# Java a hry

Ondřej Sýkora

srpen 2003

# 1 Úvod

Během diskuse v rámci mého referátu na téma *Java Virtual Machine* padlo i několik dotazů na možnost využití Javy v počítačových hrách a podobných interaktivních aplikacích. Většina uživatelů počítačů a internetu si při spojení pojmů „Java“ a „počítačová hra“ nejspíše vybaví nějaký z appletů, kterými se snaží zaujmout tvůrci webových stránek. To je ovšem jen jedna možnost... Tento text je zaměřen spíše na využití interpretovaných jazyků při tvorbě složitějších herních projektů na osobních počítačích.

Čtenář by se měl alespoň orientovat v pojmech, se kterými pracuje objektově orientované programování; znalost jazyka Java a jeho principů je velkou výhodou. Také předpokládám, že se čtenář již setkal s některou z modifikovatelných her - výborným příkladem je třeba Unreal od Epic Megagames, Quake I-III od iD Software, nebo Half-life od společnosti Valve, nebo Neverwinter Nights (BioWare) pro ty, kteří nemají rádi rychlou akci.

## 1.1 Počítačové hry

Na první pohled je patrné, že složitost počítačových her se stále zvyšuje. Nemyslím tím, jak obtížné je ve hře zvítězit, ale složitost technologickou a obsahovou - silnější počítače umožňují vytvářet stále větší a obsáhlejší (záměrně však neříkám lepší) hry. To pochopitelně klade stále větší nároky na jejich tvůrce, s současnosti je běžné, že herní vývojáři používají cizí technologie, nebo že na jednom grafickém systému autoři postaví několik různých her. U některých typů her je běžné, že autoři dávají hráčům k disposici některé nástroje, které sami použili při vývoji hry, a umožní jim tak upravovat nebo vytvářet nový obsah hry (nové mapy, herní situace a podobně). Stává se nutností, aby program hry byl v největší možné míře modifikovatelný a modulární - a tvorbu právě takových programů může Java značně usnadnit.

## 1.2 Java

Představovat Javu je v současné době zbytečné. Během téměř osmi let své existence si dokázala najít své uplatnění ve většině „informatických“ oborů. Tedy jen pro úplnost - Java je silně typovaný objektový programovací jazyk. Programy se nekompilují do strojového kódu, ale do tzv. *byte-kódu*, který je později spouštěn pomocí *virtuálního stroje*[1]. Díky tomu je možné program bez dalších úprav spustit na různých platformách. A díky tomu také fungují

[1]Speciální program, který načítá instrukce v byte-kódu a vykonává je

o něco pomaleji, než programy překládané přímo do strojového kódu - i když ani to v současné době nemusí být úplně pravda.

Pravdou je, že první interprety (virtuální stroje) neoplývaly vysokým výkonem - interpretovaný program běžel třicet i vícekrát pomaleji, než stejný program zapsaný v jiném jazyce a přeložený do strojového kódu. Vývojáři si ale byli této slabiny vědomi a postupně přišli s mnoha optimalizacemi a vylepšeními. Virtuální stroje mohou být vybaveny překladačem a provádění programu značně urychlit převedením instrukcí byte-kódu na instrukce procesoru[2]. Poměrně velkým vývojem prošly i metody správy paměti. Při využití všech optimalizací v současnosti může „interpretovaný“ program běžet rychlostí téměř srovnatelnou s programem přeloženým do strojového kódu, v některých úlohách mohou být i o něco rychlejší.

Je přirozené, že se firma Sun snaží Javu prosadit úplně všude. Také je pochopitelné, že stejná platforma nemusí vyhovovat pro všechny účely. Proto vývojáři přišli s třemi různými edicemi, které by měly pokrýt celý trh. Všechny tři edice používají stejný byte-kód a stejné formáty souborů, liší se jednak v konstrukci virtuálních strojů a v tom, jaké nabízejí rozhraní a knihovny. Pro účely tohoto článku je nejméně zajímavá pro servery určená Java2 Enterprise Edition. Zato zde najdete zmínky o Java2 Micro Edition a tvorbě her pro mobilní zařízení[3]. Hlavně nás ale bude zajímat Java2 Standart Edition, která je určena pro osobní počítače a (hlavně) uživatelské aplikace - pokud nebude řečeno jinak bude pod názvem *Java* míněna právě tato edice.

# 2 Čistá Java

Je obvyklé pro softwarový projekt (i hra není nic jiného, než softwarový projekt) zvolit jeden programovací jazyk a v něm celý projekt implementovat. Takovému přístupu přesně odpovídá používání „čisté Javy“ - tento přístup má své nesporné, na první pohled zřejmé výhody. Přeložený program můžeme bez problémů vzít a přenést na libovolnou jinou platformu (za předpokladu, že pro ni je k disposici JVM).

Už dlouho se ale ví, že „není všechno zlato, co se třpytí“ a ukazuje se, že toto rčení dobře pasuje i na tvorbu her za pomocí Javy. Jedním z prvních problémů, na které narazíme jsou výkonová omezení - to, že na dané platformě můžeme naši hru spustit ještě neznamená, že na té platformě hru bude možné hrát. Podobným problémem tohoto druhu je, že vnitřní struk-

[2]Just In Time compilation, nověji HotSpot technologie používající adaptivní překladač k určení „kritických“ míst v programech a jejich přednostnímu překladu.

[3]Java2ME je určena hlavně pro mobilní, nebo nějakým způsobem omezená zařízení, ale měl pod ní spadat i zrušený Java Game Profile učený pro herní konzole

tury virtuálního stroje zabírají poměrně dost paměti - příkladem je třeba (jinak poměrně výkonná) konzole Sony Playstation 2, která se svými 32 MB paměti RAM téměř znemožňuje použití Javy[4].

S výkonem dané souvisí i dostupnost a použitelnost jednotlivých API rozhraní. Specifikace virtuálního stroje předepisuje pouze některá. Stejně nelze očekávat například pokročilé grafické efekty zobrazované v reálném čase na zařízeních, která nejsou vybavená příslušným hardware a stejně špatně se budou uplatňovat aplikace pro mobilní zařízení na výkonném hardware - nehledě na to, že v takovém prostředí budou působit poněkud chudým dojmem.

## 2.1 Java a multimédia

Při prosazování nějaké platformy do prostředí domácích počítačů, je naprostou nutností zajistit na ní podporu multimédií. Běžného uživatele příliš nezajímá technické pozadí programů, které používá, ale chce pracovat v příjemném a snadno použitelném (grafickém) prostředí, čeká, že si bude moci pouštět video a hudbu a podobně. Pro práci s multimédii má Java k disposici rozsáhlou sadu knihoven a rozhraní souhrnně nazývaných *Java Media APIs*. Tyto jsou rozděleny na několik nezávislých balíčků. Předepsané specifikací jsou ale jen některé z nich, ostatní jsou pouze doporučené, některé z nich navíc vyžadují podporu ze strany systému, na kterém jsou provozovány a v současné době jsou proto použitelné jen na některých platformách (Windows, Solaris a s výjimkami Linux).

### 2.1.1 Java 2D

Jak už název napovídá, mluvíme zde o rozhraní pro práci s dvourozměrnou grafikou. Toto rozhraní bylo uvedeno v JDK 1.2 a je předepsané - na jeho přítomnost se tedy můžeme spolehnout[5]. Rozhraní je koncipováno jako rozšíření grafického rozhraní AWT[6] Základní rozhraní pro 2D grafiku *java.awt.Graphics2D* je rozšířením původního *java.awt.Graphics*, stejně jsou do objektové hierarchie zařazeny i ostatní třídy Java2D - tím je zajištěna zpětná kompatibilita.

Java2D umožňuje provádět běžné grafické operace jako je kreslení prakticky libovolných geometrických obrazců, textu, práci s bitmapami, aplikace grafických filtrů, podpora transformací u všech grafických operací, podpora

[4]Tento a jiné problémy měl řešit *Java Game Profile* - speciální profil pro Java2ME zařízení. Jeho vývoj byl ale v červnu tohoto roku zrušen

[5]... všude tam, kde je k dispozici Java2SE

[6]Abstract Window Toolkit - soubor tříd a rozhraní pro programování uživatelského rozhraní.

antialiasingu a další. Navíc neklade žádné předpoklady na výstupní zařízení - díky tomu je možné ho použít stejně dobře pro tvorbu grafických operací jako pro tisk.

### 2.1.2 Java3D

Java3D je soubor tříd a rozhraní umožňující snadno pracovat s trojrozměrným prostředím. Rozhraní funguje na vysoké úrovni a programátor tak při tvorbě pracuje „neposílá trojúhelníky grafické kartě“, ale operuje s objekty nacházejícími se v trojrozměrném prostředí[7]. Díky tomu se programátoři nemusí zabývat technickými detaily zobrazování scény a mohou se soustředit přímo na její obsah. Součástí navíc je zvukový systém umožňující práci se zvukem umístěným do universa.

Také je tím zajištěna naprostá nezávislost na platformě. Pokud je to možné, používá Java3D pro zobrazování svého universa hardwarovou akceleraci, například přes Direct3D (na systémech Windows) nebo OpenGL (na unixových systémech, ale i na Windows). Grafický systém obsahuje řadu optimalizací pro zrychlení zobrazování, počínaje frustum-cullingem, využíváním více vláken až po dynamický LOD[8].

Objekty (virtuálního) universa jsou uloženy ve stromové struktuře, takzvaném grafu scény[9]. Kořenovým uzlem je vždy instance třídy *VirtualUniverse*, který slouží jako kontejner. Přímým potomkem (ve smyslu stromu scény, nikoliv v rámci dědičnosti) je vždy instance třídy *Locale* určující pozici v universu a sloužící jako kontejner pro další objekty. Ty dále mohou tvořit stromovou strukturu - k disposici například jsou uzly, které všem svým potomkům přidávají nějakou transformaci, uzly, které je možné sdílet v různých částech stromu scény[10] „Viditelné“ objekty jako jsou světla, geometrie, zvuky, nebo kamera umisťujeme do listů stromu. Do listů stromu jsou umístěny i některé objekty pro nastavení parametrů scény, jako je třeba mlha (spojená s ořezáváním objektů, které jsou příliš daleko), nebo ambientní světlo.

Součástí grafu scény jsou také takzvané *Behavior objects*, které mohou zpracovávat události a měnit nastavení jiných objektů. Stejně jako všechny ostatní objekty ve scéně i tyto mají určený svůj prostor v universu a události zpracují jen ty, které svým prostorem zasahují do viditelné části universa (do

[7]Toto prostředí je terminologii Javy3D nazýváno *Virtual Universe*

[8]Level of detail - snižování množství detailů u zobrazovaných objektů v závislosti na vzdálenosti od kamery

[9]v originále *scene graph*

[10]Díky těmto objektům může vzniknout cyklus a graf scény tak nemusí být a často není strom, po grafu se ale vždy pohybujeme jen jedním směrem (od potomků k rodičům, nebo naopak) a takové cykly nám tudíž nemusí vadit.

části, které zabírá kamera). *Behavior object* se používají hlavně k rozhýbání scény, ale je pomocí nich implementován i třeba LOD, nebo billboarding[11]. Java3D může fungovat ve třech různých režimech:

- **Immediate mode** - programátor pracuje přímo s trojúhelníky, body v prostoru, atd. tak, jak to známe třeba z OpenGL. Java3D zde jen těžko může provádět optimalizace zobrazování - to zůstává úkolem programátora aplikace.
- **Retained mode** - pracuje se se stromem scény tak, jak je popsaný výše. Umožňuje dělat optimalizace ve scéně a při zobrazování.
- **Compiled retained mode** - tento mód je podobný předchozímu, ale scéna je známá předem. Části scény, které se za běhu programu nemění jsou předem zkompilovány - to umožňuje provést některé další optimalizace a ještě zvýšit výkon.

Pro tvorbu programů autoři pochopitelně doporučují spíše druhé dva módy.

### 2.1.3 Java Advanced Imaging

V honbě za maximální nezávislostí na platformě jsou grafické funkce v Javě (Java2D) oproštěny i od závislosti na grafických formátech. Zní to vznešeně, v praxi to ale znamená, že soubor tříd v Java2D obsahuje funkce pro práci s bitmapami, ale neobsahuje žádné funkce, které by usnadnily jejich načítání. To (ale i jiné funkce) má na starosti balík *Java Advanced Imaging(JAI)*.

Pomocí tříd knihovny je možné načítat a ukládat obrázky ve všech důležitých formátech[12]. Všechny operace nad bitmapami jsou reprezentovány jako objekty. Ty jsou navíc odvozené od třídy *PlanarImage*[13], takže je možné je použít jako vstup jiných operací a vytvářet z nich řetězce.

Autoři se chlubí téměř nekonečnými možnostmi využití knihovny a zdá se, že tentokrát si asi příliš nevymýšlí. Grafických filtrů je něco přes stovku a zahrnují všechny možné operace od skládání obrázků (součty, alfa-míchání, ...), přes geometrické transformace až po různé analytické funkce jako je detekce hran. Otázkou zůstává, kolik z nich najde praktické využití...

[11] zobrazování dvourozměrných obrázků v prostoru tak, že jsou vždy natočené směrem ke kameře

[12] Zahrnuje JPEG, PNG, BMP a GIF

[13] Základní objekt reprezentující obrázek pro JAI

### 2.1.4 Java Sound API

Od verze 1.3 je součástí Javy i rozhraní pro práci se zvukem, je možné pomocí něj pracovat jednak se samplovanými daty a jednak s MIDI hudbou, navíc obsahuje i funkce pro záznam a následné zpracování zvukových dat. Podle specifikace je podporováno několik základních formátů (WAV, AIFF a AU pro samplovaná data a SMF typ 0 a 1 pro MIDI).

### 2.1.5 Java Media Framework

Pro práci s médii „založenými na časováni“ je tu k disposici Java Media Framework. Stejně jako všechny ostatní Java API, i toto se snaží nabídnout co možná největší funkčnost - k dispozici tu tedy jsou nástroje pro záznam a zpracování médií. Pro potřeby počítačových her je nejdůležitější možnost přehrávat zvuková data a video. JMF dokáže pracovat se streamovanými daty ve většině standardních formátů včetně AVI, QuickTime a MPEG.

K některým druhům médií je možné vyžádat si AWT komponenty s ovládacími prvky a pokud se jedná o video a nebo přehrávač dokáže generovat jiný grafický výstup tak i AWT komponenty, do kterých je tento výstup směrován.

## 2.2 Applety

Applety jsou (grafické) prográmky, které jsou určené pro vkládání do oken jiných programů. Tolik říká definice. Pro praktický příklad není nutné chodit daleko - většina lidí zná Javu hlavně hlavně díky appletům na webových stránkách. Jejich tvorba je poměrně jednoduchá a přitom (teoreticky) není příliš omezující. Programátoři mohou své applety snadno publikovat na svých webových stránkách, prohlédnout a spustit je pak může kdokoliv, kdo má k disposici www prohlížeč s podporou Javy, přitom nemusí nic instalovat, nemusí nic spouštět - prostě přijde na správnou adresu a může si hrát.

Naprogramovat applet je navíc dost jednoduché - jedná se totiž o GUI aplikace programované pomocí rozhraní AWT. Používá se zde *na událostech založené programování* a každý, kdo někdy vytvářel aplikaci pomocí knihoven jako je Qt, wxWindows, nebo VCL bude po zběžném prohlédnutí dokumentace schopný vytvářet vlastní applety.

Teď, když jsem appletům udělal dostatečnou reklamu přišel čas pro méně příjemné informace. Bohužel většina z nich sice nesouvisí přímo s Javou, ale i tak degradují možnosti appletů na „nástroj pro vytváření pěknějších tlačítek na www stránky“.

- ***Rychlé a kvalitní připojení k internetu*** je pro tento typ aplikací naprosto nezbytné. Běžné je, že herní data zabírají i po kompresi desítky až stovky megabytů (a v extrémních případech ještě o něco více). Tato bariéra by mohla padnout už brzy díky stále rostoucí oblibě a dostupnosti připojení pomocí bezdrátových sítí, nebo pomocí ADSL.

- ***Podpora Javy v prohlížečích*** - za rozšíření zvěstí o rychlosti (v tomto případě spíše pomalosti) Javy do značné míry mohou virtuální stroje dodávané spolu s „hlavními“ prohlížeči[14] Ty se totiž zarazily na úrovni JDK 1.1[15], které jednak nedosahují příliš vysokých výkonů a jednak nemusí obsahovat rozhraní a knihovny uvedené v novějších verzích. Tento „problém“ lze snadno vyřešit instalací novější verze virtuálního stroje - ale většina uživatelů neví, že to je třeba - dokonce ani nevědí, že je něco takového možné!

Částečné řešení prvního problému vychází z toho, že v jednom okamžiku hra pracuje jen s určitou částí dat - například pokud je herní svět rozdělený do několika map, stačí udržovat v paměti tu, ve které se hráč pohybuje právě teď, ty další hra automaticky stáhne ze serveru ve chvíli, kdy bude potřeba.

## 2.3 Aplikace

Druhou možností je tvorba samostatných aplikací. Na rozdíl od appletů nejsou závislé na hostující aplikaci - mohou to být konzolové aplikace, které vůbec nepotřebují grafické rozhraní, mohou si vytvářet vlastní okna a (ve spolupráci s grafickými rozhraními) v případně pracovat ve full-screen - u počítačových her to je celkem obvyklé. Programování takovéto aplikace se v ničem neliší od programování aplikací v jiném programovacím jazyce.

## 2.4 Java pro mobilní telefony

Programování aplikací pro mobilní telefony je poměrně specifické. Jedná se o zařízení se značně omezeným výkonem a množstvím paměti, kterou má programátor k disposici - je dobré udržet nároky aplikace co možná nejnižší. Také je nutné počítat s omezenými možnostmi zobrazovacích zařízení - černobílý displej s rozlišením 128x64 pixelů[16]. Virtuální stroj používaný v

[14] V současnosti se tato kritika týká prakticky jen MS Internet Exploreru - bohužel se ale jedná o nejrozšířenější prohlížeč. Navíc díky negativnímu přístupu firmy Microsoft k Javě nelze očekávat zlepšení

[15] z roku 1997, současná verze je 1.4

[16] Ale i to se lepší - Nokia 6600 nabízí rozlišení 176x208 pixelů a 16-bitovou barevnou hloubku.

Java2SE by na takovýchto zařízeních nebylo možné spustit, popřípadě by zabral většinu zdrojů pro sebe - v této kapitole tedy bude řeč o Java2ME.

Aplikace pro mobilní zařízení (MIDlety) jsou nejen názvem podobné více appletům, než běžným aplikacím. Podobně jako applety i midlety jsou spouštěné uvnitř jiné aplikace, tentokrát to je *application-management software*, který je obvykle zabudovaný do zařízení (firmware). Takový přístup má v případě mobilních zařízení smysl, protože běh midletu může být nutné přerušovat a znovu obnovovat v závislosti na činnosti uživatele - pokud by během práce s midletem uživatel nemohl přijímat hovory, zřejmě by spokojený nebyl. A podobná situace by nastala ve chvíli, kdy by kvůli přerušení hovorem přišel o svá data.

Stejně jako applety, ani midlety nemají funkci main. Místo ní musí definovat funkci `startApp` volanou po aktivování midletu, `pauseApp` volanou když je midlet odstaven do pozadí a funkci `destroyApp` pro zrušení midletu. Displej zařízení je reprezentován objektem *Display*, tomuto přiřadíme objekt odvozený od třídy *Canvas* - objekt zajišťující vykreslování. Jednou z jeho metod je známá metoda `paint` - kreslení probíhá pomocí rozhraní Graphics podobně jako u jiných edicí Javy. Informace o činnosti uživatele jsou midletům předávány podobným způsobem jako u appletů pomocí callback funkcí.

# 3 Špinavá Java

Název tohoto řešení vznikl jako protiklad k již dříve zmíněnému přístupu *čisté Javy*. Výrobci software si byli vědomi ohromných výhod, které jim použití Javy může přinést, ale zároveň se báli nižšího výkonu při jejím nasazení. Nejlepší by bylo nějak zkombinovat modularitu a snadnost vývoje v Javě s rychlostí a „na platformě závislými“ výhodami jazyků kompilovaných přímo pro danou platformu. A i to je možné...

S vývojářským kitem pro Javu (dále jen JDK) se programátorům dostávají do rukou i nástroje pro provázání programů v Javě s nativními programy. Vývojáři tím sice obětují nezávislost svých produktů na použité platformě, nicméně v současném prostředí, kdy většina herních zařízení jsou buď herní konzole (a tedy pro Javu víceméně nepoužitelná zařízení), nebo PC s Windows, je takováto oběť přijatelná. Portování takovéto kombinované aplikace na jinou platformu je navíc jednodušší, protože stačí upravit nativní část aplikace.

Známé moudro nám říká, že po 90% času běhu programu běží 10% kódu, zbylých 90% kódu obsadí jen 10% času běhu programu - Pokud bychom dokázali zrychlit těch nejčastěji používaných 10% kódu, znatelně bychom

zrychlili běh celého programu.

## 3.1 Java Native Interface

Jedná se o soubor funkcí a struktur, pomocí kterých je možné provázat kód v jazyce C a C++ s kódem v Javě. *Java Native Interface (JNI)* navíc je součástí specifikace virtuálního stroje, takže programy vytvořené s jeho pomocí jsou na dané platformě nezávislé na použitém virtuálním stroji.

Vytvoření třídy (v Javě), která volá nativní funkce je jednoduché - v její definici stačí před název nativní metody doplnit klíčové slovo `native`. Nativní funkce pak musíme dodat virtuálnímu stroji ve formě dynamicky připojované knihovny. Funkce v knihovně jsou pojmenovávány podle klíče `Java_{plné jméno třídy}_{jméno metody}[specifikace parametrů]`. Pokud je jméno metody v dané třídě jednoznačné, není specifikace parametrů ve jméně funkce v dynamické knihovně nutná. Hlavičkový soubor se jmény potřebných metod jde ale jednoduše vygenerovat z zdrojových kódů v Javě (*.java) vygenerovat pomocí utility `javah`.

Pomocí funkcí a struktur JNI je pochopitelně možné přistupovat k metodám a položkám instancí tříd Javy. Protože C ani C++ nepodporují automatickou správu pamětí, je nutné uvolnit odkazy na objekty, se kterými nativní metody pracují v okamžiku, kdy s nimi pracovat přestanou - virtuální stroj jinak nemůže nijak zjistit, že už nejsou používané a nikdy by je nemohl uvolnit!

Aby bylo možné zajistit spolupráci tříd získaných z různých zdrojů, jsou v byte-kódu jednotlivé metody a položky označovány pomocí jejich plných názvů, ty jsou po načtení třídy převedeny na číselné identifikátory používané virtuálním strojem. Ze stejného důvod je nutné pracovat s položkami a metodami tříd pracovat pomocí jednoznačných na virtuálním stroji nezávislých identifikátorů i v nativních funkcích. Před přístupem k položce instance Javové třídy je proto potřeba zjistit si její ID. To může být různé pro dvě různé instance a může se i měnit v čase - na platnost ID se programátor může spolehnout, dokud vlastní referenci na instanci, které ID patří.

Při tvorbě rozhraní mezi C(++) a Javou je nutné dát pozor na správnost předávaných argumentů funkcí - JNI ani virtuální stroj neprovádějí žádné typové kontroly, nebo kontroly na nenulovost ukazatelů - jednak by takové kontroly mohly znatelně zpomalit běh programu a jednak často virtuální stroj nemá dost informací pro provádění takových kontrol (toto se týká zejména typových kontrol).

Při volání metod z nativního kódu je na programátorovi kontrolovat vrácené hodnoty a případně zjišťovat, jestli nedošlo k výjimce. Sám může pomocí JNI funkce `Throw` výjimky vyvolávat, nebo předávat dále.

Spouštět kombinovaný program je možné pomocí virtuálního stroje stejně jako jakýkoliv jiný program, také ale je možné napsat nativní program, který pomocí JNI vytvoří vlastní virtuální stroj, ve kterém pak spustí část vytvořenou v Javě.

## 3.2 Možnosti použití

Jak už je psáno výše, můžeme pomocí JNI snadno provázat programy zapsané v Javě s s nativními funkcemi. Většinu času program počítačové hry tráví vykreslováním herní situace - nabízí se nám tak možnost zkombinovat rychlý nativní kód pro vykreslování s flexibilitou a snadností práce v Javě. Uvedu několik možných přístupů; to ale žádném případě není výčet všech možností - pro některé aplikace může být výhodně používat „mezistupně“ vyplňující mezery mezi uvedenými přístupy, nebo ty uvedené různě kombinovat.

### 3.2.1 Zpřístupnění nativních API

Z pohledu „multiplatformní“ filosofie se jedná o volbu nejmenšího zla - pomocí JNI zpřístupníme některá nativní rozhraní a jinak budeme celou aplikaci vytvářet v Javě - při vhodném použití a správné volbě rozhraní pak podle Marnera?? není velký rozdíl ve výkonu proti nativním aplikacím[17]. Pro některá rozhraní už existují funkční „importní“ knihovny - příkladem za všechny je OpenGL a *OpenGL4Java*. Zajímavou vlastností virtuálního stroje firmy Microsoft bylo automatické zpřístupnění rozhraní COM jako běžně použitelných Javových tříd. Bohužel nedostatky tohoto stroje převažují výhody a navíc byl jeho vývoj v rámci konkurenčního boje ukončen.

### 3.2.2 Java nad nativním kódem

V tomto přístupu je stále velká část (oněch výše zmíněných 90%) programu napsaná v Javě. Pomocí nativních funkcí jsou implementované ty části programu, které se nejčastěji opakují a ve kterých se při jeho běhu tráví nejvíce času. To v případě počítačových her zahrnuje grafické funkce a případně nějaké výpočetně složité záležitosti, jako jsou fyzikální simulace.

Pokud bude program vhodným způsobem rozdělený do tříd, je možné jeho jednotlivé části měnit nezávisle za zbytku - potom je možné vyjít od programu, který bude celý zapsaný v Javě, profilováním určit kritické části a ty později přepsat pomocí nativních funkcí. Po zveřejnění rozhraní navíc

[17]Marner testoval výkon jednoduché OpenGL aplikace napsané v C a v Javě, nejlepší výsledky programů v Javě byly téměř srovnatelné s výsledky téhož programu v C

mohou sami uživatelé snadno měnit vlastnosti systému a přidávat novou funkčnost.

Tento přístup byl úspěšně použit ve hře *Prax War* (hra nebyla dokončena, ne však kvůli Javě) - S výjimkou rendereru byla hra kompletně vytvořena v Javě.

### 3.2.3 Java jako skriptovací jazyk

Téměř celý program je vytvořený jako nativní (C, C++), JNI využívá pro vytvoření vlastního virtuálního stroje a ten potom využívá jako skriptovací jazyk. Pomocí něj je možné naprogramovat například umělou inteligenci nepřátel, kterou je možné snadno měnit a pomocí dědičnosti vytvářet speciální chování. Virtuální stroj může i v tomto případě používat optimalizace jako je JIT a tím zajistit výrazně lepší výkon, než jakékoliv „vlastnoručně“ vyrobené skriptovací systémy. Navíc je Java dostatečně bohatý jazyk, pro který existují kvalitní volně dostupné nástroje. Po zveřejnění rozhraní budou uživatelé moci měnit chování některých objektů ve hře, nebo za pomoci dalších nástrojů a editorů vytvářet totální modifikace.

Úspěšným příkladem použití této metody je například *Vampire: The Masquerade* od společnosti *Nihilistic Software*.

# 4 Pohled jinam

Java není jediným interpretovaným jazykem používaným v počítačových hrách. Spíše naopak, možností je spousta, jen si vybrat tu pravou.

## 4.1 UnrealScript

UnrealScript je jazyk, který se už od pohledu Javě dost podobá. Autoři se netají tím, že při tvorbě jazyka z Javy vycházeli a že se v mnoha věcech nechali inspirovat. Ze všech ostatních uváděných jazyků je Javě asi nejblíže - jedná se o objektový jazyk používající byte kód. Proti Javě je o něco pomalejší, ale výměnou za nižší rychlost nabízí různé zajímavé jinde neviděné funkce.

Původně prý autoři chtěli využít pro Unreal jako základ skriptovacího jazyka Javu, ale zjistili, že jim proti C++ nepřináší žádné výhody, naopak že je omezuje nepřítomnost některých prvků jazyka C++ (přetěžování operátorů). Navíc dostupné virtuální stroje špatně zvládaly programy s velkým počtem běžících vláken (každý herní objekt v Unrealu totiž běží ve svém vlastním vláknu).

Autoři navíc potřebovali další vlastnosti, kterými Java nedisponuje - podporu stavů u objektů[18] a lepší podporu pro časování. Ve hrách je běžné, že některé akce (například přesun nepřítele z místa na místo) trvají určitou dobu - implementace funkcí ovládajících takovéto akce v jazyce, jako je C++ nebo Java, je poměrně obtížná a výsledkem bývá dosti nepřehledný kód - v Unrealu je možné takové funkce používat ve spojení se stavy objektu.

Spolu s výborným editorem úrovní nabízí hráčům a nadšencům UnrealScript jednu z nejlepších možností modifikovat hru. Unreal tak dává k dispozici výborné trojrozměrné prostředí, které je možné použít i mimo prostředí počítačových her.

## 4.2 Python

Stejně jako Java je i Python objektový jazyk. Je založený na modulech (jednotky kódu zapouzdřující určité funkce) - pokud zůstane zachováno rozhraní (definice funkcí a tříd), je možné používat moduly z různých zdrojů - díky tomu je možné například začít s prototypem programu kompletně zapsaným v Pythonu a pokud se prototyp osvědčí, postupně ho nahrazovat kódem naprogramovaným v jiných jazycích.

Integraci Pythonu a C navíc usnadňují nástroje, které z zdrojových kódů v Pythonu vygenerují zdrojové kódy a hlavičkové soubory v C, pomocí kterých je možné pracovat s objekty v Pythonu.

Další velice zajímavou (a jinde neviděnou) vlastností je ukládání stavu objektů do souboru - za pomocí modulu *cPickle* je možné snadno uložit data objektu do souboru a později je opět načíst. A to včetně vnořených objektů, polí a referencí.

Python jako skriptovací jazyk používá například open-source 3d engine *Crystal space*, je v něm možné vytvářet interaktivní aplikace v programu *Blender*, z komerčních her stojí za zmínku například *Blade of Darkness* od *Codemasters*.

Navíc je možné programovat hry celé v Pythonu za pomoci knihovny *PyGame* postavené na multiplatformní multimediální knihovně SDL. Pro vytváření jednodušších hříček je takovéto prostředí ideální. Seznam podporovaných platforem je poměrně široký a zahrnuje Windows, Unixové systémy (FreeBSD, Linux), ale i MacOS.

[18] Funkce definuje pro různé stavy různé varianty jedné metody - po zavolání této metody virtuální stroj použije tu, která odpovídá aktuálnímu stavu objektu

## 4.3 Lua

Lua jako skriptovací jazyk má za sebou dlouhou herní historii - poprvé byla použitá zřejmě ve hře *Grim Fandango* (*LucasArts*). Od té doby vývoj jazyka probíhal v těsné spolupráci s autory počítačových her a Lua se stala jedním z nejpoužívanějších skriptovacích jazyků. Celý interpret je napsán v ANSI C, takže není problém přenést ho na téměř jakoukoliv platformu.

Jednoduchá syntaxe a snadné použití z ní dělají ideálního kandidáta na herní skriptovací jazyk.

Tituly používající Luu zahrnují například sérii *Baldur's Gate* od *BioWare*.

## 4.4 LISP

LISP, jazyk původně určený pro programování umělé inteligence je perfektní jazyk pro tvorbu skriptů. Základní stavební kámen programu v LISPu jsou seznamy - seznam je dokonce i sám program v LISPu. Díky tomu je možné snadno přidávat do jazyka nové vlastnosti a funkce[19].

LISP byl použitý jako skriptovací jazyk například ve hře *Abuse* od firmy *Crack dot Com*.

## 4.5 INF

Aby bylo zřejmé, jakým vývojem prošlo použití interpretů v počítačových hrách, na závěr uvádím jako kuriozitu jeden z prvních skriptovacích systémů použitých ve hře. Je to systém použitý ve hře *Dark Forces* od *LucasArts*. Tento „jazyk“ neměl vlastní syntaxi - programovalo se přímo zadáváním instrukcí byte-kódu...

[19] většina interpretů LISPu obsahuje podporu pro objektově orientované programování, jeho doprogramování (jako LISPový program!) do interpretů bez jeho podpory je otázkou chvilky

# Obsah

# Reference


[1] Sun: *Java2 SDK*
`http://java.sun.com`

[2] J. Marner: *Evaluating Java for Game Development*
`http://www.rolemaker.dk/articles/evaljava/`

[3] články na serveru *Gamedev.net*
`http://www.gamedev.net`

[4] články na serveru *Gamasutra.com*
`http://www.gamasutra.com`

[5] Epic Megagames: *UnrealWiki*
`http://wiki.beyondunreal.com`