MOM). Neobsahuje ale podporu jazykových operátorů (ani defuzzifikační metodu DEE), čímž jsou značně omezeny výrazové prostředky rozhodovacích pravidel.

### 3.1.2. Použité zjednodušení teorie

Jak jsem se již zmínil, z volby konkrétní knihovny pro zpracování fuzzy rozhodovacích pravidel plyne určité zjednodušení teorie - tak, aby ji bylo možné efektivně implementovat. Jak je to tedy s fuzzy logikou ve Fuzzy Engine for Java?

Fuzzy Engine podporuje pravidla typu:

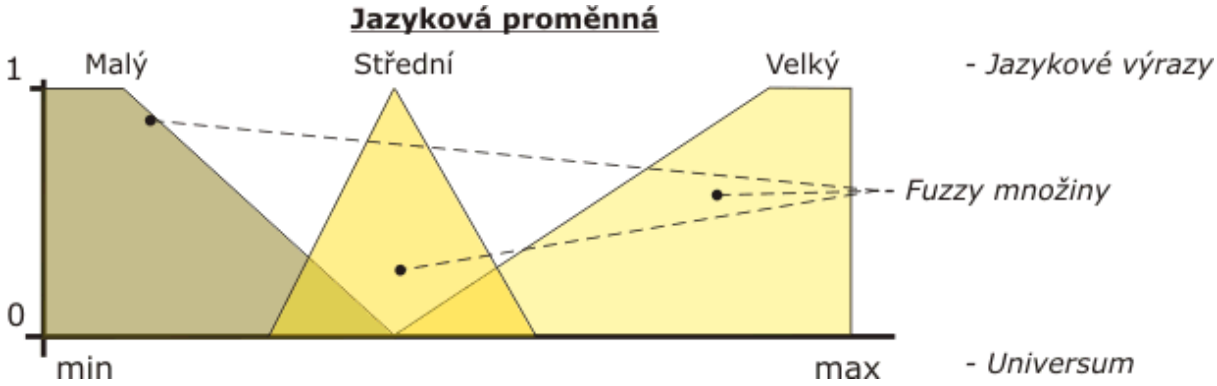
**IF** *lv1* IS <*h1*> *mf1* AND/OR *lv2* ... **THEN** *lv3* IS *mf3* AND/OR ...

Kde se vyskytují

- Jazykové proměnné (*lv1, lv2, lv3*)
- Atomické jazykové výrazy (*mf1, mf3*)
- Logické spojky (AND, OR)
- Evaluační predikace (IS)
- Jazykové operátory (*h1*)

Jazykové proměnné jsou jednoznačně určené svým jménem, universum je definované intervalem na reálné ose (Ten se ale nezadává přímo, je odvozen ze zadaných fuzzy množin reprezentujících atomické jazykové výrazy.).

Atomické jazykové výrazy jsou reprezentovány jako pojmenované fuzzy množiny v rámci jednotlivých jazykových proměnných. Fuzzy množiny však nemohou být libovolného tvaru, ale pouze tvaru **lichoběžníku** (a jeho degenerovaných případů obdélníku, trojúhelníku a přímky). To je asi nejzásadnější zjednodušení teorie, které je podle [2] příliš hrubé, podle [1] pro řízení botů dostatečné (Nezapomínejme, že důležitým faktorem je také rychlost výpočtů.). Fuzzy množina takového tvaru může být jednoduše zadána pomocí čtyř čísel určujících pozice zlomových bodů v univerzu (na reálné ose). Fuzzy množina je tedy reprezentována jako uspořádaná čtveřice vrcholů <levý dolní, levý horní, pravý horní, levý dolní>. Lépe je to vidět na obrázku 3.1 (pro srovnání viz obr. 2.2). S takto definovanými množinami se také velmi jednoduše a efektivně pracuje.




**Obrázek 3.1: Zjednodušený tvar fuzzy množin z obr. 2.2**

Logické spojky AND a OR jsou definovány jako minimum a maximum, což odpovídá vzorcům (2.2) a (2.3). Žádné další fuzzy operace podporovány nejsou (fuzzy negace NOT je implementována jako jazykový operátor, ale odpovídá (2.4)).

Stejně tak evaluační predikace odpovídá výše uvedenému významu – redukuje výraz X is A (X je objekt, A jazykový výraz) na příslušný jazykový výraz, tedy fuzzy množinu daného jména definovanou v odpovídající jazykové proměnné.


**Obrázek 3.2: Vliv jazykových operátorů na fuzzy množiny zjednodušeného tvaru**

Jazykové operátory byly původně ve Fuzzy Engine for Java definovány podle (2.6) a (2.7). Nevhodnost tohoto řešení je diskutována v kapitole 2.1.3 a také v [2] str. 52-55. Rozhodl jsem se tedy předělat implementaci jazykových operátorů tak, aby kromě změny strmosti umožňovaly změnu velikosti množiny prvků s jednotkovým stupněm příslušnosti. V původní implementaci operátory nebyly aplikovány na fuzzy množiny jako takové, ale pouze na hodnotě získané fuzzifikací. To jsem změnil a nyní jsou operátory definovány i nad fuzzy množinami (přesněji nad jejich reprezentací pomocí uspořádané čtveřice bodů) a díky tomu splňují podmínky z kapitoly 2.1.3. Vliv takto definovaných jazykových operátorů na fuzzy množiny zjednodušeného tvaru použitého ve Fuzzy Engine for Java je vidět na obr. 3.2.

### 3.1.3. Použité metody inference a defuzzifikace

V původní verzi Fuzzy Engine for Java byla implementována inference pomocí produktu (2.11) a defuzzifikace metodou COG (2.13). Produktová inference však není pro množiny lichoběžníkového tvaru příliš vhodná, neboť vynásobíme-li y-ové souřadnice jeho bodů (hodnoty funkce příslušnosti) nějakým číslem z [0,1], nezmění se x-ové souřadnice (prvky universa) těžiště ani maxim, tím pádem výsledná defuzzifikovaná hodnota vůbec nezávisí na levé straně IF-THEN pravidla. Proto jsem produktovou inferenci nahradil inferencí pomocí minima (2.12), která je pro lichoběžníkový tvar fuzzy množin vhodnější.

Původní implementace COG defuzzifikace nebyla efektivní, protože nebrala v potaz zjednodušený tvar fuzzy množin a složitým způsobem aproximovala integrál (tedy plochu) funkce příslušnosti. Defuzzifikaci jsem tedy přepracoval a rozdělil ji na tři fáze – **kompozici**, **defuzzifikaci** a **sumaci**.

Kompozice je vlastně ještě součástí inference – je to fáze, ve které se řeší situace, kdy je na pravé straně několika pravidel stejná fuzzy množina. K dispozici jsou metody uvedené v kapitole 2.1.5, tedy minimum, maximum, součet a průměr. Další fází je samotná defuzzifikace fuzzy množin, implementovány jsou metody COG (2.13), MOM (2.14) a DEE, jejich podrobný popis je opět v kapitole 2.1.5. Poslední fází je tzv. sumace, neboli spojení výsledků defuzzifikace jednotlivých fuzzy množin v rámci jedné jazykové proměnné. K tomu slouží metody minimum, maximum, těžiště a průměr, taktéž podrobněji popsané v kapitole 2.1.5.

## *3.2. Specifika řízení agentů*

V úvodu této kapitoly přesněji definuji pojem agenta a aplikaci tohoto pojmu na prostředí počítačových her. Rovněž se zabývám obecnou myšlenkou řízení takového agenta fuzzy logikou a možnými přínosy tohoto přístupu. Několikrát zmiňuji realistické chování – nad tím, co to je a jak ho dosáhnout, se zamýšlím následně. Dále popisuji problém s propojením symbolického zpracování fuzzy rozhodovacích pravidel s virtuálním prostředím, následuje zamyšlení nad problémy spjatými obecně s použitím fuzzy přístupu v počítačových hrách typu Unreal Tournament 2004 a detailnější rozbor jednoho z konkrétních problémů týkajícího se formulace fuzzy pravidel.

### 3.2.1. Obecná myšlenka fuzzy agenta

Existuje mnoho různých definic agenta a agentového programování, principiálně se jedná o abstraktní nástroj k popisu, vysvětlení a předvídání chování komplexních systémů. Agent je umístěn v nějakém prostředí, jehož stav (stav zde znamená nejen aktuální stav, ale sekvenci stavů a akcí agenta končící aktuálním stavem, tzv. běh agenta) je schopen vnímat a které může ovlivňovat svými akcemi. Agent je definován jako funkce:

$$Ag : R^E \rightarrow A_C \tag{3.1}$$

kde $R^E$ jsou běhy agenta končící nějakým stavem světa a $A_C$ jsou akce agenta. Konkrétně v našem případě se jedná spíš o tzv. **umělé bytosti** – agenty, kteří se pohybují v realistickém virtuálním prostředí (tedy prostředí, které se do určité míry snaží modelovat realitu), mají nějaké tělo jakožto součást prostředí a do určité (co největší) míry se snaží napodobovat lidské (nebo zvířecí) chování.

S tímto typem agentů (umělých bytostí) se nejčastěji setkáváme v počítačových hrách, které jsou v dnešní době většinou vybaveny realistickou grafikou, dost často i celkem uvěřitelnými fyzikálními vlastnostmi, ale opravdu realistické chování je stále spíše vzácností.

Povšimněme si, že cíl herní AI (umělé inteligence) se podstatně liší od cíle klasické AI. V klasické umělé inteligenci jde většinou o to, aby stroj vyřešil nějaký problém inteligentně, ať už za inteligentní řešení považujeme to logicky správné, nebo to člověku nejbližší. Oproti tomu ve hrách (mám na mysli hry spíše simulačního charakteru než hry dvou hráčů jako šachy apod., které jsou skutečně doménou klasické AI) nám jde o to, aby se počítačem ovládané bytosti *chovaly*, jako by je ovládal člověk (nebo jako by „byly živé“). Jde tedy do určité míry o iluzi, zajímá nás výsledné chování, postava tedy nemusí ve skutečnosti být inteligentní, stačí, když to tak vypadá. Také proto v tomto oboru dlouhou dobu nenacházely metody umělé inteligence uplatnění a spoléhalo se více na tvrdou práci designérů (skriptování, stavové automaty apod.), než na samostatně se chovající agenty. V posledních letech se však začíná ukazovat, že někdy je nejlepší způsob, jak simulovat inteligentní chování, skutečně ono chování inteligentním udělat.

Fuzzy logika je jednou z technik, se kterou se čas od času můžeme setkat na různých fórech či v knihách věnovaných modernímu přístupu k herní AI jako s alternativní metodou řízení autonomního agenta. Naděje, které jsou obvykle do uplatnění fuzzy přístupu vkládány, se dají rozdělit do dvou hlavních skupin – zvýšení realističnosti chování a zjednodušení návrhu.

Realističtější chování je výsledek, který v průmyslových aplikacích fuzzy logiky většinou nemá smysl, nicméně se zdá, že by mohlo jít o přirozený důsledek její aplikace na řízení agentů. Člověk, jehož chování se snažíme napodobit, se obvykle také nerozhoduje „ostře“, tedy podle klasické logiky, ale spíš fuzzy – chce A, ale ne úplně, trochu i B. Stejně je to s různými informacemi, většinou neuvažujeme o přesných hodnotách, ale zajímají nás spíše přibližná, fuzzy, označení. Proč by bot potřeboval vědět, že je jeho nepřítel 153,45m daleko, nestačí mu vědět, že je „velmi daleko“?

Naopak zjednodušení návrhu a řešení problému je jeden z hlavních důvodů úspěchu fuzzy logiky v průmyslových aplikacích, zřejmě by se tedy tato výhoda měla uplatnit i při návrhu chování agentů. Rozhodovací pravidla řídící chování např. bota ve hře jsou ve své podstatě expertním systémem, možnost zapisování expertních znalostí způsobem bližším přirozenému jazyku a uvažování by tedy měla jejich tvorbu usnadnit. Popsat herní strategii tímto způsobem dokáže každý trochu lepší hráč.

### 3.2.2. Otázka realistického chování

Realistické chování je komplexní problém dotýkající se téměř všech částí hry. Dojem přirozenosti navozuje zejména správné zasazení agenta do prostředí a jeho interakce s ním, interakce mezi jednotlivými agenty, reakce na hráče a v neposlední řadě rozumné rozhodování. V tomto případě však „rozumné“ neznamená „logicky správné“, ale spíš pochopitelné – hráč rozumí tomu, proč agent udělal to, co udělal (ale na druhou stranu by neměl být schopný zcela předvídat, co agent udělá v dalším okamžiku). Velmi dobrou metodou je například, aby agent oznamoval, co dělá. Tak může hráč porozumět i poměrně komplexnímu chování. Tato technika byla dovedena téměř k dokonalosti ve hře F.E.A.R., kde je rozhodování AI komentováno jako rozhovor mezi jednotlivými agenty. Výsledek popsal J. Orkin v [9] takto: „*Jeden hráč vyjádřil na internetovém fóru svůj údiv nad tím, že postavy řízené AI vypadají, jako by skutečně rozuměly své slovní komunikaci. "Nejen, že si navzájem dávají příkazy, ale skutečně dělají to, co je jim přikázáno!" Samozřejmě se ve skutečnosti jedná o pouhý trik, všechna rozhodnutí co říct jsou provedena až poté, co modul*

*chování v jednotce rozhodl, co která postava udělá.*“[*] Podpora realistického chování tedy vychází už z reprezentace světa a znalostí agenta a poměrně těžko se jí dosahuje dodatečně.

Jak nám může v této oblasti posloužit fuzzy logika? Oproti stavovým automatům a běžným IF-THEN pravidlům by mohl fuzzy přístup lépe řešit tzv. přechodná chování, tedy stavy, kdy přecházíme od jedné činnosti ke druhé (Např. bot přestává bojovat agresivně a začíná ustupovat.). Podobně je to s paralelním chováním. Může–li agent pracovat na více úkolech (s různou intenzitou), je vhodné zvážit použití např. fuzzy stavového automatu, který takový návrh usnadňuje. V obou těchto případech jsou ovšem předpokládány určité výhodné vlastnosti agentů a jejich prostředí, umožňující provádět akce s různou intenzitou, popřípadě paralelně. Kromě očekávání jakéhosi všeobecného, těžko popsatelného, „zvýšení přirozenosti“ chování můžeme z fuzzy přístupu získat i nižší předvídatelnost a strojovost, avšak existuje nebezpečí, že chování přestane být srozumitelné.

### 3.2.3. Úskalí symbolického zpracování fuzzy pravidel

Při práci na botech řízených fuzzy logikou jsem narazil na několik problémů, které se zřejmě týkají obecně symbolického modelu agentů. Konkrétně jde o to, že fuzzy rozhodovací pravidla jsou definována v přirozeném jazyce a používají symboly, jako např. „enemy“, označující nějaké objekty (nebo spíš třídu objektů) ve virtuálním světě. Co je tedy myšleno pravidlem *„if enemy is near then …“*? Odpověď se zdá zřejmá, „enemy“ je přeci nepřítel, kterého agent vidí. Problém nastává, vidí-li agent nepřátel víc. Kterého zvolit? Aplikovat pravidlo na každého? Jak ale potom interpretovat výsledek?

Existuje v zásadě několik přístupů:

- Zvolit jeden objekt z množiny objektů daného typu na základě předem daného kriteria a takto zvolený objekt dále symbolicky zpracovat.
- Považovat pravidla jen za jakési vzory a za běhu (nebo předem) vytvářet instance pravidel, kde místo symbolu třídy objektu budou symboly reprezentující jednotlivé konkrétní objekty.
- Procházet množinu objektů dané třídy a pro každý objekt zvlášť vyhodnotit pravidla, která se ho týkají.

První přístup, použitý např. v architektuře F.E.A.R. (viz [1]), je spíš únik před problémem, než jeho řešení. Navíc neumožňuje použít takový systém například pro výběr nepřítele nebo zbraně, což mi připadá jako jedno z vhodných míst pro využití fuzzy rozhodovacích pravidel. Slabinou druhého přístupu je nemožnost jakékoliv kontroly nad zpracováním výsledku, vše záleží na použitém inferenčním mechanismu (viz kapitolu 2.1.5.). Nevýhodou posledního přístupu je opakovaný běh fuzzy inference, což by mohlo být obzvlášť nepříjemné, pokud bychom chtěli např. vyhodnocovat pravidla týkající se všech navigačních bodů. Přesto se ale tento přístup zdá výhodný, neboť umožňuje poměrně přirozené zpracování množiny objektů a další zpracování výsledků. Konkrétní aplikace tohoto přístupu je rozvedena v kapitole 4.1.

V jednom pravidle se také může vyskytovat víc symbolů zastupujících třídy objektů. To by způsobilo, že by nárůst počtu zpracovávaných pravidel byl příliš vysoký (jednalo by se o kartézský součin množin objektů, tedy každý s každým) a navíc by některá pravidla vůbec nemusela mít smysl. V průběhu práce se však ukázalo, že takové situace se prakticky nevyskytují nebo se jim lze vyhnout. Proto jsem tento případ dále neuvažoval.

[*] Vlastní překlad.

### 3.2.4. Úskalí fuzzy přístupu

Fuzzy přístup s sebou ze své podstaty nese určitý stupeň vágnosti (kvůli tomu se používá), což se ale při návrhu komplexního chování může stát noční můrou. Rozpoznat kdy se ještě jedná o přirozené fuzzy chování a kdy už jde o chybu nebývá lehké, stejně tak přesně předpovědět, jaký efekt bude mít přidání nějakého pravidla. Proto je velmi vhodné dbát při návrhu fuzzy systému na jeho jasnou strukturu a na co nejpohodlnější odlaďování.

Kolem fuzzy logiky je vybudovaná rozsáhlá a komplexní teorie, jejíž aplikace však může být obtížná, zejména v prostředí s vysokými nároky na efektivitu (což prostředí herní AI bezpochyby je). V případě jejího (výrazného) zjednodušení se může, i přes zachování mnoha jiných výhod, vytratit přirozenost zápisu rozhodovacích pravidel, která hodně závisí na způsobu modelování přirozeného jazyka. Projeví se to tím, že zapsané pravidlo způsobí jiné chování než by se z jeho přirozené interpretace očekávalo. Dopad pravidel je samozřejmě také velmi ovlivněn tvarem a parametry fuzzy množin reprezentujících jednotlivé jazykové výrazy. Tím se z návrhu stává poměrně komplikovaný proces a přicházíme o požadované usnadnění.

Jak už jsem zmínil v předchozím odstavci, fuzzy rozhodovací pravidla mohou mít oproti těm klasickým značně vyšší výpočetní nároky (viz [1]). To znamená, že není možné použít efektivní algoritmy jako RETE, ale je potřeba vyhodnocovat (téměř) všechna pravidla. Práce s funkcemi příslušnosti fuzzy množin (zejména při defuzzifikaci) může být dost složitá. V situacích, kde záleží na vysokém výkonu, je tedy nutné polevit z přesné aplikace teorie, což však někdy vede ke ztrátě některých výhodných vlastností fuzzy systémů.

Zvýšení výpočetní náročnosti a případné komplikace při návrhu jsou nejspíš hlavní důvody, které zatím brání širšímu nasazení fuzzy přístupu v herní AI. Zatím také nebyl dostatečně jasně zformulován přínos použití fuzzy logiky pro hru samotnou. Je-li však přece použita, většinou je silně zjednodušena a slouží pouze k řešení nějakého konkrétního problému. Například v úspěšné hře Halo [10] měl být původně stav agenta určen čtyřmi fuzzy proměnnými – strach, agresivita, obrana, překvapení. Nakonec byl ale tento fuzzy koncept zredukován, neboť pro hráče bylo výsledné chování těžko pochopitelné. Podobně jsou ve hře The Sims modelovány pocity, v obou případech se však nejedná o plnohodnotné využití fuzzy logiky a rozhodovacích pravidel, ale pouze o nahrazení dvouhodnotových proměnných spojitou škálou. Quake III [11] sice používá o něco složitější koncept - fuzzy relace, sloužící zejména k určení toho, jak moc chce bot nějaký předmět. Stále zde však nenacházíme nasazení fuzzy přístupu tak, jak je podle [2] běžné v průmyslových aplikacích.

### 3.2.5. Úskalí jazykových proměnných

Setkáme-li se někde s návrhem fuzzy bota, často jsou uvedena velmi přirozená pravidla typu *„if enemy is near then action is shoot“* nebo ještě lépe *„if enemy is near and enemy is armed then action is panic“*. Pokud se však pokusíme taková pravidla implementovat, narazíme na zajímavý problém.

Symbol *„enemy“* totiž kromě jakéhosi objektu označuje obvykle jazykovou proměnnou s tímto objektem spjatou. Jazyková proměnná je ale definována jako pětice $< \chi, T(\chi), U, G, M >$, kde je pro nás momentálně zajímavé zejména U – univerzum. To je totiž množina (interval), nad kterou jsou definovány všechny fuzzy množiny reprezentující jazykové výrazy, které může daná proměnná nabývat. Jinými slovy v případě *„if enemy is near and enemy is armed…“* by musely být fuzzy množiny *„near“* i *„armed“* definovány nad stejnou množinou nebo intervalem, což si lze představit jen těžko.

Samozřejmě existuje jednoduché řešení: specifikovat detailněji význam symbolu enemy, tedy přepsat výše uvedené pravidlo např. do tvaru *„if enemy_distance is small and*

*enemy_armed is true then…"* (pozor, *true* je zde ve fuzzy smyslu), čímž ale trochu ztrácíme přirozenost zápisu.

Dalším možným řešením je zachovat přirozeně znějící pravidla, ale porušit matematickou definici jazykové proměnné tím, že jí místo jednoho univerza přiřadíme jejich množinu a k jazykovým výrazům pak přidáme informaci, nad kterým univerzem je definována jejich fuzzy množina. Jiný pohled by mohl být, že se vlastně jedná o sdružení více jazykových proměnných pod jedno společné jméno. Tento přístup ukazuje možnost nechat jádro fuzzy enginu matematicky korektní a toto rozšíření zahrnout až do uživatelského rozhraní. Uživatel tak může navrhovat pravidla jemu pohodlným způsobem a ta jsou následně převedena do matematicky korektní formy pro zpracování fuzzy enginem.

Rozhodl jsem se tuto možnost dále nerozvíjet, protože je možné ji v budoucnu zahrnout jako součást uživatelského rozhraní, které však není součástí této práce.

# 4. Implementace

V této kapitole jsou popsány praktické výsledky bakalářské práce – návrh architektury, návrh a implementace bota využívajícího tuto architekturu a experimenty, které jsem s tímto botem provedl.

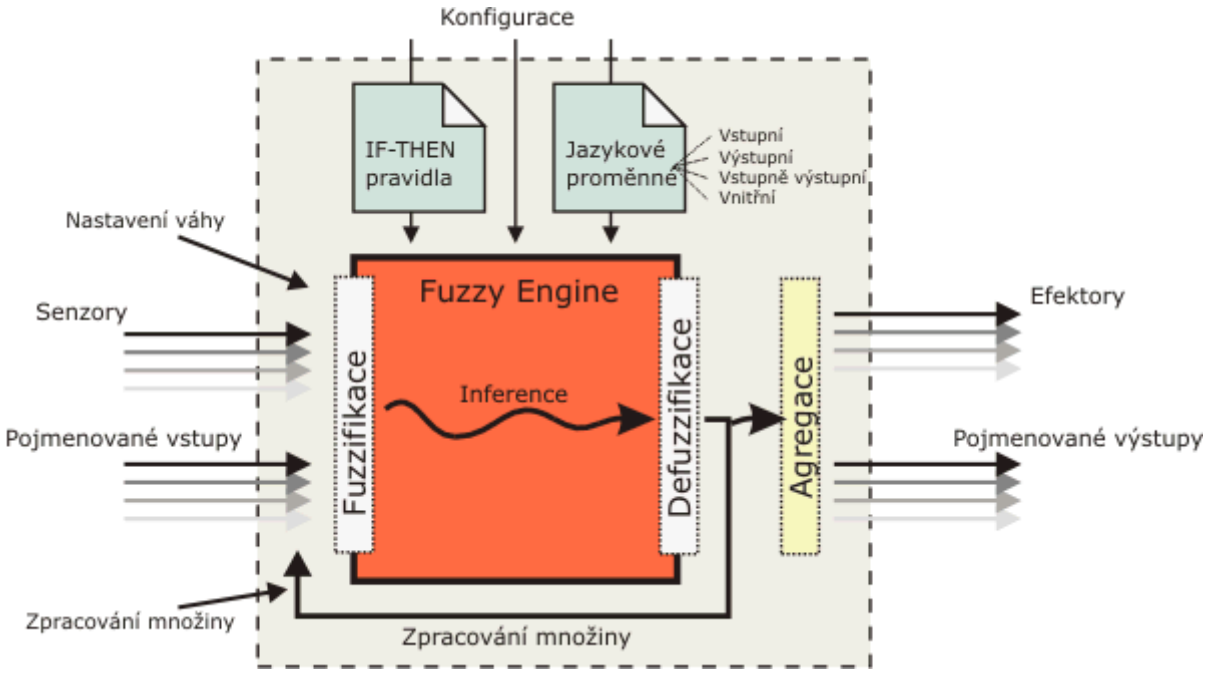
## *4.1. Architektura*

V rámci práce jsem vytvořil architekturu, která by měla usnadnit práci při tvorbě botů řízených fuzzy logikou v prostředí platformy Pogamut 2. Snažil jsem se, aby byly zachovány pokud možno všechny výhody fuzzy přístupu a zároveň eliminováno co nejvíce možných nevýhod. Jedná se o návrh, z nějž byla implementována pouze část, zejména za účelem ověření funkčnosti takto navržené architektury. I přes toto omezení je patrné zpřehlednění a zjednodušení návrhu fuzzy bota oproti práci se samotným Fuzzy Engine for Java.

Nejdříve je popsán základní prvek této architektury – tzv. fuzzy jednotka. Tyto jednotky mohou být propojeny, čímž vznikne komplexní fuzzy bot, který je popsán v kapitole 4.1.2.

### 4.1.1. Fuzzy jednotka

**Fuzzy jednotka** (Fuzzy Unit) je základním stavebním kamenem celé architektury, přičemž může být jak součástí komplexního fuzzy bota, tak samostatným prvkem v rámci jiného typu agenta.




**Obrázek 4.1: Schéma fuzzy jednotky**

Fuzzy jednotka je v podstatě „obal“ fuzzy enginu a tvoří rozhraní mezi implementací fuzzy logiky a logiky bota. Každá fuzzy jednotka je konfigurována sadou IF-THEN pravidel v textové formě a sadou jazykových proměnných následujících typů:

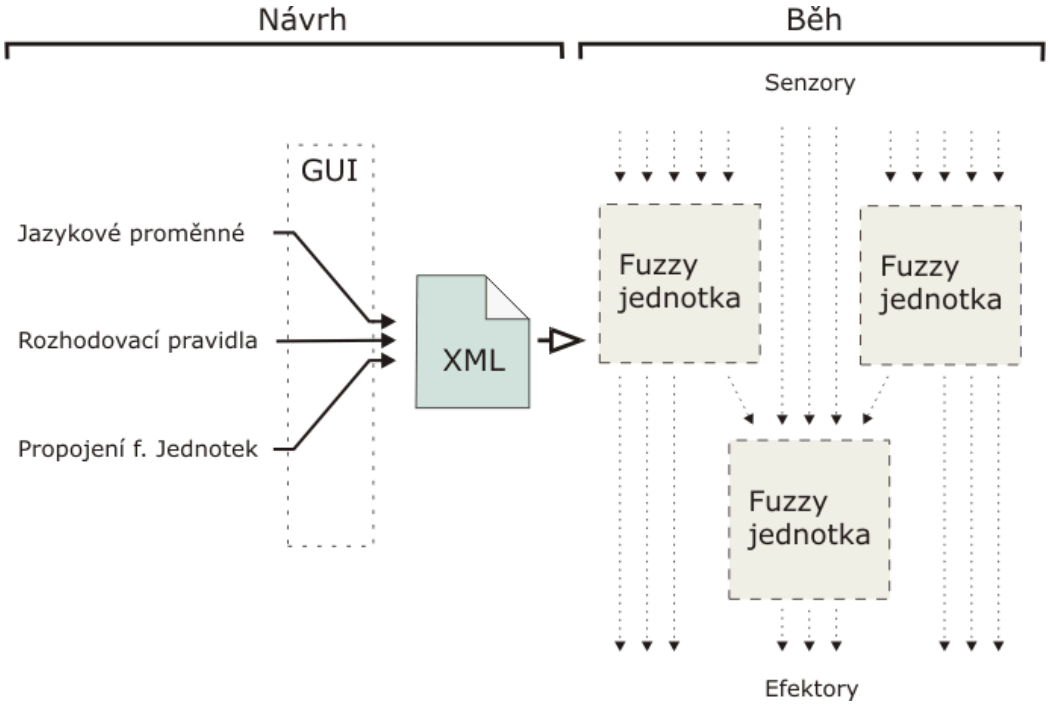
- **Vstupní** – před během fuzzy enginu probíhá fuzzifikace.
- **Výstupní** – po běhu fuzzy enginu probíhá defuzzifikace.
- **Vstupně výstupní** – probíhá fuzzifikace i defuzzifikace.
- **Vnitřní** – neprobíhá fuzzifikace ani defuzzifikace.

Fuzzy jednotka umožňuje zpracovat množinu prvků (nejvýše jednu) způsobem diskutovaným v kapitole 3.2.5., tedy opakovaným spouštěním fuzzy enginu pro jednotlivé prvky. Výstupní proměnné pak mohou být agregovány, to znamená ukládány, a po projití celé množiny zpracovány uživatelsky definovaným způsobem.

Vstupy fuzzy jednotky mohou být jak ze senzorů bota, tak z pojmenovaných vstupů. Výstupy mohou být do efektorů bota nebo do pojmenovaných výstupů, čímž vzniká provázání mezi jednotlivými fuzzy jednotkami. Navíc jsou k dispozici dva vstupy, umožňující zpracovat množinu objektů a nastavit váhu celé fuzzy jednotky. Nastavení váhy jednotlivých pravidel pomocí speciální syntaxe umožňuje Fuzzy Engine for Java od E. Sazonova (viz [4]), váha je v tomto smyslu číslo z intervalu [0,1], kterým je při inferenci vynásobena výsledná hodnota levé strany pravidla a následně je přiřazena pravé straně. Nastavení váhy celé fuzzy jednotky je tedy jen dalším využitím této možnosti.

### 4.1.2. Fuzzy Bot

Konfigurace celého bota sestávající z konfigurací jednotlivých fuzzy jednotek a jejich provázání, je definována na základě XML souboru, který je možné vytvořit ručně nebo generovat pomocí uživatelského rozhraní (schéma viz příloha 7.1).




**Obrázek 4.2: Schéma návrhu a běhu fuzzy bota skládajícího se z jednotlivých fuzzy jednotek**

Ze schématu jsou patrné dvě fáze návrhu - návrh jednotlivých fuzzy jednotek, jejich pravidel a jazykových proměnných a návrh provázání jednotlivých vstupů s výstupy. Každé provázání má jeden zdroj a jeden nebo více cílů. Zdrojem při tom může být výstupní jazyková proměnná jiné fuzzy jednotky nebo senzor definovaný v nějaké sadě chování (objektu třídy **behaviour**). Obdobně cílem může být vstupní jazyková proměnná jiné fuzzy jednotky nebo efektor definovaný v nějaké sadě chování. Kromě jazykových proměnných jsou u každe fuzzy jednotky definovány dva speciální vstupy – vstup pro nastavení váhy (chová se stejně jako běžné vstupy) a vstup pro zpracování pole (může být provázán pouze s objektem třídy behaviour).

Jak jsem již uvedl, implementoval jsem pouze část uvedené architektury, konkrétně fuzzy jednotku a několik dalších pomocných tříd. Fuzzy jednotka při tom slouží jako „obal“ knihovny Fuzzy Engine for Java [4]. Kód samotného fuzzy bota je sice stále potřeba psát ručně, i tak se jeho tvorba značně zjednodušila a zpřehlednila.

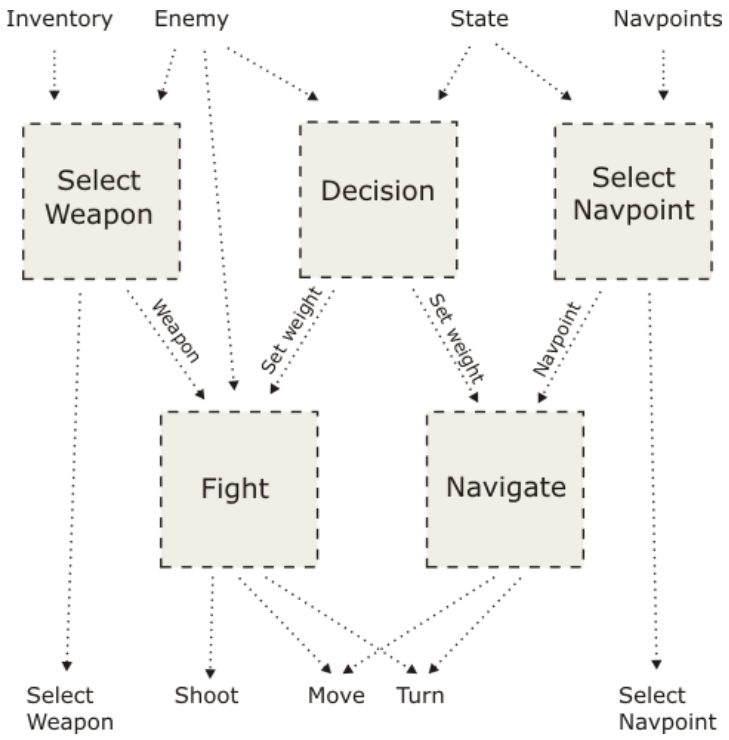
Kromě fuzzy jednotky jsou dále implementována rozhraní **Input**, **Output** a **Aggregator**, jejichž použití je nastíněno v kapitole 4.2.2. Důležitá je třída **FuzzyBehaviour**, která slouží jako základ pro implementaci senzorů a efektorů. Třída **AggregatorFactory** poskytuje několik základních implementací tříd sloužících ke zpracování výsledků běhu fuzzy jednotky (implementujících rozhraní Aggregator). Detailní dokumentace všech tříd je v příloze 7.2 a elektronicky formou javadoc.

## *4.2. Příklad fuzzy bota*

Mým cílem bylo vytvořit příklad bota, který ověří funkčnost navržené architektury a zároveň na něm budou patrné charakteristické vlastnosti fuzzy přístupu. V této kapitole je popsán návrh bota a některé implementační detaily.

### 4.2.1. Návrh

Základní myšlenka tohoto bota je velmi jednoduchá – pokud vidí nepřítele, bojuje, pokud ne, běhá po mapě a sbírá předměty. K orientaci v mapě má k dispozici navigační body a senzor, který mu říká, že je před ním blízko zeď (tzv. trace). Fuzzy rozhodovací pravidla jsou rozdělena do pěti fuzzy jednotek (obr. 4.3). První slouží k rozhodnutí zda bojovat, nebo sbírat předměty, další dvě pracují s množinami prvků – jedna vybírá navigační bod, druhá zbraň. Poslední dvě fuzzy jednotky zajišťují samotný pohyb a akce agenta – první v případě boje, druhá při sbírání předmětů.




**Obrázek 4.3: Fuzzy jednotky řídící bota a jejich provázání**

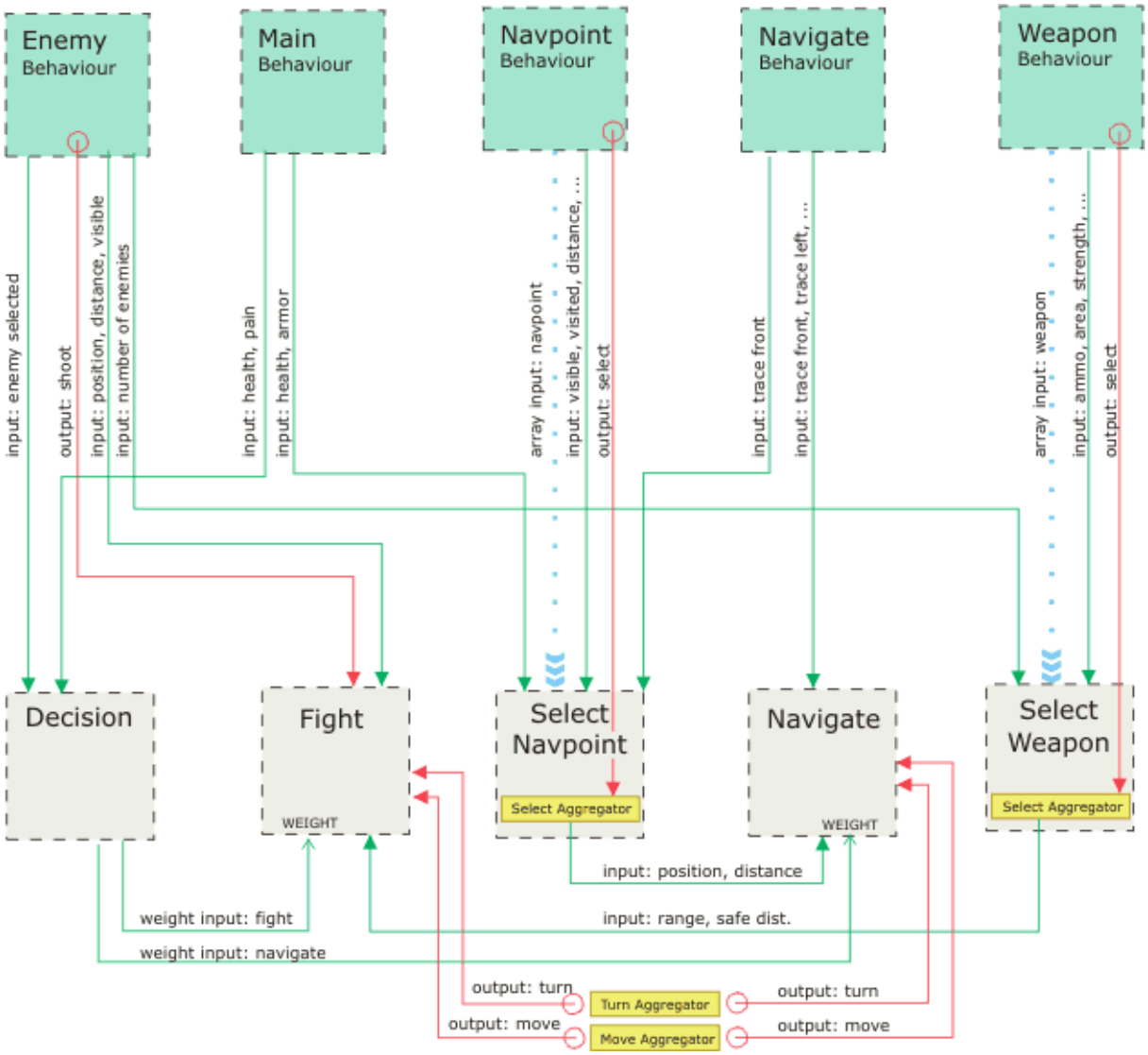
A jak by se v takovém návrhu měl projevit fuzzy přístup? Rozhodnutí, zda bojovat či ne, nezávisí pouze na tom, zda bot vidí nepřítele, ale také na jeho aktuálním zdraví. Může se tedy stát, že bot bude zároveň bojovat, ale také trochu sbírat předměty. To může mít za následek, že bot uteče z boje nebo že v průběhu boje sebere lékárničku. Také to však může způsobit, že se bude chovat poněkud zmateně a boj bude neúčinný. Další výhoda fuzzy přístupu je, že informace, jestli bot nepřítele vidí, nebo ne, nenabývá pouze dvou hodnot. To se dá využít např. tak, že pokud bot nepřítele nevidí, snižuje se postupně pravdivostní hodnota této informace, čímž vznikne jakési krátkodobé pronásledování nepřítele.

U výběru navigačního bodu a výběru zbraně je patrný snadnější návrh. Stačí napsat několik přirozených pravidel a bot si začne poměrně rozumně vybírat. Při chybné definici fuzzy množin však může dojít k různým anomáliím a nečekaným výsledkům.

Samotný pohyb může být díky fuzzy přístupu plynulejší a méně předvídatelný, ale také se může někdy jevit jako zmatený. Stejně jako se chová informace o tom, zda bot vidí nepřítele, může se chovat i informace o tom, zda bot vidí navigační bod. Díky tomu se může bot např. vrátit pro zajímavý předmět.

### 4.2.2. Implementace

Kromě výše zmíněných pěti fuzzy jednotek obsahuje implementace bota také pět objektů třídy behaviour, které poskytují přístup k senzorům, efektorům a dalším funkcím prostředí Pogamut 2. Fuzzy jednotky jsou mezi sebou a s objekty třídy behaviour propojeny pomocí rozhraní několika typů – vstupní, výstupní a vstup z pole. Konkrétní propojení pomocí rozhraní je znázorněno na obr. 4.4.




**Obrázek 4.4: Propojení fuzzy jednotek a objektů třídy behaviour pomocí rozhraní**

Vstupní rozhraní (**input**) slouží k propojení fuzzy jednotky se senzorem, propojení výstupu jedné fuzzy jednotky se vstupem jiné nebo také k nastavení váhy celé fuzzy jednotky. Výstupní rozhraní (**output**) slouží k propojení výstupu fuzzy jednotky s efektorem, chce-li však k jednomu efektoru přistupovat více jednotek, je nutné použít tzv. aggregator. Posledním typem rozhraní je rozhraní vstupu z pole (**array input**), které slouží ke zpracování množiny objektů.

Tento bot používá dvakrát obdobné zpracování množin objektů – pro výběr navigačního bodu a pro výběr zbraně. K výběru se používá jazyková proměnná *select*, která se vyskytuje na pravé straně sady pravidel. Jazyková proměnná *select* je propojena s tzv. **select aggregatorem** a s výstupním rozhraním příslušného objektu třídy behaviour. Vyhodnocení pravidel pro množinu objektů probíhá tak, že jsou postupně vyhodnocena pravidla pro každý prvek množiny a výsledek (defuzzifikovaná hodnota proměnné *select*) je uložen do select aggregatoru, ten po projití celé množiny vybere prvek s nejvyšší hodnotou a tuto hodnotu odešle na příslušné výstupní rozhraní. Nakonec objekt třídy behaviour zjistí který prvek množiny byl vybrán a dále ho zpracuje (Např. při výběru zbraně pošle signál botu, aby změnil svou zbraň.).

## 4.3. Experimenty

Funkčnost implementované architektury a kvalita vytvořeného fuzzy bota byla otestována třemi experimenty, zkoumajícími zátěž procesoru při běhu bota, herní schopnosti vzhledem k botovi, který je součástí projektu Pogamut 2 a přirozenost chování fuzzy bota.

### 4.3.1. Experiment 1 - Zátěž procesoru

Umělá inteligence v počítačových hrách patří mezi aplikace, pro které je zcela kritická efektivita, neboť téměř veškerý výkon procesoru je spotřebován na grafické a fyzikální výpočty. Navíc je často potřeba ovládat větší množství samostatných agentů. V tomto experimentu orientačně porovnávám zátěž procesoru bota řízeného fuzzy logikou a klasického bota.

#### Metodika

K měření zátěže procesoru byl použit software Process Explorer v.11.20 od společnosti Sysinternals [12], který umožňuje měřit zátěž procesoru jednotlivých procesů. Jelikož jsou boti v prostředí Pogamut 2 spouštěni ve stejné JVM (Java Virtual Machine) jako samotné vývojové prostředí, je nutné použít speciální třídu, která bota spustí v samostatném procesu.

```
public class MainSeparateProcess {
    public static void main(String[] args) throws URISyntaxException {
        Main bot = new Main();
        AgentLauncher launcher = new AgentLauncher(bot, new URI("ut://URL"), true);
        launcher.launch();
    }
}
```

**Obrázek 4.5: Kód - Třída pro spuštění agenta v samostatném procesu**

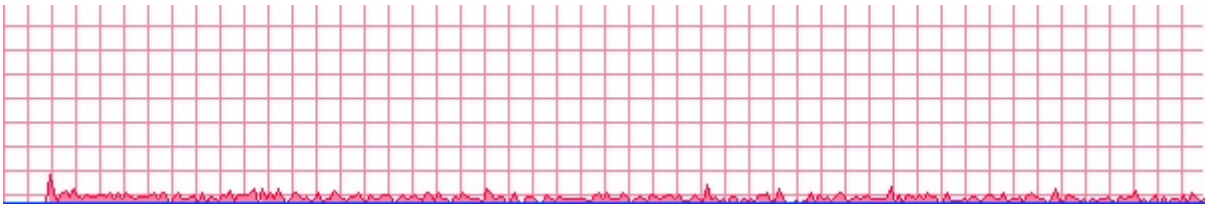
Zátěž procesoru byla měřena u obou botů zároveň během hry typu DeathMatch (boti bojovali proti sobě). Měření proběhlo dvakrát, v malé mapě (Training Day) a v rozsáhlé mapě (Tokara Forest). Zátěž procesoru byla zachycena do grafů pomocí výše uvedeného software s vzorkovací frekvencí 0,5 Hz. Ze získaných záznamů byly dále vybrány přesné hodnoty

jdoucí za sebou v intervalu 10 s po dobu 500 s, které byly dále zpracovány do přehledných grafů a ze kterých byla vypočítána průměrná zátěž procesoru. V grafu bylo vynecháno prvních 20 sekund, kdy u obou botů probíhá inicializace a vzniká extrémní zátěž (Do průměrných hodnot ale byla tato fáze započítána.).

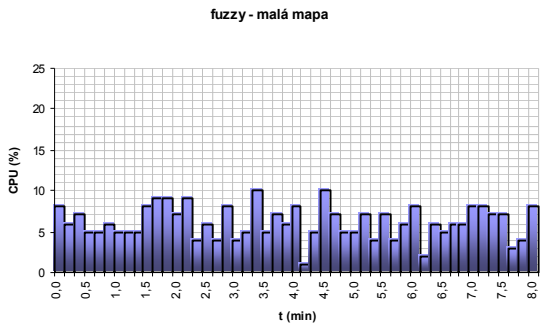
### Výsledky

Následující grafy zachycují zátěž procesoru při experimentu v malé mapě podle Process Exploreru (pro orientační představu) a přesné hodnoty získané z těchto grafů. (Tabulka obsahující přesné hodnoty, viz příloha 7.3.)

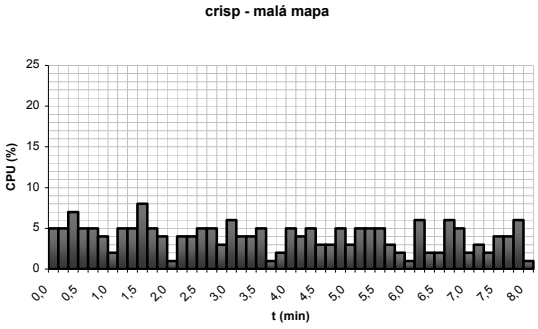
**Graf 4.1: Zátěž procesoru při běhu fuzzy bota v malé mapě**



**Graf 4.2: Zátěž procesoru při běhu standardního (crisp) bota v malé mapě**



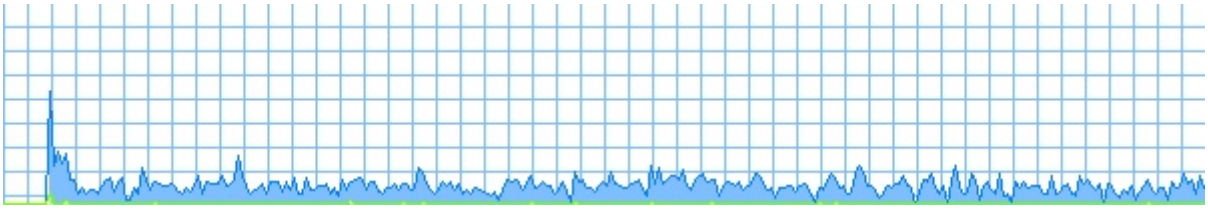

**Graf 4.3: Hodnoty zátěže (fuzzy, malá mapa)**



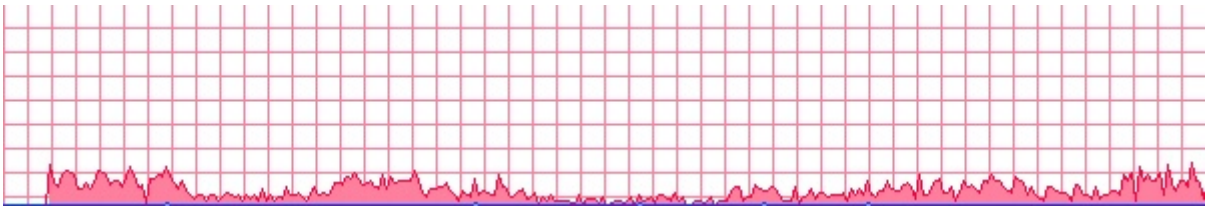

**Graf 4.4: Hodnoty zátěže (crisp, malá mapa)**

Průměrná hodnota zátěže fuzzy bota je **6,4%** a crisp bota **4,3%**.

Dále uvádím grafy vytvořené Process Explorerem při experimentu v rozsáhlé mapě (Tokara Forest).

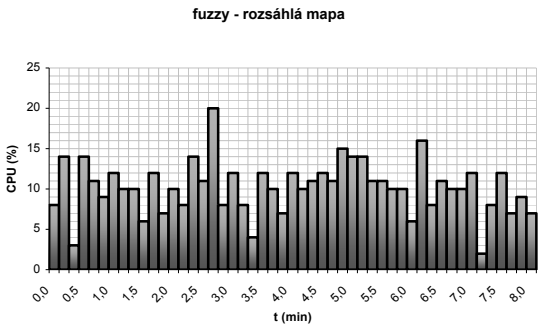


**Graf 4.5: Zátěž procesoru při běhu fuzzy bota v rozsáhlé mapě**

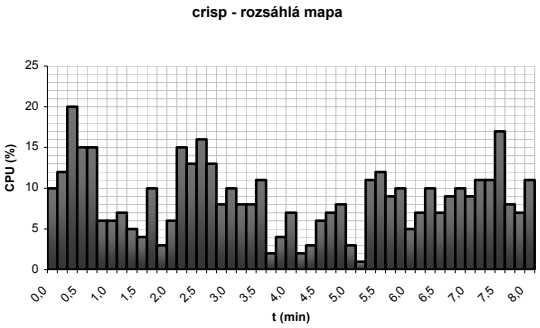


**Graf 4.6: Zátěž procesoru při běhu crisp bota v rozsáhlé mapě**

Přesné hodnoty získané z těchto grafů (Tabulka obsahující přesné hodnoty viz příloha 7.3.):




**Graf 4.7: Hodnoty zátěže (fuzzy, rozsáhlá mapa)**




**Graf 4.8: Hodnoty zátěže (crisp, rozsáhlá mapa)**

Průměrná hodnota zátěže fuzzy bota je **11,1%** a crisp bota **9,0%**, nezapočítáme-li však inicializační fázi (prvních 20 s), dostaneme zátěž **10,2%** pro fuzzy bota a **8,7%** pro crisp bota.

### Vyhodnocení

Podle předpokladů vykazuje fuzzy bot obecně vyšší zátěž procesoru (v malé mapě 1,5x zátěž standardního bota), ale při rostoucí komplexnosti a velikosti prostředí se rozdíl zmenšuje. Nezapočítáme-li inicializační fázi, která je u fuzzy bota výrazně náročnější, neboť probíhá inicializace všech fuzzy jednotek, parsování rozhodovacích pravidel atd., činí rozdíl v zátěži procesoru pouze 1,5%. Navíc standardní bot část své zátěže přesouvá na herní engine, neboť využívá vestavěný A* algoritmus na vyhledávání cest (což by se projevilo zejména v rozsáhlých mapách). Zvýšení zátěže procesoru při použití fuzzy logiky tedy není tak markantní, jak by se mohlo předpokládat.

## 4.3.2. Experiment 2 - Herní schopnosti

Funkčnost navržené architektury a její implementované části byla testována pomocí bota pro hru typu DeathMatch (každý proti každému). Tento experiment ověřuje funkčnost vytvořeného bota vzhledem k tomuto typu hry. V tomto experimentu jsem místo bota z kapitoly 4.2. použil verzi, která projevuje lepší herní schopnosti. Použitý bot se liší tím, že místo fuzzy rozhodovacích pravidel pro výběr navigačního bodu používá jednoduchou sadu klasických rozhodovacích pravidel.

### Metodika

V rámci experimentu proběhlo 25 her typu DeathMatch s limitem deseti bodů („fragů“), kde bot řízený fuzzy logikou (F) bojoval proti standardnímu botu řízenému běžnými rozhodovacími pravidly (C). Aby bylo dosaženo větší objektivity, byla pro každých 5 her použita jiná mapa (použité mapy: Albatross, Roughinery, Calandras, Training Day a Compressed).

### Výsledky

V následujících tabulkách jsou uvedeny výsledky z jednotlivých map.

| mapa: | Albatross | | Roughinery | | Calandras | | Training Day | | Compressed | |
|---|---|---|---|---|---|---|---|---|---|---|
| | F | C | F | C | F | C | F | C | F | C |
| | 7 | 10 | 8 | 10 | 7 | 10 | 10 | 8 | 10 | 4 |
| | 9 | 10 | 7 | 10 | 10 | 8 | 3 | 10 | 10 | 9 |
| | 6 | 10 | 9 | 10 | 3 | 10 | 6 | 10 | 5 | 10 |
| | 9 | 10 | 5 | 10 | 4 | 10 | 7 | 10 | 7 | 10 |
| | 10 | 6 | 9 | 10 | 10 | 8 | 5 | 10 | 10 | 9 |
| průměr: | 8,2 | 9,2 | 7,6 | 10 | 6,8 | 9,2 | 6,2 | 9,6 | 8,4 | 8,4 |
| rozdíl: | 1 | | 2,4 | | 2,4 | | 3,4 | | 0 | |

**Tabulka 4.1: Výsledky souboje fuzzy bota a klasického (crisp) bota v jednotlivých mapách**

#### Vyhodnocení

Z výsledků experimentu je patrné, že fuzzy bot projevuje horší herní schopnosti než bot klasický, ale rozdíl není příliš velký. Značnou roli zde hraje konkrétní implementace bota a čas strávený na jeho vývoji, oba typy botů je vždy možné dále vylepšovat. Přesto je však možné vysledovat některé obecné vlastnosti použitého fuzzy přístupu.

Navigace pomocí fuzzy rozhodovacích pravidel nevede bota přímo po spojnici mezi jednotlivými navigačními body, což může být výhodné v mapách s rozsáhlejšími volnými plochami (Compressed), ale kritické v okamžiku, kdy je nezbytný např. pohyb po lávce nad lávou (vyskytuje se v mapě Training Day – při pádu do lávy bot ztrácí jeden bod). Další nevýhodou ne zcela přesného pohybu je, že bot nemusí okamžitě sebrat předmět, ke kterému běží (musí se vracet nebo ho úplně mine), což je kritické zejména v malých mapách, kde je nutné se rychle vyzbrojit (Training Day, Roughinery, Albatross).

Co se týče samotného souboje botů, rozhoduje zpravidla to, který bot uvidí toho druhého dříve (druhý bot často nestačí zareagovat) a také to, jak je který bot vyzbrojen. Je tedy výhodné vyzbrojit se co nejrychleji, k čemuž napomáhá standardnímu botu využití znalostí o mapě a vyhledávání cest pomocí klasického A* algoritmu.

### 4.3.3. Experiment 3 - Realističnost chování

V tomto experimentu je zkoumána realističnost (přirozenost) chování bota. Dojem přirozenosti navozuje zejména správné zasazení agenta do prostředí a jeho interakce s ním, interakce mezi jednotlivými agenty, reakce na hráče a v neposlední řadě rozumné rozhodování. Jelikož fuzzy rozhodovací pravidla modelují přirozený jazyk, dá se očekávat, že se bude bot řízený těmito pravidly chovat přirozeněji.

#### Metodika

Hodnocení přirozenosti chování do značné míry závisí na subjektivním dojmu každého hráče. Vhodným prostředkem ke zjištění, zda se bot chová přirozeně, je tedy testování pomocí metody nestandardizovaného dotazníku (viz příloha 7.4). Jedná se však pouze o orientační průzkum, neboť velikost a složení výzkumného vzorku hráčů neodpovídá statistickým pravidlům (příliš malý a stejnorodý výzkumný vzorek).

Každý hráč hraje jednu hru typu Death Match (každý proti každému) ve standardní mapě Albatross proti botu řízenému fuzzy logikou (F) a jednu hru proti standardnímu botu řízenému běžnými rozhodovacími pravidly (C). Hra končí v okamžiku, kdy hráč nebo bot dosáhne deseti bodů („fragů“). Po každé hře hráč vyplní dotazník, ve kterém na stupnici 1-10 hodnotí čtyři parametry přirozeného chování:

- Nepřítel se pohybuje přirozeně (kategorie „pohyb“).
- Nepřítel se vyhýbá překážkám přirozeně (kategorie „překážky“).
- Nepřítel reaguje na hráče přirozeně (kategorie „reakce“).
- Nepřítel používá zbraně přirozeně (kategorie „zbraně“).

Získaná data jsou následně statisticky zpracována a vyhodnocena (použitý dotazník viz příloha). Výzkumný vzorek hráčů se skládal z osob ve věku 16-22 let, převážně mužů (průzkumu se zúčastnila pouze jedna žena), kteří většinou měli zkušenost s podobným typem hry.

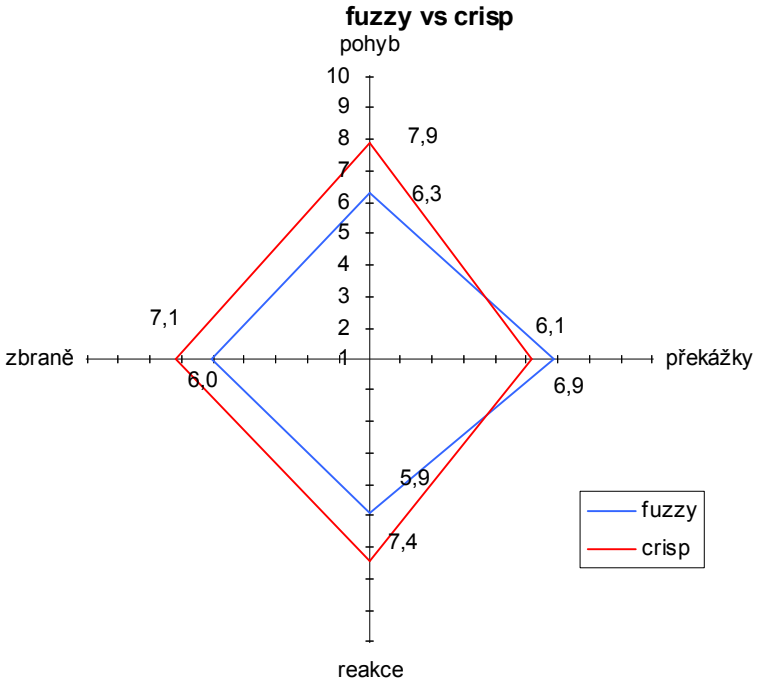
## Výsledky

Následuje tabulka s výsledky získanými od sedmi respondentů, včetně vypočítaného průměru, nejvyšší a nejnižší hodnoty.

| | F (fuzzy) | | | | C (crisp) | | | |
|---|---|---|---|---|---|---|---|---|
| **respondent** | **pohyb** | **překážky** | **reakce** | **zbraně** | **pohyb** | **překážky** | **reakce** | **zbraně** |
| 1 | 5 | 6 | 3 | 5 | 6 | 6 | 7 | 7 |
| 2 | 5 | 8 | 6 | 6 | 8 | 7 | 8 | 8 |
| 3 | 4 | 8 | 9 | 8 | 8 | 9 | 8 | 7 |
| 4 | 8 | 6 | 7 | 9 | 7 | 3 | 9 | 9 |
| 5 | 7 | 6 | 3 | 2 | 8 | 6 | 5 | 7 |
| 6 | 8 | 6 | 6 | 7 | 9 | 7 | 6 | 7 |
| 7 | 7 | 8 | 7 | 5 | 9 | 5 | 9 | 5 |
| průměr | 6,29 | 6,86 | 5,86 | 6,00 | 7,86 | 6,14 | 7,43 | 7,14 |
| minimum | 4 | 6 | 3 | 2 | 6 | 3 | 5 | 5 |
| maximum | 8 | 8 | 9 | 9 | 9 | 9 | 9 | 9 |

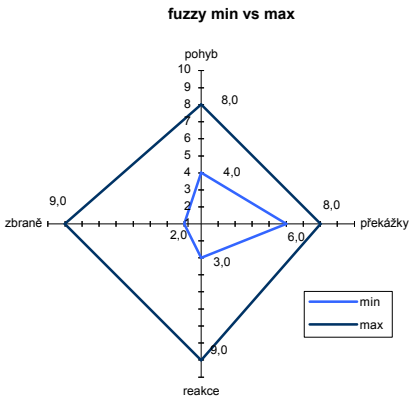
**Tabulka 4.2: Výsledky získané od respondentů**

Pro lepší přehlednost jsou výsledky znázorněny paprskovým grafem, kde každá ze čtyř os popisuje jednu z pozorovaných kategorií.

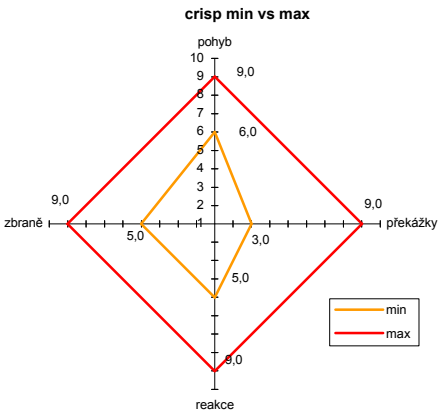



**Graf 4.9: Porovnání průměrných hodnot jednotlivých veličin u bota řízeného fuzzy logikou a bota řízeného klasickou (crisp) logikou**

Dále následují dva grafy, znázorňující rozptyl získaných hodnot u obou botů.




**Graf 4.10: Rozptyl hodnot fuzzy bota**




**Graf 4.11: Rozptyl hodnot klasického bota**

## Vyhodnocení

Je pozoruhodné, že oproti předpokladům se fuzzy přístup pozitivně projevil pouze u jedné kategorie a to vyhýbání se překážkám. Navíc je zde i malý rozptyl získaných hodnot, což značí, že chování je poměrně robustní (tzn., nenastávají výrazné odchylky, jako např. „zaseknutí se“ apod.). To může být způsobeno tím, že fuzzy bot na rozdíl od standardního bota používá senzory, které ve třech směrech (dopředu, doleva a doprava) zjišťují, není-li v blízkosti zeď (tzv. „trace“ senzory). Proč ale fuzzy bot v ostatních oblastech nedosáhl očekávaných výsledků?

Přirozenost pohybu měla spočívat zejména v plynulejším zatáčení a méně „kostrbatém“ probíhání navigačních bodů, což však kvůli poměrně rychlému spádu hry zřejmě není patrné. Navíc každý respondent měl jen krátký čas na sledování hry, která probíhala pouze v jedné mapě a některé složitější varianty chování se tedy nemusely vůbec projevit. Další výhodou standardního bota bylo použití klasického A* algoritmu, díky kterému systematicky prozkoumával celou mapu. Oproti tomu fuzzy bot k výběru dalšího navigačního bodu používá fuzzy rozhodovací pravidla a v některých situacích se mohlo zdát, že se příliš zdržuje na jednom místě.

Reakce standardního bota na zásah hráčem spočívá v přímočarém příkazu „*jsi-li zasažen a nevidíš protihráče, otoč se*“, fuzzy verze obdobného příkazu však nevykazuje potřebnou pohotovost a spolehlivost reakce, což se může zdát nepřirozené. Tento nedostatek by se ale pravděpodobně dal odstranit další prací na logice bota. Další vlastností standardního bota je to, že všimne-li si nepřítele, začne ho velmi důsledně pronásledovat. Nevýhodou takového přístupu je, že bot se takto chová vždy. Oproti tomu fuzzy bot pronásleduje nepřítele jen krátkodobě a ne vždy. Toto chování by však nejspíš našlo uplatnění až při dlouhodobějším pozorování herních vlastností fuzzy bota.

Je třeba podotknout, že průměrné hodnoty ve všech kategoriích se mezi fuzzy botem a standardním botem liší o pouhý jeden až dva stupně, což naznačuje, že je mezi oběma boty ve hře typu DeathMatch pouze malý rozdíl. Fuzzy přístup se projeví zejména v přechodných a paralelních chováních a v chováních, která je možná parametrizovat, žádný z těchto případu

však při běžném souboji dvou hráčů není příliš patrný. Z toho plyne, že fuzzy přístup nepřináší žádné výrazné vylepšení realističnosti chování pro hru typu DeathMatch.

Jak by se tedy dal fuzzy přístup lépe uplatnit v prostředí her typu Unreal Tournament 2004? To je jeden z námětů pro další práci, kterou by mělo usnadnit zpřístupnění fuzzy rozhodování v projektu Pogamut 2. Z již existujících průmyslových aplikací [14] by mohlo vyplynout několik námětů pro výzkum v tomto směru, např.:

- Rozpoznávání (klasifikace) situací v komplikovanějších typech hry (jako Capture The Flag nebo Onslaught).
- Klasifikace herního stylu nepřítele a přizpůsobení vlastního způsobu hry.
- Nízkoúrovňové ovládání končetin, např. chci-li přesunout ruku do nějaké pozice, sebrat nebo položit předmět a podobně. Fuzzy přístup by měl umožnit přesnou a přitom plynulou manipulaci.
- Balancování na kývající se plošině (Bot se pohybuje tak, aby nespadl – vyrovnává pohyb plošiny).

Jak je vidět, fuzzy přístup by mohl být výhodný zejména v oblasti klasifikace a rozpoznávání a v oblasti nízkoúrovňového řízení pohybu (jak bylo ukázáno i v případě zde uvedeného bota).

# 5. Závěr

Ve své práci jsem si dal za cíl prozkoumat možnosti řízení umělých protivníků (botů) ve hře Unreal Tournament 2004 pomocí fuzzy logiky a doplnit o tuto možnost projekt Pogamut 2.

Svůj cíl jsem z velké části splnil a tím jsem položil základ pro další práci v tomto směru. Rozšířil jsem své znalosti v oboru fuzzy modelování a prozkoumal některé oblasti její aplikace na řízení autonomních agentů. Zformuloval jsem některé problémy, se kterými je možné se setkat, a uvedl možnosti jejich řešení. Navrhl jsem architekturu fuzzy bota a částečně ji implementoval, čímž jsem ověřil její funkčnost. Na závěr jsem vytvořil příklad bota řízeného fuzzy rozhodovacími pravidly ilustrujícího možnosti a úskalí diskutované v bakalářské práci. Experimentálně jsem ověřil jeho efektivitu (zátěž procesoru), herní schopnosti a realističnost. Výsledky experimentů jsem řádně vyhodnotil.

Přínosem pro další práci je tedy zejména návrh a částečná implementace architektury fuzzy bota, která může být dále rozšířena např. o uživatelské rozhraní a která umožňuje další výzkum využití fuzzy logiky pro řízení autonomních agentů. Také bylo uvedeno několik návrhů, jakým směrem by se mohl další vývoj ubírat – zejména se jedná o oblasti řízení na nižší úrovni (ovládání končetin, pohyb v dynamickém prostředí) a rozeznávání herních situací a stylů.

Tato práce pro mě byla velkým obohacením, zejména díky možnosti spolupráce s týmem projektu Pogamut 2. Věřím, že bude přínosem i pro tento projekt a umožní další práci v oblasti fuzzy logiky a herní AI.

# 6. Seznam Literatury


[1] Champandard A.J. (2004): AI Game Development. New Riders Publishing, USA

[2] Novák V. (2000): Základy fuzzy modelování. BEN – technická literatura, Praha

[3] Pogamut 2: http://artemis.ms.mff.cuni.cz/pogamut

[4] Fuzzy Engine for Java: http://www.intelligent-systems.info/FuzzyEngine.htm

[5] FFLL: http://ffll.sourceforge.net/index.html

[6] A.Dvořák, V.Pavliska, M.Štěpnička, R.Valášek (2004): Dynamic Robot Driven by Linguistic Fuzzy Logic Controller (Research report No. 56). 16th IFAC World Congress, Praha

[7] Zadeh, L.A. (1973): Outline of a New Approach to the Analysis of Complex Systems and Decission Processes, IEEE Trans. Syst. Man and Cybern., 1

[8] Zadeh, L.A. (1973): Quantitative Fuzzy Semantics, Inf. Sci. 3 (1973), 156-176

[9] Orkin, J. (2006): Three States and a Plan: The A.I. of F.E.A.R., Game Developer's Conference Proceedings, International Game Developers Association

[10] Griesemer J., Butcher Ch. (2002): The Integration of AI and Level Design in Halo, 12

[11] Waveren J.M.P (2001): The Quake III Arena Bot, University of Technology Delft, Faculty ITS, 52-55

[12] Process Explorer: http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx

[13] Brom C., Bryson J. (2006): Action selection for Intelligent Systems (whitepaper), University of Bath

[14] Applications for Fuzzy Logic (1996): http://www.esru.strath.ac.uk/Reference/concepts/fuzzy/fuzzy_appl.10.htm, Johannes Kepler University Linz, FLLL

# 7. Přílohy

## 7.1. Schéma konfiguračního XML souboru

```
<fuzzy_units>
    <fuzzy_unit name = ”name” composition = “method” defuzzification = “method” summation = “method” >
        <linguistic_variables>
            <linguistic_variable name = “name” input = “input name” output = “output name” aggregator = “aggregetor type”>
                <membership_function name = “name” dl = “value” ul = “value” ur = “value” dr = “value”>
                ...
            </linguistic_variable>
            ...
        </linguistic_variables>
        <rules>
            <rule> if-then rule </rule>
            ...
        </rules>
    </fuzzy_unit>
    ...
</fuzzy_units>
<mapping>
    <link>
        <source donor_type = “fuzzy_unit / behaviour” donor_name = “name” output = “output name”>
        <target acceptor_type = “fuzzy_unit / behaviour” acceptor _name = “name” input = “array / weight / input name”>
        ...
    </link>
    ...
</mapping>
```

## 7.2. Implementace architektury - dokumentace

Implementoval jsem pouze část uvedené architektury, konkrétně fuzzy jednotku a několik dalších pomocných tříd. Třída **FuzzyBehaviour** slouží jako základ pro implementaci senzorů a efektorů. Třída **AggregatorFactory** poskytuje několik základních implementací tříd sloužících ke zpracování výsledků běhu fuzzy jednotky (implementujících rozhraní Aggregator). Dále jsou implementována rozhraní **Input**, **Output** a **Aggregator**. Detailní dokumentace všech tříd je přiložena elektronicky ve formě javadoc.

Ke konfiguraci fuzzy jednotky slouží metody:

- **addRule** – Přidává pravidlo v textové podobě.
- **addLinguisticVariable** – Definuje jazykovou proměnnou a s ní spjatá vstupní, výstupní a agregační rozhraní.
- **addMembershipFunction** – Definuje funkci příslušnosti fuzzy množiny reprezentující daný jazykový výraz v rámci dané jazykové proměnné.
- **addOutputInterface** – Dané jazykové proměnné přidává další výstupní interface.
- **addWeightInput** – Definuje vstup určující váhu pravidel označených daným popiskem (tzv. label).
- **setArrayInput** – Definuje vstup pro zpracování pole, jedná se o standardní rozhraní java.util.Iterator.
- **getInputInterface** – Sváže výstupní rozhraní dané jazykové proměnné se vstupním rozhraním, které vrátí. Tato metoda slouží k provázání jednotlivých fuzzy jednotek.
- **setDefuzzificationMethod** – Nastaví jednu z metod defuzzifikace (COG, MOM, DEE).
- **setSummationMethod** – Nastaví jednu z metod sumace (MAX, MIN, COG, AVG). Tato metoda je použita pro výpočet výsledné hodnoty z více defuzzifikovaných fuzzy množin v rámci jedné jazykové proměnné.
- **setCompositionMethod** – Nastaví jednu z metod kompozice (MAX, MIN, SUM, AVG). Tato metoda se používá k výpočtu stupně příslušnosti fuzzy množiny, pokud byla na pravé straně dvou a více pravidel.

Konfigurace je následně zakončena metodou **initialize( )**. Za běhu je pak volána metoda **run( )**, která spouští běh fuzzy enginu. Při práci s více fuzzy jednotkami je potřeba dbát, s ohledem na jejich provázání, na správné pořadí definic a zejména pak na pořadí volání metody run.

**Input** je rozhraní, kterým fuzzy jednotka získává hodnoty ze vstupu, používá se tedy ke zpracování senzorů. Obsahuje jednu metodu **get( )**, která vrací požadovanou hodnotu. Každá jazyková proměnná má definováno nejvýše jedno vstupní rozhraní.

Rozhraní **Output** naopak slouží k předávání výsledků ven z fuzzy jednotky, zejména k dalšímu zpracování efektory. Hlavní metodou je **set(**double value**)**, která na vstup dostane požadovanou hodnotu. Obě výše uvedená rozhraní, Input i Output slouží také k svázání dvou fuzzy jednotek dohromady. Jazyková proměnná může mít definováno žádné, jedno nebo více výstupních rozhraní.

Posledním rozhraním je rozhraní **Aggregator**, které rozšiřuje rozhraní Output, a používá se při zpracování pole objektů. Metoda **set(**double *value***)** zde slouží k uložení výsledku jednoho

běhu fuzzy enginu pro další zpracování. Metoda **result( )** vrací výsledek zpracování (je volána až po projití celého pole). Metoda **resultIndex( )** nemusí mít v některých implementacích smysl (např. počítáme-li průměr hodnot), neboť vrací index prvku, který odpovídá vrácené hodnotě (např. pro minimum to bude právě ten prvek, pro nějž bylo minima dosaženo).

Součástí fuzzy balíčku pro Pogamut 2 je i třída **FuzzyBehaviour**, která slouží jako předek pro vytváření tříd definujících chování bota (tedy sad senzorů a efektorů). Těžištěm této třídy je skupina funkcí **getInputInterface**, **getOutputInterface** a **getArrayInputInterface**, které pomocí Java Reflection API vrací na základě uživatelem definovaných metod příslušná rozhraní vhodná k zapojení do fuzzy jednotky.

Další užitečnou pomůckou je třída **AggregatorFactory** obsahující statické metody poskytující agregační rozhraní nejběžnějších typů. K dispozici jsou rozhraní vybírající minimum a maximum a rozhraní počítající aritmetický průměr. Dále je k dispozici tzv. **IndexAggregator**, který při vytvoření dostává referenci na jiné agregační rozhraní a vrací hodnotu danou indexem vybraného prvku daného rozhraní.

## 7.3. Tabulka hodnot při měření zátěže k experimentu 1

Uvedené hodnoty jsou v % zátěže CPU.

| | rozsáhlá mapa | | | malá mapa | |
|---|---|---|---|---|---|
| t(min) | fuzzy | crisp | | fuzzy | crisp |
| 0,0 | 56 | 12 | | 25 | 15 |
| 0,2 | 13 | 18 | | 3 | 7 |
| 0,3 | 8 | 10 | | 8 | 5 |
| 0,5 | 14 | 12 | | 6 | 5 |
| 0,7 | 3 | 20 | | 7 | 7 |
| 0,8 | 14 | 15 | | 5 | 5 |
| 1,0 | 11 | 15 | | 5 | 5 |
| 1,2 | 9 | 6 | | 6 | 4 |
| 1,3 | 12 | 6 | | 5 | 2 |
| 1,5 | 10 | 7 | | 5 | 5 |
| 1,7 | 10 | 5 | | 5 | 5 |
| 1,8 | 6 | 4 | | 8 | 8 |
| 2,0 | 12 | 10 | | 9 | 5 |
| 2,2 | 7 | 3 | | 9 | 4 |
| 2,3 | 10 | 6 | | 7 | 1 |
| 2,5 | 8 | 15 | | 9 | 4 |
| 2,7 | 14 | 13 | | 4 | 4 |
| 2,8 | 11 | 16 | | 6 | 5 |
| 3,0 | 20 | 13 | | 4 | 5 |
| 3,2 | 8 | 8 | | 8 | 3 |
| 3,3 | 12 | 10 | | 4 | 6 |
| 3,5 | 8 | 8 | | 5 | 4 |
| 3,7 | 4 | 8 | | 10 | 4 |
| 3,8 | 12 | 11 | | 5 | 5 |
| 4,0 | 10 | 2 | | 7 | 1 |
| 4,2 | 7 | 4 | | 6 | 2 |
| 4,3 | 12 | 7 | | 8 | 5 |
| 4,5 | 10 | 2 | | 1 | 4 |
| 4,7 | 11 | 3 | | 5 | 5 |
| 4,8 | 12 | 6 | | 10 | 3 |
| 5,0 | 11 | 7 | | 7 | 3 |
| 5,2 | 15 | 8 | | 5 | 5 |
| 5,3 | 14 | 3 | | 5 | 3 |
| 5,5 | 14 | 1 | | 7 | 5 |
| 5,7 | 11 | 11 | | 4 | 5 |
| 5,8 | 11 | 12 | | 7 | 5 |
| 6,0 | 10 | 9 | | 4 | 3 |
| 6,2 | 10 | 10 | | 6 | 2 |
| 6,3 | 6 | 5 | | 8 | 1 |
| 6,5 | 16 | 7 | | 2 | 6 |
| 6,7 | 8 | 10 | | 6 | 2 |
| 6,8 | 11 | 7 | | 5 | 2 |
| 7,0 | 10 | 9 | | 6 | 6 |
| 7,2 | 10 | 10 | | 6 | 5 |
| 7,3 | 12 | 9 | | 8 | 2 |
| 7,5 | 2 | 11 | | 8 | 3 |
| 7,7 | 8 | 11 | | 7 | 2 |
| 7,8 | 12 | 17 | | 7 | 4 |
| 8,0 | 7 | 8 | | 3 | 4 |
| 8,2 | 9 | 7 | | 4 | 6 |
| 8,3 | 7 | 11 | | 8 | 1 |
| průměr | **11,1** | **9,0** | | **6,4** | **4,3** |

## 7.4. Dotazník k experimentu 3

Nestandardizovaný dotazník použitý v experimentu 3:

# dotazník (questionary)

**Osoba (person): .....**

**Zakřížkujte prosím na stupnici dle Vašeho dojmu z chování nepřítele (cross out on the scale according to your experience with the enemy, please):**

**Nepřítel (enemy): .....**

**1. Nepřítel se pohybuje (the enemy is moving):**

**přirozeně (naturaly)**  **nepřirozeně (unnaturaly)**

**2. Nepřítel se vyhýbá překážkám (the enemy is avoiding obstacles):**

**přirozeně (naturaly)**  **nepřirozeně (unnaturaly)**

**3. Nepřítel reaguje na hráče (the enemy reacts on the player):**

**přirozeně (naturaly)**  **nepřirozeně (unnaturaly)**

**4. Nepřítel používá zbraně (the enemy is using weapons):**

**přirozeně (naturaly)**  **nepřirozeně (unnaturaly)**

**Nepřítel (enemy): .....**

**1. Nepřítel se pohybuje (the enemy is moving):**

**přirozeně (naturaly)**  **nepřirozeně (unnaturaly)**

**2. Nepřítel se vyhýbá překážkám (the enemy is avoiding obstacles):**

**přirozeně (naturaly)**  **nepřirozeně (unnaturaly)**

**3. Nepřítel reaguje na hráče (the enemy reacts on the player):**

**přirozeně (naturaly)**  **nepřirozeně (unnaturaly)**

**4. Nepřítel používá zbraně (the enemy is using weapons):**

**přirozeně (naturaly)**  **nepřirozeně (unnaturaly)**

## 7.5. Přehled obrázků a grafů