Obsah

Zoznam obrázkov

1 Úvod

1.1 Predslov

"A štvrtý stupen je skutocný cudzinec, varelse, ktorý zahrnuje všetky zvieratá, lebo s nimi sa rozprávat nedá. Sú živé, ale nevieme odhadnút, akými pohnútkami sa riadi ich chovanie. Môžu byt inteligentné, môžu si sami seba uvedomovat, ale my nemáme možnost sa to dozvediet."
Orson S. Card "Mluvcí za mrtvé"

Evolúcia jazyka je zaujímavou témou vhodnou pre pozorovanie. Táto téma a jazyk samotný je odpradávna parketou filozofov. V dnešnej dobe sa vznikom jazyka zaoberajú rôzne vedné odbory. Pri myšlienke, že by sme sa mohli v budúcnosti dorozumievat s pocítacmi, je namieste si klást otázky, co je to vlastne jazyk a ako vznikol. Tieto otázky budeme musiet mat zodpovedané, aby sme boli schopní plnohodnotne preniest jazyk tak, ako ho poznáme, aj do sveta pocítacov a rozšírit tak dnešné primitívne možnosti komunikácie so strojmi.

Pri dnešných možnostiach využitia pocítacov sa medzi vedné odbory, ktoré sa zaoberajú problematikou vzniku jazyka, zaradila aj umelá inteligencia so svojím osobitným prístupom v riešení problémov. Pomocou simulácii, takzvaných jazykových hier, o ktorých si z filozofického hladiska môžete viac precítat v [1,2,3], sa umelá inteligencia pokúša skúmat príciny a možnosti vzniku jazyka v komunite simulovaných agentov.

1.2 Ciele diplomovej práce

Stanovili sme si dva základné ciele diplomovej práce.

Prvým cielom je priblížit citatelovi tematiku experimentov s jazykovými hrami. Existuje mnoho publikácií zaoberajúcich sa evolúciou jazyka a vela clánkov popisujúcich rôzne typy experimentov v tejto oblasti. Našou snahou je ich zosumarizovat a vytvorit nový prehlad založený nie na typoch experimentov, ale na ich vlastnostiach. Takto chceme priblížit základné pojmy v problematike jazykových hier a zároven ich vysvetlit záujemcom o danú oblast výskumu.

Druhým cielom diplomovej práce je vytvorit vlastný program – prostredie, ktoré má slúžit na experimentovanie v oblasti evolúcie jazyka, hlavne na simulácie založené na multi–agentovej architektúre.

Text práce je rozdelený podla cielov na dve casti.

Druhá kapitola zacína prehladom problematiky evolúcie jazyka. V jej dalších podkapitolách sa venujeme jednotlivým vlastnostiam experimentov.

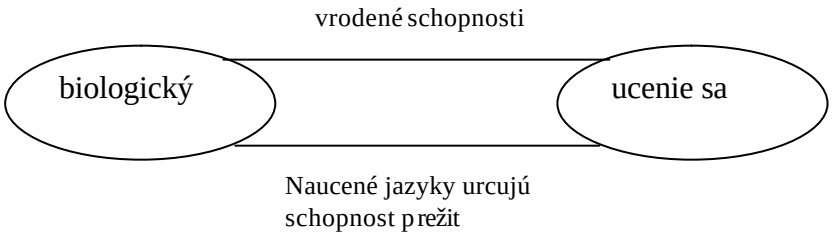
V tretej kapitole popisujeme nami vytvorené prostredie. Podkapitoly sú rozdelené na popis programu, ukážky vytvárania experimentov v programe a nakoniec popis možností rozširovania programu pre dalšie experimenty. V ukážkach experimentov sa pokúšame demonštrovat jednotlivé vlastnosti experimentov, ktoré sú vysvetlené v prvej casti textu. Podrobný manuál prostredia nie je súcastou textu ale samotného programu.

2 Prehlad jazykových experimentov

2.1 Jazyk

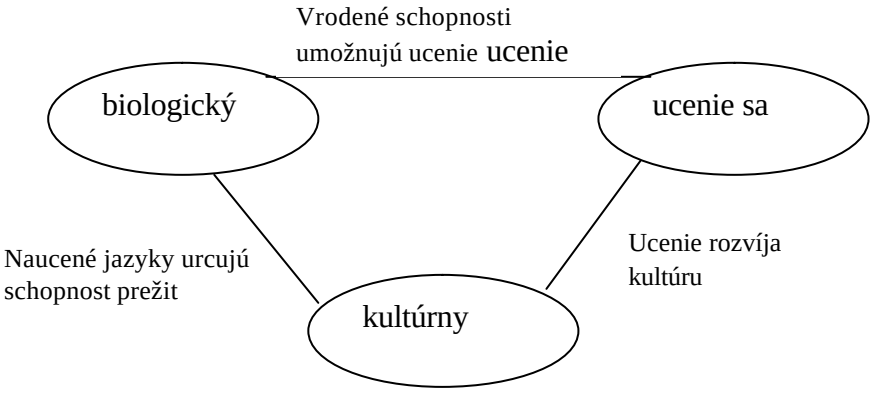
Jedným z cielov výskumu v oblasti evolúcie jazyka je zistit, ako jazyk vznikol. Touto témou sa zaoberá mnoho vedeckých tímov s rôznych oblastí výskumu (lingvistika, filozofia, umelá inteligencia, psychológia), pricom existuje niekolko smerov výskumu jazyka.

Štandardný adaptacný model je založený na Chomského jazykovej paradigme [4,35], ktorá sa sústredí na vrodené lingvistické schopnosti jedinca. Jazyk vníma hlavne ako biologickú charakteristiku. Tento model vysvetluje vznik jazyka ako dôsledok interakcie medzi biologickým vývojom ludského jazykového inštinktu [36] a ucením sa jednotlivca. Túto interakciu znázornuje [Obrázok 1 – jednoduchá interakcia].



Obrázok 1 – jednoduchá interakcia
Obrázok je uvedený v [21]

Další smer [21] pridáva do systému vývoja jazyka tretí komplexný systém – kultúru. Jazyk teda chápe ako výsledok interakcie medzi troma systémami (vid Obrázok 2 – zložitejšia interakcia).



Obrázok 2 – zložitejšia interakcia
Obrázok je uvedený v [21]

Pri vytváraní jazyka je v prvom modeli urcujúci genetický vývoj, ktorý zabezpecuje zmeny a mutácie jazykového aparátu jedinca. Nato, aby sa mohla udiat zmena v jazyku populácie agentov, je nutné, aby u všetkých prebehla podobná mutácia pri genetickom prenose ich génov. Takáto situácia je málo pravdepodobná. Samotná teória genetickej evolúcie jazyka neodzrkadluje vývoj reálneho jazyka, pri pozorovaní ktorého vidíme zmeny v jazyku casto už pocas jednej generácie populácie. Kultúrny proces prináša velký pocet interakcií – lokálnych komunikácií medzi agentmi. Takáto komunikácia dáva lepšie podmienky pre vývoj, organizáciu a šírenie jazyka medzi jednotlivcami populácie. K zmenám jazyka môže prispievat odsun a prísun jednotlivcov (napríklad demografické migrácie), prípadne nespolahlivý prenos vedomostí (napríklad šírenie ústnym podaním z generácie na generáciu).

V tejto práci sa budeme na jazyk pozerat ako na samostatný dynamický adaptívny systém, ktorý vzíde z interakcie všetkých troch komplexných adaptívnych systémov – biologickej evolúcie, ucenia sa a kultúrnej evolúcie. Avšak vela popísaných experimentov sa zameriava a využíva iba interakciu kultúrneho procesu a ucenia sa. Takéto experimenty pochádzajú hlavne z projektov Luca Steelsa a jeho vedeckého tímu pracujúceho pre Sony Computer Science Laboratory v Paríži. L Steels v [6] píše o jazyku ako adaptívnom systéme:

Sformovanie globálnej lingvistickej konvencie v skupine distribuovaných agentov môže byt porozumené pod pojmom samoorganizácia. Existujúca prirodzená variácia v chovaní sa jazyka sa upevnuje, pretože cím viac clenov komunity si privlastnuje rovnaké konvencie, tým silnejšou sa tá konvencia stáva. Avšak táto dynamika je možná iba vtedy, ked agenty menia svoje správanie tak, aby bolo viac v súlade so správaním komunity. Samoorganizácia teda vedie k záveru, aby sme sa na jazykovú komunitu pozerali ako na komplexný adaptívny systém.

2.2 Jazyková hra

Mnohí vedci pri výskume jazyka používajú ako jeden z nástrojov na experimentovanie jazykové hry. Pojem *jazyková hra* zaviedol pravdepodobne ako prvý Wittgenstein [1,2,3].

My budeme v tejto práci pod jazykovou hrou rozumiet experimenty, ktoré majú nasledujúce vlastnosti a pravidlá:

Experiment prebieha v multi–agentovom prostredí, v prostredí kde sa môže nachádzat jeden alebo viac agentov. Agenty môžu byt softvérový výtvor a prostredie softvérová simulácia alebo to môžu byt fyzické roboty a prostredie reálny svet. Dôležitou vlastnostou týchto prostredí je, že agenty sú samostatné jednotky, ich rozhodovanie a konanie v prostredí nie je centrálne riadené. Každý agent si vytvára svoj vnútorný obraz prostredia, tento je pren jedinecný. Agent nemá možnost nahliadnut do vnútornej reprezentácie prostredia iných agentov.

Agenty spolu komunikujú, prebieha medzi nimi interakcia. Taktiež komunikujú s prostredím a tak získavajú potrebné informácie. Na komunikáciu používajú agenty pri skutocných robotoch senzory a efektory, ktoré sa v pocítacových simuláciách simulujú.

Výmena informácií medzi agentmi sa deje formou slov. Pod *slovom* budeme rozumiet retazec písmen, zvukov, pohybov. Agenty sa výmenou slov snažia dorozumiet o *význame*, ktorý sa stáva témou ich rozhovoru. Hrajú spolu jazykovú hru, ktorá väcšinou prebieha medzi dvoma agentmi. Hra je úspešná, ak si agenty porozumeli, v opacnom prípade hra zlyhala.

Súcastou agentov je *slovník*, v ktorom si udržujú informácie o svojich pomenovaniach významov z prostredia ako reláciu slovo – význam. Pre jeden význam môže mat agent viac slov. Jedno zo slov je preferované, týmto agent pomenováva daný význam, ked je v roli rozprávaca. Slovník, tak ako vnútorná reprezentácia prostredia, je jedinecný pre každý agent. Agent ho nemôže priamo sprostredkovat iným agentom. Slovník je zvycajne na zaciatku simulácie prázdny.

Cielom experimentu je dosiahnut jednotný slovník medzi agentmi. Jednotný slovník neznamená, že jednotlivé slovníky agentov sú identické, ale že používajú pre pomenovanie významov, o ktorých sa dorozumievajú rovnaké preferované slová. Môžeme povedat, že jazyk týchto agentov emerguje, lebo spolocný jazyk je vedlajší produkt interakcií jazykových hier a nie ich priamy výsledok. (Agenty sú naprogramované, aby spolu komunikovali, nie aby si vytvárali spolocnú rec.) Jednotný slovník medzi agentmi vzniká

práve vtedy, ked väcšina jazykových hier – interakcií je úspešná. Hovoríme, že dochádza k porozumeniu si agentov.

Jazykové hry môžu mat aj iné ciele. Napríklad, popri vytvorení spolocného jazyka sa môže ocakávat tiež vytvorenie viet, väcšej komplexnosti slov ci dokonca celej gramatiky. Takéto ocakávania musia byt ale niecím podložené (napr. zložitejšími modelmi agentov a prostredia). Inými slovami, ciele jazykových hier môžu byt naozaj široké.

2.3 Pomenovávacia hra

Pomenovávacia hra je konkrétny prípad jazykovej hry. Hra prebieha medzi agentom *A* – rozprávacom a agentom *B* – posluchácom. Rozprávac si vyberie z prostredia lubovolný význam, ktorý sa stane témou ich rozhovoru. K tejto téme nájde slovo zo svojho slovníka, pod ktorým ju pozná a toto slovo sprostredkuje (povie) posluchácovi. Ten sa podla vypocutého slova snaží vo svojom slovníku nájst tému, ktorú pod týmto slovom rozumie on. Ak sa téma vybratá posluchácom zhoduje s témou rozprávaca, hra bola úspešná.

Po úspešnej hre si oba agenty zvýšia úspešnost relácie slovo – téma, ktorú pri hre použili. Ak hra bola neúspešná, agenty si tiež upravia svoje poznatky, ale takým spôsobom, aby nabudúce boli bližšie k úspešnej hre.

Príklad priebehu pomenovávacej hry:

Rozprávac si vyberie objekt, o ktorom chce rozprávat. Napríklad, velkú bielu škatulu, ktorú pozná pod slovom prácka. Sprostredkuje toto slovo posluchácovi. Posluchác pocuje slovo prácka a v svojom slovníku si nájde objekt, ktorému prislúcha toto slovo. Posluchác ale pod týmto slovom pozná objekt malú bielu gulu. Kedže gula a škatula sa nezhodujú, hra neprebehla úspešne. Obaja – rozprávac aj posluchác si upravia slovníky tak, aby si nabudúce lepšie porozumeli.

Treba si uvedomit, že agenty sú naozaj samostatné, teda nie je žiadne centrálne riadenie. Nemôžu si navzájom vidiet do svojich reprezentácií, ako predpokladajú niektoré iné experimenty [37].
Nasleduje presná definícia pomenovávacej hry podla Steelsa [7].

Definícia 1. Pomenovávacia hra

Predpokladajme množinu agentov $A=\{a_1,a_2,...a_{N_A}\}$, kde každý agent $a\in A$ má prístup k množine objektov $O_a=\{o_1,o_2,...o_n\}$. Niektoré alebo všetky objekty sú spolocné pre rôzne agenty. *Slovo* je postupnost písmen konecnej abecedy. Predpokladáme, že všetky agenty majú prístup k rovnakej abecede. *Slovník L* je dynamická relácia medzi objektmi a slovami a dvoma velicinami: *u* – pocet, kolkokrát bola relácia použitá a *s* – pocet, kolkokrát bola použitá úspešne. Každý agent $a\in A$ má svoj vlastný slovník $L_a\subset O_a x N_a x N x N$, ktorý je na zaciatku prázdny. Slovník môže obsahovat synonymá aj homonymá, teda jedno slovo môže byt asociované s viacerými objektmi a objekt môže byt asociovaný s viacerými slovami. Agent $a\in A$ môžeme teraz definovat ako dvojicu $a=(L_a,O_a)$.

Pomenovávacia hra $N=(s,h,o)$ je interakcia medzi dvoma agentmi: rozprávacom *s* a posluchácom *h* o téme *o,* ktorá je objektom, $o\in O_s$. Interakcia prebieha nasledovne:

1. Rozprávac si náhodne vyberie tému zo svojej množiny objektov. Rozprávac si vynúti pozornost poslucháca na vybratú tému. V prirodzenej konverzácii by sa toto rovnalo ukázaniu prstom.

2. Rozprávac *s* zakóduje tému *o* prostredníctvom slova *w*. Vybraté slovo je najúspešnejšie slovo *w*, kde $(o,w,u,s)\in L_s$. Slovo w_1 je úspešnejšie ako slovo w_2, ak $(o,w_1,u_1,s_1)\in L_s$, $u_1\geq u_2\geq 0$ a bud $\frac{s_1}{u_1}>\frac{s_2}{u_2}$ alebo $\frac{s_1}{u_1}=\frac{s_2}{u_2}$ a $u_1>u_2$. Ak je viac slov rovnako úspešných, vyberie sa z nich jedno náhodne.

3. Posluchác *h* dekóduje slovo *w* a odvodí množinu objektov *H* takú, že $o\in H\Rightarrow(o,w,u,s)\in L_h$.

4. Pomenovávacia hra je úspešná, ak $o\in H$.

Kedže interakcie v pomenovávacích hrách prebiehajú iba medzi dvojicou agentov, môžu paralelne prebiehat viaceré pomenovávacie hry medzi disjunktnými dvojicami agentov.

Adaptívna pomenovávacia hra je pomenovávacia hra $N = (s, h, o)$ definovaná vyššie, so zmenami v slovníku oboch agentov, tak rozprávaca ako aj poslucháca. To znamená, že musíme definovat casovú dimenziu v definícii agenta. Agent *a* v case *t* je definovaný ako $a_t = (L_{a,t}, O_{a,t})$. Casový bod korešponduje s udalostou, v ktorej dva agenty hrajú jazykovú hru. Ak rozprávac v case *t* použije slovo *w*, kde $(o, w, u, s) \in L_{s,t}$ a hra skoncí úspešne, potom $(o, w, u+1, s+1) \in L_{s,t+1}$ a $(o, w, u+1, s+1) \in L_{h,t+1}$. Ak hra skoncí neúspešne, zvyšuje sa použitie slova, ale nie úspešnost: $(o, w, u+1, s) \in L_{s,t+1}$ a $(o, w, u+1, s) \in L_{h,t+1}$.

Ostatné zmeny definujúce stav agenta v case *t+1* sú nasledovné:

- Chýbajúci objekt (krok 1 zlyhal)

  Rozprávacom vybraná téma nie je zdielaná posluchácom: $o \notin O_{h,t}$. Hra koncí neúspechom, ale posluchác dostal vedomost o novom objekte podla pravdepodobnosti šírenia objektov $p_o : O_{h,t+1} = O_{h,t} \cup \{o\}$.

- Rozprávac nemá slovo (krok 2 zlyhal)

  Teda neexistuje také *w*, kde $(o, w, u, s) \in L_{s,t}$. Hra koncí neúspechom, ale rozprávac si môže náhodne vytvorit nové slovo *w'* a asociovat ho s objektom vo svojom slovníku podla pravdepodobnosti vytvárania slov $p_c : L_{s,t+1} = L_{s,t} \cup \{(o, w', 1, 0)\}$.

- Posluchác nepozná slovo (krok 3 zlyhal)

  Teda $(o, w, u, s) \notin L_{h,t}$. Hra koncí neúspechom, ale posluchác si môže rozšírit slovník podla pravdepodobnosti absorpcie slov $p_a : L_{h,t+1} = L_{h,t} \cup \{(o, w, 1, 0)\}$.

- Téma nie je úspešne dekódovaná posluchácom (krok 4 zlyhal)

  Teda $o \notin H$. Hra koncí neúspechom, ale posluchác si môže rozšírit slovník podla pravdepodobnosti absorpcie slov $p_a : L_{h,t+1} = L_{h,t} \cup \{(o, w, 1, 0)\}$.

- Ak nenastane žiadna z vyššie definovaných explicitných zmien, definícia agenta sa nezmení: $a_{t+1} = (L_{a,t}, O_{a,t})$.

Uviedli sme definíciu velmi triviálnej jazykovej hry. Existuje vela modifikácii tejto definície (napríklad *analogická pomenovávacia hra* [31], *hádacia hra* [12], *rozlišovacia hra* [10, 11, 9] a iné.). Všetky však majú rovnaký ciel – dosiahnut zhodu slovníkov komunikujúcich agentov.

Jedným z variantov pomenovávacej hry je verzia, v ktorej sa do komunikácie agentov pridal šum. Aj v takomto systéme pri rozumnej miere šumu bola dosiahnutá koherencia slovníkov [13]. Iným variantom pomenovávacej hry môže byt rozšírenie definície agenta o další parameter – pozíciu v priestore [7].

2.4 Príznaky experimentov

V tejto kapitole kategorizujeme použité clánky podla niektorých klúcových vlastností experimentov a tiež podrobnejšie popisujeme dané klúcové vlastnosti. Niektoré clánky sú rozsiahlejšie a zaoberajú sa viacerými metódami. Pri jednotlivých vlastnostiach uvádzame clánky, ktoré sa nám zdali k nim relevantné.

2.4.1.1 Komplexné adaptívne systémy

Komplexný dynamický systém [8] pozostáva z množiny prvkov, medzi ktorými prebieha interakcia, pricom globálne správanie systému je nepriamym a nehierarchickým dôsledkom správania sa jednotlivých prvkov. Neexistuje v nom žiadna centrálna riadiaca jednotka a štandartne je systém otvorený, teda prvky v systéme môžu pribúdat a odbúdat.

Najcastejšie sú pozorované tri hlavné typy správania sa, ktoré závisia na nastaveniach parametrov prostredia:

- *rovnováha* – systém sa posúva bližšie k rovnovážnemu stavu, v ktorom zotrváva
- *samoorganizácia* – v systéme vznikajú disipatívne štruktúry (pokial sú na to vhodné podmienky v prostredí)
- *chaos* – v systéme sa vyskytujú casto nepredvídatelné postupnosti správania sa

Komplexné adaptívne systémy sú podtriedou komplexných dynamických systémov. Zatial, co v dynamických systémoch je správanie prvkov a priebeh interakcií konštantný, v adaptívnych systémoch sa toto správanie mení, co prispieva k vyššej dynamickosti

systému. Ako už bolo spomínané, príkladmi [8] takéhoto systému je biologický vývoj, sociálny systém, ekonomika ci ekologický systém.

Ako sme videli v úvode, jazyk môžeme chápat ako výsledok kooperácií troch komplexných adaptívnych systémov. Ak chceme vediet vysvetlit pôvod a evolúciu jazyka, je potrebné pochopit samotné komplexné systémy. Pre pochopenie pôvodu komplexnosti sa ponúkajú tri prístupy: genetická evolúcia, samoorganizácia s prispôsobovaním a genetická asimilácia. Genetická asimilácia je kombinácia prvých dvoch prístupov.

2.4.1.2 Genetická evolúcia

Vela z prvých experimentov s multi–agentovými prostrediami *využívalo genetickú evolúciu*. Takéto pocítacové simulácie využívajú princípy prevzaté z prírody, kde potomok získa svoje vlastnosti od svojich rodicov prostredníctvom génov.

Podobne sa v niektorých multi–agentových systémoch môžu správat agenty. Agent je nositelom genetickej informácie – génov, ktoré môžu predstavovat napríklad program alebo nejaké pravidlá, ktorými sa agent riadi.

Po niekolkých krokoch simulácie, v ktorých má každý agent "volnost pohybu", sa z celej populácie agentov vyberú tie najlepšie. Výber sa koná podla takzvanej *fitnes funkcie* agenta. Fitnes funkcia je zvolená tak, aby agent, ktorý je najviac schopný prežit ci riešit problém v prostredí, bol nou najlepšie ohodnotený.

Potom sa z vybratých agentov vytvoria potomkovia. Gény vybratých rodicov – agentov sa navzájom skrížia v jeden gén, napríklad prvá polovica prvého rodica a druhá polovica druhého rodica. V skríženom géne môže ešte nastat náhodná mutácia, teda niektoré informácie sa v géne náhodne zmenia – zmutujú. Tento výsledný gén bude patrit ich potomkom – novým agentom vstupujúcim do systému. Podla úspešnosti ich nových génov sa z nich za nejaký cas môžu stat rodicia. Pôvodní rodicia prostredie opúštajú alebo v nom ešte nejaký cas zotrvajú.

Jazykové experimenty založené na genetickej evolúcii vychádzajú väcšinou z nasledujúcich predpokladov:

1. V mozgu existuje *orgán slúžiaci na získavanie jazyka.* Väčšinou tento orgán neslúži na získanie konkrétneho jazyka, ale zahrna v sebe univerzálne princípy. V konkrétnom prostredí sa mu nastavia parametre pre špecifický jazyk daného prostredia.

2. Zachovanie jazyka je spôsobené prenosom génov, ktoré zahrnajú aj jazykový orgán, na potomkov.

3. Zmeny v jazyku vznikajú mutáciou a kombináciou génov, v dôsledku čoho sa mení aj jazykový orgán.

4. Tvar jazyka závisí od úspešnosti prenosu génov pri reprodukcnom procese.

5. Jazyková koherencia by mala nastat podobne, ako sa predpokladá pri iných biologických znakoch (napríklad vznik párových orgánov)

Genetická evolúcia nie je asi to najlepšie na simulovanie evolúcie jazyka. Každopádne to neznamená, že v tomto procese nemôže prispievat inou formou, než sme si popísali [8,21].

2.4.1.3 Sociálna evolúcia – samoorganizácia a adaptácia

Pri sociálnej evolúcií zohráva dôležitú úlohu *samoorganizácia* [38]. Je to proces, v ktorom systém elementov s lokálnymi interakciami a so silnou pozitívnou spätnou väzbou dosiahne globálnu koherenciu, aby sa vysporiadal s prísunom a odsunom energie alebo materiálu. V takomto procese nie sú žiadne známky genetiky. Informácie sú uchovávané len spôsobom samoorganizácie štruktúr. Samoorganizácia bola použitá na vysvetlenie niektorých negenetických biologických fenoménov, akou je napríklad tvorba cesticiek v komunite mravcov.

Pri skúmaní evolúcie jazyka môžeme samoorganizáciou vysvetlit vytvorenie slovníka v skupine distribuovaných agentov. Každý agent v prostredí je schopný komunikovat a zároven musí svoju komunikatívnost rozširovat a zlepšovat. Agenty si predávajú informácie – zvukové, textové alebo obrazové, a tak sa medzi nimi šíria "vedomosti", ktoré sa castým používaním a opakovaním stávajú spolocné. Teda cím viac agentov používa rovnaké vedomosti, tým rýchlejšie sa šíria a upevnujú.

Jazykové experimenty používajúce samoorganizáciu a adaptáciu vychádzajú z trochu odlišných predpokladov [8] než experimenty založené na genetickej evolúcii:

1. Informácie o jazyku sú uložené v pamäti jedinca a nie v jeho génoch. Neexistuje jazykový orgán ale niekolko *základných kognitívnych schopností* agenta, ktoré sa pokladajú za vrodené a jazyk si s nimi musí vystacit.

2. Jazyk sa zachováva šírením a ucením v komunite, co by sme mohli pomenovat kultúrnym procesom.

3. Zmeny jazyka môžu nastat rôznym spôsobom. Komunikácia medzi jedincami nemusí byt v inicializacnej fáze a pri komunikácii stopercentná. Do komunity môže vstúpit cudzinec, alebo sa môže chciet jedinec v komunite zámerne odlišovat a podobne.

4. Selekciu, teda to, aké formy z jazyka sa zachovajú, urcujú rôzne faktory. Príkladmi takýchto faktorov sú: snaha maximalizovat úspech komunikácie, snaha znížit pamätové nároky, zrýchlit kognitívne procesy, byt schopný popísat objekty, schopnost dorozumiet sa a iné.

5. Jazyková koherencia vzniká samoorganizáciou. Medzi komunikáciou agentov je pozitívna spätná väzba, ktorá zarucuje oznacovanie úspešnosti. Formy, ktoré sú úspešné v používaní, sa preferujú, co vedie k ich dalším úspešným použitiam.

Sledovaním priebehu interakcií medzi agentmi môžeme sledovat sociálnu evolúciu jazyka. Poznatky o priebehu môžu priniest trochu viac svetla do evolúcie jazyka vo svete ludí. Tiež nám pomôžu bližšie pochopit, ako sa jazyk vyvinul, co všetko je potrebné – akými kognitívnymi vlastnostami musí agent disponovat, aby jazyk vôbec mohol vzniknút.

2.4.2 Štruktúra systému

2.4.2.1 Multi–agentové systémy

Multi–agentové systémy sme spomínali už pri definícii jazykovej hry. Sú to systémy založené na agentoch – samostatných entitách bez centrálneho riadenia. Jedným z prvých vedeckých smerov, ktorý zacal využívat takéto systémy, bol umelý život [43].

Priebeh simulácie v multi–agentových prostrediach môže byt rôzny. Ak je prostredie reálne (fyzicky realizované) [11,12,14,32], priebeh simulácie je okamžitý. To znamená, že každá entita (agent) v systéme má vlastný výpoctový cas a ten môže využit podla vlastných potrieb. Simuláciu je niekedy tažké presne zastavit, prípadne sa v nej vrátit späť. Tieto simulácie sú vysoko paralelné.

Pocítacové simulácie majú väcšiu možnost riadit svoj priebeh. Jednak vedia simulovat paralelizmus ako pri reálnych systémoch, no môžu tiež celú simuláciu rozdelit do krokov. V každom kroku môže podla potreby spravit svoje akcie jeden alebo viac agentov. Vdaka takejto možnosti riadenia priebehu simulácie je jednoduchšie sledovat závislosti v systéme, prípadne príciny vzniku niektorých javov.

Pri jazykových hrách sa využíva simulovanie po krokoch, kde v každom kroku prebehne jedna jazyková hra – interakcia medzi dvoma vybratými agentmi.

2.4.2.2 Deterministické systémy

Inou možnostou na experimenty sú matematické modely, ktoré sú presné a navyše umožnujú dosiahnuté výsledky aj dokázat.

Ak by sme chceli vytvorit deterministický multi–agentový systém, museli by sme z neho odstránit náhodnost. V našom prípade pri jazykových hrách nastáva náhodnost vždy vtedy, ked sa rozhoduje, ci sa akcia podla nejakej pravdepodobnosti vykoná. Napríklad, ci si agent vytvorí nové slovo (ak mu chýba v slovníku), ci prijme pocuté nové slovo a podobne.

De Jong v [26] ukazuje, že zo simulácie jednoduchej jazykovej hry sa dá odstránit náhodnost tak, aby si systém zachoval svoje vlastnosti. Tento deterministický systém konverguje do atraktorov, v ktorých je dosiahnutá stopercentná komunikácia medzi agentmi. V [26] je poskytnutý dôkaz (jeho rozsah prekracuje zámery tejto práce a preto ho nebudeme uvádzat) uvedenej konvergencie. De Jong dalej uvádza, že nedeterministická verzia systému konverguje do urcitých stavov – tzv. pseudo–atraktorov. Táto konvergencia závisí od nastavenia parametrov simulácie (viac sa o tom zmienime v dalšom texte pri otvorených systémoch).

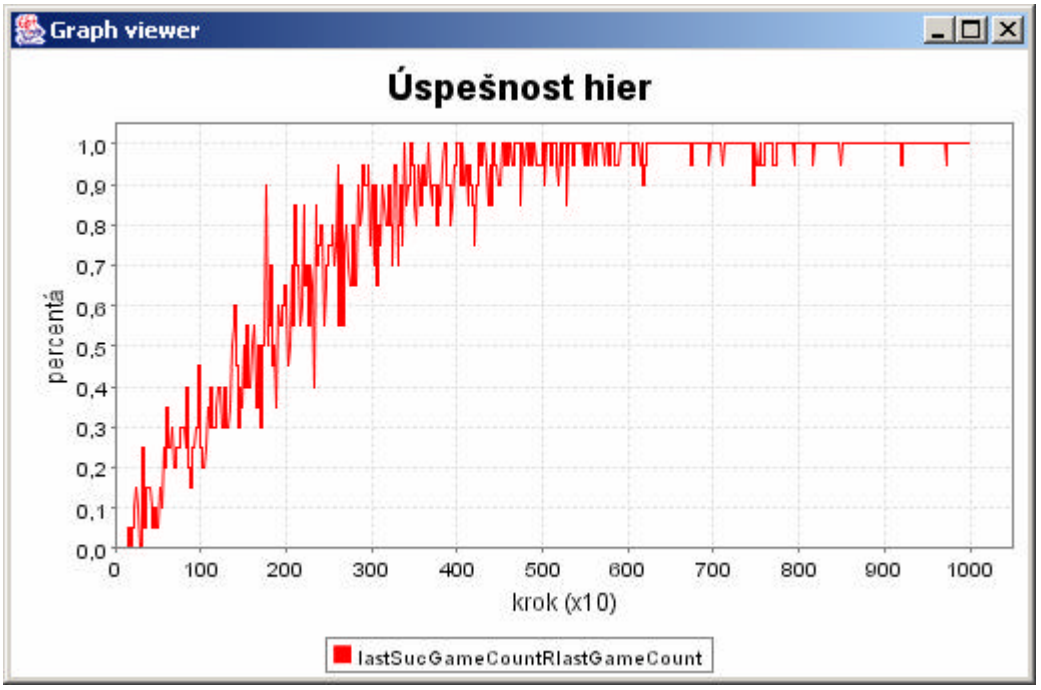
Matematické modely jazykových hier sú tiež témou výskumov [4,5,39].

2.4.3 Otvorenost

2.4.3.1 Zatvorené systémy

Multi–agentové prostredie, v ktorom sa nemení pocet agentov, sa nazýva *zatvorené*. Zatvorené prostredia nemajú význam pri genetickej evolúcii, pretože u nej je fluktuácia agentov potrebná.

V [7] sú prezentované výsledky niekolkých uzavretých jazykových hier. Po niekolkých hrách systém dosiahne stav, kedy je väcšina jazykových hier úspešná a tento stav sa nemení (vid Obrázok 3 – úspešnost hier).



Obrázok 3 – úspešnost hier
Výsledky simulácie, ktorú sme podla [7] realizovali v našom programe.

Je to spôsobené tým, že v tomto stave v systéme nie je nic, co by don prispelo nejakou radikálnou zmenou (ako napríklad nové slovo). Každý agent má väcšinu preferovaných slov rovnakých, co vedie k porozumeniu si v populácii agentov. Tým sa zvyšuje úspešné použitie tých istých slov, ktoré si stále viac a viac upevnujú svoje miesta v slovníkoch agentov. V každom kroku sa preto pocet úspešných hier približuje k maximu.

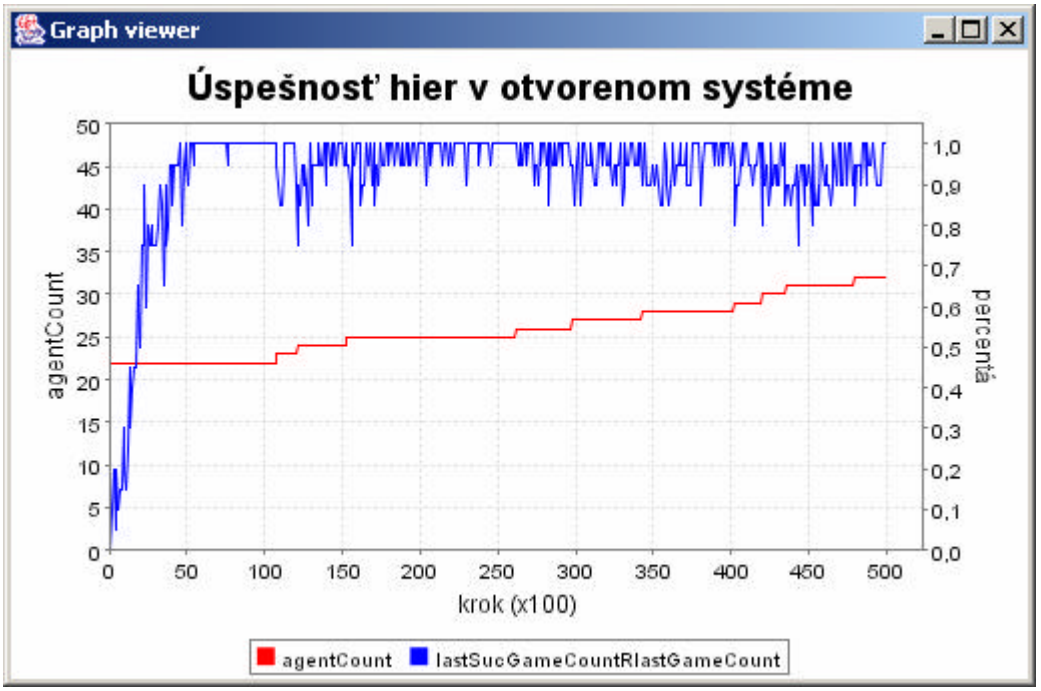
2.4.3.2 Otvorené systémy

V otvorených systémoch pribúdajú nové agenty a odchádzajú staršie agenty, ktoré boli v systéme už niekolko krokov simulácie. Pribúdanie aj ubúdanie agentov sa casto definuje

pomocou pravdepodobností. V každom cykle je pravdepodobnost, že pribudne nový agent p_p a pravdepodobnost odchodu staršieho agenta p_o.

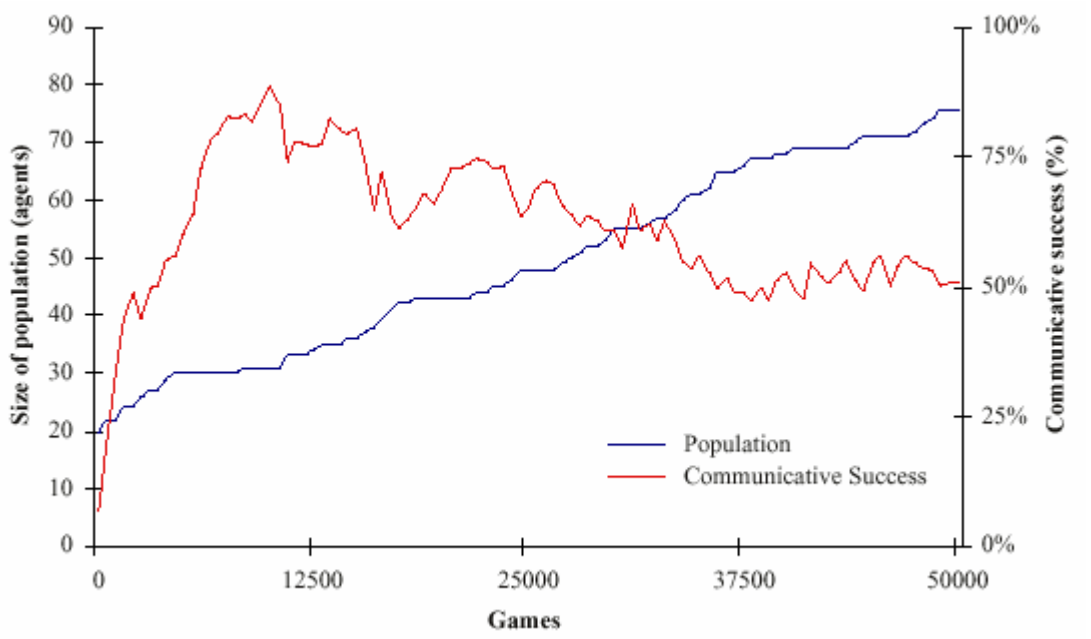
Fluktuácia agentov spôsobuje v systéme rôzne javy a ovplyvnuje výsledky experimentu. Jedným z ovplyvnených výsledkov je pri jazykových hrách pocet úspešných hier. Pribúdanie a ubúdanie agentov vplýva na úspešnost hier niekolkými spôsobmi [7,15,31].

Ak agenty pribúdajú pomaly, teda p_p je nízka (približne 0.00025), celá populácia sa vie rýchlo zosynchronizovat, a aj ked po pribudnutí agentov klesne úspešnost komunikácie, je v krátkom case napravená (vid Obrázok 4 – úspešnost hier a velkost populácie).



Obrázok 4 – úspešnost hier a velkost populácie
Spodnejšia ciara predstavuje pocet agentov, vrchnejšia úspešnost hier.
Výsledky simulácie, ktorú sme podla [7] realizovali v našom programe.

Nové agenty prinášajú do systému docasnú nestabilitu, lebo ich slovníky sú prázdne. Preto každá hra, ktorej sa zúcastnujú, je neúspešná, až kým si nedotvoria svoje slovníky. Dotvorený slovník nových agentov odzrkadluje stav pamäti (pod pamätou rozumieme všetky slovníky všetkých agentov, akoby globálnu vedomost celého systému), ktorou systém disponuje. Nové agenty môžu tiež prispiet novými slovami.



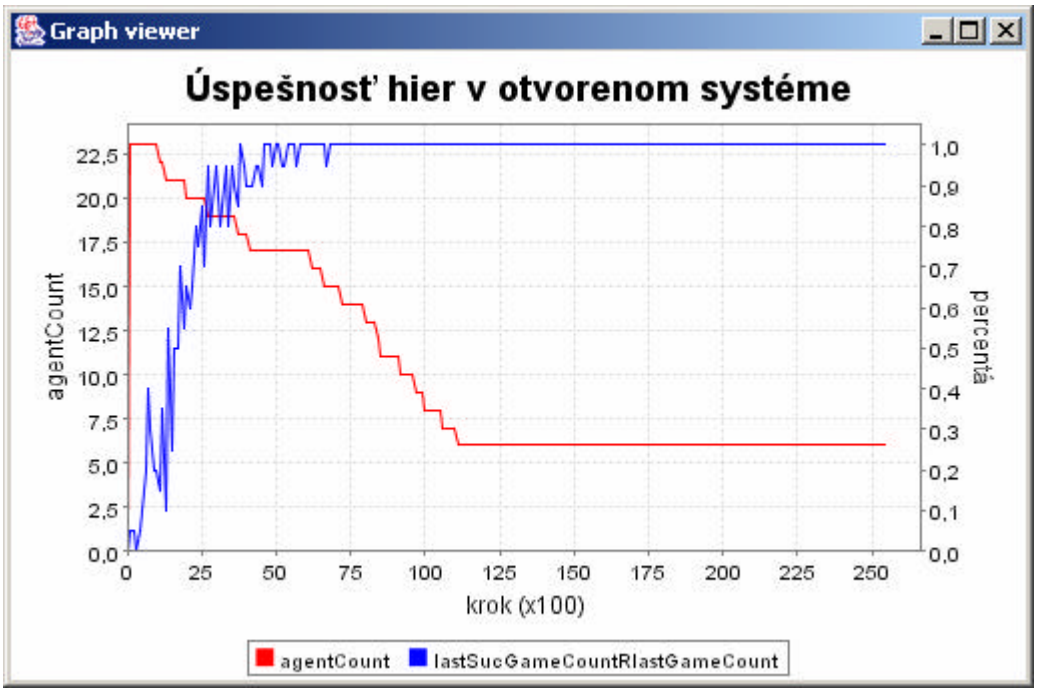
Obrázok 5 – úspešnost hier a velkost populácie 2

Ak je pravdepodobnost p_p vysoká (napr. 0.001), pribúdanie nových agentov je tak rýchle, že sa systém nestíha synchronizovat a úspešnost komunikácie stále klesá (vid Obrázok 5 – úspešnost hier a velkost populácie 2). Pocet jazykových hier nie je dostatocný na prísun nových agentov, preto sa ich slovníky nestíhajú dotvorit. Dôsledkom je velký pocet neúspechov pri hrách.

Pri odchode agenta je situácia iná. Agent zo sebou odnesie svoje vedomosti, ale taktiež možnost pre neúspešnú hru, ak jeho slovník ešte nebol dostatocne vyvinutý. Nižší pocet agentov má k dispozícii viac krokov a teda viac jazykových hier, ktoré skôr dorovnajú rozdiely v ich slovníkoch. Takže odchodom agentov stúpa úspešnost komunikácie k maximu ovela rýchlejšie (vid Obrázok 6 – úspešnost hier a velkost populácie 3).

Ostáva spomenút, ako sa systém správa, ked je v nom plná fluktuácia agentov, odchod aj príchod. Platia tu podobné výsledky, ako sme už uviedli. Ak sú pravdepodobnosti nízke, systém sa vyrovnáva so zmenou v krátkom case. Teda výsledok je podobný situácii na [Obrázok 4 – úspešnost hier a velkost populácie]. Naopak, pri velkých pravdepodobnostiach sa systém nestíha stabilizovat a priebeh experimentu sa viac podobá [Obrázok 5 – úspešnost hier a velkost populácie 2]. Treba si uvedomit, že aj ked samotný odsun agentov má priaznivý vplyv na úspešnost hier, v spolupráci s prísunom nových agentov je jeho vplyv presne opacný. Pridaním nového agenta a odobratím starého sa ešte

viac prehlbuje podiel nových agentov voci starším a tak sa systém dostáva bližšie k stavu, v ktorom zacínal.



Obrázok 6 – úspešnost hier a velkost populácie 3
Klesajúca ciara predstavuje pocet agentov a stúpajúca ciara úspešnost hier.
Výsledky simulácie, ktorú sme podla [7] realizovali v našom programe.

Asi by stálo zato preskúmat, ako sa bude systém správat, ked po pribudnutí agenta tento bude niekolko prvých hier len v pozícii poslucháca. Tento model by viac pripomínal realitu, v ktorej sa deti tiež najskôr len ucia a až neskôr aktívne prispievajú svojimi vedomostami k formovaniu jazyku.

Taktiež by bolo zaujímavé skúmat simuláciu, v ktorej sa príchody a odchody agentov neriadia pravdepodobnostami, ale sa dejú spontánne medzi agentmi. Dva agenty by "splodili" nového agenta, ak si dobre rozumejú a majú na to dost síl a staršie agenty by umierali na starobu (na pocet simulacných krokov). Takáto simulácie je už kombináciou sociálnej evolúcie s genetickou.

Otvorenost systému je dôležitá vlastnost pri experimentoch, viac približuje realitu a prináša do emergencie nové javy [6,7,17,29,31].

2.4.4 Priestor

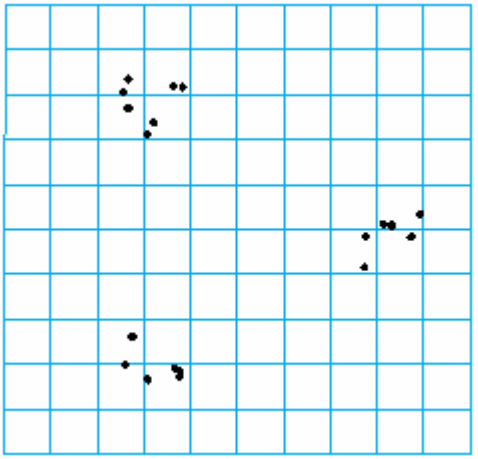
2.4.4.1 Nepriestorové hry

Agenty v jednoduchých jazykových hrách nie sú uložené v priestore. Nemajú žiadnu polohu, nepohybujú sa. Pozeráme sa na ne iba ako na množinu agentov, z ktorej náhodne vyberieme dva, medzi ktorými prebehne jazyková hra.

Takáto vlastnost systému zarucuje, že agenty si sú rovnocenné. Slová sa medzi nimi šíria s rovnakou pravdepodobnostou. Ale aj takto je možné agenty rozdelit do viacerých skupín, množín, a robit jazykové hry len v rámci skupín a neskôr aj medzi skupinami.

2.4.4.2 Priestorové hry

Jazykové hry by mohli prebiehat aj závisle od priestoru [7,31]. Nestací však len pridat agentom dalšiu vlastnost, akou je poloha v priestore. Treba urcit, akú úlohu bude tento priestor v experimente zohrávat.



Obrázok 7 – rozdelenie 20 agentov na mriežke po skupinkách
Obrázok je uvedený v [7]

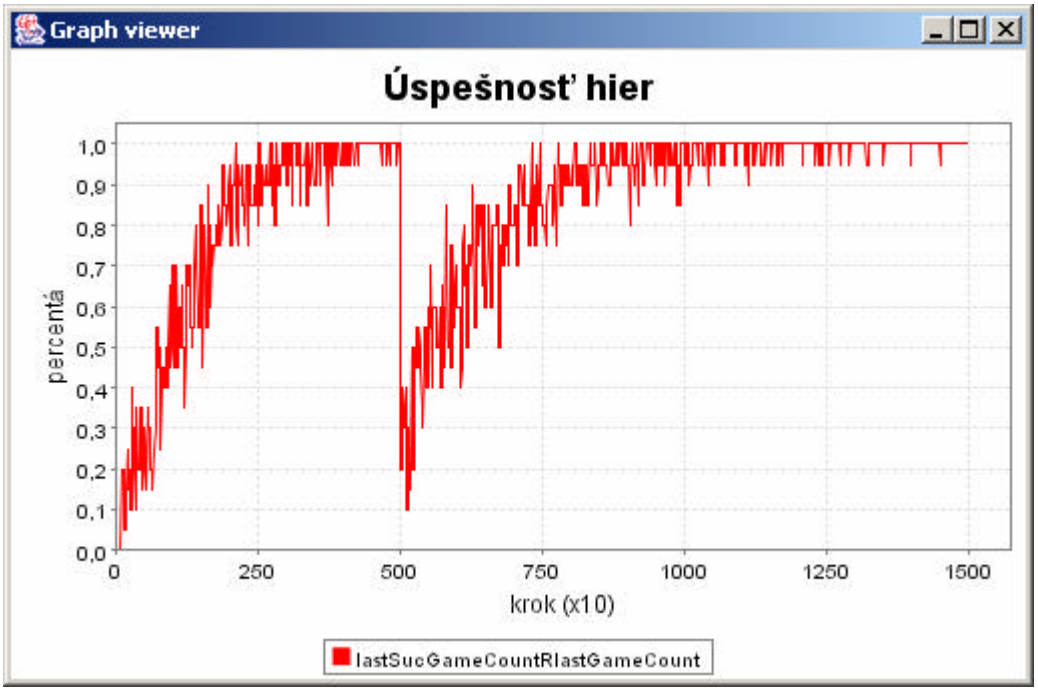
V jednom z prvých experimentov [7] sa sledoval priebeh jazykovej hry, kedy bol výber agentov pre vzájomnú komunikáciu závislý na ich vzájomnej polohe. Agenty boli rozmiestnené na 2D mriežke po niekolkých skupinkách. Na niektorých miestach boli agenty blízko seba, ale skupinky boli od seba dost daleko (vid Obrázok 7 – rozdelenie 20 agentov na mriežke po skupinkách).

Pravdepodobnost výberu dvoch agentov bola urcená podla jednoduchej priestorovej funkcie, cím dalej boli agenty od seba, tým bola pravdepodobnost menšia. Takýto výber

agentov na komunikáciu zaprícinil, že každé zoskupenie si vytvorilo vlastný jazyk. Kedže skupinky hovorili z velkej casti len medzi sebou, každá skupinka dosiahla samostatnú úspešnost v hrách. Teda v rámci jednej skupinky bola komunikácia velmi úspešná.

Napriek tomu si agenty ako celok nevytvorili jeden jazyk, ale jazyk pre každú skupinku zvlášt. Z toho plynie záver, že samotná úspešnost komunikácie sa nedá zobrat ako ukazovatel toho, že agenty majú spolocný jazyk.

V experimente sa po niekolkých krokoch zrušil výber podla vzdialenosti, agenty sa zacali vyberat lubovolne. Ako vidno na [Obrázok 8 – úspešnost hier v priestorových hrách], po tejto zmene nastal pokles úspešnej komunikácie, pretože agenty mali vytvorené slovníky iba v rámci svojej komunity.



Obrázok 8 – úspešnost hier v priestorových hrách
Najprv prebiehali interakcie podla vzdialenosti, v kroku 5000 zacali globálne interakcie. Výsledky simulácie, ktorú sme podla [7] realizovali v našom programe.

V slovníkoch agentov nastala zaujímavá situácia. Každý agent mal preferované slová podla lokálneho jazyka skupiny. Dalšie slová pre objekt boli slová ostatných skupín. Takže agent sa jednak vedel dorozumiet vo svojej skupine a tiež jeho slová poznali agenty v iných skupinách, aj ked ich nepoužívali. Takýmto spôsobom sa dá nasimulovat bilingvalizmus.

Situácia sa trochu zmenila, ked nastali globálne interakcie (vzdialenost už nehrala rolu). Potom sa do popredia všetkých agentov zacali dostávat rovnaké slová a lokálne jazyky zanikali.

V [31] bola priestorovost využitá na klasifikáciu skupín objektov. Objekty, o ktorých sa agenty rozprávali, boli uložené na mriežku po skupinkách. Pravidlá jazykovej hry sa zmenili tak, že si pri výbere nového slova agent nevytváral nové slovo, ale preskúmal svoj slovník a použil slovo iného objektu podla nejakej analogickej funkcie. V tomto prípade bola analógia podla vzdialenosti, takže najbližšie objekty si boli navzájom najpodobnejšie. Takto bolo možné pomenovat jedným slovom celú skupinu objektov. A aby agent nepomenoval všetky objekty iba jedným slovom, bol naviac pridaný parameter pre minimum slov v slovníku, po ktorých môže zacat analógia pri pomenovávaní objektov. Táto hra bola nazvaná *analogická pomenovávacia jazyková hra*.

Na [Obrázok 9 – globálny slovník po 10000 hrách] je výsledný slovník jedného z agentov po takejto hre. Slovník obsahuje asociácie pre 30 objektov a 5 slov. V matici je obsiahnutá pravdepodobnost použitia slova pre daný objekt. Objekty tvoria 3 skupiny, C1 až C3 rozmiestnené podobne ako na obrázku 7. Z tabulky vidno, že agent si vytvoril 5 tried pre skupiny objektov. W1 je trieda pre skupinu C1. Pre skupinu C2 má agent 2 triedy, W2 a W5. Tieto triedy sú rôzne!

V [31] je aj grafická reprezentácia danej tabulky. Pre každú triedu je vypocítané tažisko a priemer rozptylu. Potom sú na mriežke pre objekty nakreslené kružnice pre danú triedu s vypocítaným stredom a priemerom.

Dalej sú v [31] popísané výsledky simulácií pri fluktuácii agentov (otvorený systém) a pri rôznych nastaveniach parametra pre velkost slovníka.

Table 4: Global lexicon built after 10000 games

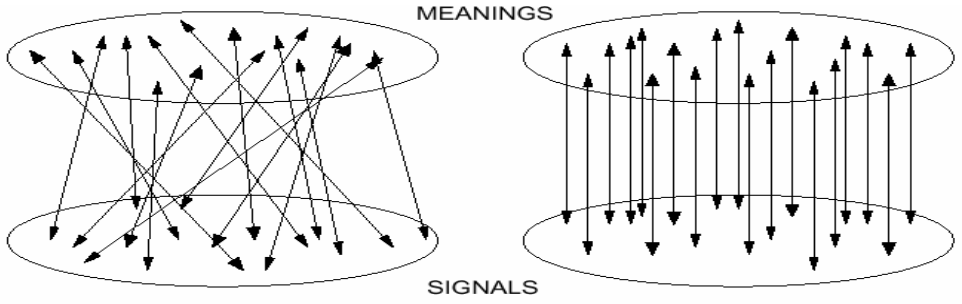
Cluster	Referent	W1	W2	W3	W4	W5
C1	object1	1.00				
C1	object2	1.00				
C1	object3	1.00				
C1	object4	1.00				
C1	object5	1.00				
C1	object6	1.00				
C1	object7	1.00				
C1	object8	1.00				
C1	object9	1.00				
C2	object10		0.46			0.54
C2	object11		0.17			0.83
C2	object12		0.96			0.04
C2	object13					1.00
C2	object14		0.27			0.73
C2	object15		0.17			0.83
C2	object16		0.98			0.02
C2	object17		1.00			
C3	object18				1.00	
C3	object19				1.00	
C3	object20				1.00	
C3	object21			0.22	0.78	
C3	object22				1.00	
C3	object23				1.00	
C3	object24				1.00	
C3	object25				1.00	
C3	object26			1.00		
C3	object27			1.00		
C3	object28			0.33	0.67	
C3	object29			1.00		
C3	object30			1.00		

Obrázok 9 – globálny slovník po 10000 hrách
Obrázok je v [31]

2.4.5 Zložitost jazyka

2.4.5.1 Holistické a primitívne jazyky

Pod *primitívnymi jazykmi* rozumieme jednoduché jazyky bez gramatickej štruktúry. Význam nadobúdajú iba samotné slová. Vo väčšine jazykových hier vznikajú primitívne jazyky. Primitívny jazyk môžeme reprezentovat jednoduchou asociacnou tabulkou ako napríklad v [Tabulka 1 – jednoduchý slovník].



Obrázok 10 – holistické a štruktúrované jazyky

Holistické jazyky [21,23,26] patria medzi primitívne. Tieto jazyky nemajú žiadnu štruktúru a teda ani nevedia zachovat štruktúru významov v prostredí a preniest ju do jazyka. Znázornenie holistického jazyka je na [Obrázok 10 – holistické a štruktúrované jazyky] vlavo.

2.4.5.2 Štruktúrované jazyky

Štruktúrované jazyky oproti holistickým, prenášajú štruktúru priestoru významov do jazyka [21] (vid Obrázok 10 – holistické a štruktúrované jazyky vpravo). [21] popisuje rad článkov, ktoré sa zaoberali možnostami simulácií, ktorých výsledkom sú štruktúrované jazyky.

Aby sa štruktúrované jazyky mohli vôbec vytvorit, musí najprv existovat štruktúra v prostredí, ktorú má jazyk mapovat. S narastajúcou komplexnostou jazyka narastá aj komplexnost agenta. Obycajná asociacná tabulka by si s takýmto jazykom neporadila, pretože nemá možnost uchovávat informácie o jeho štruktúre.

V niektorých experimentoch sa podarilo spravit jednoduché štruktúrované jazyky pomocou rekurentných neurónových sietí [28].

2.4.5.3 Gramatika

Podla Gärdenforsa [33] sa komunikacné systémy delia do 6 tried. Systémy sú vytvorené podla mentálnych reprezentácií nutných k tomu, aby mohol vzniknút jazyk – vyvolané a oddelené. Vyvolané reprezentácie sú závislé na situácii, ktorá vie reprezentáciu vyvolat – pripomenút (napríklad bolest pri popálení), zatial co oddelené reprezentácie môžu byt použité – vyvolané lubovolne (napríklad spomienka). Z hladiska možnosti skladania prvkov sa delia na: systémy so samostatnými prvkami, kompozicné systémy a systémy s gramatikou.

	Samostatné prvky	Kompozícia	Gramatika
Vyvolané reprezentácie	Typ 1 (Zvieracie signály)	Typ 2 (Včelie tance)	Typ 3 ∅
Oddelené reprezentácie	Typ 4 ("Jednoslovný jazyk")	Typ 5 (Protojazyk)	Typ 6 (Plný jazyk)

Obrázok 11 – typy jazykov
Obrázok je z [33]

Naše primitívne a holistické jazyky patria do skupiny typu 1 a 4. Kompozicné jazyk do skupiny typu 2 a 5.

Nasimulovat jazyk typu 6 je jedným z cielov evolucnej lingvistiky. Jazyk typu 6 bude asi vyžadovat silné mentálne schopnosti agenta a taktiež velkú zložitost prostredia. Inými slovami, jednoduchost doteraz popísaných experimentov neumožnuje ani emergenciu takéhoto jazyka, lebo prezentované agenty nemajú potrebné kognitívne vlastnosti na vytvorenie jazyka s gramatikou.

2.4.6 Reprezentácia významov

Pri jazykových hrách sa agenty "rozprávajú" o nejakom význame, ktorý existuje v ich prostredí, tým, že ho pomenuvávajú. Experimenty môžeme rozdelit aj podla toho, ako si agenty tento význam vnútorne reprezentujú (teda akým spôsobom je zapísaná relácia slovo – význam). Uvedené spôsoby reprezentácií nie sú jediné možné [27].

2.4.6.1 Objekty

Pod objektom v prostredí môžeme rozumiet samotné agenty, prípadne iné prvky, ktoré sme schopní zakomponovat do reprezentácie prostredia. Takto sa samotné agenty môžu stat predmetom komunikácie dvoch agentov pri jazykovej hre. Volba objektu ako významu nie je najvhodnejšia v reálnejších simuláciách, ale pre jednoduché experimenty vyhovuje. Tento spôsob reprezentácie sa používa vo väcšine experimentov s jednoduchými jazykovými hrami.

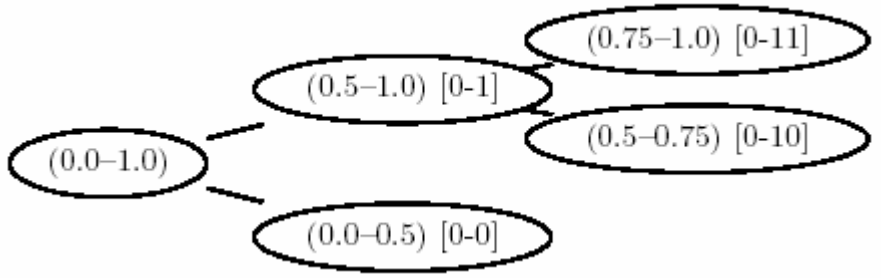
Pri použití asociacnej tabulky ako reprezentácie slovníka nám ako význam stací použit jednoznacné oznacenie objektu, napríklad meno agenta. Dalšou výhodou je fakt, že všetky agenty majú reprezentáciu v "pamäti" rovnakú, kedže používajú ako významy rovnaké oznacenia pre objekty.

2.4.6.2 Diskriminacné stromy

Diskriminacné stromy [11,12,10,22,27] sa používajú na reprezentáciu významov, ktoré disponujú crtami. Takýto spôsob reprezentácie je bližší realite a je aplikovatelný aj v aplikáciách s reálnymi robotmi so senzorickými vstupmi [9,11,12,14,32]. Agent má niekolko vstupov – kanálov z prostredia ohodnotených ako reálne císla, najcastejšie od 0

do 1 (ale môže to byt aj iný rozsah) a každý význam (objekt v prostredí) je vnímatelný týmito kanálmi (teda vnímanie objektu je realizované precítaním hodnôt na vstupoch).

Diskriminacné stromy slúžia na mapovanie reálnych hodnôt prostredia na diskrétne hodnoty. Ku každému kanálu je vytvorený jeden diskriminacný strom. Ked agent potrebuje rozlíšit objekt spomedzi niekolkých objektov (nazývame ich kontext), snaží sa ho odlíšit pomocou vytvorených diskriminacných stromov. Ak objekt nevie rozlíšit, môže napríklad rozdelit niektorú vetvu diskriminacného stromu na dve casti. Úprava diskriminacného stromu ešte neznamená úspešné rozlíšenie objektu od ostatných. Ci bola zmena užitocná, sa zistí až pri dalšom rozlišovaní objektu. Takto vzniká diskriminacný strom pre kanál a jednotlivé rozdelovanie prináša väcšiu presnost v rozlišovaní objektov (vid Obrázok 12 – diskriminacný strom).



Obrázok 12 – diskriminacný strom
Diskriminacný strom jedného kanálu, ktorý bol rozdelený 2-krát [22].

Postup na rozlišovanie objektov, ktorý sme jednoducho popísali, je známy ako *rozlišovacia hra* (discrimination game) [10,11,27]. Diskriminacné hry hrajú agenty väcšinou pred samotnými jazykovými hrami a slúžia na konceptualizáciu prostredia – agenty sa ucia rozlišovat objekty existujúce v prostredí.

Z postupu vytvárania diskriminacných stromov nie je zarucené, že si agenty reprezentujú (rozlišujú) objekty rovnako. Agenty mávajú väcšinou diskriminacné stromy odlišné.

Pri samotných jazykových hrách sa na reprezentáciu významu používajú takzvané crty objektov, ktoré sa získavajú zo vzniknutých diskriminacných stromov. Crty objektov sú vrcholy diskriminacného stromu, ktoré jednoznacne odlišujú objekt od kontextu. Agent rozlíši objekt, ktorý je témou pre jazykovú hru, od kontextu pomocou crt, ktorým priradí slovo. Posluchác si podla pocutého slova nájde crty s rovnakým pomenovaním a podla

nich sa snaží vybrat objekt z aktuálneho kontextu. Ak sa objekty zhodujú, hra bola úspešná.

2.4.6.3 Prototypy

V [30] sú na reprezentáciu významu použité prototypy. Agenty si pri tomto experimente konceptualizujú svet, ktorý pozostáva zo situácií (n – rozmerný vektor), pri ktorých môže agent zvolit nejakú akciu (reálne číslo). Prototypy spocívajú v diskriminácii priestoru pre jednotlivé situácie a akcie. V aktuálnej situácii volí agent akciu podla toho, aký prototyp k situácii nájde. Konceptualizácia sveta pre agenta znamená vytvorenie takých prototypov, aby v prostredí pri rôznych situáciach volil co najoptimálnejšie akcie. Ku konceptualizácii prispieva vonkajšia spätná väzba prostredia, ktorá ohodnocuje zvolené akcie agenta.

Napríklad agent môže zacínat s niekolkými náhodnými prototypmi. V [30] boli na zaciatku prototypy vytvorené genetickým procesom.

V samotnej jazykovej hre agent pomenováva akcie, ktoré je schopný zvolit. Akciu, ktorú má rozprávac pomenovat v danej situácii, zistí podla vytvorených prototypov. Posluchác najskôr vyberie všetky svoje akcie zodpovedajúce pocutému slovu. Potom urcí, akú akciu by zvolil on podla aktuálnej situácie. Ak sa táto akcia nachádza medzi vybratými akciami, hra bola úspešná.

Další príklad použitia prototypov popisuje [27].

2.4.7 Reprezentácia slovníka v agentoch

Existujú aj iné spôsoby reprezentácií, ako sú uvedené. Napríklad reprezentácia jazyka gramatikou [24].

2.4.7.1 Asociacná tabulka

Slovník môžeme reprezentovat jednoduchou *asociacnou tabulkou,* ktorá uchováva asociácie medzi slovom a významom. Ku každému významu, ktorý agent pozná, existuje zoznam slov, ktoré pomenovávajú daný význam. V tabulke sa môžu tiež uchovávat dalšie informácie, napríklad pocet použití páru slovo – význam, pocet úspešných použití a pod.

Príklad slovníka:

obj1	PUA, 20, 10;	LOPA, 19, 2
obj2	HATA, 1,1;	
obj3	LET, 14, 3;	TOTE, 10, 4

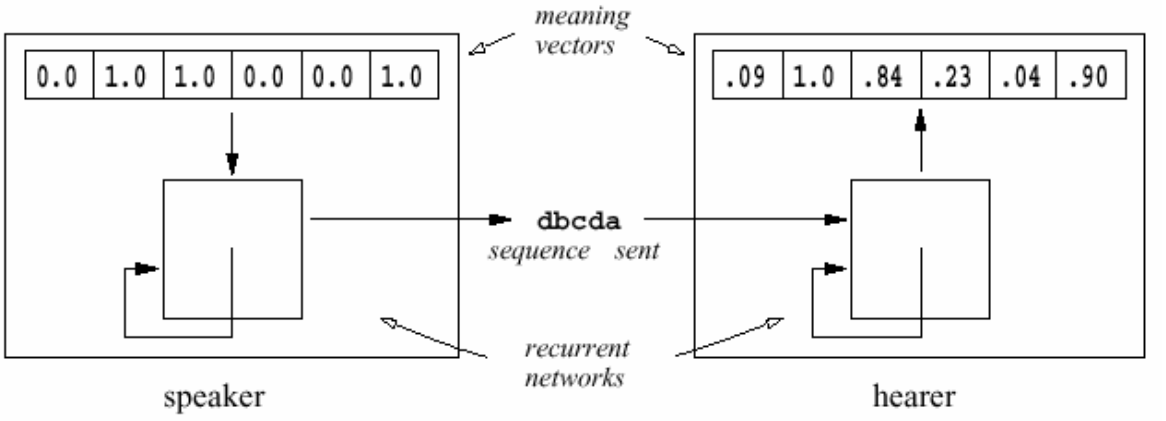
Tabulka 1 – jednoduchý slovník

Z tabulky 1 napríklad zistíme, že pre význam obj1 existujú dve slová: PUA a LOPA, pricom slovo PUA bolo v jazykovej hre použité 20-krát, z toho 10-krát úspešne.

2.4.7.2 Neurónová siet

Aby sme mohli prejst k zložitejším jazykom, akými sú napríklad štrukturované jazyky, je potrebná aj zmena reprezentácie slovníka. Asociacná tabulka pre zložitejšie jazyky nepostacuje. Môžeme však k tejto tabulke priradit iné pomocné mechanizmy, v ktorých by sme si pamätali štruktúru jazyka. Jednou z dalších možných reprezentácií sú neurónové siete [25,28,40,41].

V [28] boli na rozpoznávanie aj generovanie slov použité rekurentné neurónové siete.



Obrázok 13 – model jazykovej hry pomocou neurónových sietí
Obrázok je uvedený v [28]

Na [Obrázok 13 – model jazykovej hry pomocou neurónových sietí] je znázornený model jazykovej hry, ktorý bol použitý v [28]. Rozprávac dostal ako vstup význam. Významy mali v tomto experimente preddefinovanú štruktúru. Agent zvolený význam pomocou rekurentnej neurónovej siete zakódoval do retazca, teda pomenoval ho. Další agent – posluchác dostal retazec od rozprávaca a ten dal ho na vstup svojej rekurentnej neurónovej siete. V posluchácovi prebehne ucenie jeho neurónovej siete, ktoré má za úlohu priblížit rozpoznávanie retazca, ktorý pomenúva význam k samotnému významu.

V jazyku, ktorý si vytvoria rekurentné neurónové siete, zostane zachovaná štruktúra významov. [Obrázok 14 – vytvorené slovníky modelu s neurónovými sietami] vlavo predstavuje vytvárané slová neurónovou sietou na zaciatku simulácie a vpravo po prebehnutí simulácie. V [28] sa po analýze jazyka, ktorý vznikol pomocou neurónových sietí, ukázalo, že v jazyku ostala zachovaná štruktúra, s akou boli vytvorené významy pre komunikáciu agentov.

meaning	*sequence*
(yumi scared)	cccccccccccccccccccc
(mip hungry)	dbddddbdddbbdbddddbd
(yall sad)	ccbaccccacaadbcabcac
(they scared)	dbddccdccccccccccccc
(mip silly)	aaaaaaaaaaaaaaaaaaaa
(yall angry)	cccccccccccccccccccc
(yup sad)	ddddddddddddddddddddd
(me angry)	aaaaaaaaaaaaaaaaaaaa
(yup silly)	adddddddcddddddddadcd
(we hungry)	ccddccccccdddccccdcc
(they sad)	cccccccccccccccccccc

meaning	*sequence*
(yumi scared)	cacd
(mip hungry)	dbd
(yall sad)	acb
(they scared)	caad
(mip silly)	ada
(yall angry)	bcdbbbbbb
(yup sad)	abac
(me angry)	bdd
(yup silly)	adba
(we hungry)	ddc
(they sad)	abab

Obrázok 14 – vytvorené slovníky modelu s neurónovými sietami
Obrázok je z [28]

Jednoduchú analýzu jazyka vytvoreného neurónovými sietami zobrazuje [Obrázok 15 – analýza jazyka neurónových sietí]. Z analýzy vidno, že neurónové siete používali pre niektoré štruktúry významu rovnaké pomenovania (sufixy).

	one	*they*	*you*	*yall*	*yup*	*me*	*we*	*mip*	*yumi*	*all*
tired	--a	--ab	--c	--cb	--ba	--	--d	--db	--cd	--b
scared	--a	--ab	--c	--cb	--ba	--	--d	--db	--cd	--b
sick	--a	--ab	--c	--cb	--ba	--	--d	--db	--cd	--b
happy	--a	--ab	-c-	-c-b	--ac	--	--dc	--b	--c	--bc
sad	-ba	-bab	-c	-cb	-bac	-	-bdc	-bb	-bc	-bbc
excited	-ba	-bab	-ca	-cab	-bca	-	-cdc	-b	-cb	-bc
angry	-b	-bb	-c	-cb	-bc	-	-ddc	-db	-dc	-dbc
silly	(aa)	(aaab)	(aca)	(acab)	--ba	--d	--dc	--ad	--c	--bc
thirsty	-b-a	-b-b	-c-	-cb-	-bc-	-d-	-d-c	-b-d	-c-d	-b-cd
hungry	-bb	-bbd	-c	-cb	-bc	-d	-dc	-bd	-cd	-bcd

Obrázok 15 – analýza jazyka neurónových sietí
Významy boli skladané zo slov z prvého riadku a prvého stlpca. Pre každý význam v tabulke je analýza jeho pomenovania. [28]

2.4.8 Spolahlivost komunikácie agentov

Pri experimentoch sa väcšinou neuvažuje strata alebo zašumenie komunikácie agentov pri jazykovej hre. Dôvodom býva jednoduchost experimentu.

Zašumenie komunikácie agentov je jednou zo zaujímavých vlastností systému a približuje experiment k reálnym podmienkam. Pod zašumením komunikácie sa najcastejšie rozumie zmena prenášanej informácie medzi agentmi, ked hrajú jazykovú hru.

Mierny šum môže mat pozitívny prínos do dynamických systémov [13, 15]. Je zdrojom variácií tam, kde chýba genetická mutácia. Pri jazykových hrách sa ukázalo, že slovník emerguje aj v zašumenom prostredí, pri zašumení, ktoré predstavovalo až 10% komunikácie. Vzhladom na tieto výsledky nie je až také nutné pri každom experimente používat v prostredí šum. Výsledky experimentov nebývajú velmi rozdielne vzhladom na kvalitu prenosu (samozrejme iba do urcitej miery zašumenia).

2.5 Miery

Dôležitou castou experimentov sú výsledky a ich korektná interpretácia. Nastavovaním parametrov sa dajú získat rôzne výsledky pri rovnakých simuláciách.

2.5.1 Úspešnost hier

Jednou zo základných mier je *percentuálna úspešnost* jazykových hier. Kedže jazykové hry prebiehajú v kolách a v jednom kole prebehne jedna jazyková hra (nemusí to byt ale pravidlo), treba sa pozerat na túto mieru po viacerých hrách, napr. 20. Tak sa dá získat percentuálna úspešnost jazykových hier. Od velkosti tohto okna bude závisiet, nakolko presne budeme pozorovat výkyvy úspešnosti hier.

Vysokú úspešnost hier môžeme špecifikovat ako bod, ked má populácia agentov vytvorený spolocný slovník. Ale nemusí to byt pravda [7,31]. Treba si uvedomit, že agent môže mat k jednému významu viac slov a fakt, že s iným agentom úspešne komunikuje, ešte neznamená že ich preferované slová sú rovnaké. To znamená, že i napriek vysokej miere úspešnosti hier, sa medzi agentmi nemusel vytvorit rovnaký slovník.

2.5.2 Koherencia a lokálna koherencia

Koherencia [31] je mierou, podla ktorej môžeme povedat, ci existuje medzi agentmi rovnaký jazyk – jednotný slovník. Koherencia je tým väcšia, cím viac agentov používa tie isté slová pre významy. Preferované slová významov sa stávajú globálne v populácii agentov.

Koherenciu pocítame tak, že pre každý význam vyberieme jeho najpreferovanejšie slovo spomedzi všetkých agentov a vyjadríme percentuálne zastúpenie tohto slova pre daný význam spomedzi ostatných slov. Takto dostaneme koherenciu pre jeden význam. Z priemeru všetkých koherencií pre jednotlivé významy dostaneme celkovú koherenciu.

Dôležitá miera podobná koherencii je *lokálna koherencia*. Pri priestorových experimentoch a lokálnych jazykových hrách, sa vytvárajú slovníky v malých komunitách, podla rozloženia agentov v priestore. V takomto prípade je globálna koherencia velmi nízka, ale lokálne koherencie pre jednotlivé skupiny sú vysoké. Taktiež úspešnost hier môže byt vysoká. Agenty sa prispôsobia s viacjazycnými slovníkmi (bilingvalizmus), ktoré obsahujú preferované slová v rámci komunity a slová, ktoré sa používajú pri dorozumievaní sa s agentom z inej komunity.

2.5.3 Zhoda slovníkov

V [7] bola použitá miera na sledovanie zhody slovníkov medzi skupinami agentov. V prípade, ked sa vytvárajú skupinky s vlastnými jazykmi, táto miera bola použitá na zistenie podobnosti slovníkov dvoch skupín. Podobnost bola urcovaná medzi dvoma skupinami a zistovala, kolko preferovaných slov prvej skupiny sa vyskytuje u druhej skupiny a naopak.

2.5.4 Špecificita

Koherencia by mohla nastat aj v prípade, ked sa pre každý význam používa jedno slovo. Na sledovanie takéhoto správania sa používa miera *špecifity* [26] slovníka. Táto miera urcuje rozmanitost používaných slov pre jednotlivé významy. Ked pre každý význam existuje iné slovo, špecificita je 1 (alebo 100%). Špecificita sa dá vyjadrit ako jednoduchý pomer poctu preferovaných slov a poctu významov. Pre každý agent môžeme spocítat

jeho špecificitu slovníka. Priemer zo všetkých mier špecificity slovníkov je globálna špecificita.

2.5.5 Neistota klasifikovania

Neistota klasifikovania [26] je velmi podobná koherencii. Sleduje, s akou pravdepodobnostou bude agent v jazykovej hre úspešný pre daný význam. Miera vyjadruje, kolko iných agentov používa rovnaké slová. Ked je koherencia 1, neistota klasifikovania je 0; sú to doplnujúce sa pravdepodobnosti. Táto miera má však zmysel skôr lokálny než globálny.

2.6 Parametre

Popri samotných výsledkoch experimentu je dôležité vediet, za akých nastavení daný experiment prebiehal. Pri simuláciách s rôznymi nastaveniami parametrov sa získavajú poznatky o závislosti výsledkov na parametroch a tiež o závislosti nastavení iných parametrov, aby bol experiment úspešný. Táto kapitola popisuje iba niektoré základné parametre z definície 1 a z jednoduchých experimentov.

2.6.1 Obmedzenie slov slovníka

Tento parameter urcuje, kolko slov si agent môže zapamätat. Pri niektorých simuláciách je potrebný aj parameter pre minimálny pocet slov v slovníku. Kapacita slovníka ovplyvnuje výsledný jazyk agentov. Ak je slov velmi málo (menej než významov), v slovníku sa vytvoria homonymá. Pri velkom pocte možných slov narastá cas konvergencie globálneho slovníka.

2.6.2 Pocet krokov simulácie

Pocet krokov simulácie je dôležitý pre jej úspešnost. Tento parameter závisí napríklad na pocte agentov v prostredí. Kedže sa v každom kroku udeje jedna jazyková hra, s narastajúcim poctom agentov je nutné robit vela krokov simulácií, aby sa dostavili požadované výsledky.

2.6.3 Vytvorenie a absorpcia nového slova

Ak agent pri jazykovej hre nemá, resp. nepozná použité slovo pre význam, môže si ho vytvorit podla pravdepodobnosti vytvárania slov, resp. prijat ho za svoje podla pravdepodobnosti absorpcie slov. Pokial sú oba parametre nízke, potrvá dlhšie, kým si agenty vyrobia slovníky. Ak je nízka iba pravdepodobnost pre vytváranie nových slov, budú slovníky agentov ovela viac podobné, lebo agenty budú mat tendenciu slová radšej prijat, než si vytvorit nové. Ak je nízka pravdepodobnost prijatia slov, spolocný slovník sa bude vytvárat tažko a pomaly, lebo agenty nebudú mat tendenciu prijat cudzie slová.

2.6.4 Vstup nového agenta

Ak pocas simulácie vstupujú agenty do prostredia a zaclenujú sa do jazykovej hry, treba pocítat so spomalením vytvárania globálneho jazyka. Každý nový agent má prázdny slovník, preto je komunikácia s ním spociatku neúspešná a taktiež to negatívne prispieva ku globálnej koherencii.

Pravdepodobnost vstupu agenta urcuje možný prírastok v každom kroku simulácie. Ak agenty pribúdajú rýchlo, možnost na vytvorenie spolocného slovníka klesá. Naopak, pri pomalom pribúdaní agentov sa nové agenty stihnú prispôsobit existujúcej komunite a tak prispiet do formovania globálneho jazyka.

2.6.5 Odchod agenta

Pri odchode agenta je situácia trochu iná ako pri príchode, ale nie celkom opacná. Odchod agenta môže mat pozitívny vplyv na vytvorenie spolocného slovníka, ked patril medzi agenty, ktoré nepoužívali iba preferované slová – slová podla globálneho slovníka. V prípade, že odíde agent, ktorý prispieval na formovanie globálneho jazyka v plnej miere a jeho slovník bol s globálnym rovnaký (co sa preferovaných slov týka), spomalí to formovanie globálneho slovníka. Menší pocet agentov však na druhej strane urýchli proces formovania tohto slovníka.

3 Prostredie pre jazykové hry

Táto kapitola opisujeme nami vytvorený program ELGE – jeho komponenty, štruktúru, beh. Vysvetlujeme, ako fungujú hlavné casti programu, ako sa v nom dá realizovat experiment a tiež ako je možné program rozšírit pre dalšie experimenty. Kapitola netvorí referencnú prírucku programu, predstavuje skôr rýchleho sprievodcu v oboznamovaní sa s programom a vo vytváraní experimentov. Podrobná prírucka je[1] k dispozícii priamo s programom.

3.1 Motivácia

Pri experimentoch s agentmi, ktoré hrajú jazykové hry, sa dá postupovat rýchlo dopredu a je potrebné takémuto postupu prispôsobit aj prostredie. Dobré prostredie by malo poskytovat velkú variabilnost a nastavitelnost experimentu na užívatelskej úrovni, zároven by malo byt dostatocne otvorené, by sa dalo jednoducho rozširovat aj programátorskými prostriedkami. Existujúce produkty, ktoré boli priamo vytvorené na experimenty s jazykovými hrami, nie sú verejne dostupné. Dostupné existujúce prostredia zas nevyhovujú celkom našim potrebám. Preto jedným z cielov tejto diplomovej práce bolo navrhnút a naprogramovat prostredie s vyššie uvedenými vlastnostami, ktoré bude slúžit ako nástroj na realizáciu experimentov s jazykovými hrami.

3.2 Existujúce prostredia

Z inštitútu Sony Computer Science Laboratory Paris, priamo od priekopníkov experimentov s jazykovými hrami pochádza asi jedno z prvých prostredí nazývané BABEL. Hoci sa nám podarilo nadviazat internetovú komunikáciu priamo s autorom prostredia Angusom McIntyre [16], nedostali sme uvedené prostredie k dispozícii. Toto prostredie je napísané v programovacom jazyku LISP.

Ako alternatíva k BABELu vzniklo dalšie prostredie LEMMinS, ktoré bolo implementované v programovacom jazyku Java. LEMMinS pochádza z laboratória vo

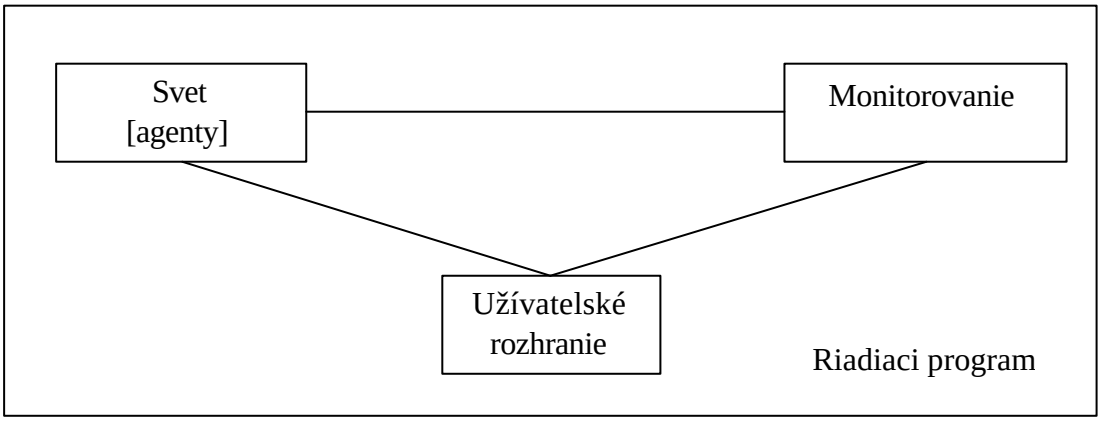
[1] referencná prírucka k programu je pri zverejnovaní tohto text u ešte v štádiu vývoja

Francúzsku, Laboratoire Dynamique Du Langage – Université Lumière Lyon 2. Žial, ani k tomuto prostrediu sme sa nedostali.

Iné simulacné prostredia ako RePast [18], Swarm [19] a AScape [20] sú navzájom velmi podobné. Umožnujú robit lubovolné multi–agentové simulácie. RePast a Swarm sú open source projekty, pricom RePast a AScape sú vytvorené v Jave, Swarm v C++. Velkou nevýhodou týchto prostredí je, že žiadne z nich neumožnuje uložit simuláciu v lubovolnom simulacnom kroku a neskôr si ju z uložených dát nahrat a pokracovat v nej. Taktiež modely vytvorené v týchto prostrediach sú málo nastavitelné, napríklad pridávanie a odoberanie agentov z modelu sa nedá realizovat jednoducho a väcšinou musí byt zakomponované priamo v modeli. V našom programe sme sa snažili uvedené nedostatky odstránit.

3.3 Štruktúra prostredia

Prostredie sa skladá z troch samostatných castí, ktoré medzi sebou spolupracujú a z jednej hlavnej riadiacej casti.



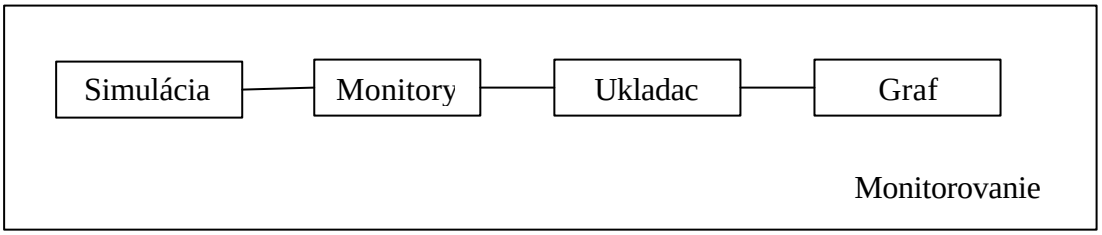
Obrázok 16 – základná štruktúra prostredia

Základné tri casti prostredia sú svet, monitorovanie a užívatelské rozhranie. Všetky zastrešuje hlavná riadiaca cast. Každá cast má svoju úlohu a je co najviac oddelená od ostatných castí.

Svet je jadrom samotnej simulácie. Zahrna v sebe agenty, ktoré sa pocas simulácie dostávajú k slovu. Jednotlivé agenty sú oživované v postupnosti, ktorá závisí od implementácie a nastavení parametrov sveta. Svet spolu s agentmi a ich obsahom tvorí

stavový priestor. Jednotlivé kroky simulácií sa dajú vyjadrit presným stavom sveta a jeho komponentov.

Monitorovanie slúži na získavanie údajov z prostredia. Systém monitorovania sa skladá z troch samostatných castí: monitorov, grafov a ukladaca. *Ukladac* má na starosti samotné uloženie dát. Podla jeho implementácie sa dáta môžu ukladat do súborov, databáz a pod. Ukladac zodpovedá za to, ako sa majú dáta ukladat. Aké dáta sa majú ukladat, definujú *monitory*. Monitory sme navrhli ako samostatné prvky prostredia (co najviac nezávislé od sveta), cím sme docielili vyššiu univerzálnost prostredia. Monitory sú znovupoužitelné, dajú sa použit vo všetkých experimentoch, pri ktorých sa sleduje rovnaký obsah, na aký boli vytvorené. *Grafy* slúžia na prezeranie výsledkov, dáta cerpajú z ukladaca, nie z monitorov.



Obrázok 17 – tok dát pri monitorovaní

Užívatelské rozhranie síce využíva všetky casti programu, ale zvyšok systému je od neho nezávislý. Samotný experiment by po nastavení mohol prebehnút aj bez neho. Užívatelské rozhranie slúži na nastavenie parametrov experimentu, jeho komfortné spustenie a sledovanie.

3.4 Základy prostredia

Riadiaci program zaclenuje v sebe jednotlivé experimenty ako projekty. Naraz môžeme vytvárat a spúštat viac projektov. Každý projekt má vlastné nastavenia a bežiace simulácie sa navzájom neovplyvnujú. Jeden projekt môže byt v riadiacom programe otvorený len raz.

Projekt zahrna všetko potrebné pre beh simulácie. Jeho súcastou je jeden svet (model), ktorý používatel zvolí pri vytváraní projektu a nedá sa už zmenit. Ak v prostredí neexistuje žiaden použitelný svet (nebol vytvorený), nedá sa vytvorit ani nový projekt.

Pred spustením simulácie je potrebné nastavit nasledovné:

- Agenty, ktoré sa stávajú súcastou sveta
- Monitory, ktoré cerpajú informácie z celého projektu a posúvajú dáta v každom kroku simulácie ukladacu
- Grafy, ktoré umožnujú priebežné sledovanie dát

Treba pripomenút, že monitory a grafy nie sú podstatné pre samotný beh simulácie, ale manipulujú s dátami, ktoré sú v každej simulácii velmi dôležité.

Ak systém neobsahuje svet, prípadne agenty, ci monitory, ktoré by vyhovovali nášmu experimentu, musíme si ich dorobit. Tomu sa budeme venovat v kapitole 3.7 Rozšírenie prostredia pre jazykové hry.

Po nastavení parametrov experimentu môže používatel prejst k samotným simuláciám. Nová simulácia vznikne vždy, ked sa spustí beh simulácie po obnovení (resete) experimentu alebo pri prvom spustení. Obnovenie modelu znamená nastavenie dát na ich pociatocné hodnoty, ktoré sa v priebehu simulácie postupne menili.

Simuláciu je možné v lubovolnom kroku prerušit a uložit. Taktiež je možné zmenit pocas simulácie všetky nastavovatelné parametre a potom v nej pokracovat už so zmenami.

Pre každú simuláciu sú dáta ukladané samostatne pod novým menom. Dáta jednotlivých simulácií sa dajú prezerat a exportovat, aby mohli byt použité v iných programoch.

Prostredie je pripravované na prírastok nových svetov, agentov a monitorov. Z tohoto dôvodu je nutné, aby prostredie vedelo vždy ponúknut používatelovi pri výbere všetky prítomné implementácie uvedených entít. To zabezpecujú zberne. *Zberne* slúžia na uchovávanie všetkých doposial implementovaných entít (svetov, agentov a monitorov). Pre každý typ entity je vyhradená samostatná zberna.

3.5 Inštalácia a pociatocná konfigurácia prostredia

Skôr, ako zacneme robit konkrétne experimenty, je nutné, aby sme mali pripravené prostredie. Inštrukcie na inštaláciu a spustenie je možné nájst aj v súbore priloženom spolu s programom.

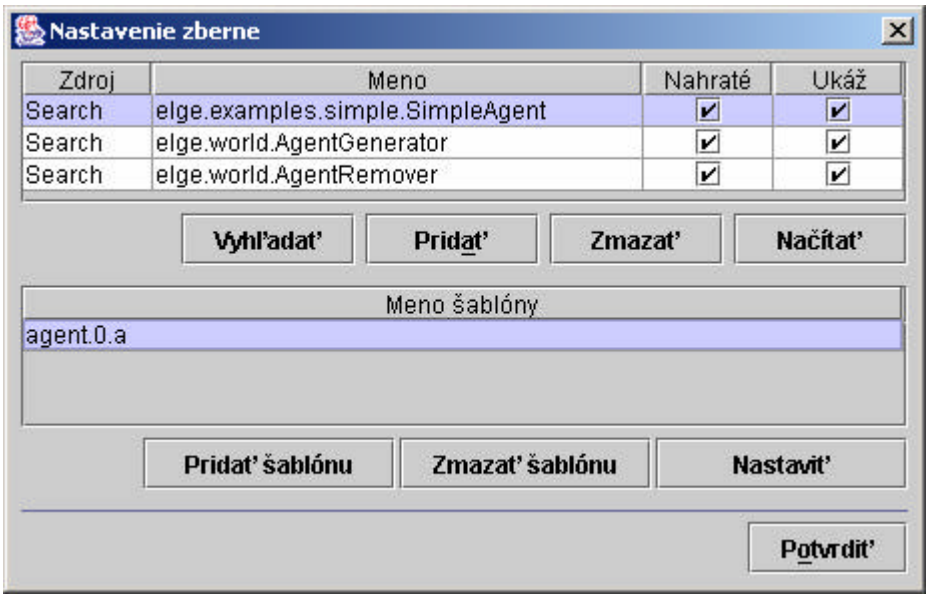
Pri inštalácii postupujeme nasledovne:

a) Adresár s programom nakopírujeme na zvolené miesto alebo program rozbalíme do zvoleného adresára.
b) Pokial nemáme v systéme k dispozícii interpreter javy[2] alebo je starší ako minimálne potrebná verzia, treba ho nainštalovat. Najnižšia podporovaná verzia je 1.4.
c) Po úspešnom nainštalovaní interpretra javy je možné spustit prostredie dávkovým súborom. Podla operacného systému treba zvolit pre OS Windows **run.bat** alebo pre UNIX systémy **run.sh** z adresára, do ktorého sme nakopírovali prostredie.
d) Ak prostredie nenabehne, treba skontrolovat body inštalácie a) a b).
e) Pri prvom spustení si prostredie vypýta adresár, kde sa budú ukladat vytvárané projekty. Je potrebné zadat existujúci adresár.

Okno úspešne spusteného programu zobrazuje Obrázok 18 – hlavné okno programu.

Obrázok 18 – hlavné okno programu

Dalším krokom pri prvom spustení programu je nastavenie zberní. Zberne treba nastavit zvlášt pre svety, agenty a monitory. V menu hlavného programu **Konfigurácia** si treba vyvolat konfigurácie pre jednotlivé zberne (vid Obrázok 19 – zberna agentov).



Obrázok 19 – zberna agentov

[2] java je registrovaná obchodná znacka firmy Sun Microsystems, Inc v Spojených štátoch a v ostatných krajinách. http://java.sun.com

Pokial zberne ešte neboli nastavené, mali by byt prázdne. Rýchlu konfiguráciu prevedieme stlacením tlacidla **Vyhladaj**. Štandardne by program po vyhladaní mal v zberni pre svet obsahovat položku s názvom SimpleWorld, v zberni pre agenty by sa mali nachádzat položky s názvami SimpleAgent, AgentGenerator a AgentRemover a v zberni pre monitory položky s názvami WorldMonitor, WorldRatioMonitor, CoherenceMonitor a GroupCoherenceMonitor.

3.6 Experimenty

Táto cast popisuje postup, ako si nastavit a spustit niekolko predpripravených experimentov. Experimenty sme zámerne zvolili tak, aby demonštrovali niektoré príznaky jazykovej hry popísané v kapitole 2.4 Príznaky experimentov.

3.6.1 Editor projektu

Obrázok 20 - editor projektu

Pre experimentovanie si musíme vytvorit nový projekt alebo nacítat do prostredia už existujúci projekt.

Vytvorenie projektu vyvoláme z hlavného okna programu cez menu **Projekt/Nový**. Program si vypýta názov projektu, ktorý zadáme, napríklad "Prvy projekt". Dalším dialógom urcíme svet, ktorý bude projekt používat. Pri výbere sveta zvolíme z dialógu položku s názvom SimpleWorld.

Po úspešnom vytvorení projektu sa zobrazí editor projektu (vid Obrázok 20 - editor projektu).

Editor projektu je hlavné okno pre experiment a jeho beh. Skladá sa z dvoch hlavných castí:

Prvú tvoria záložky, v ktorých je možné nastavovat parametre potrebné pre beh experimentu. Sú to zoznamy agentov, monitorov, grafov, dát simulácií, parametrov sveta a parametrov projektu. Každá záložka disponuje vlastnými prvkami na ovládanie.

Druhou castou editora projektu sú prvky na ovládanie a sledovanie simulácie. Položka **Krok** zobrazuje, v ktorom kroku sa simulácia momentálne nachádza a v zátvorke sa pocas behu simulácie zobrazuje, kolko krokov ešte simulácia pobeží. Simuláciu je možné odštartovat, prípadne v nej pokracovat stlacením tlacidla . Systém si vypýta pocet krokov, ktoré má simulácia vykonat. Ak zadáme hodnotu –1, simulácia bude bežat bez obmedzenia, až kým ju neukoncíme tlacidlom . (Pocas behu simulácie sa tlacidlo mení na tlacidlo .) Tlacidlo umožnuje postupovat v simulácii po jednom kroku. Tlacidlo slúži na obnovu (reset) simulácie. Všetky nastavenia, ktoré sa pocas simulácie zmenili, by sa mali dostat do pôvodného stavu. Avšak záleží od konkrétnej implementácie sveta, agenta ci monitoru, ako sa bude správat v prípade obnovy simulácie. Preto si v prípade potreby precítajte dokumentáciu k používanej implementácii sveta, agenta ci monitora.

V menu editora projektu môžeme projekt uložit, uložit pod novým menom alebo zatvorit.

3.6.2 Pridávanie agentov

Pri konfigurácii experimentu potrebujeme vediet, ako pridávat do sveta agenty. Pridávanie môžeme realizovat viacerými spôsobmi. Agenty môžeme pridávat manuálne pomocou kontrolných prvkov zoznamu v záložke **Agenty** editora projektu. Druhou možnostou je použitie generátorov. *Generátory* sú špecializované agenty, ktoré v prostredí plnia špeciálne funkcie a nepodielajú sa na jazykových hrách. V prostredí sú dva pripravené generátory. Jeden slúži na pridávanie nových agentov do sveta, druhý na ich vyhadzovanie. Pri simulácii sa generátormi pridané resp. odobraté agenty pridajú resp. odoberú vždy na zaciatku dalšieho kroku simulácie.

Najprv si popíšeme vlastnosti generátora, ktorý pridáva agenty. V zberni agentov existuje pod položkou s názvom AgentGenerator. Popis jeho parametrov:

- AgentTemplate – tu treba nastavit agent, ktorý bude generátor pridávat do sveta. Agent sa vyberá zo zberne.

- GenCycleType – urcuje, ako casto bude generátor pocas simulácie generovat nových agentov. Parameter môže nadobúdat jednu z hodnôt:
  - Tick period – generovanie prebehne v každom GenPeriod kroku
  - Every tick – generovanie prebehne v každom kroku simulácie
  - First tick – generovanie prebehne iba na zaciatku simulácie, v 1. kroku
  - Exact tick – generovanie prebehne iba raz, v kroku GenPeriod
  - No gen. – neprebehne žiadne generovanie
- GenPeriod – slúži na urcenie periódy alebo presného kroku generovania
- GenProb – pravdepodobnost vygenerovania agenta, hodnota z intervalu <0,1>
- InOneStep – oznacuje, kolko agentov sa má generovat v jednom generovacom kroku. Skutocný pocet agentov, ktoré pribudnú do sveta, závisí od pravdepodobnosti GenProb.
- MaxAgents – urcuje, kolko agentov môže daný generátor v priebehu celej simulácie vygenerovat. Pre hodnotu –1 nie je pocet agentov obmedzený.

Nasledujúce dva parametre majú význam iba pre svety, ktoré podporujejú rozmiestnenie agentov v priestore. V takom prípade si generátor od svetu vyžiada náhodnú pozíciu pre pridávaný agent.

- RndPosition – urcuje typ generovania náhodnej pozície, nadobúda jednu z hodnôt:
  - around agent – vygeneruje náhodnú pozíciu v okolí samotného generátora v maximálnej vzdialenosti RndRadius
  - rnd on map – vygeneruje pozíciu náhodne v celom priestore
  - fix position – nastaví pozíciu agenta zhodnú s pozíciou generátora. Pokial to svet umožnuje, všetky agenty budú v jednom bode.
- RndRadius – urcuje maximálnu vzdialenost agenta od generátora

Kedže generátor je tiež agent, v prostredí s ním manipulujeme rovnako ako s agentmi, t.j. musíme ho vložit do zberne agentov a patricne nastavit.

3.6.3 Nastavovanie grafov

Dalej si ukážeme, ako nastavovat grafy. Pridávanie, mazanie, editovanie a zobrazovanie grafov sa deje pomocou kontrolných prvkov v editore projektu v záložke **Grafy**. Pri

pridávaní a editovaní grafu je potrebné sa oboznámit s možnostami nastavenia, ktoré poskytuje *editor grafu* (vid Obrázok 23 – konfigurácia grafu).

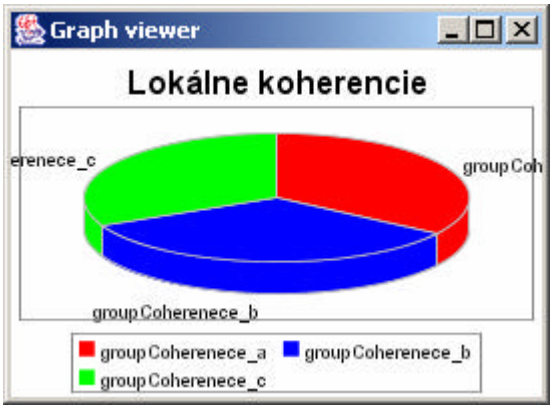
Pri nastavovaní grafu treba urcit:

- hodnoty, ktoré chceme zobrazovat
- typ grafu
- názov grafu a osí X,Y
- okno pre zobrazovanie dát

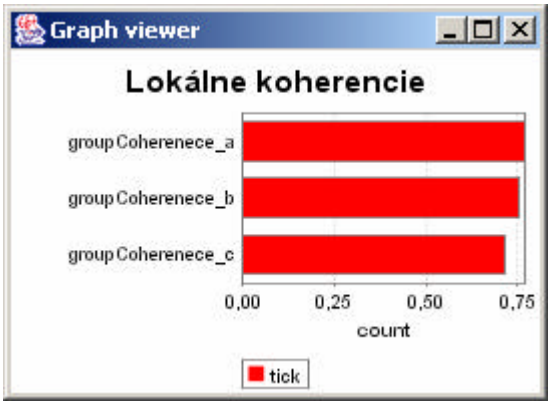
Implementovali sme štyri typy grafov.

Prvý typ je *ciarový graf*. Os Y predstavuje rozsah hodnôt zobrazovaných dát. V prípade, že graf bude zobrazovat viac dát, mali by mat približne rovnaký rozsah hodnôt. Os X grafu predstavuje pocet krokov simulácie. Jeden graf môže zobrazovat viac dát súcasne. Pre každé dáta je vykreslená jedna ciara. Ciary sú navzájom odlíšené farebne (napr. vid Obrázok 28 – koherencia a úspešnost hier v spolocenstvách).

Druhý typ grafu *(DA – dual axis)* je tiež ciarový graf. Tento graf slúži na zobrazovanie viacerých dát, ktoré nemajú rovnaký rozsah hodnôt. Graf vytvorí os Y na oboch stranách grafu, pricom na jednu nanesie rozsah hodnôt prvých dát a na druhú rozsah hodnôt ostatných dát (napr. vid Obrázok 4 – úspešnost hier a velkost populácie).



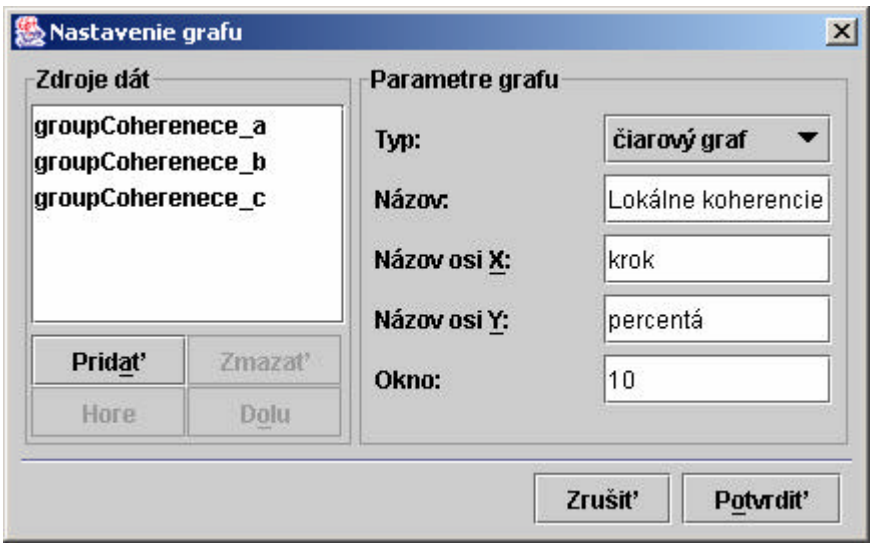
Obrázok 21 – ukážka kolácového grafu



Obrázok 22 – ukážka stlpcového grafu

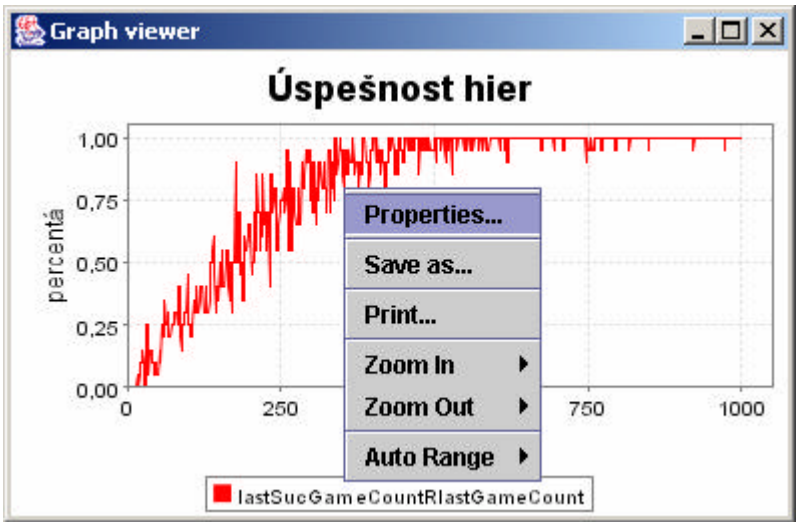
Kolácový graf a *slpcový graf* zobrazujú hodnoty dát v poslednom kroku simulácie. Oba grafy majú význam iba pri porovnávaní na sebe závislých alebo podobných dátach (vid Obrázok 21 – ukážka kolácového grafu a Obrázok 22 – ukážka stlpcového grafu).

Okno je hodnota, ktorá urcuje, po kolkých krokoch sa majú zobrazovat hodnoty dát. Napríklad, pre hodnotu 1 bude zobrazený každý krok simulácie. Hodnota 10 znamená, že bude zobrazený každý desiaty krok simulácie. Pre kolácový a stlpcový graf nemá tento parameter význam.



Obrázok 23 – konfigurácia grafu

Samotný graf vyvoláme z editora projektu v záložke **Grafy** jeho oznacením v zozname grafov stlacením tlacidla **U/S (**Ukáž/Skry). Ako názov napovedá, rovnakým tlacidlom môžeme okno grafu skryt.



Obrázok 24 – ukážka grafu a jeho menu[3]

[3] Dokumentáciu ku knižnici JFreeChart môžete nájst na http://www.jfree.org/jfreechart/

Výzor grafu môžeme upravovat z jeho vlastného menu, ktoré vyvoláme stlacením pravého tlacidla myši[4] v okne grafu (vid Obrázok 24 – ukážka grafu a jeho menu). Graf ponúka možnosti ako tlac, uloženie do súboru, zmenu farieb, nadpisov a podobne.

3.6.4 Prvý experiment

Po strucnom oboznámení sa s ovládaním programu, prejdime k prvému experimentu. Pripravený svet SimpleWorld a agent SimpleAgent splnajú definíciu 1. Pomocou tejto implementácie si prejdeme niektoré simulácie, ktorých výsledky sme mali možnost vidiet pri popisoch vlastností experimentov.

V zberni agentov si pre položku SimpleAgent vytvoríme jednu šablónu, ktorej môžeme zmenit meno podla uváženia. V editore zvolíme záložku **Agenty** a pridáme do zoznamu agentov zo zberne agentov jeden AgentGenerator. Pre pridaný generátor nastavíme parametre nasledovne:

- AgentTemplate – zo zberne zvolíme našu preddefinovanú šablónu SimpleAgent
- GenCycleType – nastavíme na First tick– agentov chceme generovat iba na zaciatku simulácie
- GenProb – nastavíme na 1 (100% pravdepodobnost, že sa agent vytvorí)
- InOneStep – zadáme 20 (tolko agentov chceme mat v simulácii)

Ostatné položky nás teraz nezaujímajú. Ak sme vyplnili všetko v poriadku, môžeme vyskúšat spustit simuláciu. Spustíme ju na 300 rokov. Ak bol generátor nastavený správne, do zoznamu agentov pribudlo dalších 20 agentov. Každý s rovnakým menom, ale rôznym ID. Môžeme si prezriet ich slovníky (v zozname agentov treba oznacit lubovolný agent, editovat ho a potom editovat jeho parameter Lexikon). Kedže už prebehli jazykové hry, niektoré slovníky by už nemali byt prázdne.

Dáme obnovit simuláciu a vymažeme vygenerovaných agentov. (V nastaveniach pre svet je parameter RemoveAgent. Ak je nastavený na remove generated, tak pri obnove sa zo zoznamu agentov vymažú všetky agenty, ktoré boli vygenerované).

[4] Rôzne OS môžu používat iné tlacidlá, prípadne iné mechanizmy na vyvolanie takzvaného popup menu.

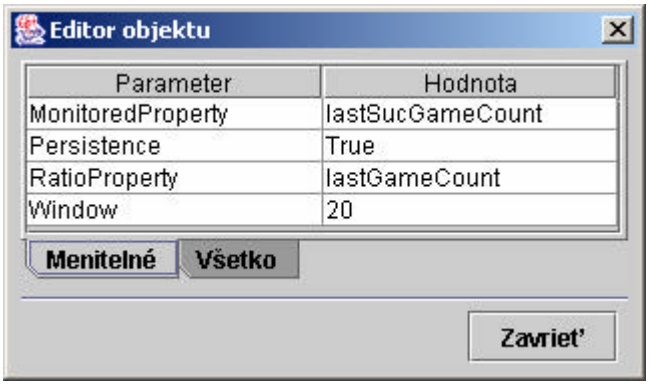
Predošlá simulácia prebehla "nasucho", lebo žiadne dáta sme neukladali a ani žiadne nezobrazovali, a tak sme nemohli sledovat jej priebeh. Teraz nastavíme monitory a grafy tak, aby sme mohli priebeh simulácie pozorovat.

Ako prvé musíme nastavit zbernu pre monitory, ak ešte nie je (vid 3.5 Inštalácia a pociatocná konfigurácia prostredia). Zberna monitorov by mala obsahovat položku s názvom WorldRatioMonitor. Tento monitor sleduje dva lubovolné parametre sveta a dáva ich do pomeru po X krokoch. To znamená že pre parametre monitoru MonitoredProperty (MP), RatioProperty (RP) a Window (X) je v krokových intervaloch po X pre interval $(K,K+X)$ hodnota vrátená monitorom vypocítaná ako:

$$\text{výsledok} = \frac{MP_K + MP_{K+1} + \ldots + MP_{K+X}}{RP_K + RP_{K+1} + \ldots + RP_{K+X}}$$

Monitor nastavíme tak, aby sme sledovali mieru úspešnosti hier (vid 2.5.1 Úspešnost hier). Kedže v každom kroku prebehne len jedna hra medzi dvoma agentmi, na graf by sa táto udalost nanášala takmer zbytocne, lebo skoky medzi 0 a 1 nie sú až také zaujímavé. Preto tento monitor dáva parametre do pomeru podla zvoleného poctu krokov. Svet poskytuje na zistenie úspešnosti a poctu hier dva parametre: lastGameCount a lastSucGameCount.

V editore projektu v záložke **Monitory** pridáme do zberne monitorov WorldRatioMonitor, ktorému nastavíme parametre podla [Obrázok 25 – nastavenie monitoru WorldRatioMonitor pre mieru úspešnost hier].



Obrázok 25 – nastavenie monitoru WorldRatioMonitor pre mieru úspešnost hier

Tu si treba dat pozor, lebo ak zadáme názvy položiek nesprávne, monitor nebude správne fungovat a dáta sa nebudú zberat. Pri názve položky treba zachovat velké a malé

písmená. Náš pridaný monitor sleduje úspešnost jazykových hier pre každých 20 krokov simulácie.

Dalším krokom je nastavenie grafu. V záložke **Grafy** editora projektu pridáme graf pomocou tlacidla **Pridaj**. Zobrazí sa nám editor grafu. Graf nazveme "Úspešnost hier". Do zoznamu **Zdroje dát** vložíme položku s názvom lastSucGameCountRlastGameCount, ktorú by nám systém mal dat k dispozícii potom, co sme pridali monitor. (Každá implementácia monitoru zodpovedá za to, ako budú pomenované zdroje dát, ktoré daný monitor bude dávat k dispozícii. V tomto prípade si monitor vytvoril názov podla oboch parametrov, ktoré monitoruje a oddelil ich písmenom 'R'.)

Po nastavení monitoru a grafu môžeme spustit druhú simuláciu. Tentoraz ale budeme mat k dispozícii aj jej priebežné výsledky – pocas behu simulácie by sme mali vidiet zmeny v dátach. Výsledný graf by mal po 10000 krokoch simulácie vyzerat ako [Obrázok 3 – úspešnost hier]. Graf na obrázku používa grafové okno o velkosti 10, zobrazované údaje nie sú nahusto. Môžeme tak nastavit aj náš graf a v editore grafu zmenit položku Okno na 10. Zmeny sa po potvrdení editovaného grafu prejavia hned. (Pokial nenastane obnovenie simulácie, nazbierané dáta zostanú grafom prístupné.)

Simuláciu môžeme opakovat pre rôzne pocty agentov, prípadne môžeme vyskúšat niektorým agentom nastavit rôzne parametre, a tak sledovat závislost úspešnosti hier na zmenených parametroch.

3.6.5 Druhý experiment – otvorenost systému

Druhý experiment je pokracovaním prvého. V predošlom experimente bola jazyková hra uzavretá. Do systému neprichádzali a ani z neho neodchádzali agenty (nerátame fakt, že sme si ich v prvom kroku simulácie dali vygenerovat).

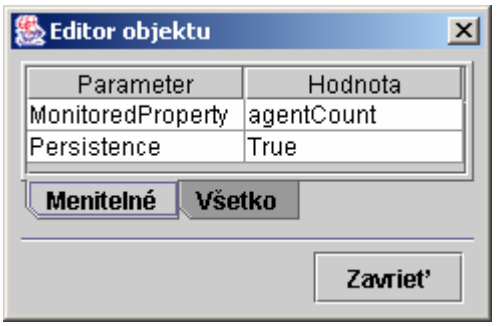
Aby sme si vyskúšali jazykovú hru v otvorenom prostredí (vid kapitolu 2.4.3.2 Otvorené systémy), stací spravit len pár zmien. Do zoznamu agentov v editore projektu pridáme další generátor agentov, ale nastavíme ho trochu inak:

- AgentTemplate – zo zberne zvolíme preddefinovanú šablónu SimpleAgent.
- GenCycleType – nastavíme na Every tick. Budeme generovat agentov v každom kroku.

- GenProb – zvolíme 0,00025. Nechceme, aby agenty pribúdali príliš rýchlo.
- InOneStep – zvolíme 1. V každom kroku sa vygeneruje maximálne jeden agent.

Aby sme mohli sledovat vývoj miery úspešnosti hier v závislosti na pribúdaní agentov, budeme musiet do systému pridat další monitor. Tentoraz nám poslúži preddefinovaný monitor s názvom WorldMonitor, ktorý by sa mal tiež nachádzat v zberni monitorov.

V záložke **Monitory** editora projektu pridáme nový monitor, ktorý má v zberni názov WorldMonitor. Tento monitor sme navrhli tak, aby zaznamenával lubovolný parameter sveta. Zoznam parametrov sveta s ich aktuálnymi hodnotami sa nachádza v záložke **Par.sveta** editora projektu. V našom príklade chceme sledovat pocet agentov, preto parameter MonitoredProperty nastavíme na agentCount (vid Obrázok 26 – nastavenie monitoru WorldMonitor).



Obrázok 26 – nastavenie monitoru WorldMonitor

Dalej treba upravit graf. Do pôvodného grafu z prvého experimentu pridáme do zoznamu **Zdroje dát** dalšiu položku s názvom agentCount. Rozsah hodnôt, ktoré chceme zobrazovat bude ale dost odlišný. Zatial, co pocet agentov môže narást na velké prirodzené číslo, úspešnost hier sa pohybuje medzi hodnotami 0 a 1 (resp. 0 až 100%). Na zobrazenie výsledkov bude preto vhodnejší typ grafu DA[5] ciarový graf.

Prírastok nových agentov by sme chceli vyskúšat až potom, co miera úspešnosti hier dosiahne okolie maxima. Preto najprv nastavíme nový generátor, aby negeneroval agenty (parametru GenCycleType priradíme hodnotu No gen.). Spustíme simuláciu a pockáme, kým sa miera úspešnosti hier neustáli v okolí 100%. Potom prestavíme generátor tak, aby zacal generovat nové agenty a necháme bežat simuláciu na dalších rádovo 40000 krokov.

[5] Dual Axis

Výsledok simulácie by mal byt porovnatelný s výsledkom na [Obrázok 4 – úspešnost hier a velkost populácie]. Graf na obrázku bol ešte trochu upravený – pre agentCount sme nastavili rozsah hodnôt manuálne s maximom na 50.

Pre dalšiu simuláciu zmeníme len parameter GenProb nového generátora – zvýšime pravdepodobnost vygenerovania nového agenta z hodnoty 0,00025 na 0,001. Experiment obnovíme a spustíme na 50000 krokov. Výsledky by sa mali priblížit k [Obrázok 5 – úspešnost hier a velkost populácie 2].

Na simuláciu dalších výsledkov z kapitoly 2.4.3.2 Otvorené systémy budeme potrebovat další generátor – AgentRemover. Tento generátor bude zodpovedný za vyhadzovanie agentov zo sveta. V editore projektu pridáme do zoznamu agentov AgentRemover, ktorý má niekolko analogických parametrov ako AgentGenerator. Rozdielne parametre sú:

- MinAgents – kolko agentov musí minimálne zostat vo svete. AgentRemover vyhodí agent, iba ak svet obsahuje viac agentov než je stanovené minimum.
- ToRemove – kolko agentov môže generátor vyhodit pocas celej simulácie. Hodnota –1 znamená neobmedzený pocet agentov.

AgentRemover nevyhadzuje systémové agenty, medzi ktoré patria aj generátory.

Nastavíme prvý generátor, ktorý vygeneruje na zaciatku simulácie 20 agentov. Druhý generátor, ktorý v predchádzajúcej simulácii generoval v každom kroku nové agenty, vypneme. A napokon, novému generátoru AgentRemover nastavíme tieto hodnoty:

- MinAgents – 6. (3 agenty – generátory + minimálne 3 agenty hrajúce jazykovú hru)
- ToRemove – necháme –1
- GenCycleType – nastavíme na Every tick.
- GenProb – nastavíme dost velkú pravdepodobnost – okolo 0,002.

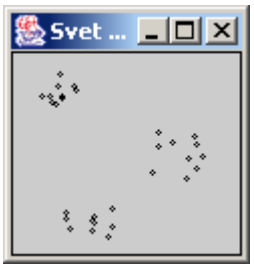
Nastavenie grafu nebudeme menit. Obnovíme a spustíme dalšiu simuláciu. Tentoraz bude stacit omnoho menej krokov, aby sa úspešnost hier ustálila na maxime 100%. Ak však chceme získat podobné výsledky ako [Obrázok 6 – úspešnost hier a velkost populácie 3], musíme nechat simuláciu bežat 25000 krokov.

3.6.6 Tretí experiment – spolocenstvá

Posledný ukážkový experiment je podobný priestorovej hre z kapitoly [2.4.4.2 Priestorové hry]. Agenty vygenerujeme tak, aby boli rozmiestnené v priestore po skupinkách. Na to ale budeme potrebovat viac šablón v zberni agentov. Vytvoríme si preto tri šablóny položky SimpleAgent a v každej šablóne nastavíme v parametri Group inú skupinu pre agenta. Skupiny pomenujeme a, b a c.

Pre každú skupinu pridáme do zoznamu agentov jeden generátor – AgentGenerator, ktorému nastavíme tieto vlastnosti:

- AgentTemplate – pre jednotlivé generátory nastavíme pripravené šablóny zo zberne agentov so skupinou a, b a c.
- GenCycleType – nastavíme na First tick, – agenty sa vygenerujú na zaciatku
- InOneStep – nastavíme na 10, v každej skupine chceme 10 agentov.
- GenProb – nastavíme na 1 (100% pravdepodobnost).
- Position – každému generátoru nastavíme inú pozíciu. V našom prípade je svet dvojrozmerný, môžeme preto použit napr. pozície [20,20], [50,80] a [80,40].
- RndPosition – nastavíme na around agent –generovanie agentov okolo generátora.
- RndRadius – nastavíme 30 – maximálna vzdialenost agentov od generátora.



Obrázok 27 – agenty v priestore po skupinkách
30 agentov v priestore v 3 skupinkách

Ak budeme simuláciu krokovat, po druhom kroku by mala reprezentácia sveta vyzerat podobne ako [Obrázok 27 – agenty v priestore po skupinkách].

V takto pripravenom svete je možné hrat priestorové hry, pokial je nato svet pripravený. SimpleWorld v každom kroku podla definície jazykovej hry vyberie dvoch hrácov a medzi nimi prebehne jazyková hra. Na ovplyvnenie výberu hrácov svet poskytuje parameter s názvom SAgentChoose (v záložke editora projektu **Par. sveta)**. Parameter SAgentChoose

urcuje, ako sa bude vyberat druhý hrác. Zatial, co prvý hrác je vybraný vždy náhodne, druhý hrác sa môže vybrat:

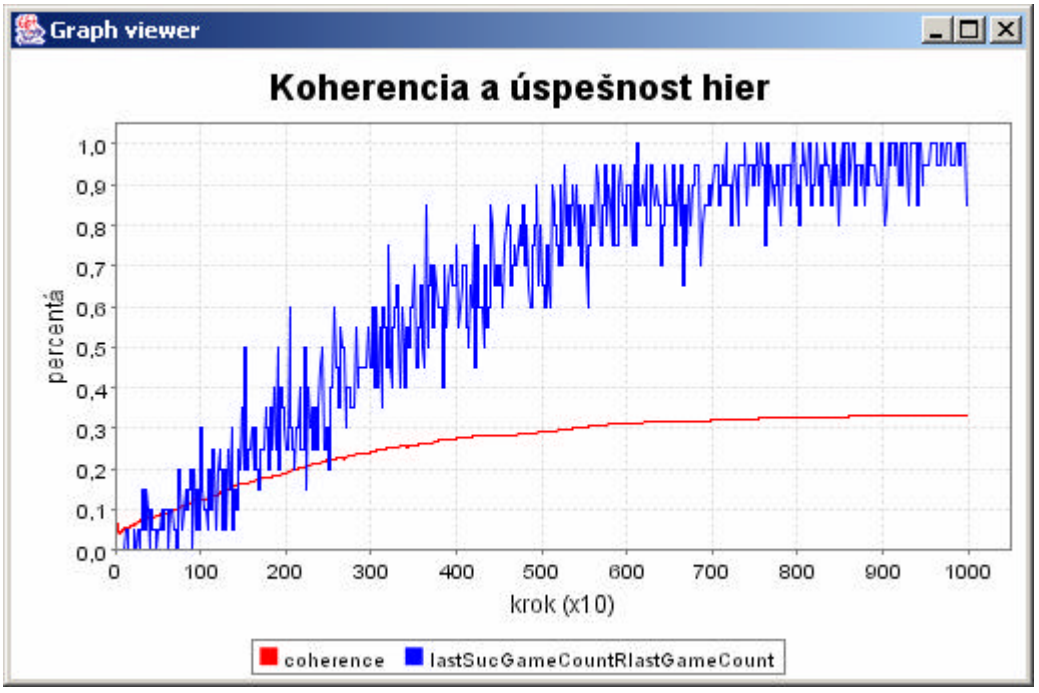
a. random – tiež náhodne

b. by group – náhodne, ale z rovnakej skupiny (s rovnakým parametrom Group)

Pre experiment si teraz nastavíme SAgentChoose na by group. Kedže každá skupinka agentov je vygenerovaná jedným generátorom, ktorý používa jednu šablónu, agenty budú komunikovat spolu len v rámci skupín, co má velmi podobný dopad na priebeh simulácie ako komunikovanie podla vzdialenosti.

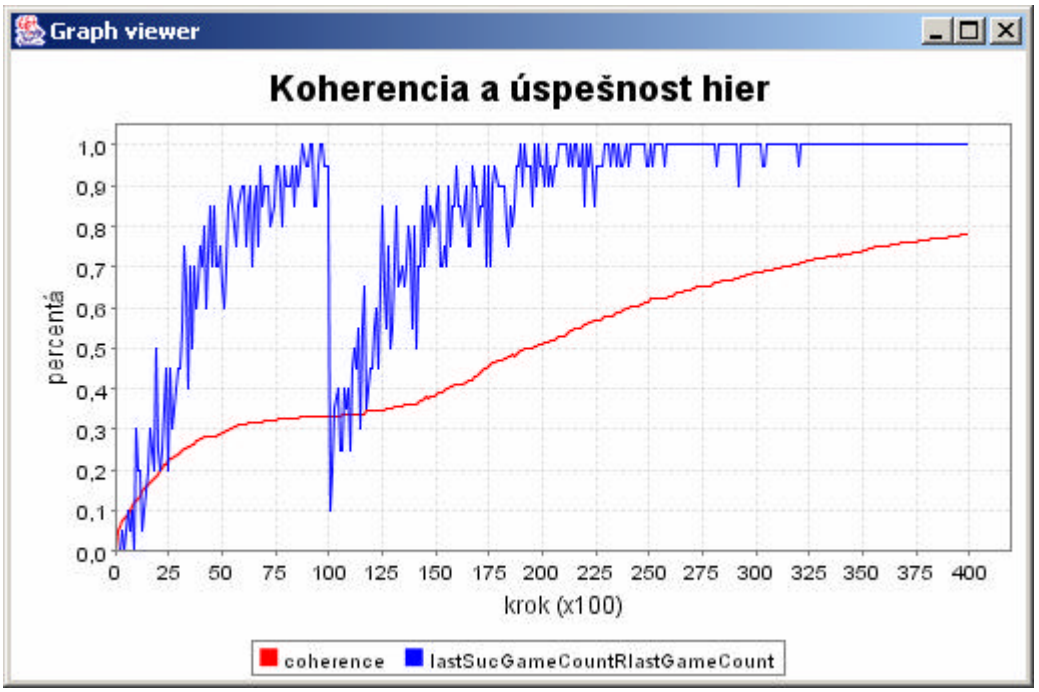
Vo *svete* by mala nastat známa situácia popísaná v kapitole [2.4.4.2 Priestorové hry]. Môžeme skontrolovat slovníky agentov z dvoch rôznych skupín. Situáciu tiež môžeme podrobnejšie sledovat pridaním monitoru koherencie. Tento monitor netreba nijak zvlášt nastavovat, stací ho pridat do zoznamu monitorov.

Obnovíme simuláciu a spustíme další beh. Na pozorovanie vývoja hry si vytvoríme graf s koherenciou a úspešnostou hier. Výsledky priebehu zobrazuje [Obrázok 28 – koherencia a úspešnost hier v spolocenstvách]. Ako na obrázku vidno, úspešnost hier je vysoká, ale globálna koherencia nie je, lebo agenty komunikujú len v rámci spolocenstva, a tak jazyky každého zo spolocenstiev sú odlišné.



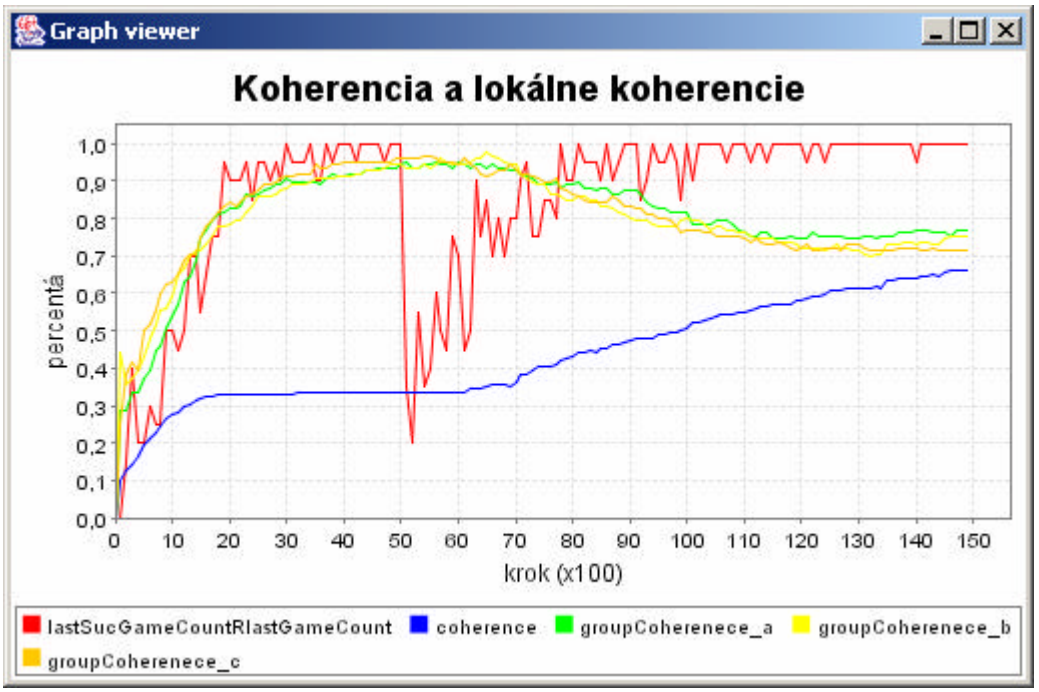
Obrázok 28 – koherencia a úspešnost hier v spolocenstvách
Spodnejšia ciara v grafe je koherencia.

Prestavíme parameter SAgentChoose na random. Odteraz budú medzi sebou komunikovat všetky agenty bez rozdielu. Simuláciu neobnovujeme, ale pokracujeme v nej dalších *30000* krokov. Na výsledkoch (vid Obrázok 29 – koherencia a úspešnost hier v spolocenstvách 2) je zjavne vidiet zmenu, ktorá nastala v komunikácii medzi agentmi. Koherencia zacala stúpat, co spôsobili globálne interakcie.



Obrázok 29 – koherencia a úspešnost hier v spolocenstvách 2
Spodnejšia ciara v grafe je koherencia. V kroku 10000 zacali prebiehat globálne interakcie medzi agentmi.

Na sledovanie priebehu experimentu pridáme monitory koherencie pre každé spolocenstvo agentov. Monitor nájdeme pod názvom GroupCoherenceMonitor v zberni monitorov. Výsledky simulácie sú znázornené na [Obrázok 30 – koherencia a lokálne koherencie spolocenstiev].



Obrázok 30 – koherencia a lokálne koherencie spolocenstiev
Najspodnejšia ciara je globálna koherencia. Ciara s najväčším zlomom v kroku 5000 je úspešnost hier. Ostatné predstavujú lokálne koherencie spolocenstiev. V bode 5000 krokoch zacali globálne interakcie medzi agentmi.

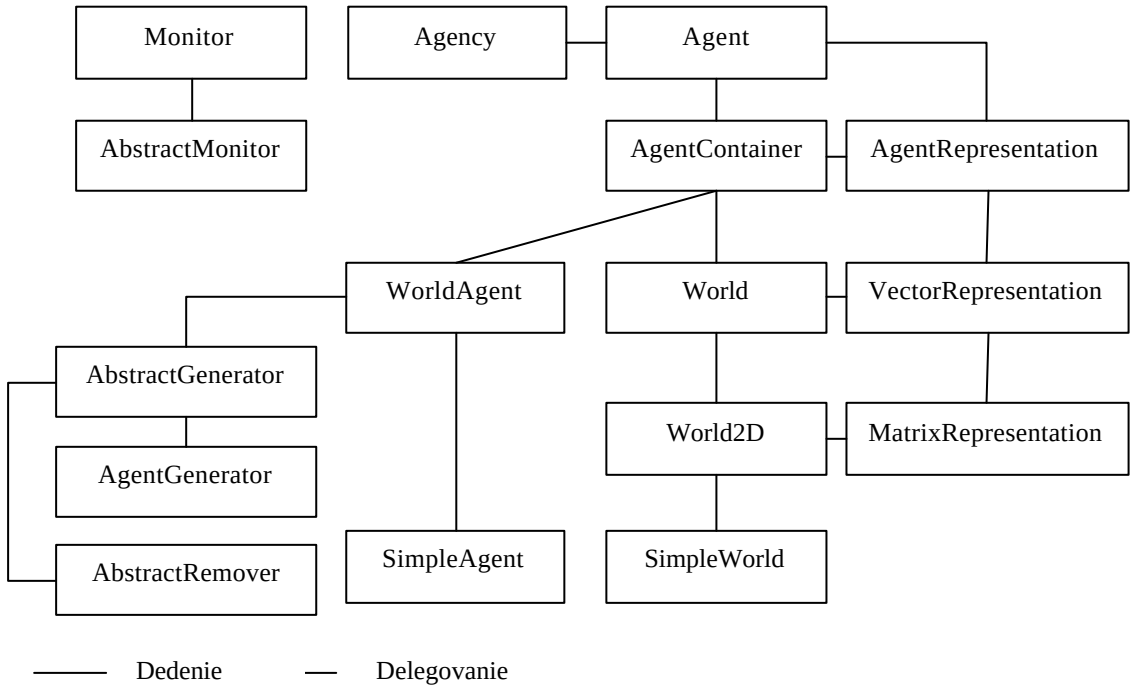
3.7 Rozšírenie prostredia pre jazykové hry

Táto kapitola sa zaoberá rozšírením programu pre dalšie experimenty. Nepopisujeme možnosti rozširovania užívatelského rozhrania alebo systémových castí programu, ale rozširovanie svetov, agentov a monitorov, ktoré musí používatel zmenit, ak chce realizovat iné typy experimentov.

3.7.1 Potrebné znalosti

Na rozširovanie prostredia je nutné ovládat aspon základy objektového programovania. Taktiež je potrebné oboznámit sa s programovacím jazykom Java [42] (jeho technológiou JavaBeans) a jeho prostredím na tvorbu aplikácií. Rozširovatel by mal taktiež vediet nieco o problematike jazykových hier a podrobnejšie sa oboznámit s našim prostredím pre jazykové hry.

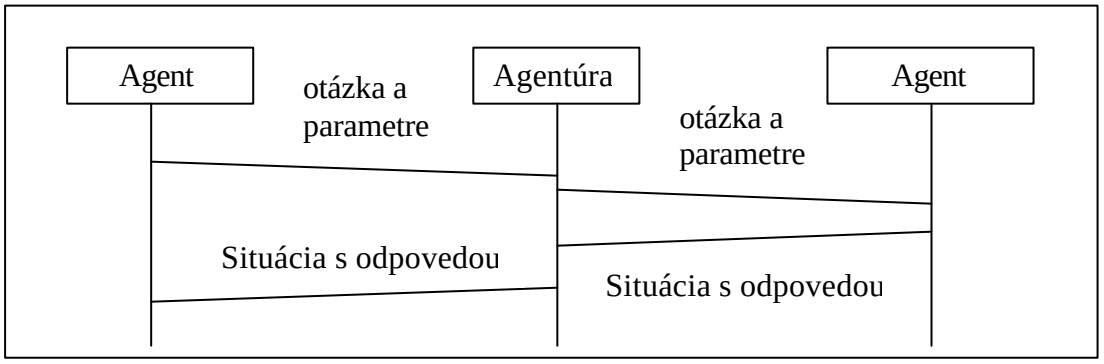
Skôr, ako prejdeme ku konkrétnym postupom a zásadám rozširovania prostredia, oboznámime sa s objektovým modelom základných castí programu (vid Obrázok 31 – objektový model rozšíritelných castí programu)



Obrázok 31 – objektový model rozšíritelných castí programu

Základnou triedou je trieda Agent. Metódy triedy Agent sme navrhli vychádzajúc z predpokladu, že inštancie triedy Agent majú byt použité v multi-agentovom prostredí.. Trieda definuje metódu life, pomocou ktorej sú inštancie triedy v multi-agentovom prostredí obsluhované. Metóda life volá tri dalšie metódy, ktoré predstavujú životný cyklus agenta. Sú to metódy sense, memorize a chooseAction volané v tomto poradí.

Trieda AgentContainer[6] rozširuje triedu Agent. Inštancia triedy AgentContainer predstavuje multi-agentové prostredie, lebo môže obsahovat v sebe agenty (objekty triedy Agent) a v cykle ich obsluhovat. AgentContainer obsluhuje agenty v metóde agentsLife, ktorú volá zo svojej metódy life. Takže krok simulácie v inštancii AgentContainer predstavuje jedno volanie metódy life. Postupnost obsluhy jednotlivých agentov závisí na konkrétnej implementácii. Štandardne sú agenty obsluhované v postupnosti, v akej boli do AgentContainer-a pridané. Na udržovanie zoznamu agentov používa AgentContainer inštanciu triedy AgentRepresentation[7].



Obrázok 32 – komunikácia agentov

Aby sme zachovali ideu multi–agentového prostredia, agenty by medzi sebou nemali komunikovat priamo ale prostredníctvom samotného prostredia. Preto má trieda Agent podporu pre komunikáciu cez agentúry – inštancie triedy Agency (vid Obrázok 32 – komunikácia agentov). Komunikáciu cez agentúry používajú agenty aj na komunikáciu so svetom. AgentContainer zarucuje, že všetky agenty obsiahnuté v nom majú rovnakú agentúru, a tak sa môžu cez nu dorozumievat.

[6] Triedy, ktoré zahrnujú správu iných objektov sa casto nazývajú kontajnery.
[7] Kedže v Jave je jednoduché dedenie, teda objekt môže mat iba jedného rodica, existuje v nej rozhranie (interface). Rozhranie je len definíciou, nieco ako abstraktná trieda, neobsahuje žiadnu implementáciu. Triedy v javy môžu implementovat lubovolný pocet rozhraní. AgentRepresentation je v skutocnosti rozhranie.

Pri komunikácii agentov cez agentúru sa na prenos informácií používa objekt triedy Situation. Táto trieda je jednoduchá asociacná tabulka a má za úlohu uchovávat dvojice klúc – hodnota. Komunikujúce agenty musia vediet, aké klúce môžu ocakávat a aké typy hodnôt uchovávajú.

Triedy WorldAgent a World sú podtriedy AgentContainer-a. Hoci sú obe odvodené z rovnakej triedy, každá plní inú funkciu. Projekt (vid 3.4 Základy prostredia) pracuje s inštanciou triedy World, ktorú používa ako multi-agentové prostredie na simuláciu agentov. Trieda World pracuje s objektmi triedy WorldAgent.

Na vytváranie nových experimentov potrebujeme nové svety a nové agenty, ktoré definujeme ako podtriedy tried WorldAgent a World.

Na vytváranie nového monitora definujeme podtrediu triedy AbstractMonitor.

V práci boli pre simulácie použité implementácie SimpleAgent a SimpleWorld, ktoré sme vytvorili podla definície 1.

3.7.2 Pravidlá rožširovania

Pre agenty, svety a monitory platia pri rozširovaní urcité zásady a mechanizmy, ktoré treba dodržiavat a ktoré si popíšeme. Pri každom pravidle je popísané, pre ktoré entity platí.

Obnova (reset) – platí pre agenty, svety a monitory

Pri obnove simulácie (vid 3.4 Základy prostredia) je volaná metóda restart**.** Ak pribudli nejaké nastavenia do objektu, ktoré treba pri obnove simulácie nastavit na pôvodné hodnoty, treba predefinovat túto metódu a urobit v nej patricné nastavenia. V metóde restart je dobré volat aj jej rodicovskú verziu.

Nastavovatelné parametre – platí pre agenty, svety a monitory

Ak chceme, aby mal agent, svet alebo monitor nejaký parameter nastavitelný z užívatelského rozhrania, je potrebné definovat dve metódy. Jednu na získanie parametru, druhú na jeho nastavenie. Deklarácia metódy slúžiacej na získavanie parametra musí vyzerat takto:

```
public <TypParametra> get<MenoParametra>()
```

Pre nastavovanie parametra je deklarácia metódy nasledovná:

```
public void set<MenoParametra>(<TypParametra> param)
```

`<MenoParametra>` nahradíme zvoleným menom parametra a `<TypParametra>` nahradíme zvoleným typom parametra. Ak má byt parameter urcený iba na cítanie, netreba vytvárat metódu na zapisovanie.

Komunikácia – platí pre svety a agenty

Ak chceme vytvorit slušnú komunikáciu medzi agentmi a medzi agentom a svetom, musíme zadefinovat príslušné metódy. V komunikácii agentov prvý agent kladie otázku a druhý agent odpovedá. Hlavným parametrom komunikácie je názov otázky. Metódy na kladenie otázky sú už definované v triede Agent a netreba ich implementovat. Sú to tri metódy ask, každá s inými parametrami. Pri kladení otázky môže prvý agent použit tieto formy:

```
ask( "mojaotazka" );
ask( "mojaotazka", mojeparametrekotazke );
ask( cielovy_agent, "mojaotazka", mojeparametrekotazke );
```

Aby agent vedel odpovedat napríklad na otázku `mojaotazka` musí deklarovat metódu v tvare:

```
public Situation getAnswer_mojaotazka(Agent agent, Object params).
```

Metóda je povinná vrátit objekt triedy Situation. Je to jednoduchý ukladac párov hodnôt. Každý, kto chce komunikovat, musí vediet, aké parametre môže pri otázke poslat a co môže ocakávat vo vrátenej situácii.

Informácie o tom, aké parametre otázka ocakáva a aké hodnoty vie vrátit by mali byt pripísané pri každej implementácií takejto metódy. Napríklad, trieda World vie odpovedat na otázky sense, addAgent, delAgent a iné. Ku každej otázke ocakáva urcité parametre a dáva urcité odpovede (vid dokumentáciu API na CD).

Životný cyklus – len svety a agenty

Ak chceme zmenit správanie sa agenta alebo sveta pocas simulácie, musíme predefinovat jednu z metód jeho životného cyklu (vid 3.7.1 Potrebné znalosti). Môžme predefinovat hlavnú metódu life (co sa neodporúca), alebo niektorú z troch metód sense, memorize a chooseAction. Názvy metód napovedajú, za aké akcie metódy zodpovedajú. Metóda sense má aj štandartnú implementáciu, v ktorej zistuje aktuálny stav prostredia pomocou otázkovej metódy ask("sense") a získané informácie o stave prostredia ukladá do premenej actualSituation, ktorá je prístupná aj podtriedam.

Zmena reprezentácie priestoru – len svety

AgentContainer si udržuje zoznam agentov v inštancii triedy AgentRepresentation. Táto trieda sa tiež používa ako reprezentácia priestoru, v ktorom sú agenty uložené. Podla implementácie to môže byt napríklad mriežka, kocka, orientovaný graf a iné. Definované sú dve reprezentácie priestoru: VectorRepresentation a MatrixRepresentation ako potomkovia triedy AgentRepresentation. Prvá trieda nedefinuje žiadny priestor, agenty sú akoby nahádzané v taške, zatial co druhá trieda definuje dvojrozmerný priestor. Trieda World používa inštanciu triedy VectorRepresentation.

Ak chceme zmenit triedu pre reprezentáciu priestoru, zavoláme v konštruktore[8] sveta metódu setRepresentation. Napríklad:

```
setRepresentation(new MatrixRepresentation(200, 200));
```

Zmena zobrazovania reprezentácie priestoru – len svety

Reprezentáciu priestoru je možné zobrazovat na obrazovku. Ku každej triede predstavujúcej reprezentáciu prostredia si musíme vytvorit jej zobrazovac. Napríklad k triede MatrixRepresentation existuje zobrazovac s názvom Display2DWorld. Zobrazovac musí byt podtriedou triedy WorldDisplay.

Podobne ako reprezentáciu prostredia, aj zobrazovac nastavíme v konštruktore sveta, tentoraz pomocou metódy setDisplay. Napríklad:

```
setDisplay(new Display2DWorld());
```

[8] konštruktor objektu sa zavolá vždy pri vytvorení objektu.

Príklad zobrazovaca, ktorý vykresluje reprezentáciu triedy MatrixRepresentation na obrazovku možno vidiet na [Obrázok 27 – agenty v priestore po skupinkách].

Inicializácia – len svety

Metóda init triedy World sa volá po vytvorení nového projektu a má slúžit na pociatocnú inicializáciu sveta. Nie je volaná pri otvorení uloženého projektu. Ak potrebujeme pri inicializácii spravit vlastné nastavenia, môžeme túto metódu predefinovat. V metóde init je dobrým zvykom volat rodicovskú verziu metódy.

Vytvorenie aktuálnej situácie prostredia – len svety

Trieda World má predefinovanú komunikacnú metódu s názvom getAnswer_sense. Pomocou tejto metódy vie odpovedat agentom, ktorí sa pýtajú na aktuálnu situáciu. V tejto metóde svet volá metódu createSituationForAgent(Agent agent, Params params), ktorá má vytvorit aktuálnu situáciu, v akej sa agent nachádza. Táto metóda dostane dva parametre: agenta, ktorý si pýta stav prostredia a parametre, ktoré posiela. Pre vytvorenie vlastných situácií stací predefinovat danú metódu. Odporúca sa zachovat situáciu vytvorenú predkom a iba ju rozšírit.

Meno a popis – len monitory

Pre meno monitora treba predefinovat metódu getMonitorName a getFullName. Prvá metóda by mala vrátit jednoduché meno monitora, napríklad MonitorVzdialenosti, zatial co druhá metóda by mala vrátit obšírnejšie pomenovanie. Napríklad rozšírené o názov zdroja dát, kam sa budú monitorované hodnoty ukladat – MonitorVzdialenosti_vzdialenost. Taktiež je dobré predefinovat metódu getDescription, ktorá by mala vrátit strucný popis monitora.

Získavanie hodnôt z monitora – len monitory

Aby plnil monitor svoju funkciu, je potrebné predefinovat dve metódy. Pri simulácii si ukladac od monitora vypýta názvy zdrojov dát, kam chce ukladat hodnoty pomocou metódy getIdentifiers. Metóda musí vracat pole identifikátorov, názvov zdrojov dát. Potom si ukladac vypýta od monitora dáta pomocou metódy getValues, ktorá vracia pole

objektov. Každá položka v poli objektov by mala prislúchat jednému identifikátoru z pola identifikátorov.

Obnova štruktúr – len monitory

Pre rozširovanie monitora sa odporúca použit triedu AbstractMonitor, ktorá definuje užitocné metódy. Napríklad, ak si potrebujeme uchovávat vypocítané dáta, ktoré závisia na agentoch vo svete, môžeme použit metódu updateVectors, ktorá je volaná vždy, ked do systému príde agent alebo z neho odíde.

3.7.3 Príklad rozšírenia

Urobíme si jednoduchý príklad rozšírenia prostredia, pricom využijeme už existujúcu implementáciu jazykovej hry. V našom rozšírení sa budú agenty vo svete pohybovat a jazykovú hru budú môct hrat iba agenty ktoré budú vedla seba dost blízko. Nebudeme uvádzat kompletný kód príkladu, iba niektoré jeho casti. Celý príklad je priložený na CD diplomovej práce.

Na písanie programu v jave môžeme použit jedno z existujúcich prostredí s užívatelským rozhraním [44,45], alebo si môžeme vystacit s obycajným textovým editorom a kompilátorom javy.

Vytvoríme si adresár, kde si uložíme náš príklad. Napríklad adresár work. Objekty, ktoré budeme editovat by sme radi umiestnili do balíka[9] org.fmfiuk.elge.extended a preto si v adresári work musíme vytvorit podadresárovú štruktúru org/fmfiuk/elge/extended.

V súbore MoveWorld.java zacneme definíciou balíka a hlavicky triedy MoveWorld.

```
package org.fmfiuk.elge.extended;
/** hlavné importy[10] */
public class MoveWorld extends SimpleWorld {
      /** tu pride telo programu */
```

[9] balík (package) v jave rozdeluje triedy do logickejších štruktúr. Pri programovaní by súbory mali byt uložené v potadresároch zodpovedajúcim názvu balíku. Napríklad pre balík elge.engine má všetky definované triedy uložené v adresári /elge/engine/

[10] netreba zabudnút, že pre každú triedu, ktorú chceme použit, musíme špecifikovat, kde ju má kompilátor hladat. Bud v programe budeme písat celú cestu k triede, napríklad elge.examples.simple.SimpleWorld, alebo použijeme klúcové slovo import a zadáme cesty, ktoré má kompilátor prehladávat. Napríklad import elge.examples.simple.*;

```
}
```

Aby sme rozšírili možnosti výberu druhého agenta pri jazykovej hre, rozšírime definíciu z triedy SimpleWorld. SimpleWorld používa na zistenie druhého agenta metódu chooseSecondAgent. V tejto metóde používa parameter agentChoose, aby zistil akým spôsobom má agent vybrat. Premenná agentChoose je typu ComboInteger, co umožnuje do nej vložit preddefinované dvojice názov – hodnota. Preto si pridáme v konštruktori triedy MoveWorld do tejto premennej náš typ výberu druhého agenta a predefinujeme metódu chooseSecondAgent:

```
/** Creates a new instance of MoveWorld */
public MoveWorld() {
    agentChoose.add(BY_RADIUS); //BY_RADIUS == "by radius";
}

public GameAgent chooseSecondAgent(GameAgent first) {
    if (agentChoose.toString().equals(BY_RADIUS)) {
        return chooseAgentByRadius(first);
    } else return super.chooseSecondAgent(first);
}
```

Naša metóda chooseAgentByRadius vyberá agenta podla nastavenia maximálnej vzdialenosti. Ak chceme aby bol tento parameter nastavovatelný v užívatelskom rozhraní, vytvoríme dvojicu metód podla pravidiel:

```
public double getMaxAgentDistance() { return maxD; }
public void setMaxAgentDistance(double max) { maxD = max; }
```

Takto pribudne v editore projektu v záložke **Par. sveta** parameter s názvom MaxAgentDistance. Pre pohyb agentov vytvoríme komunikacnú metódu, s názvom relativeMove:

```
public Situation getAnswer_relativeMove(Agent agent, Object params) {
   ...
   if (agent instanceof MoveableAgent && params instanceof Point) {
       //tuto treba zmenit poziciu agenta
   }
   //vrat situaciu agentovi
   ...
}
```

Z hladiska tvorby multi–agentového prostredia nie je dobré, aby si agent menil pozíciu sám. Preto sme vytvorili komunikacnú metódu relativeMove a zariadili, že agent si môže menit polohu kladením otázky ask s menom otázky relativeMove a s parametrom typu Point. Príklad ukazuje volanie metódy relativeMove s parametrom typu Point, ktorý hovorí že sa chcem pohnút o jeden bod dolava a o jeden bod hore.

```
ask("relativeMove", new Point(-1,1));
```

Ak bude možné agent presunút na požadovanú pozíciu, presunie sa tam. Posunutie agenta zabezpecuje trieda reprezentácie priestoru, v tomto prípade MatrixRepresentation. Samotný agent je už prispôsobený na pohyb.

V súbore MoveAgent.java (podobne ako pri definovaní sveta) zacneme s balíkom a deklaráciou triedy.

```
package org.fmfiuk.elge.extended;
/** hlavné importy */
public class MoveAgent extends SimpleAgent {
       /** tu pride telo programu */
}
```

Agent chceme rozšírit o dva nastavitelné parametre: pravdepodobnost pohybu a maximálny možný krok pohybu. Pre oba parametre vytvoríme požadovanú dvojicu metód:

```
public void setMaxMoveStep(int step) { this.maxMoveStep = step; }
public int getMaxMoveStep() { return maxMoveStep; }

public void setMoveProb(double prob) { this.moveProb = prob; }
public double getMoveProb() { return moveProb; }
```

Ešte musíme definovat, kedy sa má agent pohybovat. To zabezpecíme pomocou životného cyklu agenta. Nebudeme predefinovávat metódu life, ale metódu chooseAction, v ktorej pomocou opytovacej metódy a otázky typu relativeMove zariadime, aby agent spravil pohyb (pokial to svet dovolí):

```
public void chooseAction() {
   if (Math.random() <= moveProb) {
       ask(MoveWorld.ASK_TO_MOVE,
           new Point(Engine.getRandomNumber(–maxMoveStep, maxMoveStep),
```

```
                    Engine.getRandomNumber(-maxMoveStep, maxMoveStep)));
    }
}
```

Pohyb agenta je naozaj jednoduchý. Najprv si podla pravdepodobnosti zistíme, ci sa chceme pohybovat. Potom povieme svetu, že sa chceme pohnút. Zavoláme teda metódu ask s typom otázky na pohybu (konštanta pre názov otázky je zadefinovaná v triede MoveWorld) a potrebným parametrom. Vytvoríme náhodné relatívne súradnice podla nastaveného maximálneho kroku do lubovolnej strany.

Vytvorené triedy skompilujeme a celú adresárovú štruktúru pocnúc adresárom org skopírujeme do adresára extends, ktorý sa nachádza v inštalacnom adresári prostredia. Takto zarucíme, že prostredie bude mat k nášmu rozšíreniu prístup.

V programe vyvoláme okno pre nastavenie zberne svetov z menu **Nastavenia/Zberna svetov**. Náše nové rozšírenie program asi automaticky nenájde a tak musíme pomocou tlacidla **Pridat** (vid Obrázok 19 – zberna agentov) zadat celú cestu k svetu MoveWorld – org.fmfiuk.elge.extended.MoveWorld. Tlacidlom **Nacítat** by sa v stlpci **Nahrané** mala pre náš svet MoveWorld nastavit potvrdzujúca hodnota (fajka). Ak sa tak nestane, znamená to, že implementácia triedy MoveWorld nebola nájdená. Bud sme zadali zlé meno, alebo sme nezaradili správne súbory implementácie nášho príkladu do adresára extends. Ak sa svet podarilo nacítat, treba ešte nastavit zbernu agentov (teraz by už automatické hladanie malo fungovat). V zberni agentov pridáme položku org.fmfiuk.elge.extended.MoveAgent.

Po nastavení zberní môžeme vytvorit nový projekt. Ako svet urcíme MoveWorld. Do zoznamu agentov pridáme generátor, ktorý bude generovat agenty typu MoveAgent. V záložke editora projektu **Par. sveta** nastavíme parameter **chooseSAgent** na hodnotu by radius. Pohyb agentov pocas simulácie môžeme sledovat na zobrazovaci reprezentácie priestoru, ktorý zobrazíme pomocou tlacidla **U/S**.

Záver

Cielom diplomovej práce bolo informovat o výskume evolúcie jazyka pomocou jazykových hier a naprogramovat prostredie pre experimentovanie v tejto oblasti.

Veríme, že prehlad, ktorý sme v práci vytvorili, umožní citatelom jednoducho a rýchlo preniknút do problematiky jazykových hier a vzbudí v nich záujem o experimentovanie v tejto oblasti. Dúfame, že k experimentom ich bude motivovat aj náš implementovaný program. Poskytnutý prehlad by taktiež mohol byt dobrou základnou pre jeho elektronickú verziu, ktorá by sa s postupom casu mohla rozširovat o nové experimenty a referencie na clánky v tejto oblasti. Nebolo našim zámerom podat vycerpávajúci prehlad všetkých existujúcich prác, ale túto vlastnost by mohla mat navrhnutá elektronická verzia prehladu.

Druhým cielom bola implementácia prostredia pre experimenty s jazykovými hrami a návody na jeho použitie a rozšírenie. Prostredie sa nám podarilo implementovat podla našich predstáv a požiadaviek. Ci je prostredie dostatocne prepracované, sa ukáže až pri jeho používaní inými záujemcami, ktorí môžu pozitívne prispiet svojimi pripomienkami a návrhmi. Prostredie plánujeme dalej udržiavat a rozširovat pre nové experimenty, z tohoto dôvodu sme program sprístupnili pod open–source[11] licenciou a dávame si za ciel mimo diplomovej práce udržiavat elektronický portál k tomuto prostrediu ako i samotné prostredie na adrese http://elge.sourceforge.net.

[11] http://www.opensource.org

Literatúra

1 Wittgenstein, L.: *Philosophical Investigations*. New York, The Macmillan Company, 1965.

2 Shawver, L.: *On Wittgenstein's Concept of a Language Game, http://www.california.com/~rathbone/word.htm*

3 Shawver, L.: *Commentary on Wittgenstein's Philosophical Investigations, http://users.rcn.com/rathbone/lwtocc.htm*

4 Nowak, M. A., Komarova, N. L., Niyogi, P.: *Computational and evolutionary aspects of language, http://www.ptb.ias.edu/nowak*

5 Nowak, M. A., Krakauer, D. C.: *The evolution of language***,** Proc. Natl. Acad. Sci. USA, Vol. 96, pp. 8028–8033, July 1999, Evolution

6 Steels, L.: *The spontaneous self–organization of an adaptive language,* In Muggleton, S., editor, Machine Intelligence 15, Oxford University Press. Oxford, 1997

7 Steels, L., McIntyre, A.: *Spatially Distributed Naming Games,* Advances in complex systems, 1(4), January 1999

8 Steels, L.: *The synthetic modeling of language origins,* Evolution of Communication Journal, 1(1):1–34 1997

9 Steels, L.: *Synthesising the origins of language and meaning using co–evolution, self–organization and level formation,* In Hurford, J. and Knight, C. and Studdert–Kennedy, M., editor, Approaches to the Evolution of Language: Social and Cognitive bases, Edinburgh University Press. Edinburgh, 1997

10 Steels, L.: *Constructing and Sharing Perceptual Distinctions.* In: van Someren, M. and G. Widmer (eds.), 1997

11 Steels, L.: *Perceptually grounded meaning creation, 1996*

12 Steels, L.: *The Talking Heads Experiment, Volume I. Words and Meanings.* Antwerpen, 1999

13 Steels, L., Kaplan, F.: *Stochasticity as a source of innovation in language games.* In C. Adami, R. Belew, H. Kitano, and C. Taylor, editors, Proceedings of Artificial Life VI, Los Angeles, June. MIT Press, 1998

14 Steels, L., Kaplan, F.: *AIBO's first words: The social learning of language and meaning. Evolution of Communication,* 2002

15 Steels, L., et al*.: Crucial factors in the origins of word-meaning,* in A. Wray, ed., *The Transition to Language.* Oxford (UK): Oxford University Press, 2002

16 McIntyre, A.: *Babel: a testbed for research in origins of language,* Proceedings of COLING-ACL 98, Montreal, 1998

17 Marsico, E., Coupé, C., Pellegrino, F.: *Evaluating the influence of language contact on lexical changes, http://www.infres.enst.fr/confs/evolang/actes/_actes46.html*

18 The University of Chicago's Social Science Research Computing, *RePast*, *http://repast.sourceforge.net*,

19 Santa Fe Institute, *Swarm*, *http://www.swarm.org*,

20 *AScape, http://www.brook.edu/dybdocroot/es/dynamics/models/ascape/*

21 Smith, K., Brighton, H., Kirby, S.: *Language Evolution in a Multi–agent Model: the cultural emergence of compositional structure*, 2002

22 Smith, A.D.M.: *Establishing Communication Systems without Explicit Meaning Transmission.* In J. Kelemen and P. Sosík, Prague: Springer, 2001

23 Kirby, S.: *Natural Language from Artificial Life,* in Artificial Life, 2002

24 Kirby, S.: *Learning, Bottlenecks and the Evolution of Recursive Syntax*., in Linguistic Evolution through Language Acquisition: Formal and Computational Models edited by Ted Briscoe. Cambridge University Press, in Press, 1999

25 Hurford, J. R: Expression/induction models of language evolution: dimensions and issues, in Briscoe, Ted, Eds. *Linguistic Evolution through Language Acquisition: Formal and Computational Models*. Cambridge University Press., 2002

26 De Jong, E. D.: *Autonomous Formation of Concepts and Communication*, PhD thesis, Vrije Universiteit Brussel, Brussels, Belgium, 2000.

27 De Jong, E. D., VOGT, P.: *How should a robot discriminate between objects? A comparison between two methods*, in *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior SAB '98*. Cambridge (MA): MIT Press, 1998

28 Batali, J.: *Computational simulations of the emergence of grammar*. In Hurford, J. R., Studdert–Kennedy, M. and Knight C., editors, *Approaches to the Evolution of Language – Social and Cognitive Bases*, 1998

29 Takác, M.: *Emergencia lingvistických fenoménov v jazykových hrách,* Internet Distance Education Program Cognitive Sciences, 2001.

30 Takác, M.: *Model prototypovej konceptualizácie sveta s komunikáciou*, internal technical report UI FMFI UK, 2000.

31 Kaplan, F.: *A New Approach to Class Formation in Multi–Agent Simulations of Language Evolution.*, in Demazeau, Y., editor, Proceedings of the third international conference on multi–agent systems (ICMAS 98), Los Alamitos, CA, 1998.

32 Kaplan, F.: *Talking AIBO: First experimentation of verbal interactions with an autonomous four-legged robot*, in: A. Nijholt, D. Heylen, K. Jokinen, eds., *Learning to Behave: Interacting agents CELE-TWENTE Workshop on Language Technology*. 2000

33 Gärdenfors, P.: *Language and the Evolution of Cognition*, in: V. Rialle and D. Fisette (eds.): *Penser l'esprit: Des sciences de la cognition? une philosophie cognitiv*e, Presses Universitaires de Grenoble, Grenoble, 1996.

34 Chomsky, N.: *Rules and Representations*. Columbia University Press, 1980.

35 Chomsky, N.: *Knowledge of Language: its Nature, Origin and Use*. Praeger, 1986.

36 Pinker, S.: *The Language Instinct: How the Mind Creates Language*. New York: HarperCollins, 1994

37 Oliphant, M., Batali. J.: *Learning and the emergence of coordinated communication*. The newsletter of the Center for Research in Language, 11(1), 1997

38 Prigogine, I, Stengers, I.: *Order Out Of Chaos*. Bantam Books, New York, 1984

39 Brighton, H.: *Compositional Syntax from Cultural Transmission*. Artificial Life, 8(1), 2002

40 Cangelosi, A. and Parisi, D.: *The emergence of a language in an evolving population of neural networks*. Connection Science, 1998

41 Berthouze, L., Pic M.: *Emergence of language in interactive systems*, in Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, Tucson (USA), 2001

42 Sun Microsystems, Inc. *The Java Tutorial – A practical guide for programmers, http://java.sun.com/docs/books/tutorial/index.html*

43 Umelý život, *http://alife.tuke.sk*

44 Netbeans, *http://www.netbeans.org*

45 Eclipse, http://www.eclipse.org