Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Jiří Vorba

Editor pro IVE – objekty a lokace

Kabinet software a výuky informatiky

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

Studijní program: Informatika, programování

2008

Děkuji panu Mgr. Cyrilu Bromovi, Ph.D. za odborné vedení mé bakalářské práce, za všechny rady, čas a především trpělivost, kterou mi věnoval.

Poděkování za spolupráci na softwarovém projektu IVE Editor, ke kterému se tato práce váže, a za trpělivost s mou osobou patří rovněž studentu Martinu Juhászovi.

Můj vřelý dík také věnuji studentu filozofické fakulty UK Rostislavu Taudovi za četné přípomínky a rady ke gramatické a stylistické stránce této práce.

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

V Praze dne 1.4. 2008 Jiří Vorba

Obsah

Název práce: Editor pro IVE – objekty a lokace
Autor: Jiří Vorba
Katedra: Kabinet software a výuky informatiky
Vedoucí bakalářské práce: Mgr. Cyril Brom, Ph.D.
e-mail vedoucího: brom@ksvi.mff.cuni.cz

Abstrakt: Hlavním cílem této práce je implementace editoru pro IVE, což je simulátor virtuálních lidí vyvinutý na MFF UK. Editor by měl především zpřístupnit IVE pro potřeby výuky algoritmů pro řízení virtuálních postav. Součástí práce je také rešerše z oblasti nástrojů pro prototypování algoritmů pro řízení virtuálních postav v počítačových hrách a výukových aplikacích. V textu je také představen a přiblížen samotný projekt IVE. V rámci testování editoru byl vytvořen ukázkový svět a vznikly také dva tutorialy, které jsou součástí rozsáhlé uživatelské dokumentace.

Klíčová slova: IVE, IVE Editor, level-of-detail AI, afordance, prototypování algoritmů pro řízení virtuálních postav

Title: Editor for project IVE – objects and areas
Author: Jiri Vorba
Department: Department of Software and Computer Science Education
Supervisor: Mgr. Cyril Brom, Ph.D.
Supervisor's e-mail address: brom@ksvi.mff.cuni.cz

Abstract: The main goal of this work is to implement an editor for IVE which is the simulator of virtual humans developed at MFF UK. Our editor should primarily open up IVE for teaching of algorithms for control of virtual humans. The part of the work is also the search from field of tools for prototyping control algorithms of virtual agents in computer games and teaching applications. There is also introduced the project IVE itself. Also our own demo world was made for testing purposes and there arised two tutorials which are part of the extensive user documentation.

Keywords: IVE, IVE Editor, level-of-detail AI, affordance, prototyping algorithms for control of intelligent agents

Kapitola 1

Úvod

Projekt IVE (Intelligent Virtual Environment) (IVE) je simulátorem virtuálních lidí (Brom, 2006, kapitola 1) vyvinutý studenty na MFF UK. Virtuální lidé imitují lidské chování v prostředí, které modeluje lidský reálný svět. Jedním z hlavních rysů IVE je možnost simulovat rozsáhlé virtuální světy s velkým počtem jejich obyvatel, kteří jsou řízeni speciální technikou reaktivního plánování. Projekt IVE detailněji přiblížím ve druhé kapitole, kde rozeberu jeho hlavní rysy se zaměřením na součásti týkající se mé práce.

Motivace. Jestliže chceme využívat IVE k jeho původnímu účelu, tedy pro prototypování chování virtuálních lidí, musíme být schopni popsat si své vlastní prostředí „šité na míru“ našim konkrétním potřebám. Součástí frameworku IVE ale není žádný nástroj umožňující vytvářet si různá prostředí efektivně a pohodlně. Formátem pro popis prostředí (v textu volně zaměňováno za *svět*) byl v IVE zvolen jazyk XML v kombinaci s jazykem Java. Přestože data zapsaná v XML jsou pro člověka srozumitelná a lze je poměrně snadno editovat přímo, je časově náročné popsat tímto způsobem i velmi malé světy. Vytvoření opravdu rozlehlých světů, které IVE umožňuje simulovat, je tak velmi složité a zdlouhavé.

Cíle. Cílem této práce je vytvořit komplexní nástroj pro pohodlné a rychlé vytváření virtuálních světů, který by mimo jiné částečně zpřístupnil IVE také lidem, kteří se nepohybují v oblasti IT a nemají například žádnou znalost formátu XML. Komplexnost zde znamená možnost editovat vše „pod jednou střechou“, tedy jak vytváření samotné topologie světa a objektů, tak psaní reaktivních plánů pro řízení postav. Navíc byla na počátku uvažována možnost budoucího rozšíření editoru o další části, jako je např. modul pro editaci HTN plánování, které je v současnosti do

IVE implementováno (Holeček, 2008), nebo modul pro editaci epizodické paměti obyvatel světa (Reidinger, 2007). Přestože bylo nakonec od této integrace upuštěno, byla tímto původním cílem značně ovlivněna architektura celé aplikace. Ta je popsána v oddíle 3.1.

Rozdělení práce. Na editoru jsem spolupracoval s Martinem Juhászem. První problém, kterému jsme museli čelit bylo rozdělení práce tak, abychom se vzájemně co nejméně ovlivňovali a zdržovali. Kvůli záměru snadné budoucí rozšiřitelnosti, byla aplikace navržena modulárně, aby jednotlivé moduly bylo možné vyvíjet nezávisle na sobě. Má práce se týká:

- editace objektů a topologie světa (viz Kapitola 2);
- implementace a návrhu jádra, které tvoří styčnou plochu pro jedntolivé moduly (detailně viz Kapitola 3);
- repositáře objektů (detailně viz oddíly 3.2 a 3.3).

Martin pak ve své práci (Juhász, 2008) řešil a implementoval:

- editaci UI pro IVE (reaktivních plánů);
- všechny jednotlivé součásti zajišťující nahrávání a ukládání vytvářeného světa v rámci projektu, export projektu přímo do prostředí IVE, zakládání nových projektů, nastavení projektu;
- repositář činností, které může virtuální člověk (v-člověk) ve světě vykonávat.

V tomto textu se omezuji na skutečnosti týkající se mé práce. O věcech týkajících se Martinovy práce se zde okrajově zmiňuji pouze tehdy, pokud je zde nějaká souvislost s mojí prací. Na těchto místech čtenáře odkazuji na příslušnou literaturu.

1.1. Obsah jednotlivých kapitol

Ve druhé kapitole čtenáři přiblížím samotný projekt IVE, přičemž vysvětlím jeho hlavní koncepty a s tím související termíny. Také zavedu některé akronymy používané v této práci.

Ve třetí kapitole rozvedu problémy, kterým jsem během řešení projektu musel čelit, a popíši, jakým způsobem jsem k jejich řešení přistupoval.

Čtvrtá kapitola pak pojednává o testování aplikace a jejích výstupů, tedy výsledných světů.

Pátá kapitola je rešerší v oblasti nástrojů pro prototypování algoritmů umělé inteligence pro řízení virtuálních postav. Podává stručný přehled nástrojů v jistém smyslu blízkých našemu editoru.

V šesté kapitole především diskutuji hlavní nedostatky a omezení ve vytváření světů pro IVE, které nám použití našeho editoru odhalilo. Kapitola také obsahuje seznam budoucích prací.

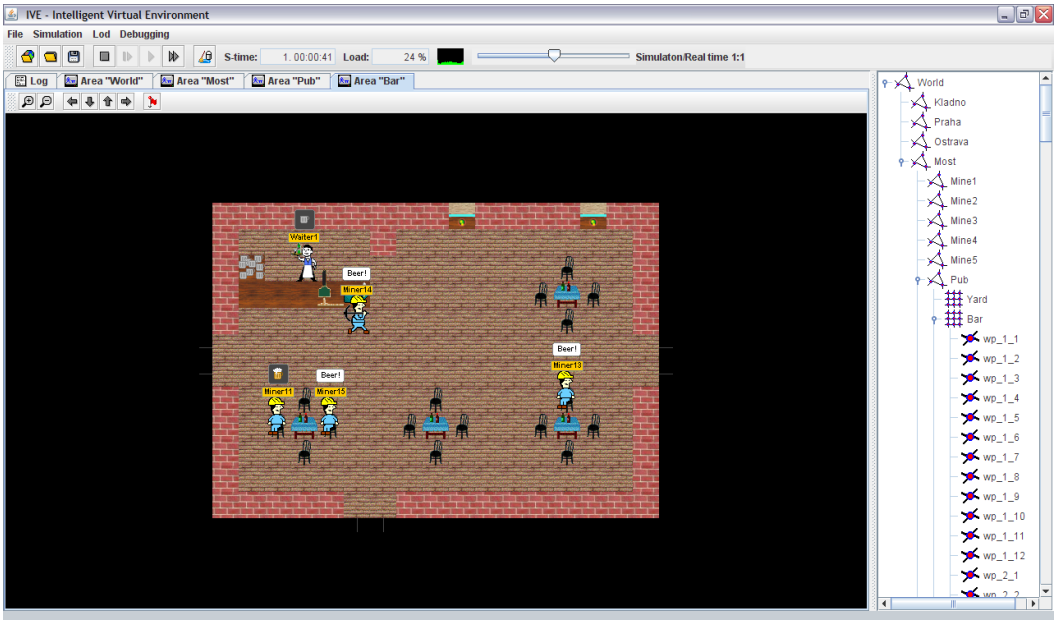
Poslední sedmou kapitolu tvoří samotný závěr, shrnující naší práci.

Kapitola 2

Přiblížení projektu IVE

V této kapitole detailněji představím projekt IVE především na konceptuální úrovni a v jejím průběhu zavedu některé termíny, které budu používat v dalších částech práce, zejména pak ve třetí kapitole. Nicméně zmíním pouze ty nejdůležitější rysy IVE a části, které souvisí přímo s mou prací na editoru. Proto je tato kapitola spíše přehledová a má za úkol člověku neseznámenému s IVE poskytnout alespoň základní představu o tom, k čemu se náš editor váže, a umožnit mu další čtení práce s lepším povědomím o řešených problémech. U míst, která detailněji nerozvádím, jsou uvedeny relevantní odkazy, kde se zájemce může dočíst bližší podrobnosti.

Představím zejména pojmy *way-point, lokace, objekt, genius, aktor, ent, senzor.*



Obr 2.1.: Screenshot z projektu IVE; pohled na hospodu na nejvyšší úrovni detailů.

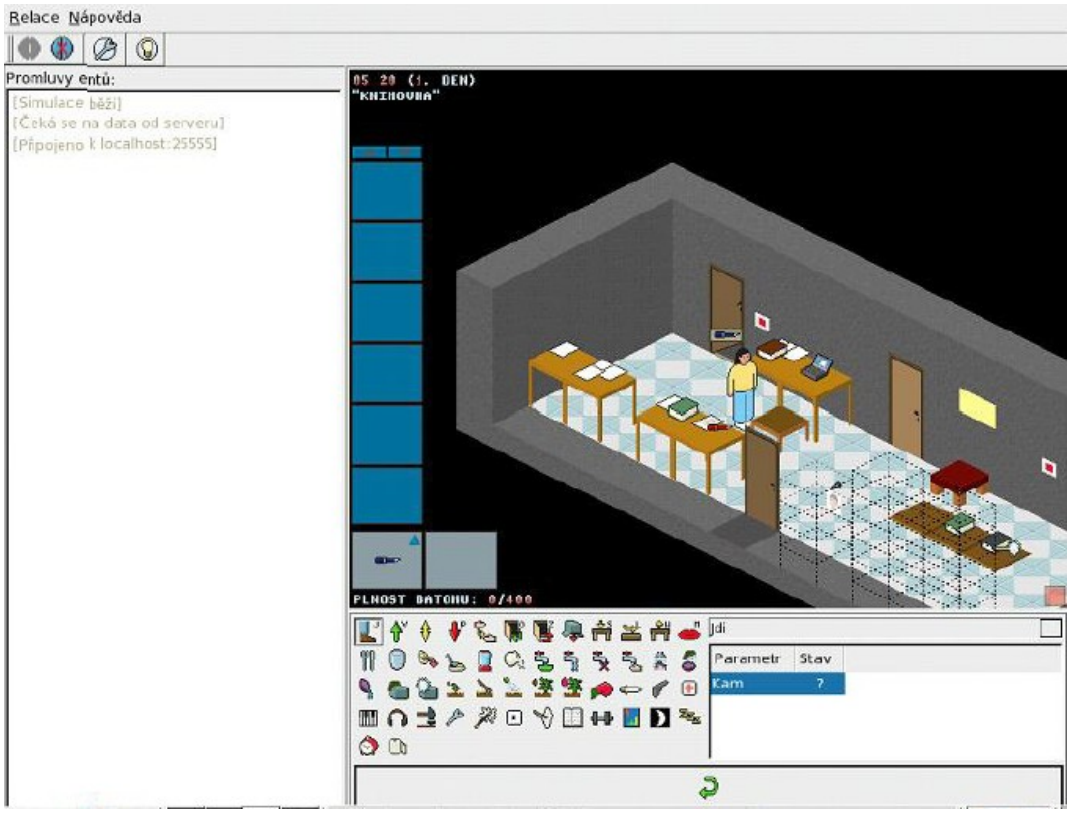
Projekt IVE vznikl roku 2005 a byl inspirován simulátorem přirozeného světa lidí, který taktéž vznikal na MFF UK jako studentský projekt pod názvem Enti (Enti). Nicméně projekt IVE na rozdíl od Entů staví na dobře popsaném teoretickém

základu. Ve své podstatě je verifikací modelové reprezentace ISMA („intention-suitability-materialisation-advice“) (Brom ad., 2006a) a techniky level-of-detail (Brom, 2006, kapitola 8), které byly obě navrženy Cyrilem Bromem a které vznikly právě z důvodů odstranění nedostatků Entů. Těmito nedostatky jsou především:

1) nemožnost rozšiřovat stávající svět Entů o nové objekty a místa za běhu simulace;

2) neúnosné zpomalení simulace již při třech postavách ve světě.

ISMA je řešením na konceptuální úrovni, které projekt IVE implementuje. ISMA a s ní související speciální algoritmus reaktivního plánování S-GHRP zavádí a detailně popisuje Brom (2006). Zde tyto pojmy jen stručně nastíním. Popis implementační úrovně v IVE pak spíše spadá do bakalářské práce Martina Juhásze (Juhász, 2008), který implementuje modul editoru pro modelování reaktivních plánů.



Obr. 2.2.: Screenshot z projektu Enti; převzato z Brom (2006, str. 14).

První problém (1) je odstraněn právě díky representaci ISMA. Zde se mimo jiné přechází z takzvaného objektivního náhledu na svět (kdy virtuální lidé vnímají svět

jako soubor „nálepek“ fyzikálních vlastností objektů), použitého v projektu Enti, k nahlížení relativnímu, inspirovanému teorií afordancí percepčního psychologa J. J. Gibsona (Brom, 2006, oddíl 4.3).

Z důvodů objektivního náhledu na svět v projektu Enti pak změna světa, kdy je přidán nový objekt, vede k nutnosti změny reaktivního plánu bytosti.

Při relativním nahlížení vidí virtuální lidé svět jako soubor potenciálních akcí, které můžou ve světě vykonat, přičemž každá bytost vnímá svět trochu jinak. To znamená, že dvě různé bytosti mohou na stejném objektu pozorovat rozdílné akce, které by s objektem mohly vykonat. Říkáme, že objekt poskytuje dané bytosti afordance. Tímto se autor ISMA zbavuje nutnosti „překladu“ objektivních vlastností světa do subjektivního vnímání dané postavy v její mysli, tato mysl tak nemusí být přepsána pokaždé, kdy je do světa přidán nový objekt.

V-lidé tedy nejsou nahlíženi jako inteligentní a autonomní, ale nechají se navádět *inteligentním prostředím* (odtud akronym IVE), kde je distribuována veškerá inteligence. Nicméně je zachována iluze inteligentních bytostí.

S druhým problémem (2) přímo souvisí členění světa, které se dotýká editování lokací. Problém zpomalení simulace při větším počtu bytostí ve světě je v IVE řešen technikou LOD (level-of-detail), kdy je simulace plynule zjednodušována v místech, kam se uživatel zrovna „nedívá“. Tato technika šetření výpočetní kapacity je převzata z oboru počítačové grafiky a přenesena na pole virtuálních bytostí. Jde tedy o zcela stejný princip. Pokud je nějaké místo v obraze vzdáleno od pozorovatele, není nutné počítat zanedbatelné detaily vykreslovaných objektů, neboť ty nebudou v obraze podstatné a neupoutají pozornost diváka. Zjednodušování detailů postupuje plynule. Čím dál je objekt vzdálen od diváka, tím menší je jeho úroveň detailů. Analogicky se tak děje i při simulaci virtuálního světa. Tato technika značně šetří výpočetní kapacitu a umožňuje tak simulovat rozsáhlé světy s velkým množstvím jeho obyvatel.

2.1. Názvosloví

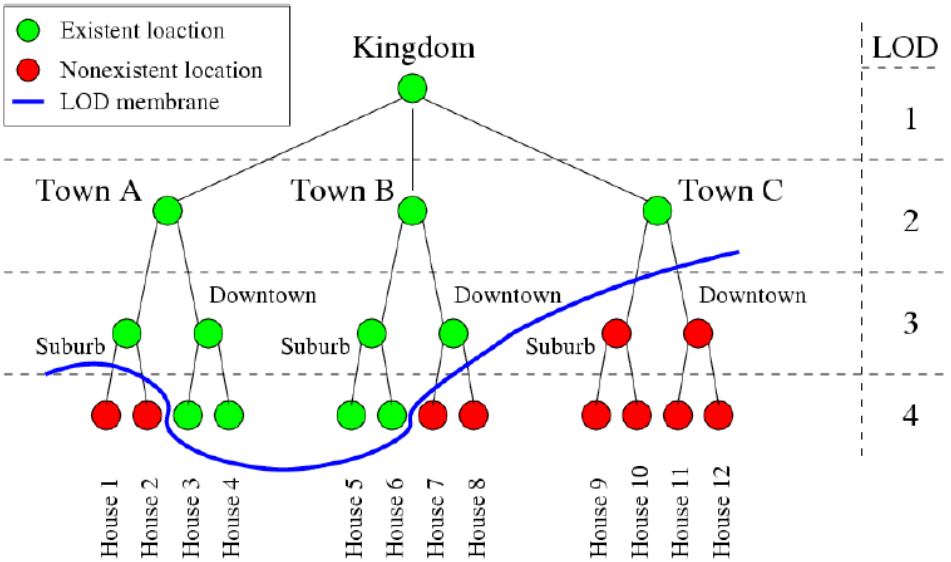
Lokace a way-pointy. Aby bylo možné techniku LOD použít, je nutné členit svět hierarchicky. Pak lze definovat, kdy se uživatel někam dívá a jak moc detailně se dívá.

Svět je členěn na místa, nazývaná *way-pointy*, která nabízí aktorům afordanci stání a tvoří uzly topologického grafu. Tento graf má vždy podobu stromu. Tyto way-pointy nicméně neslouží pouze jako navigační body, použité v algoritmech hledání cesty, jak tomu bývá zvykem – např. v prostředí hry Unreal Tournament využívané projektem Pogamut (Bída ad., 2006) –, ale opravdu představují „fyzickou" realitu světa. Hrany tohoto grafu nabízí afordanci přejít na sousední místo. Přestože se v dokumentaci IVE (Kubr ad., 2006a) o way-pointech píše jako o kterémkoli uzlu popsaného stromu, zde bude termín way-point vždy označovat pouze listy představující atomická místa světa, tedy místa s největší možnou úrovní detailů.

Uzly stromu, které nejsou listy, budou označovány jako *lokace*. Ty představují jakýsi „vzdálenější" pohled na svět. Nejvzdálenější pohled pak představuje kořen stromu, tzv. *kořenová lokace*, která reprezentuje celý svět. Například podlaha domu může být tvořena way-pointy, na kterých se mohou pohybovat aktoři. Všechny tyto way-ponity budou mít společného předka – dům, což odpovídá skutečnosti, že podlaha je položena v tomto konkrétním domě. Při pohledu z „větší dálky" uvidíme, že dům je na předměstí atd., podobně jako na obrázku 2.3.

Jednotlivá patra stromu označujeme čísly z intervalu 1..n a mluvíme o nich jako o i-té úrovni LOD, přičemž nejnižší úroveň, kterou tvoří kořenová lokace, je označena číslem 1.

Popis souvislosti mezi hierarchickou strukturou světa, dále označovanou jako *topologie světa*, a řešením problému se zpomalováním simulace rozlehlých světů však spadá spíše do práce Martina Juhásze (Juhász, 2008). Detailně se o této souvislosti lze také dočíst v dokumentaci k IVE (Kubr ad., 2006a, oddíly 2.3.4 a 2.3.5) a v článku (Brom ad., 2006b).



Obr 2.3.: Hierarchická struktura virtuálního království spolu se znázorněním membrány určující rozsah zjednodušení simulace. O tomto je pojednáno v dokumentaci k IVE v článku 2.3.5, odkud je obrázek převzatý.

Objekt. Je jednou ze dvou „fyzických entit“ světa, která má *atributy* různých typů (v současnosti IVE podporuje čtyři typy atributů objektu, viz Kubr ad. (2006a, oddíl 3.2.6)) popisující vlastnosti objektu. Objekty jsou typicky umístěny na nějakém way-pointu, nicméně z důvodu možné kontrakce lokací, která souvisí se zjednodušováním simulace a LOD (zde není popisována, detaily viz Kubr ad. (2006a, oddíl 2.3.5)), může být toto pravidlo porušeno a objekt bude umístěn v některé z lokací.

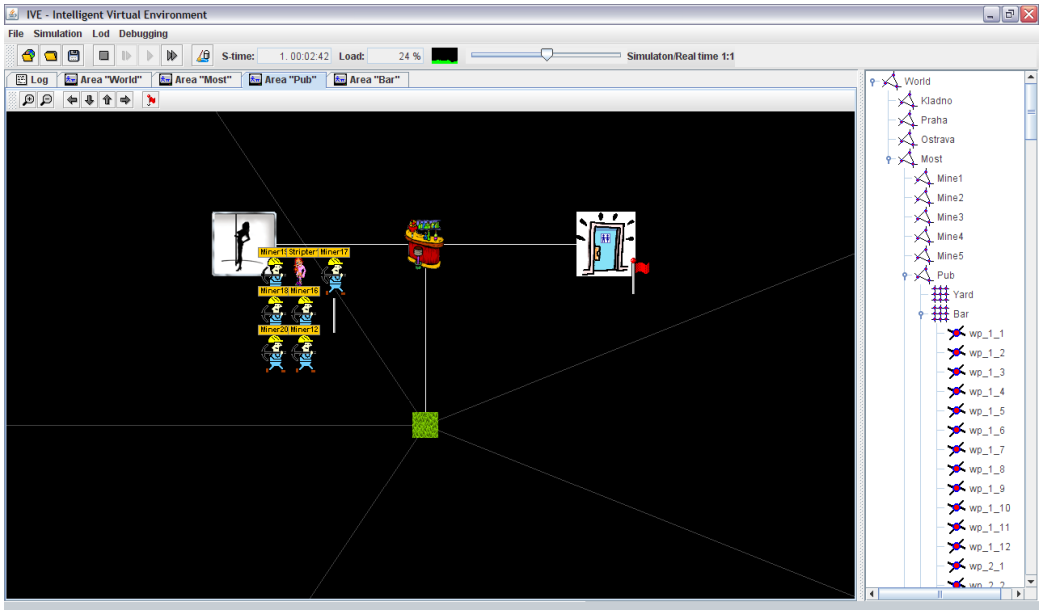
Aktor. Jde o druhou „fyzickou entitu“ ve světě, pro kterou platí vše, co bylo řečeno o objektech. Aktora tak lze chápat jako speciální typ objektu. Aktor je tělem virtuálního člověka bez mysli a lze ho v jistém smyslu chápat jako projekci v-člověka do graficky znázorněného prostředí.

Génius. Na rozdíl od aktora a objektu jde o „nehmotnou“ entitu. Tedy takovou, která nemá žádnou grafickou reprezentaci ve světě. Je určena pro navádění aktorů realizací reaktivně plánovacího algoritmu S-GHRP („simple goal driven hierarchical reactive planning“) (Brom, 2006, kapitola 4). Základním případem ve světě je

kombinace <aktor, génius>. Framework IVE umožňuje ale i složitější případy, kdy jeden aktor je naváděn více génii, či dokonce kdy jeden génius řídí chování více aktorů. V takovém případě je pak využita takzvaná metoda *role-passing*, kterou detailně popisuje Brom ad. v článku (2005/2006) a Brom v práci (2006, kapitola 9). V této metodě génius pro řízení aktora předává tohoto aktora jinému géniu, který bude centrálně řídit chování více aktorů najednou. To umožňuje daleko propracovanější chování více aktorů při činnostech, vyžadujících kooperaci aktorů ve smyslu uvěřitelnosti.

Ent. Je chápán jako dvojice <aktor, génius>. Tedy pokud mluvíme o entovi, nejde již jen o „loutku bez mozku“, ale o bytost schopnou rozhodování a tedy vykazující inteligentní chování ve virtuálním prostředí. Jde vlastně o synonymum ke spojení v-člověk na technické úrovni vyjadřování projektu IVE.

Senzor. Senzor je pouze speciálním objektem ve světě, který je schopen entovi, jemuž je přiřazen, zprostředkovávat určitou množinu informací o svém okolí. Právě díky senzorům je svět IVE pouze částečně pozorovatelný. Jeden aktor může užívat více senzorů najednou. Senzor ve své podstatě funguje jako filtr úplné informace o prostředí, ve kterém se ent nachází a umožňuje mu vnímat své okolí.



Obr. 2.4.: Screenshot z projektu IVE; Pohled na hospodu s menší úrovní detailů. Vidíme tančírnu s několika aktory a jedním objektem, která není expandovaná. Dále lokaci bar, toalety a dvůr. Šedivé úsečky a polopřímky naznačují spojení lokací a nabízí entům affordanci procházení mezi lokacemi.

2.2. Závěr

Pro detailnější seznámení s projektem IVE uvádím odkaz na jeho web (IVE) a doporučuji nastudovat zejména jeho dobře napsanou dokumentaci (Kubr ad., 2006a). Teoretické koncepty, na kterých IVE staví jsou pak prezentovány v již zmiňované disertační práci Cyrila Broma (Brom, 2006) a v článcích (Brom ad., 2006b) a (Brom ad., 2006a). Jak již bylo řečeno, předpokládám u uživatele IVE Editoru dobrou znalost IVE a konceptů, na kterých staví a vřele doporučuji prostudovat odkazy uvedené v tomto odstavci.

Kapitola 3

Analýza řešených problémů

V této kapitole budu diskutovat problémy, se kterými jsme se museli vypořádat během psaní editoru. Nejprve popíši jak byla pojata celková architektura editoru, jejíž návrh a implementace je součástí mé práce.

V další části uvedu problémy, o kterých si myslím, že stojí za zmínku, a jimž jsem čelil při řešení části editoru, který umožňuje editovat objekty, lokace, way-pointy, senzory a enty.

V následujícím oddíle pak tyto problémy detailněji analyzuji včetně popisu koncepce mého přístupu k jejich řešení.

3.1. Architektura editoru

Cíle architektury. Architektura editoru je navržena tak, aby umožňovala:

1. rozdělení práce na editoru mezi dva lidi;
2. rozšiřitelnost editoru o nové nástroje pro framewok IVE.

Na vývoji editoru jsem spolupracoval s Martinem Juhászem a architektura editoru byla navržena tak, abychom se vzájemně co nejméně zdržovali a byli nuceni řešit minimum společných problémů.

Co se týče druhého požadavku, již v úvodu jsem zmínil, že ačkoli tento rys editoru nakonec nebyl plně využit tak, jak se očekávalo, tato vlastnost editoru zůstala zachována. Protože tento požadavek rozšiřitelnosti silně ovlivnil návrh architektury celého editoru, uvádím jej zde jako ospravedlnění mého přístupu.

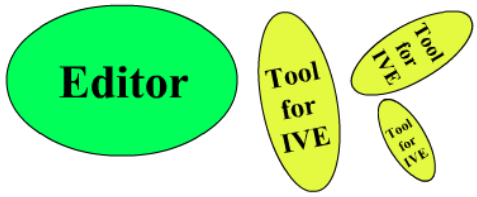
Řešení architektury. Editor je implementován v jazyce Java stejně jako framework IVE.

Vzhledem k požadavku (2) se náš editor stal součástí větší modulární aplikace jako soubor několika jejích modulů. Tím se rozšiřitelnost editoru přesouvá o patro výš na rozšiřitelnost aplikace, sdružující různé nástroje pro IVE. Jedním z těchto nástrojů je právě náš editor. Jestliže budu ve zbytku této kapitoly mluvit o *aplikaci*, budu mít na mysli právě tuto modulární aplikaci, jejíž součástí je i náš editor.

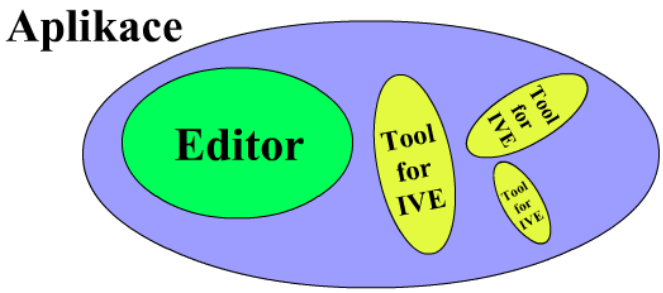
Toto rozlišení je pro uživatele nepodstatné. Jak již bylo řečeno, náš editor je jedinou součástí této aplikace, a proto ji uživatel jako celek vlastně vnímá pouze jako editor. (My se v tomto duchu dopouštíme stejné nepřesnosti, když tuto aplikaci, místo např. „Toolset for IVE“ nazýváme IVE Editor.) Existenci editoru jako modulární součásti větší aplikace si uživatel neuvědomuje, čímž je ospravedlněna naše nepřesnost. Toto rozlišení je ale nyní potřeba k popsání architektury našeho projektu.

Obsah předchozích dvou odstavců je pro lepší představu znázorněn graficky na následující straně.

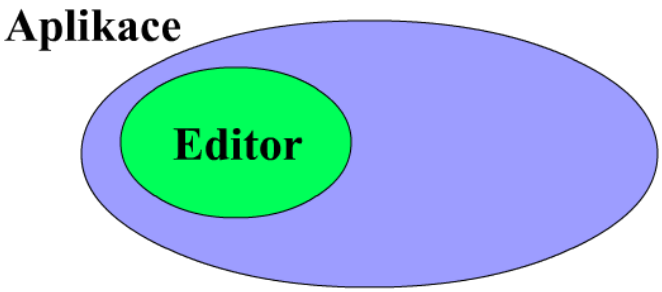
Před návrhem architektury; chceme editor, chceme možnost přidat další nástroje pro IVE



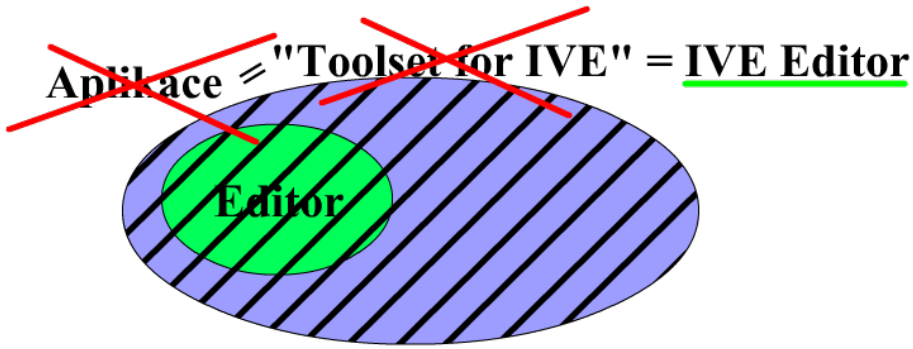
Logické vyústění předchozí situace; sdružení nástrojů pro IVE do společné modulární aplikace



Rozsah naší implementace;
Žádné jiné nástroje nebyly přidány, přestože to naše aplikace umožňuje.



Jak to vidí uživatel a proč se tedy přesně nerozlišuje mezi něčím, co by se mohlo nazývat "Toolset for IVE" a IVE Editorem; Šrafovaná část představuje uživatelův skutečný vjem



Prvky architektury

Aplikace v obecném případě sestává z několika částí. Z jednoho *jádra* a libovolného počtu *modulů* a *komponent.*

Modul je část aplikace, která:

i. rozšiřuje aplikaci o novou funkcionalitu;

ii. může přidat do aplikace nové služby pro ostatní moduly a komponenty (týká se komunikace, popsané v dalším textu);

iii. lze vytvořit nezávisle na ostatních modulech a komponentách;

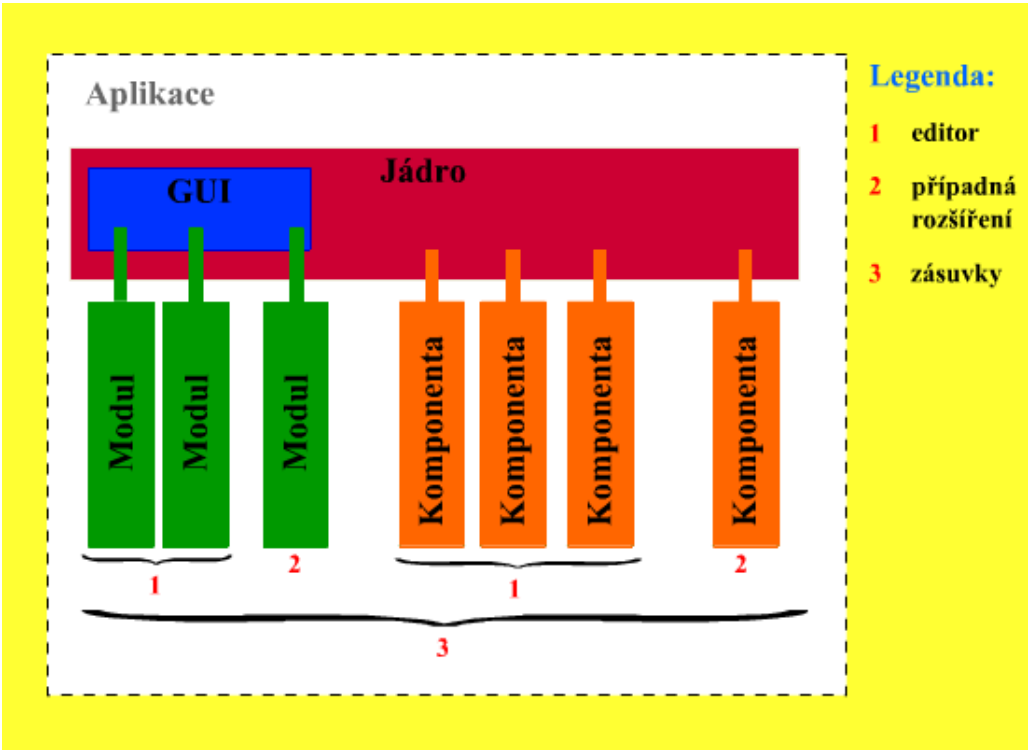
iv. může využívat libovolné služby jádra (viz odstavec o jádře);

v. definuje vlastní GUI.

Komponenta je pak část aplikace, pro kterou platí body i.–iv. z definice modulu. Hlavním rozdílem je tedy absence GUI. Je zřejmé, že možnost rozšiřování aplikace o zásuvky je v architektuře začleněna zejména pro umožnění dekompozice vývoje aplikace. Nicméně i komponenta může své funkce prezentovat přímo uživateli skrze specifickou službu jádra, což je vysvětleno v odstavci pojednávajícím o jádře.

Zásuvka. Tento termín zavádím jako zobecnění pojmů modul a komponenta. Budu jej v následujícím textu používat vždy, když nebude potřeba speciálně rozlišovat mezi komponentou a modulem. Tedy pokud budou v daném kontextu důležité pouze společné vlastnosti i.–iv..

Jádro. Tvoří styčnou plochu pro jednotlivé zásuvky. Je navrženo zcela minimalisticky, přičemž zajišťuje zobrazení GUI celé aplikace a jednotlivých modulů. GUI každého modulu se zobrazuje v samostatné záložce v hlavním okně aplikace, aby měl uživatel možnost přepínat mezi jednotlivými moduly.



Obr. 3.1.1: Znázornění modulární architektury aplikace. Naznačen rozdíl mezi termíny modul, komponenta a zásuvka; editor je „pouze“ součástí rozšiřitelné aplikace.

Rozšíření architektury

Architektura také řeší následující dva požadavky, které přirozeně vyplývají z cílů zmíněných na začátku oddílu 3.1.:

1) některé součásti by si zásuvky neměly definovat samy, ale měly by být společné pro všechny zásuvky a být pevnou součástí „minimalistického“ jádra;

2) komunikace mezi jednotlivými zásuvkami.

Hlavní nabídka. Samotné jádro poskytuje rozhraní, přes které se zásuvky mohou prezentovat uživateli a dávat mu přímo k dispozici některé ze svých funkcí. V případě modulu lze definovat položky nabídky dynamicky. Tj. aktuální nabídka se bude měnit podle toho, který modul má právě uživatel v rámci GUI aktivovaný.

Koncept hlavní nabídky přesně odpovídá požadavku 1) a je jedinou součástí, která byla takto implementována.

Rozhraní pro ovládání hlavní nabídky jednotlivými zásuvkami patří do souboru služeb, které jádro poskytuje všem zásuvkám. Mimo jiné je to například možnost aktivovat/deaktivovat konkrétní modul, vytvářet okna jako potomky hlavního okna aplikace a jiné. Rozšíření architektury o služby jádra je součástí obrázku 3.1.2.

Komunikace. Řešení komunikace mezi zásuvkami plně respektuje druhý cíl architektury editoru – rozšiřitelnost. Proto je navržena velice obecně a flexibilně.

Každá zásuvka může nabízet svému okolí (tedy ostatním zásuvkám) služby. Tyto služby mohou být skrze jádro prezentovány ostatním zásuvkám, které je taktéž prostřednictvím jádra mohou bez jakýchkoli omezení využívat.

Návrh formátu konkrétní služby je plně v režii autora dané zásuvky. Tedy služba může mít libovolné parametry i výstupy a může být využita pro předávání nejrůznějších typů dat.

Zřejmě byl popsán jeden směr komunikace, kdy nějaké služby konkrétní zásuvky jsou využívány z „jejího okolí“. Co když je ale potřeba informovat okolí o události vzniklé v zásuvce?

Tento směr komunikace si implementuje sám autor zásuvky, která takovýto typ komunikace vyžaduje, prostřednictvím dosud popisovaného systému komunikace. Autor zveřejní službu na své zásuvce, která umožní „okolí“ registrovat se jako její listener. Na všech listenerech pak v případě vzniklých událostí volá konkrétní služby, které zásuvka na listenerech vyžaduje implementovat.

Konkrétní forma komunikace realizovaná mezi zásuvkami, které tvoří náš editor, bude popsána v oddílech 3.2 a 3.3.

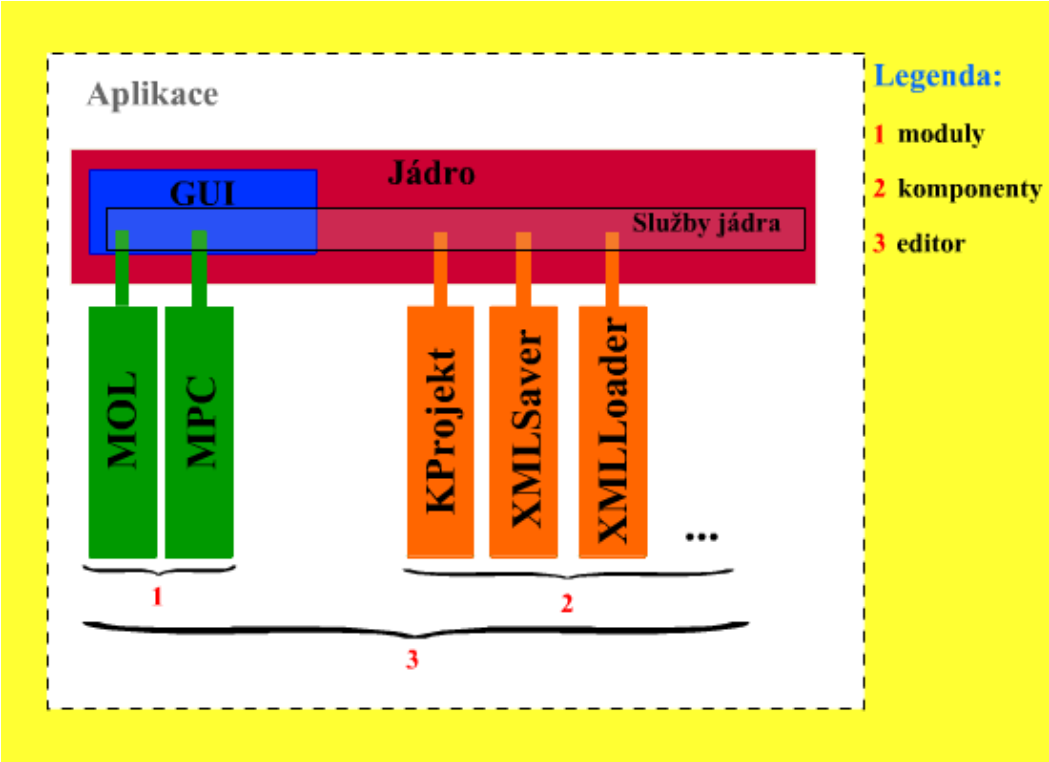
Editor jako součást aplikace

Vlastní editor je tvořen dvěma konkrétními moduly MOL a MPG a několika komponentami. Všechny komponenty jsou součástí práce Martina Juhásze (Juhász,

2008) a řeší některé jeho dílčí úkoly jako nastavení projektu, zakládání/ukládání/nahrávání nových projektů a mnohé jiné. Zde nejsou popisovány.

MOL. Modul objektů a lokací je stěžejní částí mé práce na editoru. Umožňuje uživateli editovat topologii světa, objekty, enty a senzory. V rámci něj je implementován i repositář objektů. MOL je detailněji rozebrán v oddílech 3.2 a 3.3, kde jsou také diskutovány problémy, které byly řešeny v rámci jeho návrhu a implementace.

MPG. Modul procesů a cílů dává uživateli možnost editovat reaktivní plány a vůbec vše, co se týká UI v IVE. MPG je součástí práce Martina Juhásze (Juhász, 2008) a zde se mu dále nebudu věnovat.



Obr. 3.1.2.: Konkrétní dva moduly MOL a MPG a některé komponenty, které tvoří samotný editor v rámci aplikace. Jádro poskytuje stejné služby všem zásuvkám. Služby jádra zásuvkám umožňují mimo jiné i ovládání hlavní nabídky a implementují koncept komunikace mezi zásuvkami, neboť umožňují zásuvkám prezentování a vzájemné využívání jejich služeb.

Závěr k návrhu architektury

Aplikace byla navržena modulárně s ohledem na budoucí rozšiřitelnost a možnost oddělení práce na editoru pro více lidí. Editor je „pouhou“ součástí této aplikace, přičemž sestává z několika autonomních zásuvek. Těm je mezi sebou umožněna komunikace prostřednictvím jádra.

Zde si dovolím malou poznámku k procesu návrhu architektury. Tomu předcházelo nastudování problematiky návrhových vzorů, které celý proces značně usnadnilo, zejména prostudování publikace Shalloway ad (2002), díky které jsem krom povědomí o postupech a možnostech designování aplikací získal i nový náhled na paradigma objektově orientovaného programování.

3.2. Problémy řešené v MOL

V tomto oddíle uvedu hlavní problémy, se kterými bylo nutné se vypořádat během práce na MOL. Následně se ke každému z nich vyjádřím v krátkém odstavci, přičemž přístup k jejich řešení je uveden v oddíle 3.3.

Při vývoji modulu MOL bylo nutné zejména:

1. vytvořit pro uživatele *„user-friendly“ prostředí* umožňující snadnou editaci;
2. implementovat *repositář objektů;*
3. zvolit *nástroj pro práci s XML;*
4. vyřešit *komunikaci* s ostatními zásuvkami v rámci editoru.

1. Vstřícnost k uživateli – neboli „user-friendly“ prostředí – je základním předpokladem úspěchu každého editoru. Je velice nepříjemné, pokud na uživatele číhá změť tabulek, tlačítek a formulářů. Mojí snahou tedy bylo v MOL vytvořit co nejjednodušší GUI umožňující snadnou orientaci a ovladatelnost.
2. Repositář objektů umožňuje přenášet vytvořené objekty, enty, a senzory mezi různými projekty a sdílet sady těchto vytvořených entit mezi různými uživateli.

Řešení repositáře s sebou přineslo několik dílčích problémů, které plynou z nutnosti držet se v „mantinelech“ reprezentace světa navržené tvůrci IVE. A to:

- navrhnout vhodnou fyzickou reprezentaci repositáře a formát, v jakém se budou data uchovávat a přenášet;
- řešit konflikty při nahrávání objektů z/do repositáře, pokud v projektu/repositáři již daný objekt existuje;
- řešit konflikty grafických šablon (pojem vysvětlen v oddíle 3.3 ve druhém odstavci).

3. Hlavním datovým formátem, ve kterém IVE přijímá popis světa je XML. Proto musel být zvolen vhodný nástroj umožňující programové zpracování tohoto formátu.
4. Editor je dohromady složen z několika zásuvek. Je tedy zřejmá nutnost kooperace těchto částí. Na tomto místě mě především zajímá komunikace MOL s ostatními zásuvkami. Konkrétně výměna informací při:
   - načítání/ukládání dat;
   - editaci géniů;
   - získávání informací o aktuálním projektu.

3.3. Přístup k řešení problémů v MOL

V této části podrobněji popisuji problémy uvedené v oddíle 3.2 a uvádím přístup k jejich řešení.

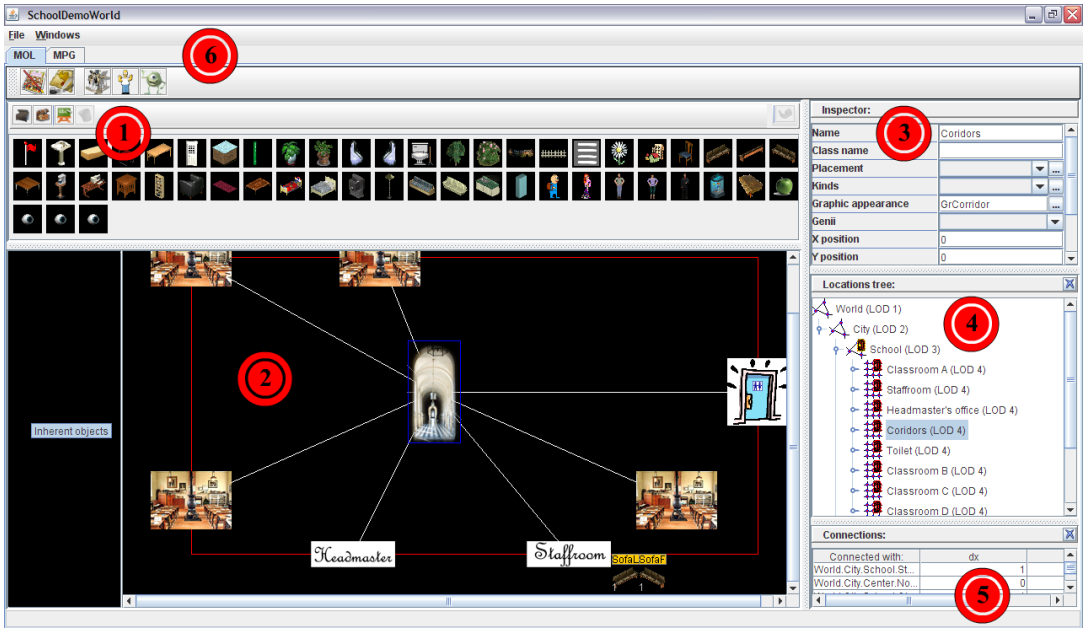
1. *Přívětivé GUI pro uživatele*. GUI se skládá z pěti hlavních oblastí, panelu nástrojů a voleb v hlavní nabídce aplikace.

Velikost pěti hlavních oblastí na obrazovce může uživatel libovolně škálovat. Zobrazení dvou z těchto oblastí může uživatel zcela potlačit, a dát tak získaný prostor k dispozici ostatním oblastem.

Uživateli je dána možnost více pohledů na jednu skutečnost. Například topologii světa může uživatel editovat prostřednictvím pohledu odpovídajícímu výsledku, který bude zobrazen po načtení světa v IVE, nebo může pro editaci využít pohled na stromovou hierarchii lokací.

Pro editaci hlavních vlastností objektů, lokací, entů a senzorů je určen inspektor objektů. Inspiraci pro tuto techniku jsem získal v IDE různých programovacích jazyků, kde lze vytvářet GUI stylem WYSIWYG. Tento přístup umožňuje uživateli snazší orientaci v editoru (vlastnosti všech entit může měnit na jednom místě) a rychlý přehled o stavu zvolené entity.

Objekty, senzory a enty lze editovat všechny v jedné oblasti díky jejich podobnosti. Zde lze také pracovat s repositářem objektů.

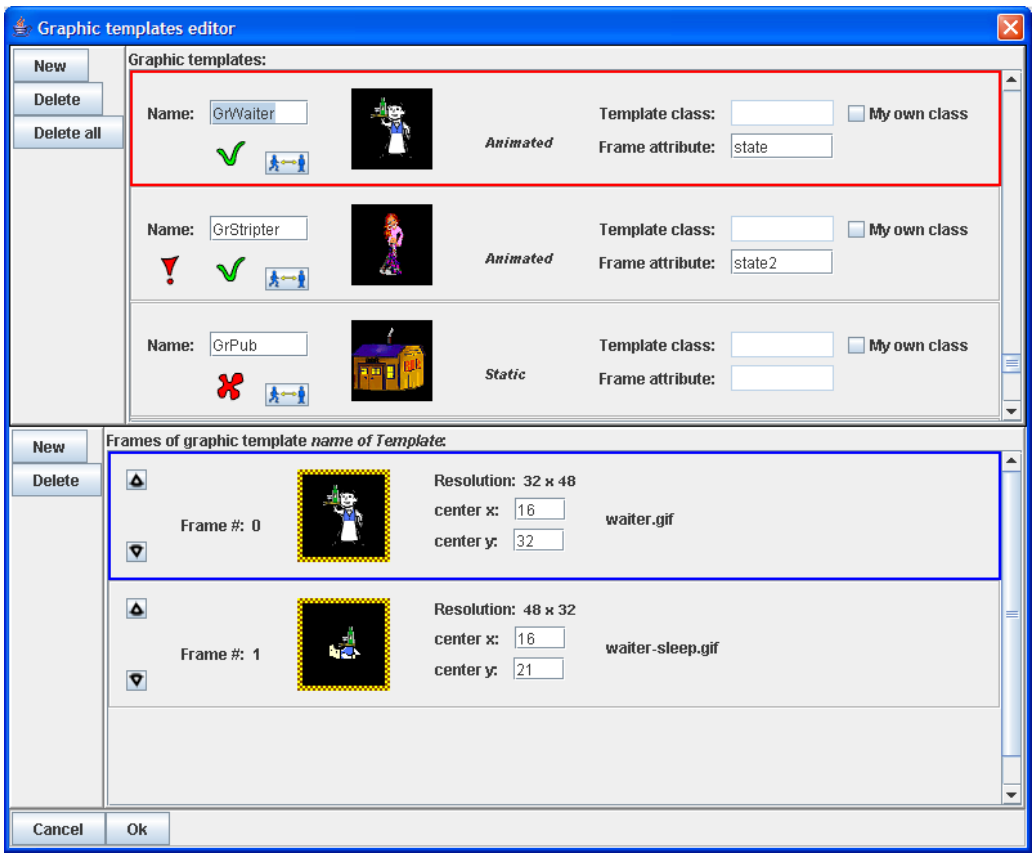


Obr. 3.3.1.: GUI modulu objektů a lokací: 1) oblast s objekty, enty a senzory; zde lze zapojit repositář objektů; 2) pohled na část světa v podobě, v jaké bude vypadat v IVE; v zobrazeném projektu jde o pohled do školy na LOD 3; 3) inspektor objektů; zde lze upravovat vlastnosti zvolené lokace „Corridors“; 4) Jiný pohled na topologii světa – stromová hierarchie lokací a way-pointů; 5) úprava spojení lokací; 6) hlavní nabídka, záložky na přepínání mezi moduly MOL a MPG, panel nástrojů (vyjmenováno shora dolů); velikost oblastí 1–5 může uživatel libovolně škálovat; zobrazení oblastí 4 a 5 lze úplně potlačit.

Grafická podoba všech editovatelných entit je určována pomocí grafických šablon. Vytvořenou grafickou šablonu pak uživatel přiřazuje vybraným entitám. Ty

všechny sdílí informace o své grafické podobě, kterou nese daná šablona, a při její změně se změní i vzhled všech entit, kterým byla přiřazena.

Šablony lze editovat skrze speciální dialog, který lze vyvolat jak z panelu nástrojů, tak z inspektoru objektů. Tato možnost dostat se k úpravě jedné věci z několika různých míst je v editoru dána uživateli poměrně často. To umožňuje uživateli zvyknout si na své vlastní postupy při editování a dává mu větší šanci intuitivně nalézt to, co v danou chvíli potřebuje.

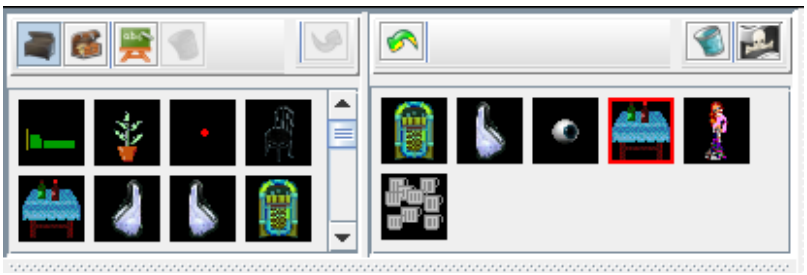


Obr. 3.3.2.: Speciální dialog pro editaci grafických šablon.

Implementace. Pro implementaci GUI v javě jsem využil frameworku JFC, zejména knihoven Swing. Bylo nutné mnohokrát použít návrhový vzor model-view-controller pro vytvoření výše zmíněných různých pohledů na jednu skutečnost.

Závěr. Popsal jsme prostředí MOL a jeho přístup k editaci světa. Ten byl takto zvolen za účelem vyřešení problému číslo 1, tj. naplnění požadavku „user-friendly“ editoru.

2. *Repositář objektů.* Repositář je ovládán v části GUI, kde se nachází všechny objekty, enti a senzory (v kontextu repositáře budu označovat souhrně jako entity). Tam lze vytvářet nový repositář, mazat existující repositáře, otevírat existující repositáře. Po otevření repositáře se uživateli zobrazí jeho obsah, který může nyní importovat do projektu nebo může z projektu ukládat jednotlivé entity do repositáře.



Obr. 3.3.3.: Otevřený repositář objektů. V levé části jsou objekty existující v projektu, jak bylo popsáno na obr. 3.3.1. pod číslem 1); v pravé části jsou objekty v právě otevřeném repositáři. Objekty lze mezi projektem a repositářem přesouvat tažením myší, nebo tlačítky v horní liště.

Fyzická reprezentace. Ve skutečnosti je každý repositář tvořen adresářovou složkou na pevném disku, ve které se nachází specifický XML soubor, který nese informace o daném repositáři a jeho obsahu. Každé entitě uložené v repositáři odpovídá jeden XML soubor, který nese informace o dané entitě. Spolu s ním je přítomný další XML soubor, popisující grafickou reprezentaci entity, a s ním i potřebné grafické soubory. Jestliže je součástí vytvořeného objektu i implementace nějaké třídy v Javě, je tato také uložena do repositáře.

Přenos entit. Jednotlivé entity jsou do/z repositáře ukládány/nahrávány prostřednictvím GUI stylem „drag & drop“ nebo pomocí tlačítek. Při přenosu mohou vznikat konflikty jak ve jménech objektů, tak i v pojmenování grafických šablon, které jsou ukládány spolu s objekty.

Při konfliktu je uživateli dána možnost vyřešit vzniklou situaci:

- přejmenováním přenášené entity;
- přejmenováním entity na cílovém místě;
- nahrazením entity na cílovém místě přenášenou entitou;
- zrušením požadované akce.

Toto řešení lze uplatnit jak při konfliktu jmen entit, tak při konfliktu jmen grafických šablon. Někdy může nastat i situace, kdy uživatel musí při jednom přenosu vyřešit oba předchozí typy konfliktů najednou.

Výše uvedené řešení je důsledkem přístupu projektu IVE k objektům a grafickým šablonám (Kubr ad., 2006b, str. 22 oddíly 4.2 a 4.3), který byl do editoru převzat.

3. *Zpracování XML.* Pro programové zpracování XML jsem se rozhodl použít rozhraní JDOM, které je určené speciálně pro jazyk java. JDOM také poskytuje uživatelsky lepší OO programové rozhraní než rozhraní DOM. To bylo zásadní pro rozhodnutí použít jej v našem editoru. Jelikož XML data je potřeba nejen číst, ale i vytvářet a dynamicky modifikovat, je nutné mít celý strom XML dat načtený v paměti. Proto nepřipadaly v úvahu žádné implementace XML parserů založené na rozhraní SAX.

Nicméně nevýhodami JDOM jsou pomalejší práce s XML daty a poměrně velké paměťové nároky (právě z důvodů nutnosti vytváření celého stromu objektů). Rozhraní je poměrně nové a použitá verze 1.0 není zcela dokončená. Pro naše účely se však ukázala být zcela dostačující.

4. *Komunikace.* Pro komunikaci mezi jednotlivými zásuvkami v aplikaci byl navržen flexibilní systém, popsaný již v oddíle 3.1 v části nazvané „Rozšíření architektury“. Ten umožňuje mimo jiné vypořádat se s konkrétními požadavky komunikace jednotlivých zásuvek i později během vývoje editoru. Tedy opomenutí či špatný návrh některých prvků konkrétní komunikace nemusí nutně vést k fatálnímu přepracování celého editoru.

Nyní rozvedu konkrétní komunikaci mezi MOL a ostatními zásuvkami.

Načítání/ukládání dat. Editor musí umět jak načíst již vytvořené světy, tak uložit pozměněné či zcela nově vytvořené světy.

V uloženém XML jsou dohromady popisovány jak části světa určené pro zpracování v MOL, tak části o jejichž editaci se stará MPG. Proto existuje v editoru komponenta, popsaná v práci (Juhász, 2008), která má za úkol tyto části rozpoznat a odeslat je ke zpracování do správného modulu.

Obdobně při ukládání vytvořeného světa je zase potřeba data poskytnutá z MOL a MPG „slít“ dohromady. To je opět centrálně v režii speciální komponenty popsané v práci (Juhász, 2008).

Jak načítání, tak ukládání je ze strany MOL realizováno „pasivně“, kdy MOL pouze poskytuje pro tyto účely vhodné služby. Jinými slovy MOL při načítání nic nevyžaduje a při ukládání nikomu nic nevnucuje. Pouze čeká a reaguje na vzniklé události.

Editace géniů. Přestože v rámci projektu IVE je informace o přiřazení génia aktorovi resp. lokaci uchována v popisu enta resp. lokace, rozhodli jsme se ponechat editaci genia do modulu MPG, nikoli do MOL. MPG je totiž zodpovědný za editaci všech aspektů UI v IVE, s čímž koncept géniů nepochybně souvisí.

Aby bylo možné aktorovi nebo lokaci přiřadit génia, musí mít MPG informaci o existenci těchto entit v MOL. MOL tedy tuto informaci zveřejňuje prostřednictvím svých služeb.

Nicméně uživatel může entity vytvořené v MOL jak přejmenovávat, tak i mazat. Pokud k této události dojde, musí být zřejmě sdělena i modulu MPG. Ten musí přijmout náležitá opatření, aby byla zachována konzistence vytvářeného světa (MPG například nemůže přiřazovat génia neexistujícímu aktorovi). MOL tedy MPG informuje o těchto událostech prostřednictvím jeho služeb.

Získávání informací o aktuálním projektu. Editor umožňuje práci uživatele zapouzdřovat do jednotlivých projektů. V rámci projektu uživatel vytváří právě jeden svět a přizpůsobuje prostředí editoru speciálně pro konkrétní projekty. Rozpracované projekty lze ukládat a načítat. Spolu s projektem se načte i nastavení editoru, jaké měl uživatel v době uložení tohoto projektu. Jde o podobný koncept používaný v různých typech editorů, jako například Microsoft Word nebo v moderních IDE pro programátory.

O projekty se obecně stará komponenta, kterou vytvořil Martin Juhász a je popsána v jeho práci (Juhász, 2008). Pro MOL jsou důležité zejména informace o aktuálním umístění adresářových struktur projektu, uložení grafických souborů pro daný projekt, souborů s třídami v javě apod.

Tato komponenta implementující koncept projektů poskytuje ostatním zásuvkám služby, jejichž prostřednictvím se může kterákoli zásuvka, tedy i MOL, ptát na potřebné informace.

3.4. Závěr

V této kapitole jsem popsal návrh architektury celé aplikace, koncept a řešení repositáře objektů v rámci modulu objektů a lokací (pojem modul byl zaveden v oddíle 3.1). Také jsem vyjmenoval stěžejní problémy modulu objektů a lokací a jeho možnosti. Ty jsem pak diskutoval v oddíle 3.3. Za tyto tři prvky editoru (architektura, repositář objektů, modul objektů a lokací) jsem byl od počátku zodpovědný.

Kapitola 4

Testování

V této kapitole popíši náš přístup k testování editoru, o němž v rámci této kapitoly budu mluvit převážně jako o aplikaci. Součástí testování aplikace byl i vznik většího testovacího světa.

Testování probíhalo ve třech hlavních fázích a bylo neoddělitelnou součástí vývoje a samotné implementace aplikace od začátku do konce. Pro lepší orientaci je nazývejme fáze:

i. testování na úrovni specifikace;

ii. syntaktického testování;

iii. sémantického testování.

Toto řazení odpovídá chronologicky průběhu testování aplikace. Mezi první a druhou fází je ostrá hranice, přičemž druhá fáze plynule přešla během vývoje ve fázi třetí. Třetí fáze pak skončila až po uspokojivém chování aplikace.

Nyní přiblížím každou fázi samostatně.

4.1. Testování na úrovni specifikace

Po napsání specifikace bylo zejména nutné ověřit, zda jsou námi požadované rysy implementovatelné vůči stávajícímu IVE. Jinými slovy, zda není specifikováno něco, s čím IVE vůbec nepočítá nebo k čemu přistupuje zcela odlišně. Tato fáze byla velmi důležitá, neboť její opomenutí by mohlo mít ve fázi implementace zcela fatální důsledky.

Tato fáze spočívala především v detailnějším zkoumání IVE a způsobů, jakým funguje. Ručně byly vytvářeny malé světy, konkrétně zaměřené na prvky, jejichž

editaci jsme požadovali ve specifikaci. Tak byly verifikovány specifikované způsoby úpravy světů pomocí námi navrhovaného editoru.

4.2. Syntaktické testování

Pro osvětlení významu této fáze nejprve popíši vstupy a výstupy editoru.

Výstupem je zřejmě hotový svět ve formátu IVE, který uživatel vytvořil nebo jednotlivé repositáře objektů. Formát IVE je především určen XML schématem, který musí popis světa dodržovat.

Vstupy mohou být na té nejvyšší úrovni pohledu dvojího typu. Jednak může být vstupem již vytvořený svět, nebo některá jeho část (např. pouze popis bez obrázků a java souborů, tedy jen XML soubor), a jednak jím můžou být samotné uživatelovy akce v editoru (vytvoření či úprava objektů, lokací, reaktivních plánů).

My jsme v této fázi zejména využili prvně zmíněný druh vstupů, neboť k samotnému IVE byly vytvořeny dva testovací světy. Úkolem tedy bylo testovat schopnost editoru tyto světy načíst do svých vnitřních struktur a dát pokud možno ekvivalentní výstup z pohledu IVE. Tuto fázi jsem nazval syntaktickou, neboť se hledělo zejména na validitu daného výstupu oproti schématům v IVE. Až poté, co tyto světy bez problémů prošly testy, se začalo experimentovat s druhým popsaným typem vstupů. Zde začíná plynulý přechod do třetí fáze testování.

Automatizace testování v této fázi spočívala především v programovém testování validity výstupů oproti XML schématu IVE a propojením editoru s IVE tak, aby se skrze editor dal vytvářený svět přímo pustit do IVE.

4.3. Sémantická fáze testování

V této fázi se zejména používalo výše zmíněných uživatelských vstupů, tedy světy se vytvářely převážně ručně. Ohled byl brán zejména na:

- sémantickou ekvivalenci výstupu – tj. aby výsledný svět odpovídal tomu, co skutečně uživatel v editoru provedl za kroky;

- pohodlí uživatele při editaci.

Je asi zřejmé, že tuto fázi lze automatizovat jen velmi obtížně. Nám se ji bohužel nepovedlo automatizovat vůbec.

Při testování sémantické ekvivalence se postupovalo stylem od jednoduššího ke složitějšímu, takže byly do testování nejprve zahrnuty jen některé součásti světů, jako například pouze hierarchie lokací, pouze objekty bez grafiky apod. Postupně se přecházelo k úplnějším výstupům, jejichž součástí byla mimo jiné jak grafika, tak i části světa popisované v javě.

Pohodlí uživatele při editaci bylo testováno v závěru této fáze, kdy byl vytvářen přiložený testovací svět, který je popisován v následujícím oddíle. Tato část testování se ukázala být alespoň stejně tak důležitou, jako testování již na úrovni specifikace. Zde se totiž častokrát prokázalo, že některé implementované postupy editování jsou pro rozsáhlé světy uživatelsky zcela nevyhovující. Například často opakovaná činnost při editaci vyžaduje neúnosně mnoho dílčích úkonů (např. klepnutí myší), chybí volby pro hromadné zopakování některé činnosti (např. zkopírování více entit najednou), orientace v rozsáhlejších světech je obtížná apod.

Nejpalčivější problémy tohoto typu byly odstraněny, bohužel zdaleka ne všechny. To je v této fázi projektu již velmi obtížné.

4.4. Testovací svět

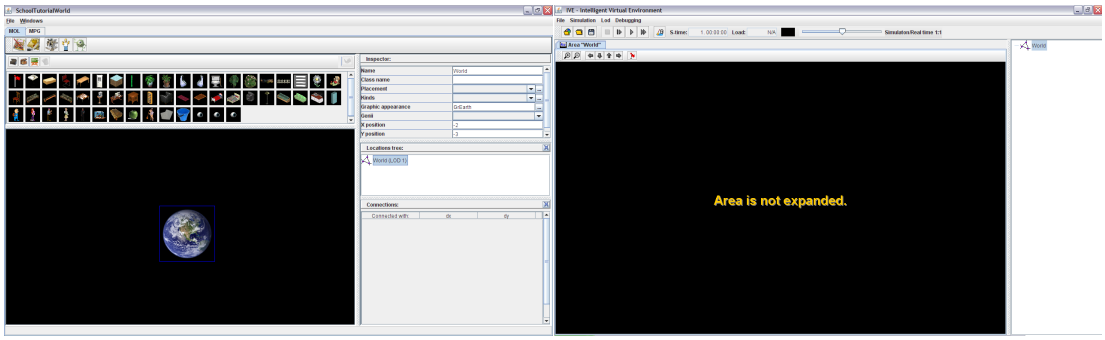
Jak již bylo zmíněno v předchozím oddíle, součástí závěrečné fáze testování bylo vytvoření vlastního světa pomocí našeho editoru.

Náš svět rozlišuje pět úrovní LOD a je obýván devětadvaceti v-lidmi. Simulace představuje zjednodušený život školáků a učitelů ve městě. Dvacet čtyři v-lidí představují školou povinní žáci, z toho dvanáct je dívek a dvanáct chlapců. Čtyři obyvatele světa tvoří pedagogický sbor, který čítá dva učitele a dvě učitelky. Posledním v-člověkem v simulaci je pan ředitel.

Topologie světa

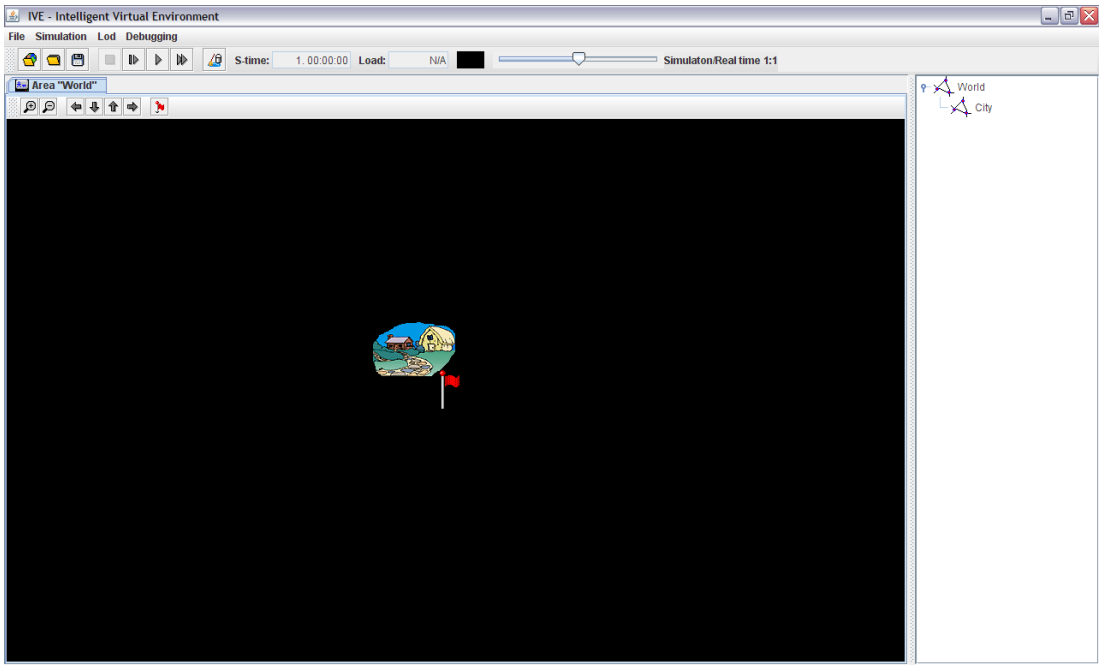
Postupně bude po jednotlivých úrovních LOD popsána topologie testovacího světa, ve kterém jsou lokace a všechny ostatní věci pojmenovány v angličtině. Proto je v závorkách u českého popisu lokací uváděn název v angličtině v takové podobě, v jaké se daná lokace doopravdy v testovacím světě nazývá.

LOD 1. Na této úrovni je vidět svět jako celek. Vzhled lokace na této úrovni není vůbec podstatný, neboť samotné IVE ji nezobrazuje.

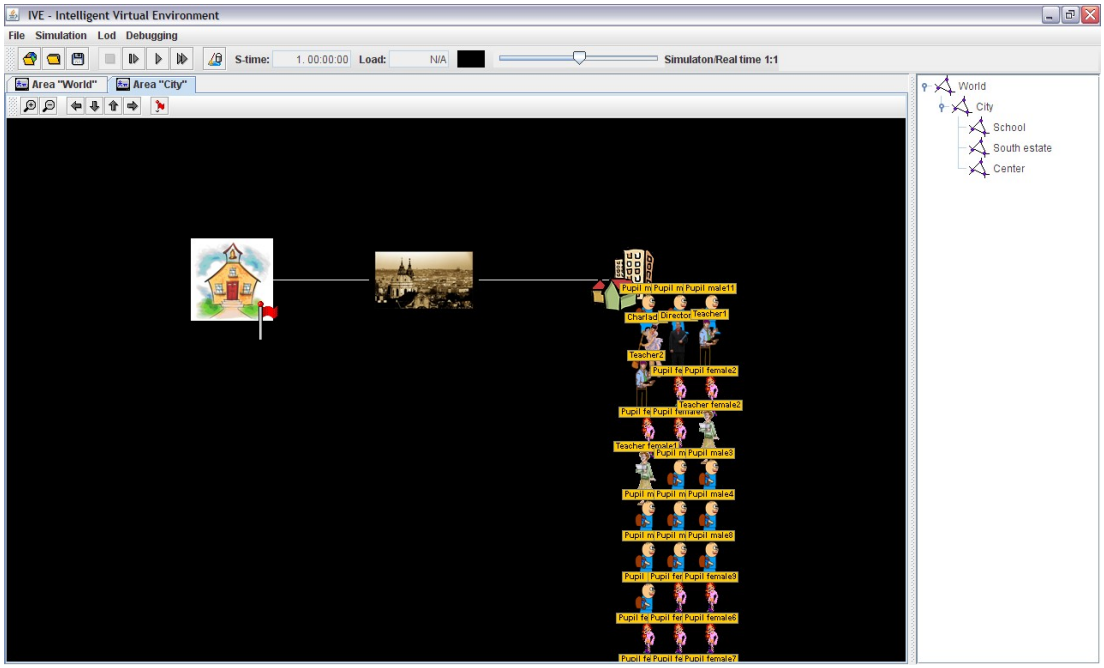


Vlevo vzhled světa na LOD1 v editoru; vpravo je vidět, že lokaci na LOD1 nelze v IVE zobrazit.

LOD 2. V našem světě je pouze jedno město.



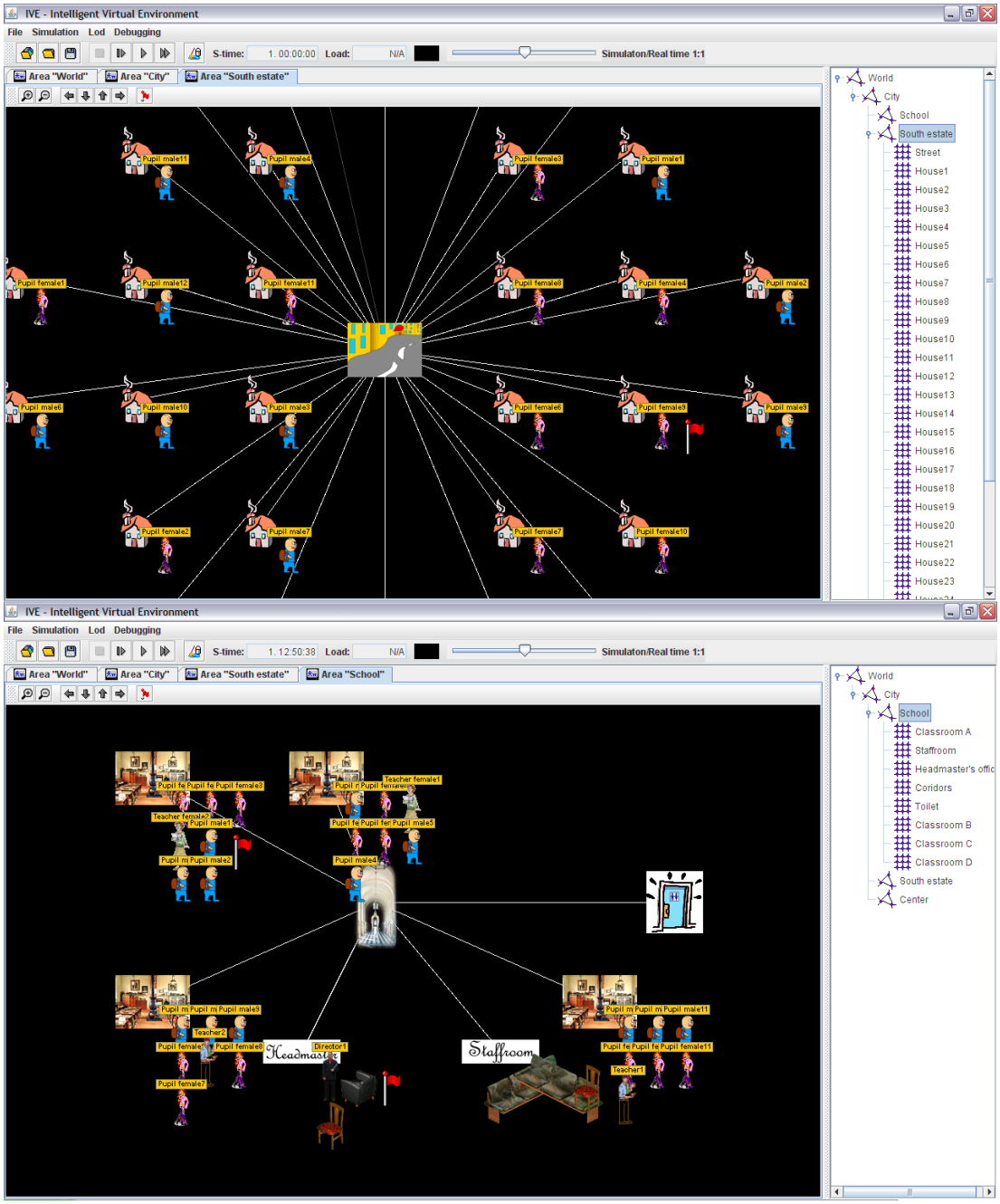
LOD 3. To je členěno na tři místa. Centrum, školu a sídliště (Centrum, School, South estate). Aby se aktoři dostali ze sídliště do školy, musí projít centrem.



LOD 4. Ve skutečnosti je škola při bližším pohledu umístěna v severní části centra (North) a do sídliště na jižním městě se lze přirozeně dostat z jižní strany centra (South).

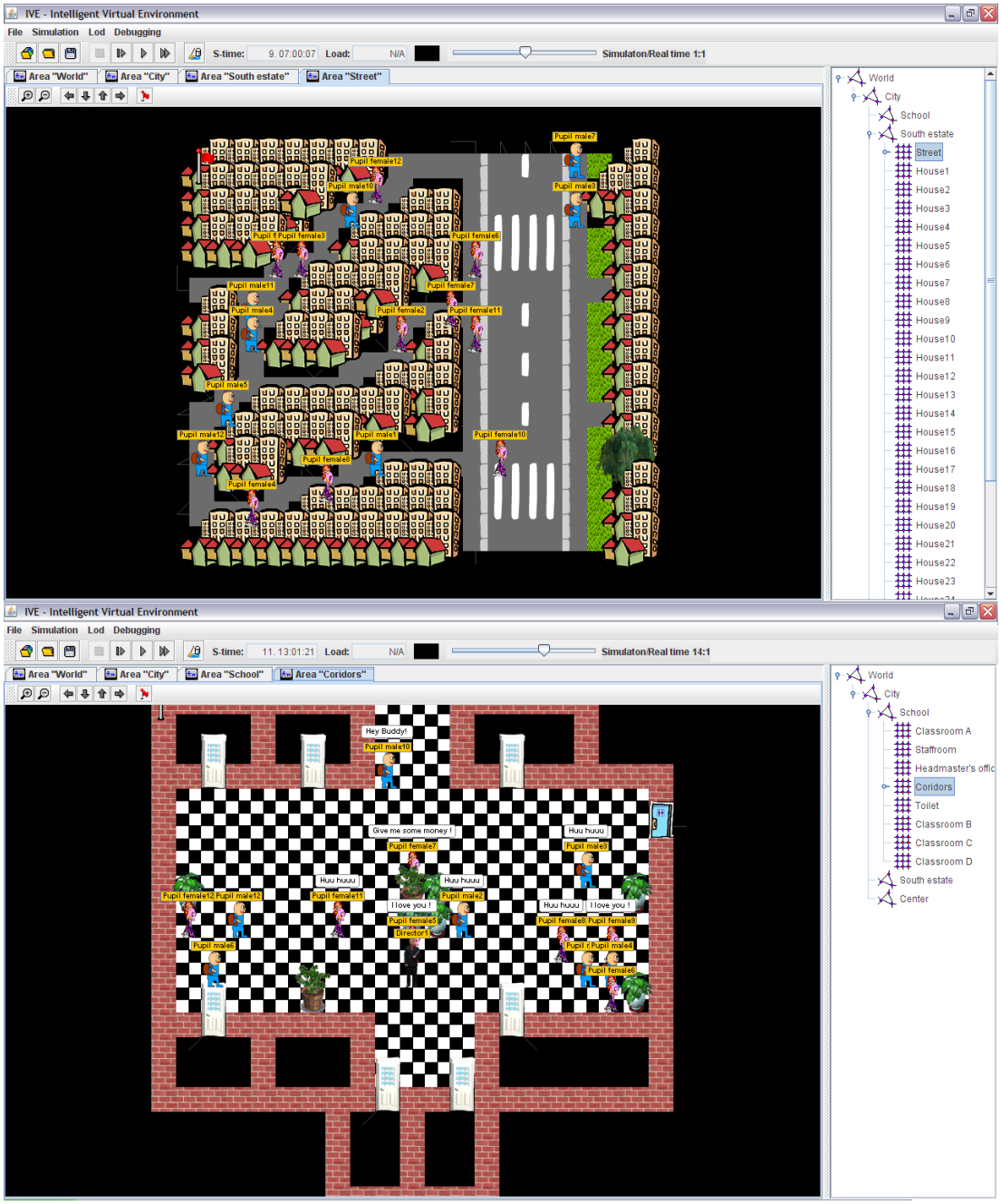
Samotné sídliště představuje jedna ulice (Street), ze které se dá vejít do jednotlivých devětadvaceti domů (House1, … House29), ve kterých bydlí obyvatelé světa. Každý jeden aktor bydlí ve svém vlastním bytě.

Škola má ve svém centru systém chodeb (Coridors), ve kterém se nacházejí vstupy a výstupy do jednotlivých místností a také ven ze školy. Těmito místnostmi jsou čtyři třídy rozlišené písmeny A až D (Classroom A, … Classroom D), ředitelna (Headmaster's office), sborovna (Staffroom) a také toalety (Toilet).



Nahoře je vidět sídliště na jižním městě; dole je detailnější pohled na školu.

LOD 5. Na nejvyšší úrovni detailů jsou pak podrobně vyobrazeny lokace uvedené na LODu 4.



Nahoře je vidět ulice na LODu 5 a žáci, kteří zrovna vychází z domů na cestu do školy; dole je zobrazena chodba ve škole během přestávky.

Život aktorů

Obyvatelé testovacího světa mají pevně daný rozvrh činností, které vykonávají během dne. Tento rozvrh je stejný pro stejné typy aktorů (studenti, učitelé, žáci, ředitel) a během jednotlivých dní se nemění.

Studenti vykonávají následující činnosti:

- v 7:00 vstávají z postelí a vyrážejí do školy;
- cestují přes ulici na jižním městě, přes jižní a severní město až do školy;
- do 8:00 tráví čas na chodbě;
- 8:00 zasedají do lavic ve svých třídách k první vyučovací hodině;
- Během hodin:
    - dávají pozor a odpovídají na učitelovy otázky z oblasti matematiky, zeměpisu, českého jazyka a angličtiny;
    - nedávají pozor (vykřikují nebo nic nedělají a jen sedí).
- o přestávkách pobíhají po chodbě a neustále pokřikují;
- žáci se učí 4 vyučovací hodiny a v 15:00 opouštějí školu a cestují domů;
- doma tráví zbytek dne hraním her na počítači, odpočinkem na posteli, četbou knih, sledováním televize nebo konzumováním ovoce z ledničky;
- ve 22:00 uléhají do postele a usínají.

Učitelé vykonávají následující činnosti:

- vstávají v 6:00 a vyráží do práce;
- Před začátkem vyučování tráví čas ve sborovně kde:
    - pracují na počítači;
    - vysedávají na pohovce;
    - studují u knihovny.
- v 8:00 odchází na svou první hodinu, každý podle svého rozvrhu do příslušné třídy, kde vyučuje předmět podle své aprobace;
- při výuce chodí po třídě, kladou žákům otázky a vykládají;
- přestávky tráví ve sborovně;
- každý učitel učí 4 hodiny;

- po skončení poslední hodiny v 15:00 tráví ještě hodinu ve sborovně a pak odcházejí domů;
- zbytek dne tráví stejně jako studenti;
- ve 22:00 uléhají ke spánku.

Pan ředitel vykonává během dne následující činnosti:

- vstává v 6:00 a vyráží do práce;
- pracuje od 7:00 do 16:00;
- V práci:
  - úřaduje v ředitelně ve svém luxusním křesle;
  - zalévá květiny;
  - stará se o rybičky v akváriu;
  - pracuje na počítači v ředitelně;
  - během vyučování občas zajde na inspekci;
  - v odpoledních hodinách se prochází po chodbách.

4.5. Závěr

V této kapitole jsem popsal náš přístup k testování aplikace. Logicky a chronologicky zároveň jsem rozdělil testování na tři fáze. Dále jsem také zdůraznil neoddělitelnost testování od celého vývoje aplikace, bez nějž by se samotná implementace neobešla. Testování začalo specifikací a skončilo až prohlášením aplikace za dostatečně funkční. Také jsem zmínil částečné využití automatického testování ve specifické fázi vývoje aplikace.

Kapitola 5

Rešerše v oblasti nástrojů pro prototypování algoritmů umělé inteligence

IVE je framework a middleware, který umožňuje vývoj inteligentních virtuálních agentů (IVA). Náš IVE Editor je vývojovým nástrojem pro tento framework, který zpřístupňuje vývoj IVA i uživatelům z jiných než informatických oborů, kteří nemají zkušenosti s programováním a obecně vývoj IVA usnadňuje a zrychluje. Jedním z obecných cílů projektu IVE je poskytnout umělou inteligenci (UI) jako nástroj oblastem výzkumu, zábavy a výuky. V této rešerši se tedy zaměřím na obdobné nástroje, které usnadňují nebo umožňují vývoj IVA pro nejrůznější oblasti použití.

Vzhledem ke skutečnosti, že reaktivní plánování v IVE je rozšířením (Brom ad., 2006c) plánovací architektury POSH, představím nejdříve tuto architekturu a nástroje, které jsou na ní založeny. Všechny tyto nástroje jsou vyvíjeny na akademické půdě a jsou volně dostupné. To je jejich hlavní výhodou. Navíc jejich obecné cíle jsou v zásadě podobné cílům IVE, zmíněným v předcházejícím odstavci, a proto jsou frameworku IVE zcela jistě nejblíže.

V další části nastíním možnost využití komerční hry Civilization IV pro účely akademické sféry.

Dále je představen projekt ScriptEase, který také vznikl na akademické půdě, a který ve spolupráci s komerční sférou dal vznik nástroji pro vývoj UI do počítačové hry Neverwinter Nights.

Nakonec jsou v krátkosti zmíněny některé komerční nástroje.

Akademické nástroje založené na POSH architektuře

POSH („Parallel-rooted, Ordered Slip-stack Hierarchy“) architektura byla vyvinuta J. J. Bryson z University of Bath (Bryson a) pro vývoj řídících algoritmů autonomních agentů.

Tato symbolická reaktivní architektura je založena na paradigmatu BOD („Behavior Oriented Design“) (Bryson, 2001) (Bryson ad., 2001), který především umožňuje vývoj jednoduchých, srozumitelných a snadno udržovatelných agentů. Systémy vyvinuté na základě BOD (Bryson b) jsou složeny z tzv. behaviorálních modulů popisujících akce, vnímání a učení agentů. Moduly popisují *jak* bude agent vykonávat jednotlivé činnosti (jíst, spát, lovit) v daném prostředí a *jak* bude agent prostředí vnímat. POSH plány pak určují, *kdy* bude agent tyto činnosti vykonávat.

POSH plány jsou zapisovány v jazyce podobném jazyku Lisp a moduly v nějakém objektově orientovaném jazyce (v současných implementacích jsou jimi Python, nebo Jython). Moduly mohou být vykonávány pseudoparalelně. Tedy např. agent může každých pět vteřin vstát a rozhlédnout se kolem sebe, zatímco se snaží rozdělat oheň.

```
(
 ;AP - Action Pattern, akce v sekvenci jsou vykonávané jedna za
druhou
 (AP zabec-a-otoc-se (seconds 1) (zabec otoc-se) )

 ;C – Competence, slouží pro zápis BOD chování
 (C hledej-potravu (seconds 1) (goal((ovce-je-nazrana)) )
  (elements
   (
    (zer-travu      (trigger((vidis-travu)))    zer-travu)
    (hledej-travu   (trigger((nevidis-travu))) hledej-travu)
   )
  )
 )

;RDC - Realtime Drive Collection - základní struktura POSH plánu
 (RDC ovci-zivot (goal ((fail))) ;cíl není nikdy splněn
  (drives
    (( utek-pred-predatorem   ( trigger ((ovce-vidi-vlka)) )   utikej ))
    (( neco-do-ovce-vrazilo   ( trigger ((ovce-citi-naraz)) )  zabec-a-
otoc-se ))
    (( ziskavani-potravy      ( trigger ((ovce-ma-hlad)) )     hledej-
potravu ))
    (( zkoumani-okoli         ( trigger ((succeed)) )          behej-kolem
))
    ;succeed reprezentuje vždy splněný předpoklad
   )
 )
)
```

Obr 1: Příklad části POSH plánu pro ovci, uvedený v článku (Bída ad., 2006).

POSH byl nejprve implementován v jazyce Python (pyPOSH) a později portován do jazyka Jython (jyPOSH), aby byla možná spolupráce s knihovnami v Javě. Dále v současné době existuje POSH IDE, nazvané ABODE, a dvě různé platformy pro prototypování algoritmů řídících autonomní agenty. Jsou jimi BOD/MASON a Pogamut. Tyto nástroje budou přiblíženy v následujících odstavcích.

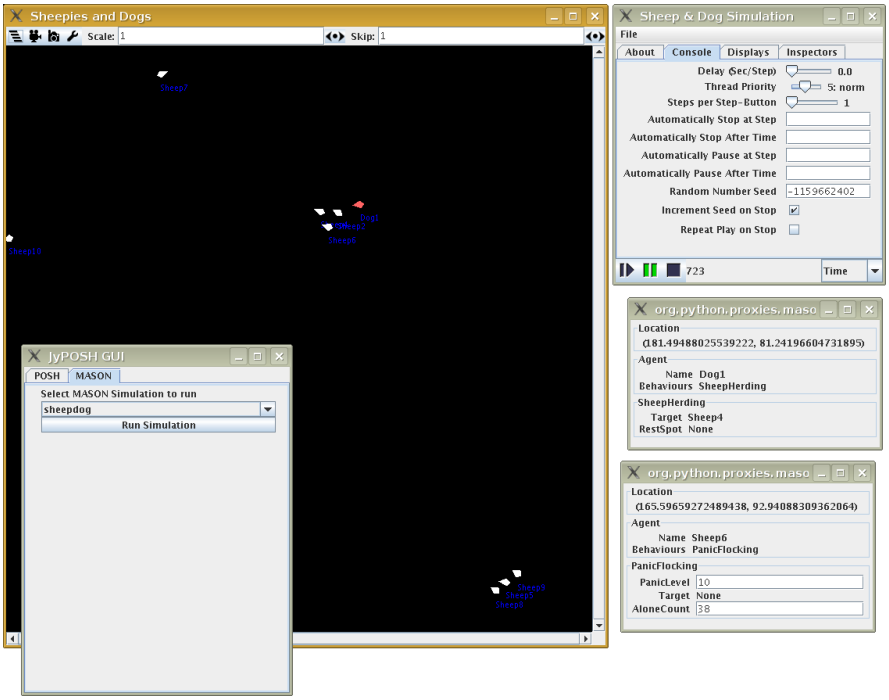
PyPOSH a BOD/MASON

PyPOSH je mechanismem, který interpretuje POSH plány a na jejich základě předává řízení behaviorálním modulům popsaným výše. Jejich implementace zřejmě závisí na prostředí, ve kterém má být agent nasazen.

Vizualizace prostředí, ve kterém se agenti pohybují je zcela zásadní pro efektivní vývoj IVA. Umožňuje sledovat jak vzájemnou interakci agentů v prostředí, tak např. i výsledek vykonávání paralelních cílů jednoho agenta. V neposlední řadě může značně napomoci ladění programů pro řízení agentů.

BOD/MASON (Bryson ad., 2005) této platformě poskytuje právě virtuální prostředí. Vznikl integrací aplikace pyPOSH s knihovnou MASON (Luke ad., 2004), která poskytuje jádro pro diskrétní multiagentní simulace. Vhodným jazykem pro implementaci behaviorálních modulů v prostředí BOD/MASON je Jython, protože samotná knihovna MASON je implementována v Javě.

Celkově jde o velmi slibnou aplikaci pro modelování IVA, která je distribuována i s demoverzí, na které lze snadno pochopit fungování celého systému. Navíc pro seriózní vývoj v tomto prostředí je dobře popsána možnost integrace s Eclipse IDE a pro editaci POSH plánů lze využít grafický editor ABODE, o kterém se píše níže. Tato platforma je spolu s podrobnými informacemi volně dostupná na internetu (BOD/MASON).



Obr 2: Ukázka prostředí BOD/MASON zobrazující sheep/dog demo.

Pogamut

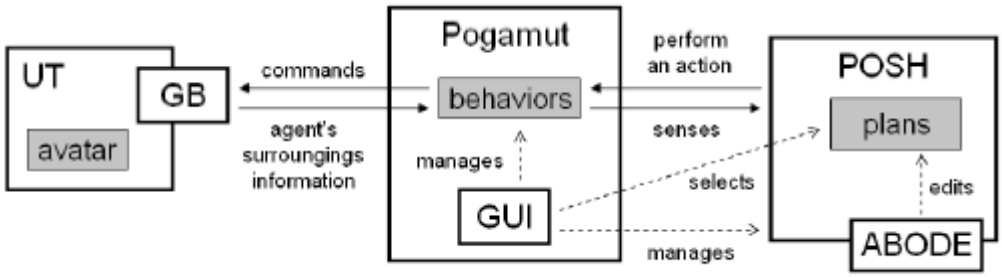
Pogamut (Bída ad., 2006) je middleware vyvíjený ve spolupráci MFF UK v Praze s University of Bath ve Velké Británii. Kombinuje pyPOSH se specifickým vizuálním prostředím, kterým je komerční hra Unreal Tournament (UT).

Kód této hry byl před několika lety částečně otevřen a tím zpřístupněn i akademické komunitě. Projekt naplňuje současný trend využití prostředí počítačových her pro seriózní výzkum umělé inteligence.

Pogamut umožňuje prototypování logigy botů, tedy postav ve hře typu „first person shooter“ řízených počítačem, pomocí hierarchického reaktivního plánování (POSH plánů). To je vhodným přístupem především z důvodu kladených nároků na rychlost rozhodování bota v real-time prostředí.

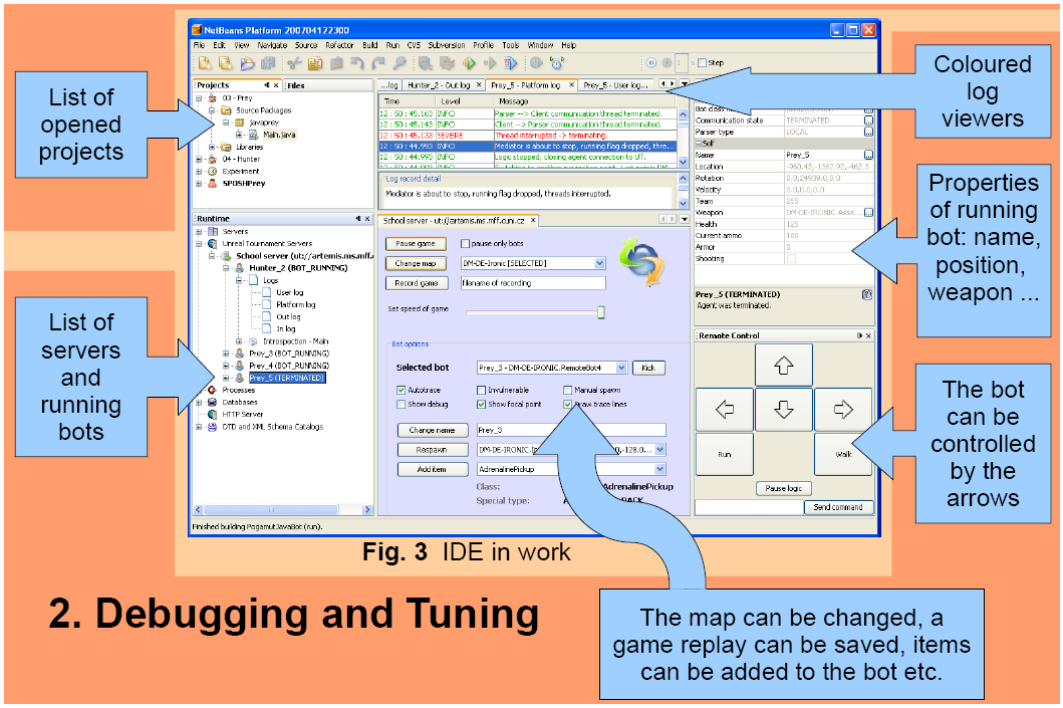
Propojení Pogamutu s UT je realizováno skrze server Game Bots. Ten je napsán v nativním jazyce UT. Tento server předává klientovi zprávy, které jsou generovány serverem hry a v UT vyvolávaly události, které původně řídily chování botů. Díky

použití UT jako virtuálního prostředí získává tato platforma také možnost vytvářet si vlastní mapy dle potřeby. Na Obr 3 je zobrazena architektura celého frameworku pro prototypování logigy botů i se zapojením zmíněného editoru POSH plánů ABODE. Tento obrázek je převzatý z článku (Brom ad., 2006c).



Obr 3: Architektura frameworku pro prototypování logigy botů.

V současnosti již existuje nová verze frameworku Pogamut2 (Kadlec ad., 2007), která se od předchozí verze odlišuje především novým flexibilním IDE, které je vytvořeno jako plug-in do známého prostředí Netbeans IDE. Framework umožňuje vytvářet celkem tři typy botů, které se odlišují jazykem, v jakém je napsána jejich logiga. Těmi mohou být Java, Python nebo již zmíněný POSH, resp. jeho dialekt SPOSH vytvořený speciálně pro účely Pogamutu.



Obr 4: Vývojové prostředí frameworku Pogamut2; obrázek je výřezem z plakátu vytvořeného autory projektu.

Tato volně dostupná (Pogamut2) platforma pro rychlé prototypování IVA má velmi dobrou podporu ze strany autorů: existují podrobné návody i instruktážní videa a je tak velmi snadné začít tuto platformu používat pro vlastní projekty. Jediná její součást, která není volně dostupná, je hra Unreal Tournament 2004, nicméně tu lze dnes pořídit za přijatelnou cenu okolo čtyř set korun.

Obr 5: Ukázka ze hry UT spolu s ladícími nástroji middlewaru GameBots 2004; obrázek je výřezem z plakátu vytvořeného autory projektu.

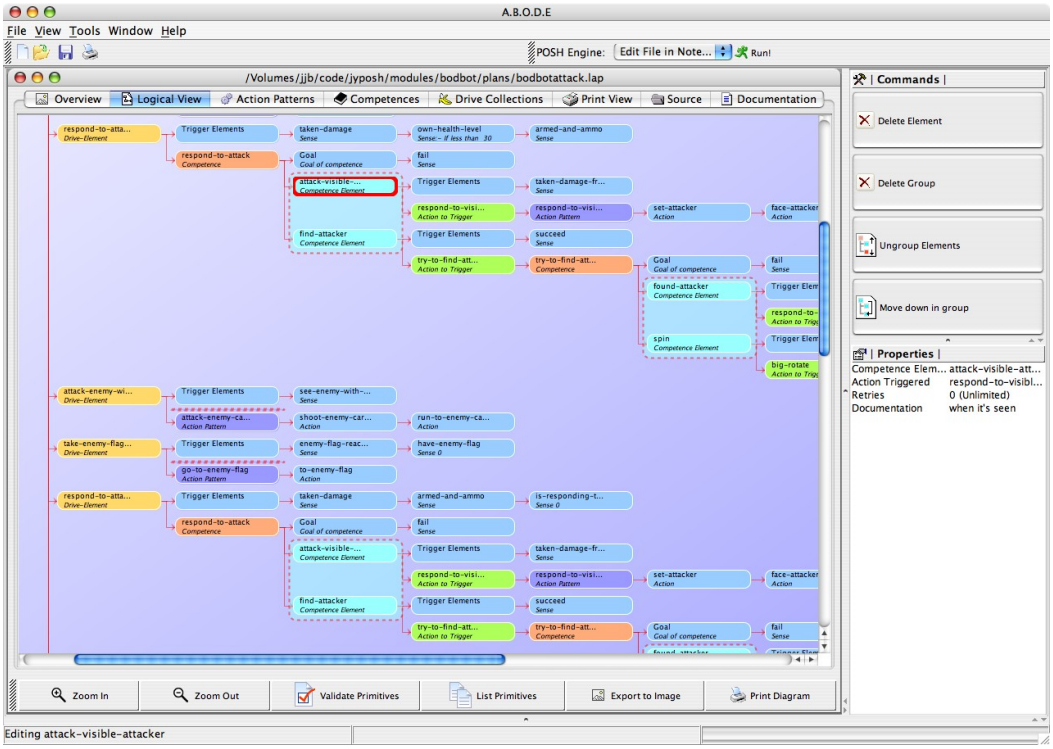
A.B.O.D.E. IDE

A.B.O.D.E. (Advanced Behavior Oriented Design Environment) (ABODE) je grafické vývojové prostředí, které značně usnadňuje a zpřehledňuje tvorbu POSH plánů pro řízení inteligentních agentů. ABODE v současnosti vyvíjí Steve Gray z Cobalt Software pod vedení J. Bryson.

ABODE by se dalo přirovnat k modulu IVE Editoru, ve kterém se vytvářejí reaktivní plány pro řízení agentů v IVE a který implementoval Martin Juhász (Juhász, 2008). ABODE umožňuje uživateli plány editovat v grafickém režimu a zároveň nahlédnout do vygenerovaného kódu jazyka architektury POSH. Podobně je tomu tak i v IVE Editoru, kde vygenerovaným kódem je zápis procesů a cílů v jazyce XML.

Podstatným rozdílem mezi IVE Editorem a ABODE je však skutečnost, že v IVE Editoru se přímo popisují také atomické akce, tedy chování postav ve virtuálním

prostředí, kdežto v ABODE jsou popisovány pouze reaktivní plány. Behaviorální moduly, tedy chování aktorů (atomické akce), se již v ABODE needitují. Tento rozdíl je způsoben především tím, že IVE jako framework obsahuje i zabudované virtuální prostředí. Behaviorální moduly však závisejí na prostředí, ve kterém se agenti budou pohybovat. Dalším rozdílem je pak přehlednější grafické provedení plánů v ABODE, na rozdíl od IVE Editoru.



Obr 6: ABODE IDE.

Otevřená otázka využití hry Civilization IV

Civilization IV (Civilization IV) je strategická komerční hra z roku 2005, ke které bylo roku 2006 vydáno SDK, obsahující zdrojové kódy jádra hry. K dispozici je editor map a navíc lze prostředí hry snadno upravovat pomocí XML a všudypřítomného skriptovacího jazyka Python. Podle samotných autorů je hra nyní libovolně škálovatelná včetně UI.

Nabízí se otázka, zda by nebyla hra Civilization IV dalším vhodným virtuálním prostředím, které by mohlo posloužit akademické sféře pro seriózní studium UI.

Stejně jako je ve zmíněném projektu Pogamut využíván middleware GameBots pro spojení s hrou Unreal Tournament, mohl by zde být implementován middleware, který by umožňoval propojení prostředí hry s nástroji pro prototypování UI.

Další možností by mohla být integrace aplikace pyPOSH. Vzhledem k přítomnosti skriptovacího jazyka Python by snad i bylo možné pro prostředí Civilization IV implementovat nějaké behaviorální moduly (viz výše), které by určovaly chování agentů.

Sám jsem blíže nezkoumal možná omezení hry Civilization IV ani nástrojů, které jsou k ní k dispozici. Není mi ani znám způsob implementace UI ve hře samotné. Podrobnější průzkum na tomto poli je mimo rozsah této práce.

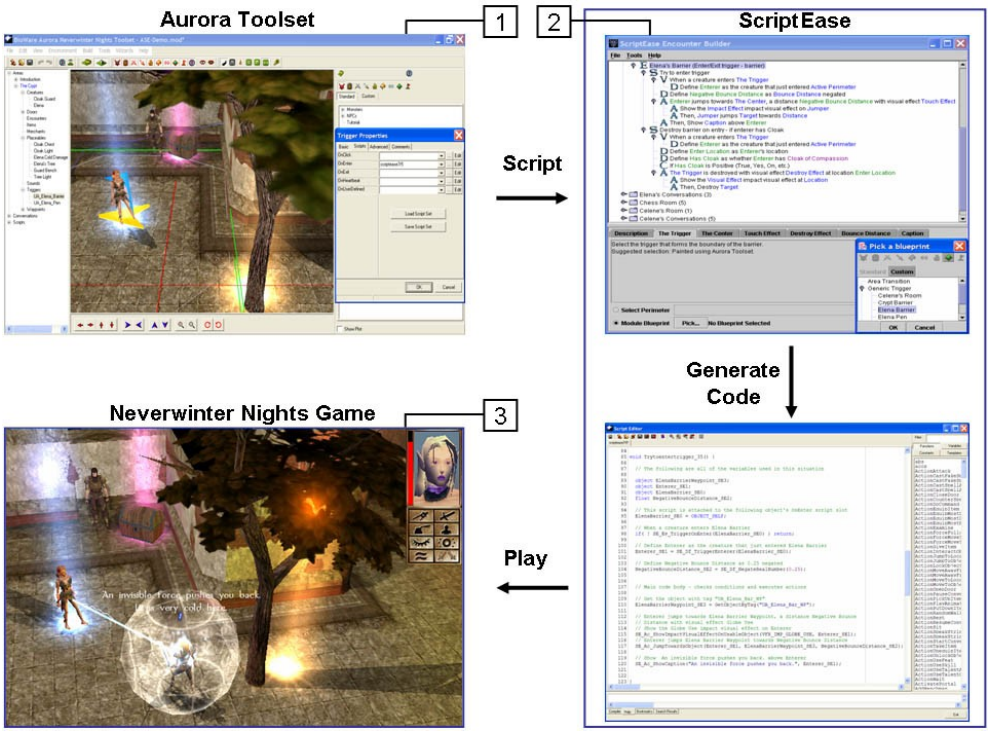
ScriptEase

Projekt ScriptEase (Cutumisu ad., 2007) je projekt vedený na kanadské University of Alberta v Edmontonu. Projekt je zaměřen na usnadnění prototypování UI do počítačových RPG (role-playing game) her. Cílem je umožnit autorům herních příběhů implementovat svůj příběh do hry, aniž by byli zběhlí v konkrétním programovacím jazyce, ve kterém je hra implmentována.

Model ScriptEase umožňuje designérům rychle budovat komplexní chování využíváním a upravováním předdefinovaných šablon chování. Takto vytvořený model je pak převeden do skriptovacího jazyka dané hry. Předdefinované šablony jsou vytvořeny jako vzory pro chování NPC (postav řízených ve hře počítačem) pro situace, které se v dané hře neustále opakují.

V rámci výzkumu autoři implementovali ScriptEase pro využití v komerční hře Neverwinter Nigths firmy BioWare (Neverwinter Nights). ScriptEase se váže k použití vývojového nástroje Aurora, který v podstatě umožňuje vytvořit si vlastní hru založenou na enginu hry původní.

Tento nástroj, ač vyvíjený na akademické půdě, není z důvodu spolupráce s komerční sférou volně dostupný. Nicméně je součástí distribuce samotné hry, jejíž cena není příliš vysoká. Neumožňuje však tak flexibilní prototypování algoritmů UI jako dříve zmíněné nástroje.



Obr 7: ScriptEase v použití s vývojovým nástrojem Aurora pro komerční hru Neverwinter Nights.

Komerční nástroje

Dnes existuje celá řada nástrojů, které usnadňují vývojářům počítačových her vývoj UI. Tyto nástroje umožňují rychlé prototypování chování virtuálních agentů a vývojáři se tak mohou soustředit na ostatní aspekty hry. To samozřejmě zvyšuje kvalitu vyvíjených her. Jednak díky ušetřenému času a jednak proto, že tyto nástroje pomalu vnášejí řád a standardizaci na pole implementace UI. Vývojáři her tak nemusí začínat v každém projektu od začátku a implementovat znovu stále stejné věci.

Tyto nástroje umějí efektivně řešit celou řadu problémů. Zvládají řídit například i obrovské davové scény s desítkami tisíců aktorů, kteří se pohybují rozlehlým 3D prostorem zcela uvěřitelně. Nové přístupy v oblasti pathfindingu umožňují i inteligentní chování aktorů v dynamicky se měnícím prostředí. Posouvají významně hranice uvěřitelnosti UI nejen v hrách, ale také ve filmovém průmyslu, kde se s úspěchem využívají především při davových scénách.

Bohužel tyto nástroje jsou zcela nevhodné pro výukové účely a to hned z několika důvodů. Jejich značnou nevýhodou je především příliš vysoká cena. Dalším důvodem jsou jejich požadavky na znalosti mnoha pokročilých algoritmů UI, které implementují. V neposlední řadě bývají tyto nástroje ve formě middlewaru a implementace jejich připojení k nějakému virtuálnímu prostředí nebývá zdaleka triviální. Z těchto důvodů jsou komerční nástroje nevhodné pro využití akademickou komunitou.

Známými a nejvíce používanými zástupci těchto nástrojů jsou např. AI.implant (AI.implant), nástroj XSI pro kompletní 3D modelování firmy Softimage (Softimage) se systémem pro simulaci chování davů či řada produktů firmy Xaitment (Xaitment).

Kapitola 6

Diskuse

Během práce na vlastním testovacím světě za použití editoru se ukázalo být psaní kódu v Javě, kterým se částečně popisuje prostředí v IVE a chování aktorů, značně omezující. Náš editor na příslušných místech sice umí otevřít externí editor pro vložení kódu v Javě a v příslušném kódu také umí po kompilaci světa zobrazit nalezené chyby kompilátorem Javy, nicméně toto není pro uživatele dostačující k napsání větších a složitějších kusů kódu pro rozsáhlejší světy.

Pro efektivní práci bylo nutné mít k dispozici zdrojové kódy samotného IVE a některé z moderních integrovaných vývojových prostředí (IDE) pro jazyk Java. Potřebné třídy se pak dopisovaly přímo do IVE, čímž jsme díky IDE získaly snadný přehled o funkcích IVE, které jsou nezbytné pro psaní našeho kódu a stejně tak jeho okamžitou kontrolu. Výsledné zdrojové kódy tříd pak bylo nutné pouze nakopírovat zpět do editoru, aby se zakomponovaly do výsledného světa.

Práce na uvedené rešerši mi pomohla vyjasnit si pozici IVE mezi podobnými projekty a zejména pak uvědomit si, jaký je nedostatek obdobných nástrojů na poli prototypování algoritmů pro řízení virtuálních postav pro účely výuky UI.

6.1. Budoucí práce

Editor je možné v budoucnu rozšířit o následující části:

- Integrovaný editor javovských tříd, který by měl přehled o funkcích IVE použitelných ve vytvářených kusech kódu a který by řešil stávající omezení popsané na začátku této kapitoly;
- Nový modul pro pohodlné, efektivní a snadné propojování lokací. Propojování lokací je v současném editoru vyřešeno značně nešikovně.

Lokace lze v editoru propojit pouze na úrovni way-pointů, přičemž na nižších úrovních LOD se poté lokace propojí automaticky. Toto propojení se ve větších světech stává značně nepřehledným, propojování většího množství lokací zase velmi pracným.

- Editor umožňuje importovat a exportovat objekty do repositáře objektů. Neumí však pracovat s repositářem lokací.

- Lokace se dají kopírovat na stejné úrovni LOD. Nelze však kopírovat lokace mezi různými úrovněmi LOD.

- Zlepšení komfortu uživatele; Během testování se ukázalo, že některé akce v editoru vyžadují zbytečně mnoho klepnutí myší nebo že se věci needitují dostatečně pohodlně a přehledně. Například pro vytvoření deseti kopií jedné lokace musí uživatel desetkrát uplatnit metodu „copy & paste“ bez možnosti hromadného kopírování. Jiným příkladem je nedostatečný přehled v tzv. třídách objektů (objectClasses) a chybí nástroje pro jejich pohodlnou úpravu.

- IVE umožňuje nastavovat plynulost zjednodušování simulace (Kubr ad., 2006a, str. 17-18) (Kubr ad., 2006b, str. 25). S tímto atributem „influences“ editor pracovat neumí.

- XML popis IVE světa vystupuje ze současné verze editoru ve značně nafouklé podobě. IVE Editor nyní implementuje algoritmus, který stejné lokace sdružuje do jedné šablony popisující tyto lokace, čímž výrazně stlačuje objem XML dat rozsáhlých světů. Nicméně XML reprezentace grid lokací je stále zbytečně redundantní. Je potřeba implementovat algoritmus, který by rozumně vyhledával čtyřúhelníková pole naprosto stejných way-pointů a popisoval je pouze jedním elementem.

- IVE Editor nezohledňuje potřeby současného rozšíření IVE o HTN plánování (Holeček, 2008), drama manager (Abonyi, 2008) (Baláš, 2008) a epizodickou paměť aktorů (Reidinger, 2007). Editor by mohl být rozšířen o nové moduly umožňující vytváření světů pro IVE s uvedenými rozšířeními.

Kapitola 7

Závěr

Cílem práce bylo rozšířit framework IVE o samostatný editor. Ve spolupráci s Martinem Juhászem jsme vytvořili komplexní nástroj pro prototypování umělé inteligence, který je navíc rozšiřitelný o libovolné další nástroje díky své „zásuvkové“ architektuře.

Pro účely testování jsme v našem editoru pro IVE vytvořili vlastní simulaci a v rámci uživatelské příručky jsme uživateli vytvořili tutoriál, který tento svět rozšiřuje za pomoci našeho editoru o jednu novou postavu. Byl napsán i druhý kratší tutorial, který provádí po vytvoření stejného světa, jaký vznikne podle původního tutorialu k frameworku IVE.

IVE Editor by měl být krůčkem k přiblížení frameworku IVE především studentům oborů bez výuky programování a tedy propagací UI jako nástroje pro výzkum v neinformatických oblastech.

Literatura

(ABODE) A.B.O.D.E.,
homepage: http://www.cs.bath.ac.uk/~jjb/web/BOD/abode.html [28.3.2008]

(Abonyi, 2008) Abonyi, A.: *Řízení příběhu pomocí Petriho sítí – drama manager.* Bakalářská práce, UK MFF Praha (2008)

(AI.implant) AI.implamnt,
homepage: http://www.ai-implant.com/ [28.3. 2008]

(Baláš, 2008) Baláš, D.: *Řízení příběhu pomocí Petriho sítí – integrace drama manageru do IVE*. Bakalářská práce, UK MFF Praha (2008)

(Bída ad., 2006) Bída, M., Burket, O., Brom, C., Gemrot, J.: *Pogamut – Platforma pro prototypování botů v Unreal Tournamentu* In: Jozef Kelemen, Vladimír Kvasnička (eds.): Sborník příspěvků z konference Kognice a umělý život. Slezská Universita v Opavě, Třešť (2006) 67-74 (in Czech)

(BOD/MASON) BOD/MASON,
homepage: http://www.cs.bath.ac.uk/~jjb/web/BOD/BOD-MASON.html
[28.3.2008]

(Brom ad., 2005/2006) Brom, C., Lukavský, J., Šerý, O.: *Affordances, level-of-detail AI, and role-passing for virtual humans*. Technical Report 2005/6. Department of Software and Computer Science Education, Charles University in Prague, Czech Republic.

(Brom, 2006) Brom, C.: *Řízení virtuálních lidí ve velkých virtuálních světech*. Disertační práce, UK MFF Praha (2006).

(Brom ad., 2006a) Brom, C., Lukavský, J.: *Proč je hranice mezi virtuální bytostí a jejím světem neostrá*. In: Sborník příspěvků z konference Kognice a umělý život

(Brom ad., 2006b) Brom, C., Lukavský, J., Šerý, O., Poch, T., Šafrata, P.: *Affordances and level-of-detail AI for virtual humans*. In: Proceedings of Game Set and Match 2, The Netherlands, Delft (2006)

(Brom ad., 2006c) Brom, C., Gemrot, J., Bída, M., Burket, O., Partington S. J., Bryson, J.: *POSH Tools for Game Agent Development by Students and Non-Programmers*. In: *CGAMES IEEE* Dublin (2006)

(Bryson a) Joanna J. Bryson, homepage:
http://www.cs.bath.ac.uk/~jjb/ [28.3.2008]

(Bryson b) Bryson. J.: *How to Make a Monkey Do Something Smart*. University of Bath, http://www.cs.bath.ac.uk/~jjb/web/how-to-monkey.pdf [28.3.2008]

(Bryson, 2001) Bryson, J.: *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents*. PhD thesis, Massachusetts Institute of Technology (2001)

(Bryson ad., 2001) Bryson, J., Stein, A.L.: *Modularity and Design in Reactive Intelligence*. In: Proceedings of IJCAI'01 (2001)

(Bryson ad., 2005) J. J. Bryson, T. J. Caulfield, and J. Drugowitsch: *Integrating life-like action selection into cycle-based agent simulation environments*. In: Proceedings of Agent 2005: Generative Social Processes, Models, and Mechanisms, M. North, D. L. Sallach, and C. Macal, Eds. Chicago: Argonne National Laboratory, October 2005.

(Civilization IV) Civilization IV, homepage: http://www.2kgames.com/civ4/home.htm [23.8.2008]

(Cutumisu ad., 2007) Cutumisu, M., Szafron, D., Schaeffer, J., Waugh, K.: *Motivational Ambient and Latent Behaviors in Computer RPGs*, In: 3rd Annual Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-07), Stanford, CA, USA, June 6-8, 2007, 74-76.

(Enti) Projekt Enti. MFF UK, Praha. Projektová stránka: http://ufal.mff.cuni.cz/~bojar/enti/ [28.3. 2008]

(Holeček, 2008) Holeček, O.: *HTN plánování pro projekt IVE*. Bakalářská práce, UK MFF Praha (2008)

(IVE) Projekt IVE: inteligentní virtuální prostředí. MFF UK, Praha. Projektová stránka: http://urtax.ms.mff.cuni.cz/ive/public/about.php [28.3. 2008]

(Juhász, 2008) Juhász, M.: *Editor pro IVE – procesy a řídící algoritmy*. Bakalářská práce, UK MFF Praha (2008)

(Kadlec ad., 2007) Kadlec, R., Gemrot, J., Burkert, O., Bída, M., Havlíček, Brom, C.: *POGAMUT 2 - A platform for fast development of virtual agents' behaviour*, In: Proceedings of CGAMES 07, La Rochelle, France (2007)

(Kubr ad., 2006a) Kubr, J., Kulhánek, J., Poch, T., Šafrata, P., Šerý, O., Šulc, Z.: *IVE Project Documentation*. Dokumentace k projektu, UK MFF Praha.

(Kubr ad., 2006b) Kubr, J., Kulhánek, J., Poch, T., Šafrata, P., Šerý, O., Šulc, Z.: *IVE Project User Documentation*. Dokumentace k projektu, UK MFF Praha.

(Luke ad., 2004) Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K.: *MASON: A New Multi-Agent Simulation Toolkit*, In: Proceedings of the 2004 SwarmFest Workshop (2004)

(Neverwinter Nights) Neverwinter Nights, homepage:
http://nwn.bioware.com/ [28.3.2008]

(Pogamut2) Pogamut2, homepage:
https://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php [28.3.2008]

(Reidinger, 2007) Reidinger, J.: *Episodická paměť pro postavy z IVE*. Bakalářská práce, UK MFF Praha (2007)

(Xaitment) Xaiment, homepage:
http://www.xaitment.com/ [28.3. 2008]

Přílohy

Součástí práce je i CD, které obsahuje:

- instalaci IVE Editoru, jejíž volitelnou součástí jsou:
  - testovací svět;
  - testovací svět rozšířený podle tutorialu v dokumentaci;
  - výsledný svět druhého tutorialu se zahradníkem;
  - původní demo svět k IVE jako projekt v IVE Editoru;
  - uživatelská dokumentace;
  - zdrojový kód;
  - tři repositáře objektů;
  - javadoc.
- zdrojový kód IVE Editoru;
- uživatelskou dokumentaci IVE Editoru, jejíž součástí jsou i dva tutorialy;
- programátorskou dokumentaci IVE Editoru (Jádro, MOL);
- javadoc;
- text této práce ve formátu .pdf;
- instalaci IVE-1.0.