ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA STROJNÍ

Studijní program: N2301 Strojní inženýrství
Studijní obor: 2301T007 Průmyslové inženýrství a management

DIPLOMOVÁ PRÁCE

Využití balíku DIGITOV pro praktické účely

Autor: **Jiří POLCAR**

Vedoucí práce: **Ing. Petr HOŘEJŠÍ, Ph.D.**

Akademický rok 2013/2014

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta strojní
Akademický rok: **2013/2014**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jiří POLCAR**
Osobní číslo:	**S12N0034P**
Studijní program:	**N2301 Strojní inženýrství**
Studijní obor:	**Průmyslové inženýrství a management**
Název tématu:	**Využití balíku DIGITOV pro praktické účely**
Zadávající katedra:	**Katedra průmyslového inženýrství a managementu**

Zásady pro vypracování:

1. Úvod
2. Shrnutí předchozí práce
3. Oblasti vývoje a využití balíku DIGITOV
4. Popis vybraného směru vývoje nebo využití
5. Zhodnocení
6. Závěr

Rozsah grafických prací: **0 výkresů**

Rozsah pracovní zprávy: **50 - 70 stran**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

1. **HOŘEJŠÍ, P., GÖRNER, T., KURKIN, O.** *VYZTYMDP: Virtuální realita a DP, e-book.* **Plzeň: ZČU-KPV, 2012. ISBN 978-80-87539-07-1.**
2. **BURDEA, G., GRIGORE, C.** *Virtual Reality Technology.* **Second Edition. Wiley-IEEE Press, 2003. ISBN 0-471-36089-9.**

Vedoucí diplomové práce: **Ing. Petr Hořejší, Ph.D.**
Katedra průmyslového inženýrství a managementu

Konzultant diplomové práce: **Ing. Tomáš Görner**
Katedra průmyslového inženýrství a managementu

Datum zadání diplomové práce: **23. září 2013**

Termín odevzdání diplomové práce: **23. května 2014**

Doc. Ing. Jiří Staněk, CSc.
děkan

L.S.

Doc. Ing. Michal Šimon, Ph.D.
vedoucí katedry

V Plzni dne 23. září 2013

Prohlášení o autorství

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě strojní Západočeské univerzity v Plzni.
Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů, uvedených v seznamu, který je součástí této diplomové práce.

V Plzni dne: ……. . . . . . .

podpis autora

ANOTAČNÍ LIST DIPLOMOVÉ PRÁCE

AUTOR	**Příjmení** Polcar	**Jméno** Jiří
STUDIJNÍ OBOR	2301T007 „Průmyslové inženýrství a management“	
VEDOUCÍ PRÁCE	**Příjmení (včetně titulů)** Ing. Hořejší, Ph.D.	**Jméno** Petr
PRACOVIŠTĚ	ZČU - FST - KPV	

DRUH PRÁCE	**DIPLOMOVÁ**	~~**BAKALÁŘSKÁ**~~	**Nehodící se škrtněte**
NÁZEV PRÁCE	Využití balíku DIGITOV pro praktické účely		

FAKULTA	strojní		**KATEDRA**	KPV	**ROK ODEVZD.**	2014

POČET STRAN (A4 a ekvivalentů A4)

CELKEM	54		**TEXTOVÁ ČÁST**	31	**GRAFICKÁ ČÁST**	23

STRUČNÝ POPIS **ZAMĚŘENÍ, TÉMA, CÍL POZNATKY A PŘÍNOSY**	Diplomová práce popisuje historii a vývoj projektu, jehož výstupem je DIGITOV, balíček zdrojů pro Source Engine. Tento balíček upravuje Source Engine na unikátní vývojové prostředí pro tvorbu interaktivních podnikových layoutů. Práce analyzuje Source Engine za účelem jeho úprav a dále se zabývá rozšířením DIGITOVu pro praktické využití. Tímto rozšířením je vývoj prototypu konvertoru layoutů ze software VisTable Touch, nový instalátor a spouštěč a tvorba animací výrobních procesů za použití zmíněného vývojového prostředí.
KLÍČOVÁ SLOVA **ZPRAVIDLA JEDNOSLOVNÉ POJMY, KTERÉ VYSTIHUJÍ PODSTATU PRÁCE**	DIGITOV, Source Engine, layout, interaktivita, animace, virtuální realita, vývojové prostředí

SUMMARY OF DIPLOMA SHEET

AUTHOR	**Surname** Polcar	**Name** Jiří	
FIELD OF STUDY	2301T007 „Industrial Engineering and Management“		
SUPERVISOR	**Surname (Inclusive of Degrees)** Ing. Hořejší, Ph.D.	**Name** Petr	
INSTITUTION	ZČU - FST - KPV		
TYPE OF WORK	**DIPLOMA**	~~**BACHELOR**~~	**Delete when not applicable**
TITLE OF THE WORK	Use of the DIGITOV package for practical purposes		

FACULTY	Mechanical Engineering		**DEPARTMENT**	Industrial Engineering and Management		**SUBMITTED IN**	2014

NUMBER OF PAGES (A4 and eq. A4)

TOTALLY	54		**TEXT PART**	31		**GRAPHICAL PART**	23

BRIEF DESCRIPTION **TOPIC, GOAL, RESULTS AND CONTRIBUTIONS**	The diploma thesis describes the history and development of a project, whose output is the DIGITOV, a package of resource for Srouce Engine. This package modifies Source Engine as a development environment for interactive enterprise layouts creation. The paper analyses Source Engine in order to modify it and describes the DIGITOV’s expansit for more practical purposes. These expansions are the development of a VisTable Touch layout converter, new installator and launcher and creation of manufacturing processes animations while using the described development environment.
KEY WORDS	animation, development environment, DIGITOV, layout, Source Engine, virtual reality

Obsah

Seznam zkratek

BSP	Binary Space Partitioning
DLL	Dynamic Linked Library
EULA	End User License Agreement
EP2	Episode Two
GUI	Graphical User Interface
HDR	High Dynamic Range
HL2	Half-Life 2
KPV	Katedra průmyslového inženýrství a managementu
PLM	Product Lifecycle Management
SDK	Software Developer's Kit
VHE	Valve Hammer Editor
visleaf	Visibility leaf (mn.č. visleaves)
VMF	Valve Material File
VMT	Valve Material Type
VPK	Valve Pak File
VTF	Valve Texture File

1 Úvod

Interaktivní modely výrobních podniků jsou v poslední době fenoménem, který si zaslouží pozornost podnikové i akademické sféry. Jedná se o virtuální svět, který si může jeho uživatel dle libosti procházet a ovlivňovat svou činností. Pokud je takové virtuální prostředí vytvořené podle reálného, vzniká velmi silný vzdělávací nebo propagační nástroj.

Využití takovýchto interaktivní prostředí je mnoho. Uživatel se může naučit zásady chování, pohybu a orientace v nebezpečném prostoru, kde například bude pracovat, aniž by při tom ohrožoval zdraví nebo majetek. Piloti letadel a závodních vozů se cvičí na trenažérech a uplatnění je i v armádě, kde vojáci podstupují virtuální výcvik pro zacházení s různými typy vojenské techniky a zbraní [1][2].

Bohužel se na trhu nevyskytuje příliš mnoho možností a softwarových nástrojů, které umožňují takové virtuální prostředí vytvořit a provozovat. Nástroje jsou často drahé a uživatelsky velice náročné [3]. Předpokladem je velmi dobrá znalost práce s mnoha počítačovými programy a znalost programování. Tato práce popisuje alternativní cestu, jak docílit tvorby a provozu virtuálního prostředí. Je jí využití počítačových her a jejich enginů.

Na *Katedře průmyslového inženýrství a managementu* probíhá od roku 2010 projekt, který je zaměřen na využití *Source Engine* a jím poháněné počítačové hry *Half-Life 2: Episode Two* pro tyto účely. V rámci projektu vznikla knihovna *DIGITOV* s novými texturami, modely a objekty pro použití v této hře za účelem tvorby virtuálních prostředí s podnikovou tematikou. Výhodou je, že takovýto nástroj je uživatelsky přívětivý a finančně nenáročný na pořízení. *Source Engine* je přitom stále aktualizovaný, takže se dá očekávat softwarová podpora i do budoucna.

Tato práce se zabývá analýzou *Source Engine* z hlediska struktury virtuálních prostředí. Konkrétně se jedná o strukturu zdrojových kódů mapy, strukturu používaných souborů, entity, textury, detailní modely a osvětlování. Je uveden i základní popis používaných softwarových nástrojů. Účelem této analýzy je shrnutí nejdůležitějších poznatků pro další tvorbu autora této práce i pro autory dalších projektů, jejichž předmětem budou úpravy *Source Engine* pro účely tvorby vývojového prostředí interaktivních podnikových modelů i samotných modelů.

Následuje popis vzniku a vývoje *DIGITOVu* včetně okolností, které je provázely. Jedná se především o dvě aktualizace *Source Engine*, které měly za následek ztrátu kompatibility *DIGITOVu*. Další kapitola se zabývá *DIGITOVem* z hlediska jeho využití do současné doby.

Další tři kapitoly jsou stěžejními kapitolami této práce, které využívají poznatky zjištěné během výše popsané analýzy.

První z nich se zabývá vývojem algoritmizovaného překladače podnikových layoutů z programu *VisTable Touch*. Program je ve fázi prototypu a umožňuje převést již vytvořený layout výrobního podniku do zdrojového kódu *VMF* (*Valve Map File*). Autor práce po zkušenostech s těmito algoritmizovanými převody předpokládá využití jako podklad pro další tvorbu. Plně algoritmizovaný převod by dle jeho názoru byl nad rámec výhodnosti takového postupu. V současné době převedené layouty však mohou velmi urychlit tvorbu, protože všechny bloky jsou přesně rozmístěny a autor layoutu nemusí přesná umístění složitě hledat.

Práce se dále zabývá vývojem nového instalátoru a spouštěče. Tento program, pojmenovaný *DIGITOV Launcher*, je plně funkční a je určený k uživatelsky příjemnému opatření, instalaci, spouštění, aktualizaci a odinstalaci *DIGITOVu*. Program byl vyvíjen tak, aby práce s ním byla rychlá a uživateli stačilo minimum klepnutí myší pro provedení všech jeho funkcí a tak, aby ovládání programu bylo intuitivní a uživatel tím pádem nepotřeboval žádnou příručku.

Třetí z těchto kapitol se zabývá využitím *DIGITOVu* jako nástroje pro tvorbu animací výrobních procesů, konkrétně soustružení. Aby bylo možné tyto animace snadno a rychle vytvářet, musely být poupraveny *DLL* knihovny obsahující entitní třídy. Součástí výstupu této kapitoly je i program v *MS Excel*, sloužící k algoritmizované tvorbě řídících příkazů pro entity ovládající animaci.

Poslední kapitola se zabývá možným vývojem a využitím *DIGITOVu* v budoucnu. *DIGITOV* i přes svou pokročilost není možné považovat za hotový. Další práce na *DIGITOVu* budou zahrnovat tvorbu webových stránek, aby jej bylo možné uvolnit a distribuovat. V neposlední řadě to může být rozšíření o další zdroje a kompatibilitu s dalšími nástroji pro tvorbu layoutů.

2 Source Engine jako nástroj k tvorbě virtuálních prostředí

Možností tvorby interaktivních virtuálních prostředí je mnoho, ne všechny jsou ale snadno použitelné nebo cenově dostupné pro školy a studenty. Od roku 2010 vzniká na Katedře průmyslového inženýrství a managementu (KPV*)* nástroj spočívající v rozšíření standardních vývojových nástrojů herního enginu *Source*. Toto řešení má výhodu, že jeho užití je, v případě využití některých softwarových nástrojů třetích stran, uživatelsky příjemné a licence k jeho užití je finančně dostupná každému. V této kapitole se nachází detailní popis, co *DIGITOV* je a je zde popsán i *Source Engine* z hledisek, které jsou zajímavé pro jeho úpravu a tedy i tvorbu *DIGITOVu* a jeho použití. Více informací o okolnostech výběru herního engine *Source* obsahuje bakalářská práce autora [4].

2.1 Co je to Source Engine

Source Engine je produkt americké firmy *Valve Software* a je užíván jako engine (programové jádro) pro 3D počítačové hry. Historicky se jedná o evoluci enginů *Quake Engine* (*Quake* – rok vydání 1996) a *idTech 2 Engine* (*Quake II* – rok vydání 1997), oba od firmy *id* [5][6][7]. Firma *Valve Software* z těchto dvou evolučně navazujících enginů vytvořila *GoldSrc Engine*, který poháněl první díl hry *Half-Life* (rok vydání 1998). Další evolucí byl pak v roce 2004 představen *Source Engine* ve hře *Counter-Strike: Source* (*Counter-Strike* byl původně modifikací pro *Half-Life* běžící na *GoldSrc*). Zanedlouho poté byla ve stejném roce vydána hra *Half-Life 2*, pro kterou tento engine primárně vznikl [8].

Source Engine si tak i přes svůj vývoj zachovává spoustu vlastností z původního *Quake Engine*. John D. Carmack, zaměstnanec *id* a tvůrce herních enginů od *Doom Engine* až po současný *idTech 5*, tvrdí, že v *Source Engine* ještě dodnes existují stejné kusy zdrojového kódu jako v původním *Quake Engine* [9].

Prostor je v *Source Engine*, stejně jako v původním *Quake Engine*, rozdělen metodou *BSP* (*Binary Space Partitioning* – binární dělení prostoru) do *BSP* stromu (*BSP Tree*) a do *visleaves* (množné číslo zkratkového slova *visleaf – visibility leaf*) určujících, které jednotlivé listy tohoto *BSP* stromu jsou mezi sebou viditelné. *Source Engine* obsahuje moderní *Direct3D* vykreslovače (rendery) schopné využít *Phongovu* osvětlovací metodu, metodou navyšování dynamického kontrastu *HDR* (*High Dynamic Range*) a užívá i fyzikální engine *VPhysics* odvozený *od Havoc Physics* [10].

Sada oficiálních vývojářských nástrojů se souhrnně nazývá *Source SDK* (*Software Developer's Kit*), v případě *DIGITOVu* je použita verze *Source SDK Base 2013 Singleplayer*. Hlavním z těchto nástrojů je level-editor *Valve Hammer Editor* (dále *VHE*) a mnoho dalších nástrojů, povětšinou překladačů formátů například 3D modelů, textur atd.

Mapy (též levely) jsou uloženy ve zkompilovaných souborech formátu *BSP*, přičemž proces kompilace je ztrátově vratný. Ve zdrojovém kódu mapy, který je metodou *WYSIWYG* tvořen ve *VHE*, je definována hrubá geometrie, jako je terén, stěny, budovy apod. s vloženými externími modely ve vlastním formátu *MDL* jako detailní geometrií. K těmto modelům náleží zjednodušené kolizní modely, které jsou podkladem pro fyzikální engine. Více o tomto zdrojovém kódu v další kapitole.

2.1.1 Struktura virtuálního prostředí

Virtuální prostředí vykreslované enginem je definováno mapou, která je uložena v souboru ve formátu *BSP*. Jedná se o komprimovaný formát a existuje v mnoha verzích. Formát *BSP* užíval už *Quake Engine*, nicméně každý engine (i v rámci označení *Source Engine*) používá verzi jinou. To znamená, že *BSP* soubory obsahující mapy jsou použitelné pouze v jedné

konkrétní hře [11]. *BSP* soubor je vytvářen sadou tří kompilačních programů dodávaných se *Source SDK* a vstupem kompilačního procesu je *VMF* (*Valve Map File*) soubor, který je zdrojovým kódem mapy a je vytvářen uživatelem v programu *VHE*.

Obsahem *BSP* souboru jsou informace o geometrii mapy včetně texturování a výpočtu statického osvětlení, o entitách, umístění modelů, *visleaves*. Textury, modely, texty, zvuky atd. jsou uloženy externě ve zvláštních souborech a je tedy možné je používat ve více mapách. Tyto zdroje je však možné zkomprimovat pomocí programu *BSPZIP* [12] přímo do *BSP* souboru a distribuovat vytvořené virtuální prostředí pouze v jediném souboru.

Při procesu kompilace (tedy převodu *VMF* do *BSP*) dojde k definování vnitřních a uzavřených (konečných) objemů prostoru rozděleného dle binárního stromu (pozn.: formát se jmenuje *Binary Space Partitioning*). Co se nachází vně těchto prostorů, není pro engine podstatné a *BSP* soubor nemusí obsahovat veškeré informace o geometrii, hranách, bodech atd. mimo tento prostor. Naproti tomu, ve zdrojovém *VMF* není nedefinován vnitřní objem, ale jednotlivé trojrozměrné geometrické objekty, které tento vnitřní objem obklopují. V průběhu kompilace tedy dochází ke ztrátě ploch, hran a bodů těchto geometrických objektů nazývaných *brush*, které se nedotýkají vnitřního objemu. Z tohoto odstavce tedy vyplývají dva poznatky. Za prvé, při tvorbě mapy ve *Valve Hammer Editoru* musí být vnitřní objem prostoru uzavřený a nesmí nikde „vytékat“ (přeloženo doslova z *leak* – termín používaný pro označení této chyby). Za druhé, proces kompilace není vratný. Dekompilace mapy sice možná je a existují na ní nástroje, dojde však k opětovnému vytvoření nových *brushů*, které se s původními nemusí shodovat [13]. Protože *BSP* není snadno čitelný a upravitelný, bude se tato práce nadále zabývat pouze formátem *VMF*.

Soubor *VMF* je v textovém formátu a je tedy snadno sekvenčně čitelný. V podstatě jde o výčet objektů a jejich vlastností (klíčových hodnot - *keyvalues*). Kód je rozdělen pomocí tagů a složených závorek. Tag uvozuje objekt a na následujícím řádku je pod tagem začátek složené závorky. Klíčové hodnoty nebo další tagy spadající pod nadřazený tag jsou odsazeny tabulátorem a název klíčové hodnoty a samotná hodnota jsou obě ve vlastních uvozovkách (jak řetězcové, tak číselné hodnoty). Celý tag je pak ukončen opět složenou závorkou. Výhodou je, že kód je takto velmi přehledný. Výpis 2.1 ukazuje část *VMF* kódu mapy. V tomto i následujících výpisech jsou odstraněny některé řádky, které se opakují nebo budou vysvětleny dále.

Na prvních místech jsou tagy s hodnotami pro účely editoru a evidence: informace o verzi editoru a souboru, seznam skupin objektů (*VisGroups*) a nastavení uložených pohledů. Následuje první speciální entita *Worldspawn* reprezentující statickou část mapy pod tagem *world*. Po ní přichází na řadu seznam všech ostatních entit (tag *entity*). Editor umožňuje schovávání objektů, což je ve *VMF* souborech řešeno umístění tagů reprezentujících objekty do nadřazeného tagu *hidden*. Schovávání objektů je možné realizovat i schováváním celých VisGroups, a to jediným kliknutím. Pro účely editoru může většina tagů obsahovat podřízený tag *editor*, kde je určeno, do jaké skupiny objekty patří a jakou barvu budou mít ve 2D a 3D pohledech jejich drátové modely (v pohledu *3D wireframe*). Všechny objekty typu *world*, *entity* a *solid* mají svoje pořadové číslo pojmenované jako *id*.

Obsah mapy je dělen na statické *brushe*, které spadají pod *Worldspawn*. Tyto *brushe* musí definovat již popsaný uzavřený objem mapy. Pod tagem *world* se nacházejí další hodnoty, z nichž nejpodstatnější pro účely popisované v této práci je hodnota pro volbu textury pro v nekonečnu promítanou oblohu označená jako *skyname*. Několik parametrů entity *world* následují jednotlivé *brushe* pod *tagem solid*. Hned za koncem tagu *world* začíná výčet jednotlivých entit (tag *entity*), jejichž struktura je shodná s *Worldspawn*. Entity začínají výčtem parametrů, kterými jsou definovány její nastavení a vlastnosti. Dále může (dle třídy

entity) následovat výčet *brushů*, které pod tuto entitu spadají. Editor rozděluje entity na *bodové* a *brushové* (*Point Entities*, *Brush Entities*) a běžné použití editoru nedovoluje jejich záměnu. Nelze tedy použít pro *brushovou entitu* třídu určenou pro *bodovou* a naopak. *Brushové* jsou takové *entity*, které ke své funkci vyžadují definovaný prostor a k tomu je využito právě přiřazených *brushů*. Na druhou stranu existují entity, které jsou definovány buď v bodě, nebo jejich umístění není vůbec podstatné. Následuje ukázka kódu *brushové entity* třídy *func_detail* (Výpis 2.2) a ukázka bodové entity třídy *info_player_start* (Výpis 2.3).

```
versioninfo
{
	"editorversion" "400"
	"editorbuild" "6238"
	"mapversion" "2"
	"formatversion" "100"
	"prefab" "0"
}
visgroups
{
	visgroup
	{
		"name" "skupina"
		"visgroupid" "5"
		"color" "164 181 218"
	}
}
viewsettings
{
	"bSnapToGrid" "1"
	"bShowGrid" "1"
	"bShowLogicalGrid" "0"
	"nGridSpacing" "64"
	"bShow3DGrid" "0"
}
world
{
	"id" "1"
	"mapversion" "2"
	"classname" "worldspawn"
	"skyname" "sky_day01_01"
	"maxpropscreenwidth" "-1"
	"detailvbsp" "detail.vbsp"
	"detailmaterial" "detail/detailsprites"
	solid
	{
		...
	}
}
entity
{
	...
}
```

Výpis 2.1. Obsah *VMF* souboru. Parametry nahrazené tečkami budou ukázány v dalších výpisech.

```
entity
{
	"id" "3"
	"classname" "info_player_start"
	"angles" "0 0 0"
	"origin" "-320 320 0"
	editor
	{
		"color" "0 255 0"
		"visgroupid" "5"
		"visgroupshown" "1"
		"visgroupautoshown" "1"
		"logicalpos" "[0 500]"
	}
}
```

Výpis 2.2. Ukázka *VMF* kódu *bodové entity* třídy *info_player_start.*

```
entity
{
	"id" "25"
	"classname" "func_detail"
	solid
	{
		"id" "23"
		side
		{
			"id" "24"
			"plane" "(512 832 64) (960 832 64) (960 512 64)"
			"material" "BRICK/BRICKFLOOR001A"
			"uaxis" "[1 0 0 0] 0.25"
			"vaxis" "[0 -1 0 0] 0.25"
			"rotation" "0"
			"lightmapscale" "16"
			"smoothing_groups" "0"
		}
		side
		{
			"id" "23"
			"plane" "(512 512 0) (960 512 0) (960 832 0)"
			"material" "BRICK/BRICKFLOOR001A"
			"uaxis" "[1 0 0 0] 0.25"
			"vaxis" "[0 -1 0 0] 0.25"
			"rotation" "0"
			"lightmapscale" "16"
			"smoothing_groups" "0"
		}
		side
		{
			...
			...
		}
		...
		...
	}
	editor
	{
		"color" "0 180 0"
		"visgroupid" "5"
		"visgroupshown" "1"
		"visgroupautoshown" "1"
		"logicalpos" "[0 1500]"
	}
}
```

Výpis 2.3. Výpis *VMF* kódu *brushové entity* třídy *func_detail. T*ečkami nahrazené parametry jsou až na souřadnice vektorů definující stěnu *brushe* stejné. Celkem má tento *brush* 6 stěn – side.

Zatímco struktura *bodových entit* je jednoduchá, protože jde pouze o výčet klíčových hodnot, u *brushů* i *brushových entit* je situace podstatně složitější. *Brushe* jsou definovány jako prostor ohraničený rovinami, přičemž průniky dvou těchto rovin pak tvoří jednotlivé hrany a průniky tří hranic vrcholy. Například kvádr je ohraničený šesti takovými rovinami. Z toho vyplývá jednak to, že nejsou přípustné oblé tvary (ostatně jako u drtivé většiny herní grafiky) a jednak to, že je možné definovat pouze konvexní tvary. Oblé hrany tedy musí být tvořeny vhodným množstvím plošek a konkávní útvary musí být tvořeny z více *brushů*.

Stěny *brushů* jsou definovány pod tagem *side*. Každá *side* má svoje vlastní číslování *id.* Může tedy existovat *solid* a *side* se stejnou hodnotou *id*, nikoli však dva *solidy* nebo dvě *sides* se shodným *id*. Stěna je geometricky definována množinou tří bodů, každý o třech kartézských souřadnicích (tedy 9 čísel) pod klíčem *plane*. Následuje jméno textury, která je na tuto stěnu nanesená pod klíčem *material*, definování směrů vektorů *U* a *V* a posun v jejich směrech pro *UV mapování* textury (*UV Mapping*) a měřítka v těchto dvou osách, rotace textury a parametry pro výpočet osvětlování, například *lightmapscale* (viz kapitola 2.1.6).

2.1.2 Entity

Jak již bylo řečeno v kapitole 2.1.1, v rámci *Source Engine* existují entity dvojího typu: *bodové* a *brushové* (*point entities*, *brush entities*). Tyto dvě skupiny se liší tím, jestli k jejich funkci nebo definici je potřeba definovat geometrii. Například entita reprezentující tlačítko (*func_button*) je *brushová entita*, geometrie definovaná pomocí *brushů* bude při aktivaci tlačítka ve hře pohyblivá. Naproti tomu *entita logic_auto*, která má za úkol spouštět ostatní entity po startu mapy, žádnou geometrii nepotřebuje a její třída je tím pádem určena pro *bodové entity*.

Atributy entit definují různé vlastnosti (například délka a směr zdvihu tlačítka, dobu jeho vynulování atd.) a jsou různých datových typů, nejčastěji celá čísla, reálná čísla, vektory, řetězce a boolean. Tyto atributy je možné nastavit vyvoláním vlastností označeného objektu ve *VHE* (klávesovou kombinací *Alt+Enter* nebo položkou *Properties* ve vyvolané nabídce po stisknutí pravého tlačítka myši) v záložkách *Class Info* a *Flags*. Třídám entit následně náleží vstupy a výstupy, které zajistí interakci entit mezi sebou. Akce je vyvolána výstupem (*Output*). Příkladem výstupu zmíněné entity *func_button* je *OnPressed* (při stisknutí), u *logic_auto* například *OnMapSpawn* (při načtení mapy). Vstupem entity je například otevřít dveře (*Open*) nebo rozsvítit světlo (*TurnOn*). Pokud má entita přijmout nějakou akci (vstup), pak samozřejmě musí být pojmenovaná – atribut *Name*. Entitám jsou definovány pouze výstupy. Akci tedy může vyvolat i nepojmenovaná entita. Výstup musí vždy cílit na jinou entitu.

Ve *VMF* souboru jsou výstupy entit definovány tagem *connections* v rámci tagu *entity*.

2.1.3 Textury

Textury jsou v rámci engine označovány jako materiály. Jedná se o několik bitmapových vrstev, z nichž nejdůležitější je základní textura (*albedo*, *base texture*). Dalšími vrstvami jsou například *bump mapa* nebo *detail mapa*. Všechny tyto bitmapy jsou uloženy v souborech vlastního formátu pro *Source Engine VTF* (*Valve Texture File*) a jsou samostatně nevyužitelné, protože *Source* jako texturu považuje materiál uložený v souboru textového formátu *VMT* (*Valve Material Type*). Tento soubor je textového formátu a obsahuje kód, který přiřazuje danému materiálu textury ve formátu *VTF* a určuje jeho funkci. Dalšími parametry je metoda referování povrchu nebo to, jaký reálný materiál (tzn. dřevo, ocel atd.) je reprezentován. Toto je důležité pouze z estetického hlediska, protože toto nastavení má vliv pouze na zvuky, které budou vydávány při fyzické interakci s povrchem, na kterém je tato textura nanesena – například při nášlapu nebo srážce s jiným objektem. *VMT* kód pro materiál s názvem *digitov/podlaha_dlazdice3modralesk* je uveden jako příklad ve Výpis 2.4. Je zde vidět, že je použita *bump mapa* (pro vytvoření reliéfu), *detailmapa* (pro vytvoření drsnosti při pohledu zblízka) a textura má možnost tvorby *environment mapy* (odráží okolní prostředí).

```
"LightmappedGeneric"
{
$basetexture "digitov/podlaha_dlazdice2modra"
$bumpmap "digitov/podlaha_dlazdice2bump"
$envmap env_cubemap
$envmapcontrast 1
$detail detail\noise_detail_01
$detailscale 7.74
$detailblendfactor 0.8
$detailblendmode 0
}
```

Výpis 2.4. Soubor materiálu *digitov/podlaha_dlazdice3modralesk* ve formátu *VMT*.

VTF může obsahovat i více snímků. Tyto snímky se můžou měnit v čase, případně textura může být volumetrická (více vrstev textury tvoří například otvor, který se renderuje trojrozměrně). Dále obsahuje *mipmapu*, tedy menší rozlišení uložené textury určené pro renderování z dálky.

VTF soubor je možné vytvořit programem *VTEX*, který je přiložený v *Source SDK*. Vstupem je soubor ve formátu *TGA* a program je ovládán pomocí spouštěcích parametrů. Uživatelsky velmi pohodlný nástroj je *VTFEdit*, který bude popsán dále.

2.1.4 Modely – popis

Modely pro *Source Engine* jsou uloženy v několika souborech vlastního formátu. Interpretace modelů enginem je poměrně složitá záležitost, jejíž znalost není pro tento projekt nutná, a proto se jimi tato práce detailně nezabývá. Přípony těchto souborů s modely jsou *MDL*, *VVD*, *ANI*, *VTX* a *PHY*. Soubory obsahují informace o fyzice modelů, síti (*mesh*), texturování, pohybech a animaci a mnoha dalších. Textury pro modely jsou jiné, než pro *world*. Liší se použitým *shaderem* a aby nedošlo k záměně, mají ve složce *materials* svou složky *models*. Jejich relativní cesta ve složce se hrou je tedy *materials/models*.

Pro tvorbu virtuálních světů postačí znát, jak je možné modely vytvářet a používat. Modely jsou vkládány do mapy pomocí entit tříd, jejichž název začíná na *prop_*. K dispozici je několik tříd s různým určením a vždy se jedná o třídy pro bodové entity.

Základní třídou je *prop_static*, pomocí které je možné do mapy vkládat statický model, se který nelze za běhu interagovat a nemůže se tedy zúčastnit žádných akcí. Tato třída entity nemá dokonce ani atribut *name*, čili nemůže figurovat jako cíl pro výstupy jiných entit. Entita jako taková během kompilace zaniká.

Pokud je potřeba vložení pohyblivého modelu, pak jednou z možností je entita třídy *prop_dynamic*. Tato třída však nemá možnost samostatného pohybu, ale je možné vytvořit pohybovou vazbu (*parent*) s jinými pohyblivými objekty, například s dveřmi. Entita pak sama o sobě nemá žádný stupeň volnosti, její pohyb je plně závislý na pohybu rodiče. Tato entita již může nést jméno a má několik typů vstupů, například přehrání některé z animací modelu.

Samostatně pohyblivý model respektující fyzikální zákony vkládá entita třídy *prop_physics*. Model pak reaguje na působící síly, které mu udělují zrychlení a je pohyblivý v šesti stupních volnosti. Stupně volnosti je možné odebírat pomocí entitních tříd s názvem začínajícím *phys_*. Je možné například definovat posuvnou vazbu, rotaci okolo kloubu nebo uvázání na lano.

Při tvorbě virtuálního světa je nutné pamatovat na to, že není možné používat modely libovolně pro kteroukoli entitu z výše popsaných tříd. Každý model má ve svých atributech určeno, pomocí které z těchto tříd může být vložen do virtuálního světa, což uživatel vidí při jeho výběru pomocí *Model vieweru*. Důvodem je, že při tvorbě fyzických modelů je třeba ještě definovat, která část modelu bude mít jakou hmotnost. Špatně použitá *prop_* entita se projeví tím, že model ve zkompilované mapě nebude existovat. V případě, že správný druh modelu není k dispozici, je stále možné je použít pomocí entit tříd *prop_dynamic_override* a *prop_physics_override*, která model převede na *dynamický* nebo na *fyzický* nebo naopak za cenu zvýšení výpočetních nároků na běh mapy.

Model se skládá z geometrie s nanesenou texturou, kterou zpracovává *render* a je to ta část modelu, která je viditelná. Další složkou je *kolizní geometrie*. Ta definuje prostor, který je neprostupný pro ostatní hmotné (*solid*) objekty a zpracovává ji fyzikální engine, tudíž viditelná není. Fyzikální engine zajišťuje realistické chování modelů při pohybu a srážkami s jinými modely, světem (zdmi, terénem apod.) nebo hráčem.

2.1.5 Modely – tvorba

V této kapitole je popsán nástin osvědčené metody, jak vytvářet nové modely a převádět existující z jiných formátů. Společnost Valve dodává s *SDK* nástroj pro kompilaci modelů z formátu *SMD* do formátu *MDL*. Existují volně dostupné moduly pro export do formátu *SMD* z programu *3ds MAX* pro verze *9*, dále *2008* až *2012* a to jak pro 32bitovou, tak pro 64bitovou verzi.

Původní model je možné vytvořit prakticky v jakémkoli programu, ze kterého je možné exportovat do takového formátu, který lze importovat ve *3dS MAX*. Po importu do tohoto programu je pomocí nástroje Collapse vytvořena tzv. *Editable Mesh*. Tím vznikne síť polygonů (*mesh*), která může vstoupit do kompilace *SMD* soboru. Nejdříve je ale nutné na jednotlivé polygony nanést texturu, jejíž jméno souboru je shodné se jménem souboru *VMT*, obsahující texturu vhodnou pro model. Po otexturování je již možné exportovat *SMD* model, k němu napsat řídící *QC* soubor a dále zkompilovat pomocí přiloženého programu *StudioMDL* v rámci *Source SDK*, nejlépe doplněného volně dostupnou grafickou nástavbou *GUIStudioMDL*.

Protože je tvorba modelu poměrně dlouhý a pracný proces náročný na popis, který není předmětem této práce, její autor doporučuje pro studium problematiky výukový e-book [14], který je zaměřen na tvorbu virtuálních prostředí za použití *Source SDK*. Problematice převodu a tvorby modelů je v tomto e-booku věnována celá kapitola.

2.1.6 Osvětlování

Osvětlování je důležitou součásti všech virtuálních modelů, protože správně osvětlený model je z estetického hlediska mnohem kvalitnější. V rámci *Source Engine* a herní grafiky obecně rozlišujeme dva hlavní způsoby osvětlování, a to statické a dynamické. Statické osvětlování je realizováno přímým zabarvením povrchových textur virtuálního světa při jeho kompilaci. Dynamické osvětlování je naproti tomu renderováno v reálném čase za běhu virtuálního světa.

Statické osvětlování probíhá během kompilace mapy a jeho výsledkem je *lightmapy*, v podstatě nová vrstva všech textur, která jednoduše povrchy zbarvuje. Kromě toho mapa uchovává i informaci v objemu prostoru mapy, která zajistí přímé osvětlení modelů. Pohyblivé modely jsou pak osvětlovány výhradně dynamicky.

Jako zdroje světla slouží entity tříd, jejichž název začíná na *light*. Existuje několik typů statických světel, z nichž nejpodstatnější jsou bodové, směrové a přirozené. Bodové světlo má na starosti entita třídy *light*. Dá se nastavit dosah, intenzita, barva světla a vzorec blikání. Zdrojem směrového světla je entita *light_spot*, která dědí všechny vlastnosti bodového světla, vyzařování je však omezeno na kužel, jehož vrcholový úhel je možné nastavit. Jinak se chová entita *light_environment*, která nastavuje přirozené (denní, zbytkové noční apod.) světlo. Přirozené osvětlování je realizováno rovnoběžnými paprsky vycházejícími z *faces* (stran *brushů*), které nesou texturu *tools/toolsskybox*. Tato speciální textura reprezentuje oblohu, která je renderována promítáním *albeda* (základní textura – base texture) v nekonečnu. Směr světla, stejně jako intenzitu a barvu, je možné nastavit touto entitou. Protože zdrojem světla není tato entita samotná, není její umístění v mapě podstatné.

Tyto tři entity se podepisují na podobě *lightmapy* po konečném zkompilování celé mapy. *Lightmapa* má čtyři vrstvy, mezi kterými lze v průběhu běhu mapy přepínat. Toto je značně omezující způsob osvětlování, protože engine takto neumožňuje, aby jeden *face* byl osvětlován více než dvěma na sobě nezávislými světly najednou. Použití více takových světel má za následek díky nedostatečné kapacitě lightmap nepřirozené přechody v osvětlování jednotlivých *faces*, viz Obr. 2.1. Další vlastnost *lightmap* je, že je možné přímo v editoru volit

jejich rozlišení, přičemž výchozí je 16 herních délkových jednotek *units*. Dle potřeby lze zadat jakoukoli mocninu dvou a dosáhnout tak ostrých stínů nebo jemných přechodů. Výhodou jemné *lighmapy* je, že statické osvětlení příliš nezatěžuje běh spuštěné mapy, ale příliš jemná síť se promítne do delší doby kompilace mapy. Vliv lightmapy na ostrost stínů je znárodněn na Obr. 2.2. Vlevo je na zdi nanesena textura s velikostí lightmapy 1, uprostřed 4 a vpravo 16.

Obr. 2.1. Nedostatečný počet vrstev lightmapy jako příčina nespojitosti osvětlení.

Obr. 2.2. Vliv rozlišení lightmapy na ostrost stínu.

Dynamické osvětlování je náročnější na výpočetní výkon počítače, ale neprodlužuje dobu kompilace. Jedná se o světla, která se mohou pohybovat. Jejich účinek je bohužel patrný pouze na modelech, nikoli na *faces* tvořených *brushi*. Dynamické světlo je reprezentováno například entitou *light_dynamic*. Hráč může dynamické světlo používat v podobě svítilny, která je k dispozici v původní hře *Half-Life 2* a tedy i při hraní map vytvořených pomocí sady *DIGITOV*.

2.1.7 Struktura souborů, mody

Většinu modifikací hry využívající *Source Engine* lze provést přidáním nových souborů nebo přepsáním existujících v adresáři hry. Protože tvůrci počítali s možností takovýchto modifikací, umožnili přidávání tzv. *add-onů* neboli *modů*, což je v podstatě složka existující paralelně s původní složkou hry, ve které jsou tyto nové soubory umístěny a nedojde tak ke změně původního obsahu. Pokud uživatel nenastaví umístění herních souborů mimo složku distribuční služby *Steam* a ponechá původní nastavení, pak herní soubory s původním obsahem nalezne ve složce *Steam\SteamApps\common\Half-Life 2*. Složka s modifikací je umístěna v rovněž v této složce. Systém souborů modifikací je shodný se složkami nemodifikované hry. Následuje seznam důležitých složek s popisem:

- *cfg (configuration)* – Složka obsahuje *CFG* soubory, které obsahují parametry s uživatelským nastavením hry. Je umožněno zde vkládat vlastní *CFG* soubory a pouštět je z konzole příkazem *„exec [jméno souboru]“*.
- *maps* – Zde se nachází všechny soubory ve formátu *BSP*, což jsou zkompilované mapy. Jednotlivé mapy lze z konzole spouštět příkazem *„map [jméno mapy]“*.
- *materials* – Tato složka obsahuje veškeré materiály (textury), tedy soubory *VTF* a *VMT*.
- *models* – Zde jsou umístěny veškeré modely *MDL*, které je možné vložit jako detailní součásti do virtuálního světa.
- *resource* – Různé texty, ikony pro GUI hry a příběhové titulky.
- *scripts* – Povětšinou soubory textového formátu sloužící pro nastavení skriptů a přiřazení zvuků jednotlivým akcím apod.
- *sounds* – Soubory s veškerými zvuky ve formátu *VAW*.

Protože není žádoucí, aby na discích bylo velké množství malých souborů (ty pro původní hru zabírají přes 8 GB), jsou tyto soubory komprimovány do souborů *VPK* (*Valve Pak File*). Herní soubory mají každý svoji vlastní prioritu, mohou tedy existovat dva se stejným názvem v různých *VPK* souborech, aniž by došlo ke konfliktu. Modifikace i původní hry jsou řízeny souborem *gameinfo.txt*, který se nachází v její kořenové složce. Soubor obsahuje seznam přiřazených *VPK* složek. Pořadím v tomto seznamu je určena priorita načítání souborů – pokud není požadovaný soubor umístěn ve složce modifikace, bude vyhledáván v jednotlivých VPK souborech dle pořadí určeném v *gameinfo.txt*, dokud není nalezen první výskyt.

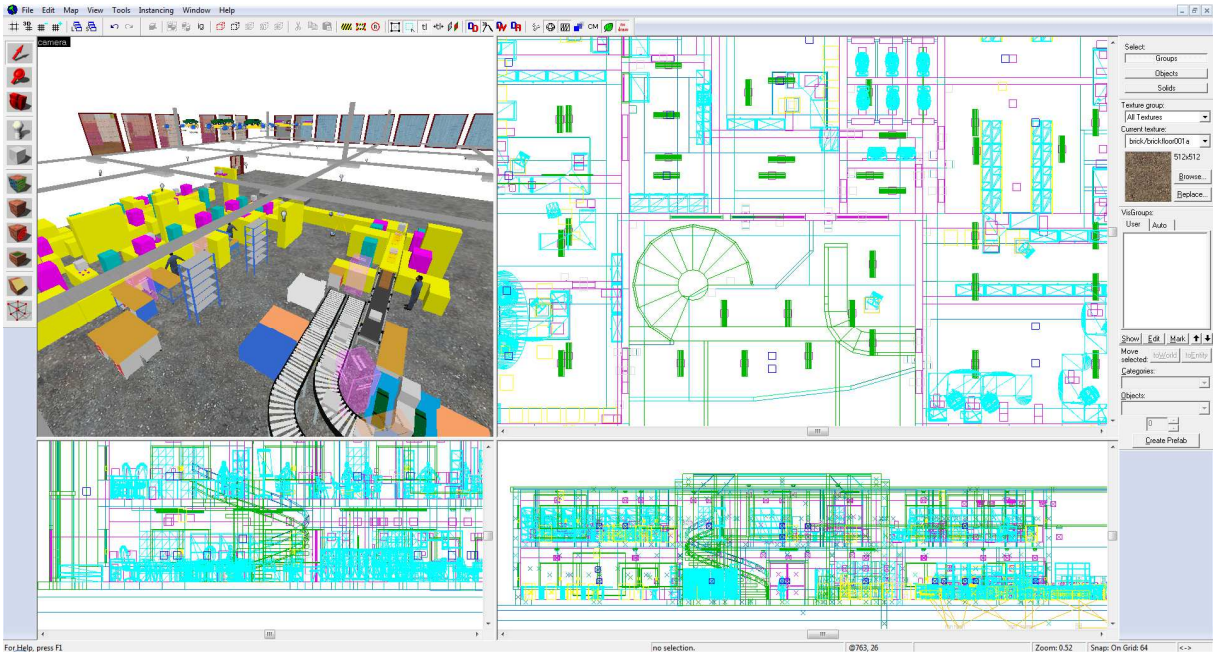
Balíček *DIGITOV* je umístěn ve složce *Steam\SteamApps\common\Half-Life 2\digitov*, ve které jsou veškeré soubory, které jsou nutné pro prohlížení tematických virtuálních prostředí a téměř všechny soubory nutné pro jejich tvorbu.

2.2 Používaný software

Tato podkapitola obsahuje popis softwarových produktů, které mohou být využity k tvorbě zdrojů pro vytváření virtuálních prostředí v *Source SDK*, a tedy i k vytvoření *DIGITOVu*.

2.2.1 Source SDK a Valve Hammer Editor

Již několikrát zmiňovaná oficiální vývojářská sada *Source SDK* je základním nástrojem pro tvorbu čehokoli v *Source Engine*. Její nejdůležitější součást je *Valve Hammer Editor* (*VHE*), což je oficiální level-editor pro *Source Engine* a je nástrojem pro tvorbu virtuálních prostředí. Tento program je uživatelsky sice docela přívětivý, jeho ergonomie však již zastarává. GUI tohoto programu se zásadně nezměnilo již prvních verzí od jeho předchůdce, programu *WorldCraft*, který existoval již v roce 1996 [15]. Program obsahuje nástroje na tvorbu geometrie mapy, texturování, tvoření entit a vazeb a mnoho dalších věcí umožňujících poměrně pohodlnou práci. Vše, co mapa může obsahovat, lze tímto programem vytvořit a pokud jsou k dispozici všechny zdroje jako textury, zvuky, modely atd., autor virtuálního prostředí nepotřebuje žádný další program. Náhled na program je na Obr. 2.3.



Obr. 2.3. Náhled modelu tvořeného ve *VHE*.

Mapu nelze spustit přímo ve *VHE* a musí projít procesem kompilace, jenž obstarávají tři programy v rámci *Source SDK*, a to *VBSP*, *VVIS* a *VRAD* v tomto pořadí. Spustitelná mapa ve formátu *BSP* vzniká již po zkompilování programem *VBSP*. Tento program přeloží *brushe* v geometrii srozumitelnou enginem, stejně tak entity zkompiluje do jiného kódu. Program *VVIS* slouží k optimalizaci renderování v reálném čase, kdy mezi jednotlivými *visleaves* (j.č. *visleaf*), tedy vnitřními objemy mapy, proběhne výpočet, zda jsou mezi sebou viditelné. Dochází tak k tomu, že geometrie v pozadí není zbytečně renderována. Poslední kompilační program, *VRAD*, spočítá statické osvětlení a *lightmapu*. Proces kompilace je schematicky znázorněn na Obr. 2.4.

Obr. 2.4. Schéma procesu kompilace mapy.

Dalšími programy *Source SDK* jsou prohlížeče modelů, různé konvertory, kompilátory a další nástroje. Většinou se jedná o programy bez grafického uživatelského rozhraní a jejich použití je díky tomu obtížnější. Většinou lze tyto programy nahradit nástroji, které jsou uživatelsky přívětivé a pocházejí od třetích stran pod jednou z volných licencí, často od nadšenců patřících do komunity *modderů* hry (*modder* je osoba vytvářející modifikace pro hru).

2.2.2 VTFEdit

VTFEdit je softwarový nástroj původem třetí strany ve freeware licenci, který nahrazuje program *VTEX* ze *Source SDK*. Jedná se o nástroj na tvorbu textur a materiálů, tedy *VTF* i *VMT* souborů. *VTEX* je ovládán pomocí parametrů při spouštění a tedy uživatelsky ne příliš přívětivě. *VTFEdit* obsahuje jednoduché, ale intuitivní grafické uživatelské rozhraní. Program umožňuje převést obrázek v jakémkoli běžném formátu do formátu *VTF* a automaticky k němu vytvořit *VMT* soubor [16].

VTFEdit nabízí výběr všech shaderů pro *Source Engine*, kompresních metod a dalších parametrů. Zároveň zabezpečuje podmínku velikosti textury, která stanovuje, že oba rozměry velikosti musí být mocninami dvou.

2.2.3 3ds MAX 2012

Tento program od společnosti *Autodesk* je světově známý nástroj pro tvorbu 3D modelů za účelem grafické tvorby. Autor této práce a ostatní autoři projektu jej využívají místo oficiálně podporovaného nástroje pro tvorbu modelů pro *Source Engine XSI Mod Tool*. Důvod je ten, že verze *XSI Mod Tool*, která je potřebná, již není podporována a *3dS MAX* je rozšířenější a je snazší pro něj najít podporu ve formě návodů a různých modulů. Další výhodou je, že podporuje import modelů většiny nejpoužívanějších formátů a v neposlední řadě také to, že *Autodesk* uvolnil svoje nástroje zdarma pro studenty [17]. *3dS MAX* vyžaduje plugin pro export *SMD* souborů, přičemž existují dvě varianty od třetích stran pro různé verze *3dS MAX* [18][19].

Modely jsou většinou převáděny z modelů ze zdrojů KPV, které byly vytvořeny v programu *Siemens NX 7.5*. Z něj jsou pomocí formátu *VRML* importovány do *3dS MAX*. Druhá možnost je stažení modelu z volně dostupných knihoven, například [20]. Ve *3dS MAX* jsou importované modely převedené na síť polygonů (*Editable Mesh*), nanesení textury a export do formátu *SMD*.

2.2.4 GUIStudioMDL

Obdobně jako *VTFEdit* nahrazuje původní nástroj ovládaný příkazovou řádkou *VTEX* pro převod textur, *GUIStudioMDL* přináší grafické uživatelské rozhraní pro program *StudioMDL*, který slouží ke kompilaci *SMD* souborů do konečného modelu ve formátu *MDL* a dalších doprovodných souborů [21].

SMD soubory obsahují geometrii nebo animace. Proces finální kompilace je, stejně jako u původního *StudioMDL* řízen *QC* souborem v textovém formátu, který nastavuje parametry výsledného modelu a jak nakládat s jednotlivými *SMD* soubory.

2.2.5 Microsoft Visual C++ 2010

Microsoft Visual C++ 2010 je vývojové prostředí nad programovacím jazykem *C++*, ve kterém je *Source Engine* naprogramovaný. Tento program bude použit ke kompilaci upraveného kódu, pokud bude potřeba přidat nějaké třídy entit nebo další funkce [22][23].

Toto vývojové prostředí bylo také použito k vývoji prototypu *VIS2VMF* pro převod layoutu podniku v programu *VisTable Touch* do zdrojového souboru mapy *VMF* (více v kapitole 5).

2.3 DIGITOV

DIGITOV je knihovna zdrojů pro tvorbu virtuálních prostředí s podnikovou tematikou v *Source Engine*. Formálně je modifikací počítačové hry *Half-Life 2: Episode Two*. Obsahuje nové modely, textury a objekty pro snadnou a rychlou tvorbu modelů výrobních a jiných podniků.

2.3.1 Využití DIGITOVu

Práce s balíčkem *DIGITOV* předpokládá použití standardních vývojářských nástrojů dodaných ke hře v ceně její uživatelské licence. Výhodou je, že tato licence se dá pořídit za 12,98 euro (na distribuční službě *Steam*) pro jednu osobu a je tak dobře dostupná. Druhou výhodou je ta, že základ hlavního vývojářského nástroje – level editoru *Valve Hammer Editor* – byl vytvořen členem komunity podle dnes již velmi staré hry *Quake* a vývojáři *Valve Software* jej adaptovali již pro první díl *Half-Life*. Nejedná se tak o nástroj určený pouze profesionálům, ale i běžným uživatelům, což lze vidět na snadnosti tvorby za jeho použití. Je

tak vhodný kromě snadné tvorby virtuálních prostředí i jako podpůrný nástroj k výuce problematiky virtuální reality.

Přímo za pomoci *Valve Hammer Editoru* jsou modelovány zdi budov, terén a další hrubá geometrie. Následuje vyplnění prázdných prostorů zařízením jako je nábytek, stroje, květiny atd. Tyto detailní části virtuálního světa se vkládají jako objekty, které slouží k promítání externích detailních modelů ve formátu *MDL*. Na řadu přichází interaktivní prvky, jako je osvětlení, ozvučení, skriptované sekvence (pohyby, činnost a řeč postav). Výsledkem je interaktivní virtuální prostředí, které je líbivé a dobře vypadá. Kromě textur a zmíněných modelů ve formátu *MDL* si tvůrce vystačí pouze se snadno ovladatelným level editorem.

Nevýhoda balíčku *DIGITOV* je ta, že *Valve Software* neumožňuje komerční využití enginu. To se vztahuje i na případné v něm vytvořené modely. Dle smlouvy *EULA* ke hrám série *Half-Life 2* a *Source SDK* umožňuje však osobní a výukové využití.

2.3.2 Popis instalace

První částí instalace je instalace samotných her *Half-Life 2*, *Half-Life 2: Episode Two* a sady vývojářských nástrojů *Source SDK Base 2013*. To lze provést pomocí distribuční služby *Steam*, kde je možné hry zakoupit dohromady za cenu 12,98 euro (stav k prosinci 2013) nebo je možné zakoupit edici *The Orange Box*, na základě které byl *DIGITOV* vytvořen. Licence pro *SDK* je pak k dispozici zdarma pro vlastníky licence jakékoli hry běžící na *Steamu*. Licence pro *Half-Life 2: Episode Two* je však nutná, protože *DIGITOV* využívá některé její zdroje.

Instalační balíčky *DIGITOVu* je možné stáhnout a nainstalovat pomocí programu *DIGITOV* Launcher, o němž pojednává tato práce v kapitole 6. Tento program je možné stáhnout z osobní univerzitní stránky [25] autora této práce a hlavního autora *DIGITOVu*.

2.3.3 Návod k použití

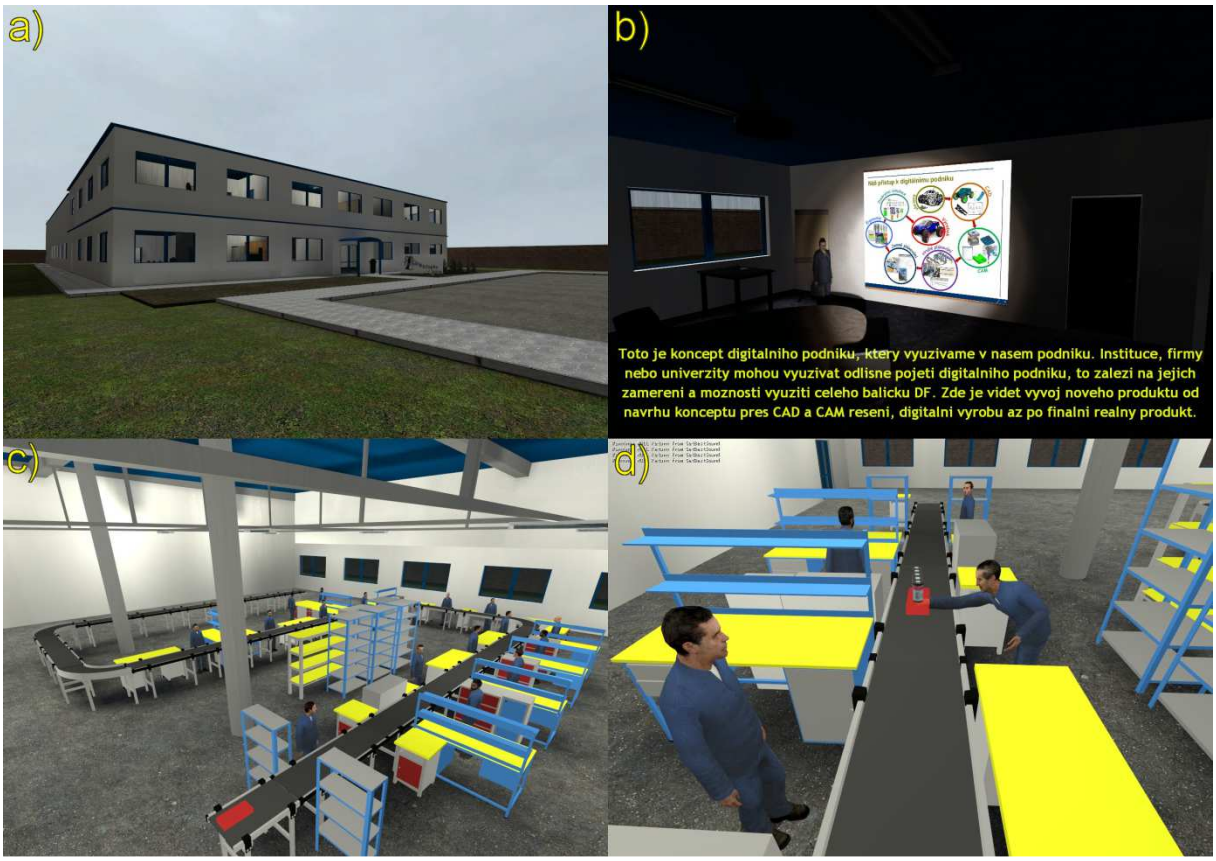
Spolu s vývojem *DIGITOVu* byl vytvářen autorem této práce, jejím vedoucím a dalšími kolegy výukový e-book *Tvorba virtuálních prostředí v Source SDK* [14].

3 Vývoj DIGITOVu

Balíček *DIGITOV* (zkratka slov digitální továrna) začal vznikat v roce 2010 na *KPV* a v současné době je knihovnou modelů, objektů a textur pro snadnou tvorbu virtuálních prostředí s průmyslovou a podnikovou tematikou. Od doby svého vzniku v něm přibyla spousta modelů a textur. Tato kapitola popisuje vývoj a historii *DIGITOVu* a jeho současný stav.

3.1 Demonstrační model

Projekt tvorby *DIGITOVu* předcházel jiný projekt, který měl za cíl potvrdit hypotézu, že k modelování výrobních podniků lze použít i počítačové hry. Vedoucí této práce na projektu začal pracovat začátkem roku 2010 a použil k tomu počítačovou hru *Half-Life 2: Episode Two* v edici *The Orange Box*. V létě 2010 začal na tomto projektu dále pracovat autor této práce, který jej dále rozvíjel.



Obr. 3.1. Demonstrační model vytvořený v *Source SDK*.

Zadání projektu bylo vytvořit za použití počítačové hry *HL2:EP2* model smyšleného výrobního podniku, kde se vyrábí modely aut na dálkové ovládání. Tento podnik byl navržen v rámci projektu integrujícího moderní metody *PLM* (*Product Lifecycle Management*).

Projekt začal převodem potřebných modelů pásových dopravníků, dílenského nábytku, textur a strojů. Následovalo dovybavení rozpracované hrubé stavby o nábytek a byly vytvořeny interaktivní prvky. Ukázka tohoto demonstračního modelu je na Obr. 3.1. Popisy jednotlivých obrázků následují:

a) Pohled na exteriér budovy podniku s parkovištěm.

b) Ukázka převedené Power Point prezentace do hry a titulky. Prezentace je realizována převodem snímků na materiály a střídáním viditelnosti *brushů*, kde je textura nanesena.
c) Pohled na výrobní linku, kde se vyrábí model auta na dálkové ovládání.
d) Interaktivní dopravník. Předmět na dopravníku je možné uchopit, zastavit tak linku a prohlédnout si jej. Pak lze výrobek vrátit na dopravník, čímž linka začne opět pracovat.

Protože byl demonstrační model úspěšný, bylo rozhodnuto projekt rozšířit. Výstupy projektu měly být nástroj pro výuku principů virtuální reality a návod použití ve formě e-booku vytvořeného v programu *ProAuthor 6*.

3.2 Vznik DIGITOVu

Existoval požadavek na snadno šiřitelnou knihovnu zdrojů pro *Half-Life 2: Episode Two*. Demonstrační model měl veškeré soubory nahrané přímo ve složkách se hrou, čímž byla ohrožena její původní funkčnost a navíc značně ztížena distribuce, instalace a odinstalace. Řešením bylo vytvořit modifikaci pro hru, která by byla ve zvláštní složce. Původní pouze pracovní název *DIGITOV* složený ze slov „digitální továrna“ se ujal a došlo tak k oficiálnímu pojmenování začátkem roku 2011 vzniklé modifikace.

Na rozšiřování knihoven se podíleli studenti předmětu *průmyslové inženýrství* (*KPV/PI*) v rámci dobrovolné výpomoci a převedli několik modelů z katedrálních zdrojů. Tito studenti byli Jan Berezňák, Olha Ilinykh, Ladislav Novák a Jiří Štěno.

Vedoucí práce, Ing. Petr Hořejší, PhD., vytvořil k *DIGITOVu* uživatelsky přívětivý instalátor. K instalaci *DIGITOVu* stačilo pouze zadat uživatelské jméno a cestu k distribuční službě *Steam*. Soubory se automaticky nakopírovaly a došlo k úpravě souboru *GameConfig.txt*, který řídí nastavení *Source SDK*.

3.3 E-book

Výukový e-book Tvorba virtuálních prostředí v *Source SDK* byl z větší části napsán autorem této práce, na jeho vzniku se dále podíleli Jiří Běl, Lukáš Fiedler, Ondřej Kolman, Jan Pomahač a Kamil Štěpnička. Obsahuje návod k použití *Source SDK* spolu s *DIGITOVem*, včetně návodu na zprovoznění, tvorbu map, převody modelů a tvorbu textur. Cílový čtenář tohoto e-booku je začátečník v této problematice, a proto obsahuje snadno srozumitelné texty doplněné obrázky a videi.

E-book je k dispozici studentům předmětu *KPV/DPVR* (*digitální podnik a virtuální realita*), kteří jako semestrální práci tvoří virtuální prostředí s jimi zvolenou tematikou za použití knihovny *DIGITOV*.

3.4 Aktualizace Source Engine MP na podzim 2011

V souvislosti s vydáním *Source Engine MP* byly přesunuty soubory *Source Engine 2009* do jiné složky a do původní složky umístěna právě verze *MP*. Oba enginy byly s *DIGITOVem* nekompatibilní a modifikace musela být upravena spolu s instalátorem.

3.5 Aktualizace Steampipe v létě 2013

Další významnou aktualizací bylo jednak vydání *Source Engine 2013*, což přineslo výhled k úpravě *DIGITOVu* pro novou verzi *Source Engine*. Práce na této úpravě byly urychleny aktualizací *Steampipe*, která spočívala v úplné změně systému souborů *Half-Life 2* včetně

druhé epizody. Byla také lehce změněna struktura samotného nástroje *Source SDK*. Instalátor přestal být zcela kompatibilní. *DIGITOV* byl po úpravě zprovozněn pro novou verzi enginu. Vzhledem k tomu, že předmět *KPV/DPVR* nebyl v letošním školním roce pro malý počet studentů na celé katedře otevřen, nebyl tlak na tvorbu nového instalátoru, která se uskutečnila začátkem roku 2014. Nový instalátor bude podrobně popsán v kapitole 6.

3.6 Současný stav

Obsah *DIGITOVu* a výukového e-booku ukazuje Tab. 3.1.

Balíček DIGITOV	
modelů	116
textur pro mapy	28
textur pro modely	44
statických objektů	8
interaktivních objektů	20
velikost v MB (komprese ZIP)	40
velikost v MB (bez komprese)	158
Výukový e-book	
kapitol	5
článků	50
obrázků	189
videí	6
slov	27 649
stran neupraveného RTF výstupu	186
velikost e-booku v MB	165

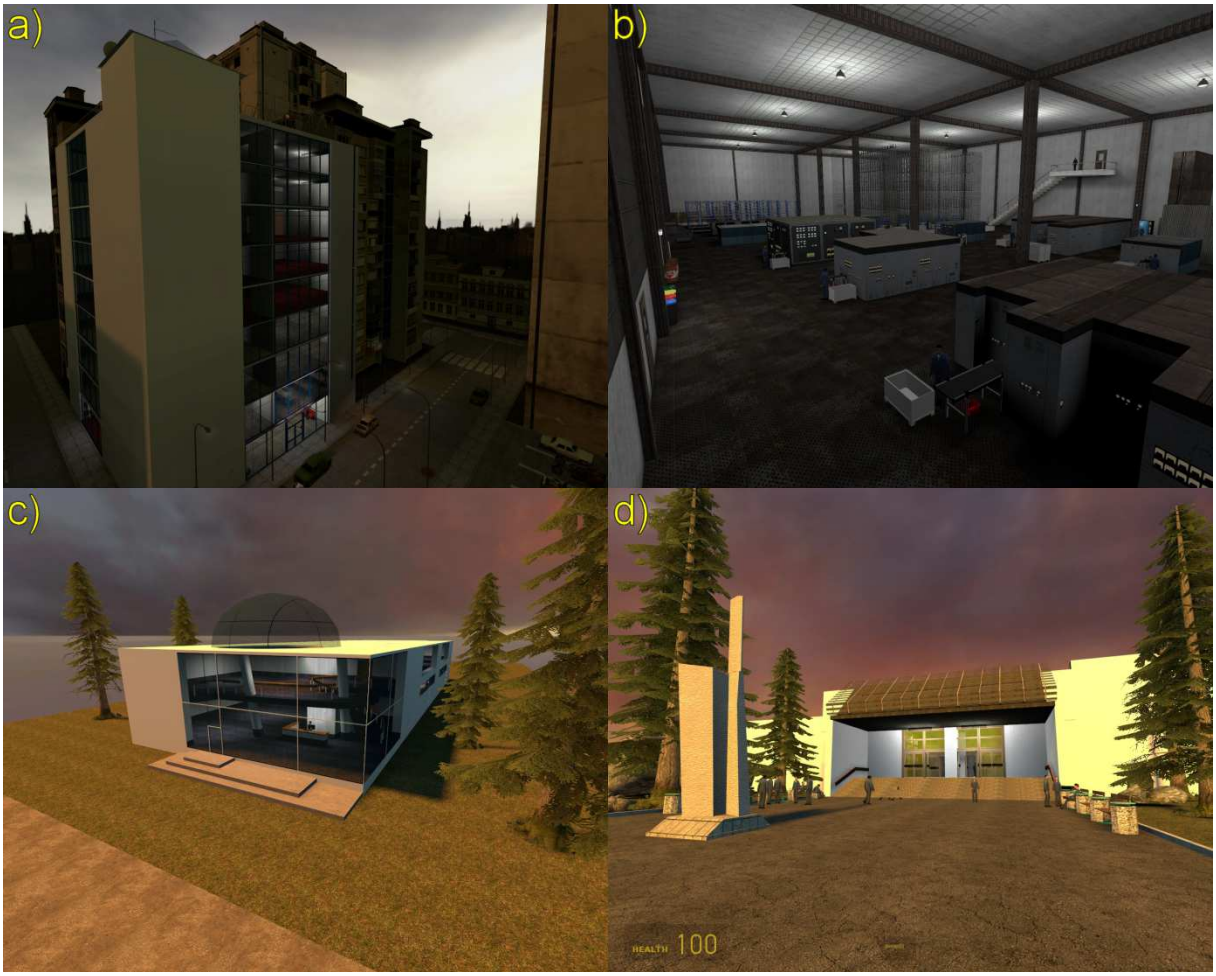
Tab. 3.1 - aktuální statistika obsahu *DIGITOVu*

4 Oblasti využití DIGITOVu

V současné době je *DIGITOV* využíván především jako výukový nástroj v problematice tvorby virtuálních prostředí a nahrazuje profesionální programy, které jsou drahé a poměrně náročné na naučení [4]. Takto je *DIGITOV* využíván v předmětu *KPV/DPVR*. Za stejným účelem byl využit v rámci *Letní školy virtuální reality* konané v létě 2013 určené pro studenty středních a vysokých škol. Katedra jej také používá k prezentaci své činnosti na veřejných akcích, například na *Dnech vědy a techniky* konaných každoročně v Plzni.

4.1 Předmět KPV/DPVR

Cílem předmětu je seznámit studenty pátého ročníku oboru Průmyslové inženýrství a management navazujícího studijního programu s principy a možnostmi využití virtuální reality a tvorby prostředí. Jak už bylo řečeno a v [4] vysvětleno, pro začátečníky je poměrně složité se v problematice zorientovat a pro tyto účely bylo nutné najít snadno použitelný vývojářský nástroj. Výhodou herních *SDK* je ta, že jsou určeny pro širokou veřejnost a nejen pro profesionály. Použití *DIGITOVu* je tak při výuce pro podobnost s principy profesionálních programů vhodné.



Obr. 4.1. Ukázky semestrálních prací studentů *KPV/DPVR*.

Jako semestrální práci studenti v tomto předmětu tvoří virtuální prostředí dle jimi zvoleného tématu a prezentovat svoje dílo v laboratoři virtuální reality typu CAVE na KPV. Předmět doposud proběhl již dvakrát, studenti jej hodnotili velmi pozitivně a probíraná problematika i

práce na semestrálních pracích je bavily, což dokazují jejich odpovědi v anketě pro hodnocení kvality výuky.

Studenti se za užití *Source SDK* naučili základy tvorby virtuálních prostředí, což dokazuje Obr. 4.1, na kterém jsou výsledky jejich prací:

a) Autor Bc. Jakub Jirsa vytvořil kus města s výškovou administrativní budovou, kde sídlí podnik.
b) Model Bc. Milana Krpejse znázorňuje halu výrobního podniku s velkými obráběcími stroji.
c) Bc. Jiří Šrajbr vytvořil model moderní budovy, v jejímž přízemí je výrobní linka a v patře administrativní část.
d) Část exteriéru budovy Fakulty strojní a Fakulty aplikovaných věd v podání Bc. Jana Mareše.

4.2 Letní škola virtuální reality

Stejně jako v předmětu *KPV/DPVR* se posluchači *Letní školy virtuální reality* (*LŠVR*) seznamovali s principem tvorby virtuálních prostředí během tří dnů z celkových dvou týdnů této akce, jejíž obsah zahrnoval i některé oblasti z tohoto předmětu. Účastnilo se pět posluchačů a několik hostujících lektorů, včetně autora této práce Obr. 4.2. Akce garantována vedoucím této práce, Ing. Petrem Hořejším, PhD., opět sklidila velmi kladné hodnocení.

Obr. 4.2. Výuka při použití *DIGITOVu* na *LŠVR* v létě 2013.

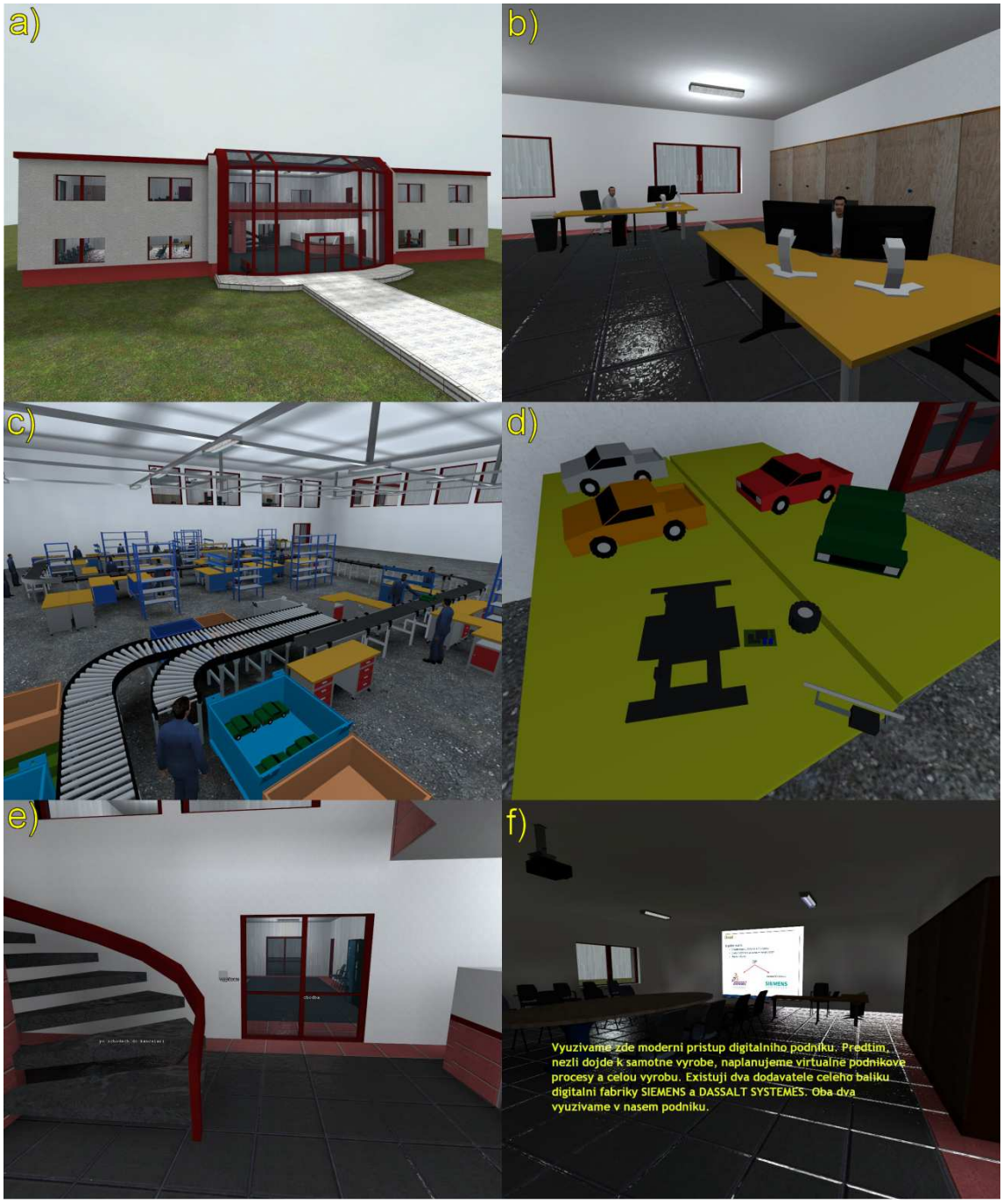
4.3 Středoškolská odborná činnost

V současné době soutěží tři studenti *Střední průmyslové školy v Plzni* se svým virtuálním modelem, na kterém v současné době ještě pracují. Cílem je vytvořit interaktivní a velmi detailní model počítačové laboratoře na jejich škole za využití *DIGITOVu* a *Source Engine*.

4.4 Dny vědy a techniky

Na dnech vědy a techniky katedra KPV organizovala expozici *Laboratoř virtuální reality*. *DIGITOV* zde byl prezentován jako jedna z možností tvorby virtuálních prostředí a k demonstraci byl použit model, který vytvořil autor této práce. Jedná se o nový výrobní podnik, kde se opět vyrábí auta na dálkové ovládání, jehož náhled je na Obr. 4.3. Cílem modelu bylo znovu vytvořit model použitý při demonstraci *DIGITOVu* (viz kapitola 3.1) tak, aby byl estetičtější a více interaktivní:

a) Pohled na budovu zvenku. Oproti demonstračnímu modelu je designově vymodelována prosklená vstupní hala.
b) Kanceláře jsou obsazené pracovníky a byly k dispozici lepší modely pro vybavení kanceláří.
c) Výrobní linka ztratila možnost sundat z pásu rozpracovaný výrobek, ten ale na druhou stranu projde celou linkou a je lépe znázorněna montáž.
d) Místo toho je ale možné brát jednotlivé díly a hotová auta, která jsou rozmístěna na stolech.
e) Při stisknutí pravého tlačítka myši dojde k zobrazení nápovědy, která upozorňuje na interaktivní prvky a pomáhá k orientaci.
f) Stejná prezentace, jako v případě demonstračního modelu na Obr. 3.1 b).



Obr. 4.3. Model určený pro Dny vědy a techniky. Tento model je přiložen k balíčku *DIGITOV* jako demo.

5 Konvertor z VisTable

Při práci s balíčkem *DIGITOV* citelně chybí převod již existujících podnikových layoutů vytvořených v jiných programech do formátu *VMF*, což by vedlo k ušetření práce, která již byla vykonána tvorbou layoutu právě v takovém programu. Vyřešení tohoto problému znamená vyvinout algoritmizovaný převaděč formátu. Pro ověření možností takového algoritmizovaného převodu bylo rozhodnuto o vývoji prototypu takového převaděče.

Vstupem programu budou layouty vytvořené programem *VisTable Touch*. Tento program je na KPV využíván k tvorbě podnikových layoutů a jsou tedy k dispozici reálná data k posouzení vhodnosti výstupů. *VisTable Touch* umožňuje pomocí nástroje *List of Equipment* (v menu na Obr. 5.1) exportovat layout jako seznam zařízení, jeho polohy, rozměrů a úhlu natočení do formátu *CSV* odděleného středníkem, který je čitelný a je možné jej snadno programově zpracovávat.

Obr. 5.1. Umístění výpisu vybavení (červeně orámováno, *List of Equipment*) v nabídce programu *VisTable Touch*.

Zdrojový soubor mapy *VMF* je rovněž čitelný a tak postačí poměrně jednoduché přeformátování textu v případě entit a výpočet všech parametrů pro *brushe*. Zároveň jsou předpokládány potíže v přesnosti převedených layoutů ve formátu *VMF* a oproti *VisTable Touch* způsobené nemožností jednoduše měnit rozměry modelů v *Source SDK*. Ve *VisTable* je možné všechny modely rozměrově upravovat ve všech třech směrech. V modelové situaci uživatel upraví délku jednoho ze segmentů pásového dopravníku zkrácením, aby se vešel mezi dva jiné, aniž by došlo k jejich překrytí. Toto zkrácení se v layoutu převedeného do *VHE* neprojeví a dopravníky se tedy překrývat budou. Další problém pravděpodobně vznikne při velkém množství převedených *brushů*, které vytvoří obrovské množství *visleaves*, jehož důsledkem budou velmi dlouhé časy při kompilaci programem *VVIS*. Tento krok kompilace mapy je však možné vynechat.

Zápis modelů nebo *NPC* (*Non-Player* Character) do *VMF* souboru je podstatně jednodušším procesem než zápis *brushů*, protože v prvním případě jde o bodovou entitu s udanou polohou a její zápis je pouhých několik řádek. V případě *brushů* je nutné vypočítat polohy bodů určující roviny stěn, tedy polohy bodů všech vrcholů *brushe* a následně ještě směry *UV vektorů*, které definují lokální souřadný systém pro rastr jednotlivých *texelů*.

Vývoj programu začal jednodušší částí – z každé položky v seznamu vybavení vytvořit entitu třídy *prop_static* s nějakým modelem. Na základě zkušeností a postřehů z vývoje této první části byl navržen algoritmus pro tvorbu a zápis kódu reprezentujícího *brush*. Další podkapitoly podrobněji popisují vývoj programu pro popisovanou konverzi. Pro vývoj tohoto

programu bylo použito vývojové prostředí *Microsoft Visual C++ 2005 Express Edition*, konečná úprava kódu proběhla ve verzi *2010*.

5.1 Analýza vstupních a výstupních formátů souborů

Ještě před tím, než začaly veškeré jiné práce na tomto dílčím projektu, bylo nutné zjistit, zda vůbec *VisTable Touch* nebo jiný vhodný program pro tvorbu layoutů má možnost jeho textového čitelného výstupu. V případě *VisTable Touch* se tato funkce nachází v menu jako položka *List of Equipment* (viz Obr. 5.1). Výstupem je soubor ve formátu *CSV* oddělený středníkem. Náhled tohoto seznamu vybavení je na Obr. 5.2. Původní layout ve *VisTable* je na Obr. 5.3.

Name	Position X	Position Y	Position Z	Width	Length	Height	Angle	Area [m²]
Administrativa	18,967 mm	69,033 mm	0 mm	19,533 mm	29,933 mm	10 mm	0	584.7
Flachdach_Brettschichtträger.3	56,250 mm	69,000 mm	4,000 mm	30,500 mm	13,500 mm	2,778 mm	90	411.7
Fläche 10x10 m.5	69,138 mm	70,908 mm	0 mm	34,183 mm	8,317 mm	10 mm	90	284.3
Wand Ziegel weiss 6000x6000mm.23	60,751 mm	53,876 mm	2,500 mm	4,000 mm	249 mm	2,500 mm	0	1.0
Wand Ziegel weiss 6000x6000mm.10	51,001 mm	53,876 mm	0 mm	23,500 mm	249 mm	2,500 mm	0	5.9
Wand Ziegel weiss 6000x6000mm.27	46,760 mm	53,876 mm	2,500 mm	24,000 mm	249 mm	2,500 mm	0	6.0
Wand massiv grau 6m.9	21,161 mm	60,730 mm	2,000 mm	1,000 mm	150 mm	1,000 mm	90	0.2
Sicherheitsbereich.1	59,640 mm	89,500 mm	0 mm	4,500 mm	5,000 mm	2,500 mm	0	22.5
Fläche 10x10 m.4	48,200 mm	89,517 mm	0 mm	40,400 mm	6,550 mm	10 mm	0	264.6
Kreis 10 m	68,310 mm	87,800 mm	0 mm	10,000 mm	10,000 mm	15 mm	0	78.5
Treppe lichtgrau x1.1	24,700 mm	85,625 mm	1,000 mm	2,751 mm	500 mm	50 mm	90	1.4
Wand Ziegel weiss 6000x6000mm.92	9,125 mm	57,000 mm	2,500 mm	6,000 mm	249 mm	2,500 mm	90	1.5
Wand Ziegel weiss 6000x6000mm.91	9,125 mm	63,000 mm	2,500 mm	6,000 mm	249 mm	2,500 mm	90	1.5
Wand Ziegel weiss 6000x6000mm.34	19,751 mm	53,876 mm	2,500 mm	6,000 mm	249 mm	2,500 mm	0	1.5
Wand Ziegel weiss 6000x6000mm.55	14,000 mm	84,125 mm	0 mm	6,000 mm	249 mm	1,000 mm	0	1.5

Obr. 5.2. *List of Equipment* modelu továrny.

Obr. 5.3. Pohled na původní layout ve *VisTable Touch*.

Je tedy zřejmé, že exportovaný seznam vybavení obsahuje veškerá data, která budou nadále potřeba. Jsou to jména zařízení a jednotlivých objektů, jejich poloha (souřadnice a natočení) a rozměry. Údaj, který je zde navíc, je půdorysná plocha. Po otevření *CSV* souboru v textovém editoru byla odhalena komplikace spočívající v tom, že každý jednotlivý znak je pravidelně střídán jakýmsi balastním znakem o ASCII kódu 0 (viz. Obr. 5.4, který vyplývá z použitého kódování *Unicode*. Jazyk *C++* bohužel nemá v základních knihovnách funkce pro správné čtení textu ze souboru v tomto kódování.

Pro vývoj programu toto bylo vyřešeno v programu *Microsoft Excel* jednoduchým překopírováním dat a uložením nového souboru *CSV*. Pro finální verzi bude třeba tento krok programově vyřešit. Program by zároveň měl rozpoznat, zda jsou přítomny tyto znaky nebo ne a podle toho vstupní data upravit. Důvodem, proč by se některé vstupní soubory měly lišit je to, že uživatel může *CSV* soubor ručně upravit a autor si není jistý, zda tento problém nastane na různých počítačích a různých systémech.

```
N a m e ; P o s i t i o n   X   [ m m ] ; P o s i t i o n   Y   [ m m ] ; P o s i t i o n   Z
[ m m ] ; W i d t h   [ m m ] ; L e n g t h   [ m m ] ; H e i g h t   [ m m ] ; A n g l e   [
° ] ; A r e a   [ m ˛ ] ;

 F l ä c h e   1 0 x 1 0   m . 2 ; 8 0 5 0 0 ; 1 0 0 5 0 0 ; 0 ; 3 0 0 0 ; 9 0 0 0 ; 1 0 ; 0 ;
2 7 ;

D e x i o n   S c h r a u b r e g a l   4 e r . 4 4 ; 1 0 8 2 5 5 ; 5 3 2 7 8 ; 2 2 4 0 ; 1 4
5 0 ; 8 0 0 ; 2 5 4 0 ; 0 ; 1 . 2 ;

W a n d   m a s s i v   g r a u   6 m . 2 1 0 ; 1 3 7 9 9 0 ; 5 2 8 4 7 ; 3 0 0 0 ; 9 0 0 0 ;
3 8 0 0 ; 1 0 0 ; 0 ; 3 4 . 2 ;

 W a n d   6 m . 4 7 ; 1 3 7 9 8 4 ; 5 2 8 4 6 ; 6 0 0 0 ; 9 0 0 0 ; 3 8 0 0 ; 1 0 0 ; 0 ; 3 4
. 2 ;

 T ü r   1 m . 1 0 7 ; 1 4 3 1 6 9 ; 8 7 8 4 6 ; 3 0 0 0 ; 1 0 0 0 ; 1 2 0 ; 2 0 0 0 ; 2 7 0 ;
0 . 1 ;

 W a n d   6 m . 1 0 ; 1 4 3 1 4 3 ; 8 9 5 4 1 ; 3 0 0 0 ; 4 4 2 0 ; 1 0 0 ; 3 0 0 0 ; 9 0 ; 0
. 4 ;

F e n s t e r   0 . 1 3 0 ; 2 0 9 4 1 0 ; 9 0 1 1 7 ; 1 0 0 0 ; 8 0 0 ; 1 0 0 ; 1 5 0 0 ; 0 ;
0 . 1 ;
```

Obr. 5.4. Ukázka seznamu vybavení ve formátu *CSV* v kódování *Unicode*.

Objekty stejného typu nebo jména ve *VisTable Touch* jsou rozlišeny číselnou příponou za tečkou. Takové zakončení jména může způsobit problémy a bude nutné tyto přípony odstranit.

Struktura souborů *VMF* byla představena v kapitolách 2.1.1 a 2.1.2 včetně popisu zápisu *brushů*. Protože všechny *brushe* jsou zapsány pod entitou *world* a všechny ostatní entity samostatně, je nutné do výstupní mapy nejdříve zapsat všechny *brushe* najednou a teprve po nich jednotlivé entity. Podstatné pro program je také to, že *Source Engine* nepodporuje *brushe* menší, než 1 *unit*. Toto se musí zohlednit, jestliže některá položka bude v některém z rozměrů menší.

Další potřebné soubory poskytují informaci, jak s každou jednotlivou položkou seznamu vybavení naložit. Je důležité rozhodnout, zda se má položka převést jako pracovník reprezentovaný bodovou entitou třídy *npc_citizen*, statický model *prop_static* včetně jména modelu nebo *brush* včetně jména textury. V případě modelů nemusí odpovídat střed modelu ve formátu *MDL* a jeho natočení s modelem objektu ve *VisTable Touch*, budou proto nutné

korekce polohy. Tato data budou uložena v jednotlivých souborech jména shodného se jménem položky ze seznamu vybavení a příponou *DAT*. Následuje výpis souboru *Wand_massiv_grau_6m.dat* (Výpis 5.1).

```
DIGITOV/stena_barva1bila brush 0 0 0 0
```

Výpis 5.1. Obsah souboru *Wand_massiv_grau_6m.dat.*

Jméno tohoto souboru, kromě číselné přípony, odpovídá jménu položky v seznamu vybavení. Pro každý druh položky tedy existuje jeden soubor *DAT*. Obsahem je několik parametrů oddělených mezerou, takže je v *C++* lze pohodlně číst příkazem *cin*. Druhý parametr je řetězec, který kromě druhu položky určuje účel prvního řetězce dle následujícího seznamu:

- *brush* – Položka se přeloží jako *brush*, na kterém bude nanesena textura určená prvním řetězcem.
- *model* – Položka bude přeložena jako bodová entita třídy *prop_static* s modelem určeným prvním řetězcem (musí se jednat o statický model).
- *npc* – Položka bude přeložena jako bodová entita třídy *npc_citizen* s modelem určeným prvním řetězcem (musí se jednat o model pro *NPC*).
- *alias* – Tento *DAT* soubor odkazuje na jiný *DAT* soubor jména určeného prvním řetězcem.
- *skip* – Tato položka bude přeskočena, nebude převedena vůbec.

Je možné, že soubor *DAT* neexistuje. V tom případě je položka automaticky přeložena jako *brush*.

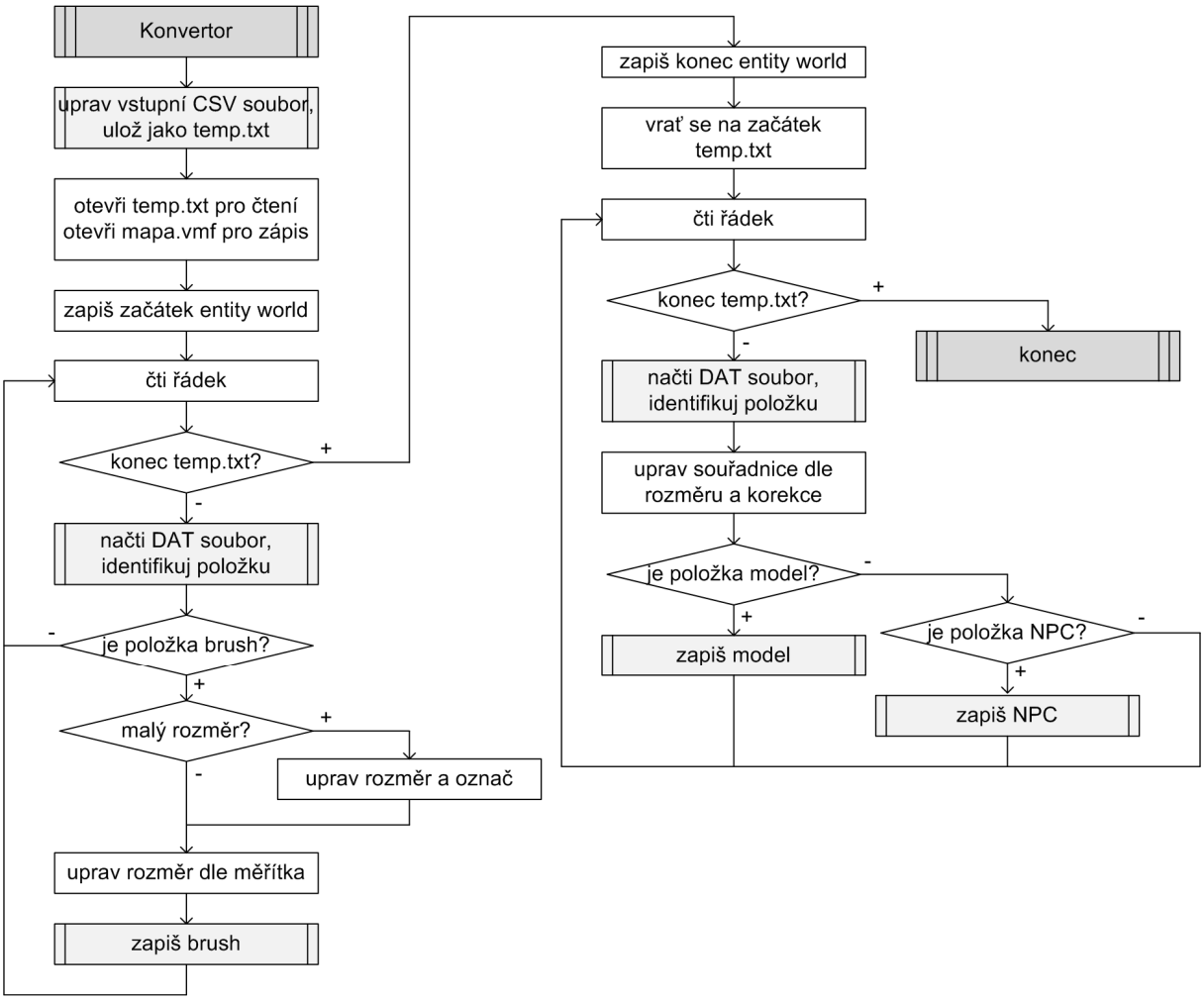
Další parametry jsou celočíselné a obsahují polohovou korekci. První dvě čísla jsou pro půdorysnou korekci středu modelu, třetí číslo je korekce pro výškovou korekci středu podstavy modelu a poslední číslo je korekce úhlu natočení. Jedná se tedy o vektor, který bude přičítán k poloze právě zapisované položky.

5.2 Návrh algoritmu

Na základě předchozí podkapitoly byl sestaven velmi hrubý, rámcový algoritmus bez detailního popisu provedení jednotlivých kroků. Je to z toho důvodu, že autorovy zkušenosti s programováním v *C++* byly před vývojem tohoto programu minimální a před začátkem vývoje předpokládal, že jej bude muset průběžně měnit v závislosti na možnostech a výbavě jazyka pro řešení jednotlivých dílčích úloh. Navržený algoritmus je na Obr. 5.5.

5.3 Zápis bodových entit do VMF souboru

V případě zápisu bodových entit, tj. pokud je načten soubor *DAT* s řídícím řetězcem *model* nebo *npc*, je postup řádově jednodušší než v případě *brushů*, proto byl tento případ realizován jako první. V podstatě stačí přepočítat souřadnice z milimetrů do *units*. To se provede jednoduchým vynásobením koeficientem *0,052* (*52 units* přibližně znázorňuje 1 metr). Zápis do souboru provede procedura *zapis_model*, respektive *zapis_npc* se vstupními parametry pozice, korekce pozice a jméno modelu. Vlastní realizace zápisu je několik tvrdě naprogramovaných řetězců, nemá tedy smysl vypracovávat vývojový diagram. Procedury je možné prohlédnout v příloze I této práce, která obsahuje výpis celého programu.



Obr. 5.5. Algoritmus programu pro generaci mapy ve formátu *VMF* ze seznamu vybavení z programu *VisTable Touch*

5.4 Zápis brushů do VMF souboru

Pro zápis *brushe* je nejprve nutné provést výpočet souřadnic všech jeho osmi vrcholů. Všechny rozměry a pozice je nutné převést z metrů do *units*, což je opět provedeno vynásobením všech rozměrů koeficientem 0,052. Z pozice středu, rozměrů a úhlu natočení je proveden výpočet souřadnic všech jeho osmi vrcholů. Vztah pro tento výpočet vyplývá z rovnice transformace ve dvourozměrném prostoru (1). Program i zadané parametry uvažují pouze rotaci kolem svislé osy, body nad sebou tedy mají dvě souřadnice stejné. Autorovu zavedenou konvenci číslování vrcholů *brushů* ukazuje Obr. 5.6.

$$X = q\,R\,X_0 + k = q \begin{pmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a \\ b \end{pmatrix} \qquad (1)$$

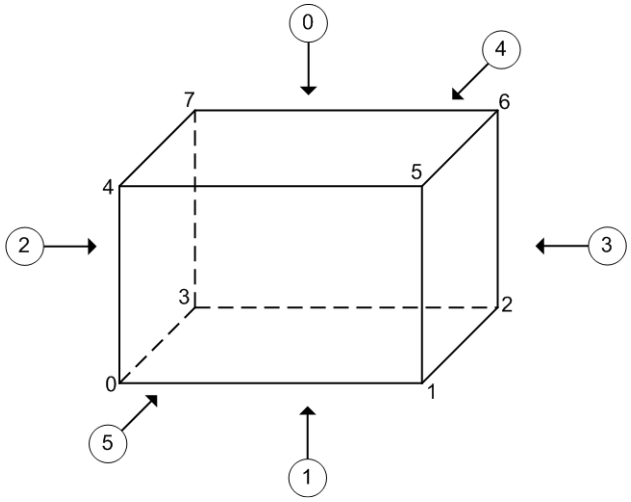
X … bod po transformaci
X_0 … bod před transformací
q … měřítko
R … matice rotace
β … úhel natočení
k (a, b) … posun [28]

V tomto případě jsou rozměry již vynásobené měřítkem při vstupu do procedury, tedy *q = 1*. Z rovnice (1) vyplývají následující v *C++* interpretované vztahy dle Výpis 5.2.

```
double s = sin(pfi/180*3.14159265359), c = cos(pfi/180*3.14159265359);
double bod[8][3],vektor[4];
bod[0][0] = bod[4][0] = px - lx/2*c + ly/2*s;
bod[0][1] = bod[4][1] = py - ly/2*c - lx/2*s;
bod[1][0] = bod[5][0] = px + lx/2*c + ly/2*s;
bod[1][1] = bod[5][1] = py - ly/2*c + lx/2*s;
bod[2][0] = bod[6][0] = px + lx/2*c - ly/2*s;
bod[2][1] = bod[6][1] = py + ly/2*c + lx/2*s;
bod[3][0] = bod[7][0] = px - lx/2*c - ly/2*s;
bod[3][1] = bod[7][1] = py + ly/2*c - lx/2*s;
bod[0][2] = bod[1][2] = bod[2][2] = bod[3][2] = pz;
bod[4][2] = bod[5][2] = bod[6][2] = bod[7][2] = pz + lz;
```

Výpis 5.2. Realizace transformačních vztahů dle (1) v C++, kde:
px, py, pz … pozice v *units*,
lx, ly, lz … velikost v *units*,
pfi … úhel natočení ve stupních,
bod[Č][S] … S-tá souřadnice Č-tého bodu,
Č … číslo bodu (slouží k jeho identifikaci)
S … číslo souřadnice – 0, 1, 2 pro x, y, z

Souřadnice všech vrcholů jsou nyní uloženy v poli reálných čísel *bod[Č][S]*. Na Obr. 5.6 odpovídají čísla vrcholů parametru *Č*, v kroužku jsou pak označeny čísla stěn. Nyní lze přistoupit k zápisu jednotlivých stěn, z nichž každá je definována třemi vrcholy. Pořadí těchto stěn, které na Obr. 5.6 odpovídá číslu stěny v kroužku, bylo převzato ze zdrojového kódu tak, jak je zapisuje *VHE*. Nezáleží, které tři ze čtyř bodů pro každou stěnu a v jakém pořadí budou využity, ale pro budoucí zápis *UV vektorů* je potřeba je volit tak, aby mohly definovat souřadnou soustavu, tedy ab byly kolmé. Tyto body byly určeny dle Tab. 5.1, která je programově realizována ve Výpis 5.3.



Obr. 5.6. Autorova konvence číslování stran a vrcholů *brushe*.

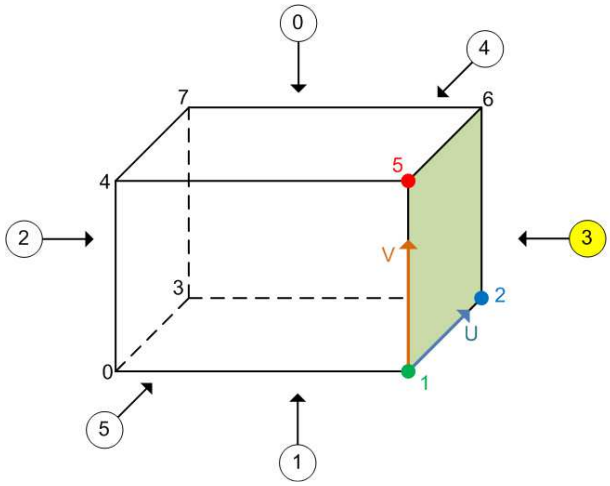
Číslo stěny W	bod[W][0]	bod[W][1]	bod[W][2]
0	4	7	6
1	3	0	1
2	0	3	7
3	2	1	5
4	3	2	6
5	1	0	4

Tab. 5.1. Čísla bodů (Č) pro jednotlivé stěny.

```
int cislo_bodu[6][3];
cislo_bodu[0][0] = 4; cislo_bodu[0][1] = 7; cislo_bodu[0][2] = 6;
cislo_bodu[1][0] = 3; cislo_bodu[1][1] = 0; cislo_bodu[1][2] = 1;
cislo_bodu[2][0] = 0; cislo_bodu[2][1] = 3; cislo_bodu[2][2] = 7;
cislo_bodu[3][0] = 2; cislo_bodu[3][1] = 1; cislo_bodu[3][2] = 5;
cislo_bodu[4][0] = 3; cislo_bodu[4][1] = 2; cislo_bodu[4][2] = 6;
cislo_bodu[5][0] = 1; cislo_bodu[5][1] = 0; cislo_bodu[5][2] = 4;
```

Výpis 5.3. Realizace Tab. 5.1 v *C*++.

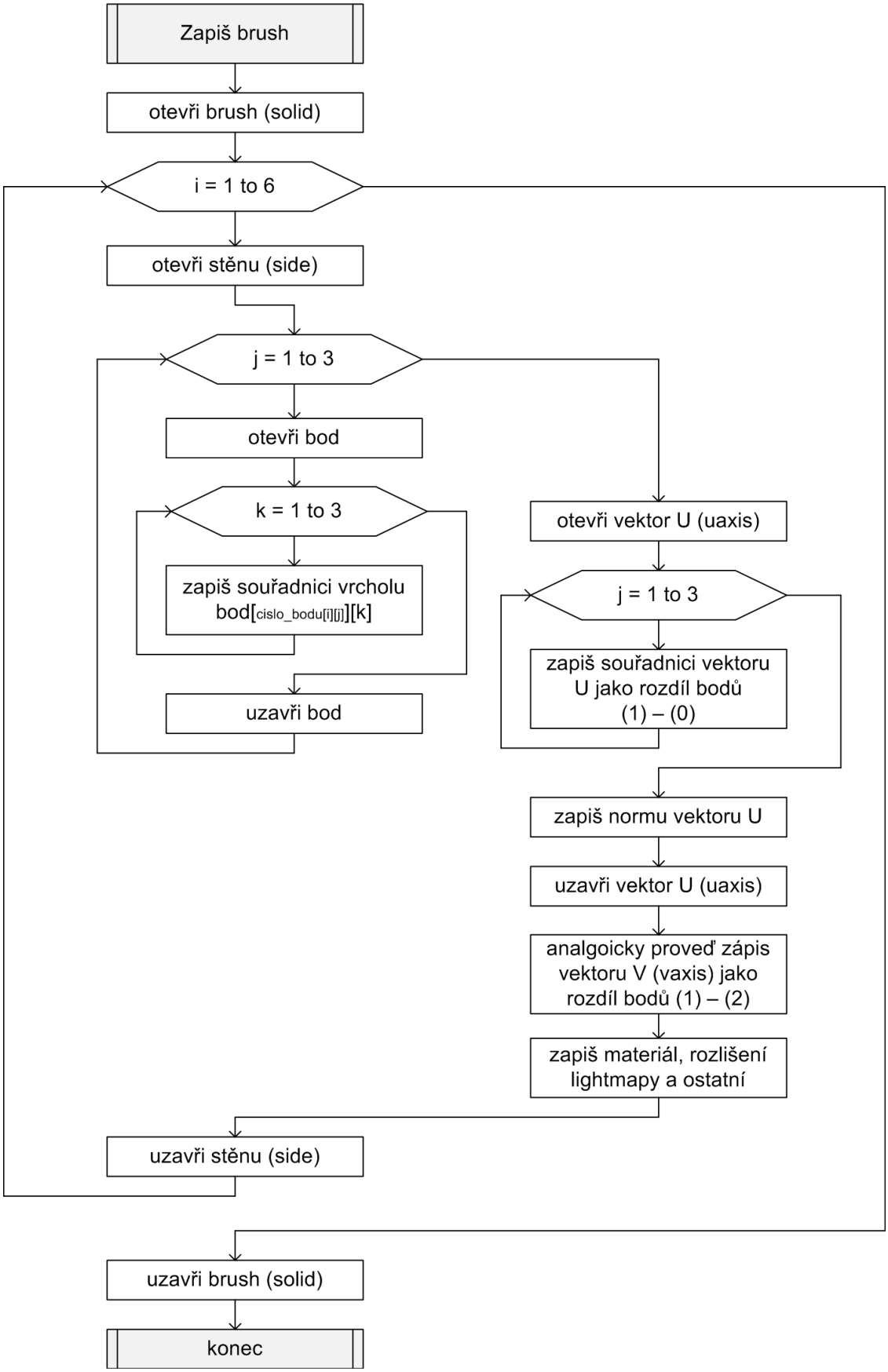
Algoritmus pracuje tak, že například pro číslo stěny 3 (viz Obr. 5.7) budou vybrána *cislo_bodu[3][i]* (i = 0 až 2). To znamená, že pravá stěna kvádru bude určena vrcholy číslo 2, 1 a 5 o souřadnicích *bod[2][j]*, *bod[1][j]*, *bod[5][j]* (*j = 0, 1, 2* pro osy *x, y, z*). Vektor *U* je pak definován jako rozdíl vrcholů číslo *cislo_bodu[w][0] – cislo_bodu[w][1]* a vektor *V* jako rozdíl *cislo_bodu[w][2] – cislo_bodu[w][1]*, to znamená v tomto případě *U* = |*21*| a *V* = |*51*|.



Číslo stěny W	bod[W][0]	bod[W][1]	bod[W][2]
0	4	7	6
1	3	0	1
2	0	3	7
3	2	1	5
4	3	2	6
5	1	0	4

Obr. 5.7. Příklad použití 3 vrcholů *brushe* pro definování stěny číslo 3 a pro definování vektorů pro *UV mapování* této stěny.

Algoritmus procedury *zapis_brush* je uveden na Obr. 5.8. Celý kód je uveden v příloze I této práce.



Obr. 5.8. Algoritmus pro zápis *brushe* do *VMF* souboru.

5.5 Funkce výsledného programu

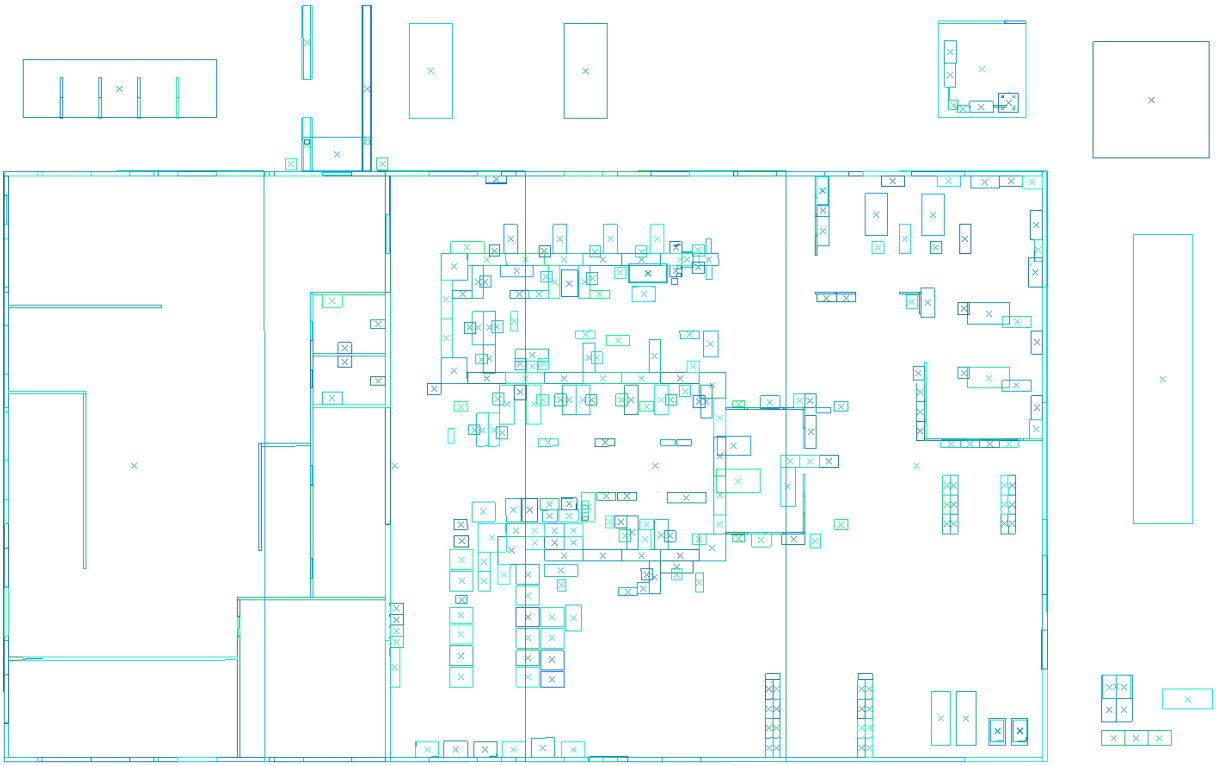
Po odladění programu se potvrdily autorovy hypotézy o tom, že výsledné layouty nebudou zcela kompatibilní z důvodu nemožnosti měnit rozměry modelu v *Source*, na rozdíl od *VisTable Touch*. Celý výpis zdrojového kódu je v příloze I.

Funkci programu demonstrují následující obrázky. Původní layout otevřený v programu *VisTable Touch* (viz Obr. 5.3 na s. 25) byl programově převeden na zdrojový soubor mapy ve formátu *VMF*, který je na Obr. 5.9 otevřený ve *VHE*. Na Obr. 5.10 jsou tytéž obrázky upravené a překryté tak, aby byl vidět rozdíl mezi těmito dvěma layouty.

5.6 Shrnutí

Tato kapitola popisuje funkční prototyp programu pro algoritmizovaný převod layoutu z programu *VisTable Touch* do zdrojového kódu mapy pro *Source SDK* ve formátu *VMF*. Program převádí všechny objekty jako *brushe* o přibližně shodných rozměrech (nepřesnost je daná nejvyšší jemností mřížky odpovídající ve skutečnosti asi 20mm). Pokud existují v knihovně *DIGITOV* odpovídající modely k objektům, pak jsou převedeny přímo jako entity zobrazující tyto modely. Při tvorbě modelu v knihovně *DIGITOV* je však nutné pro spřažení s tímto programem vytvořit odpovídající *DAT* soubor.

Plně algoritmizovaný převod by byl velmi složitý a vzhledem k tomu, že dosud se nerozhodlo o dalším vývoji, je program ponechán v současném stavu ve stadiu demonstračního prototypu. Již nyní se ale ukazuje, že i takový neúplně převedený layout je možné použít jako podklad pro tvorbu ve *VHE*, přičemž jeho zpracování je mnohem rychlejší než tvorba od začátku. Uživatel již zná polohu všech objektů v layoutu a nemusí objekty přesně vynášet na svá místa.



Obr. 5.9. Výsledek konverze layoutu z *VisTable Touch*.

Obr. 5.10. Porovnání původního layoutu ve *VisTable Touch* (černobíle) a převedeného ve *VHE* (červeně).

6 Instalátor a spouštěč DIGITOVu

Po aktualizaci *Steampipe* v létě 2013 došlo ke změně uspořádání souborů všech epizod a modifikací *Half-Life 2* i *Source SDK* a tím pádem k tomu, že původní instalátor přestal být funkční. Další změnou je to, že ke spuštění hry *Half-Life 2* s modifikací *DIGITOV*, ani ke spuštění *Valve Hammer Editor* není třeba, aby běžela distribuční služba *Steam* nebo *Source SDK Launcher*, který už nadále ani není obsažen v rámci *Source SDK Base 2013 Singleplayer*. Tím byl balíček zároveň připraven o funkci pohodlného spuštění přes službu *Steam* nebo *SDK Launcher*.

Byly určeny požadavky na vývoj programu, který by výše uvedené funkce zajistil. Tento program se bude jmenovat *DIGITOV Launcher* a bude přiložen ke každé distribuci *DIGITOVu*.

6.1 Požadavky na funkce programu

O vývoji nového instalátoru bylo rozhodnuto záhy po této aktualizaci. Tento instalátor by měl také sloužit i k odinstalování *DIGITOVu* a spuštění hry i *VHE*. Požadavky na funkce tohoto programu jsou následující:

1. automatické vyhledání instalačních cest z registru *Windows* nebo jiným způsobem
2. možnost ručního zadání instalačních cest
3. možnost instalace spouštěče *DIGITOVu* i *DIGITOVu* pro *Source SDK* zvlášť
4. automatické vyhledávání aktualizací balíku *DIGITOV* na internetu, možnost automatického stažení a instalace
5. možnost automatického stažení instalačního balíčku, pokud nebude přítomný; instalátor je tedy možné stáhnout samostatně
6. odkaz na otevření internetových stránek a e-booku
7. možnost spustit hru s modifikací *DIGITOV* nebo *VHE*
8. možnost při spuštění hry vybrat mapu
9. možnost odkázání na složky se zdroji
10. překlad do více jazyků

Steam i *Half-Life 2* instalační cesty ukládá do registru *Windows*, čímž se nabízí možnost automatického vyhledání instalačních cest. Pokud by tomu tak nebylo například z důvodu, že by se jednalo o jinou verzi nebo distribuci této hry, správné klíče v registrech by nebyly nalezeny a instalátor by byl nefunkční. Jednalo by se tak o znemožnění instalace a proto je nutné přidat možnost ručního zadání těchto instalačních cest.

Pokud na počítači není nainstalované *Source SDK Base 2013 Singleplayer*, například kdyby daný počítač sloužil pouze k účelům prezentace modelů vytvořených *DIGITOVem*, program nesmí ztratit v tomto případě funkčnost a nesmí se snažit konfigurovat *Source SDK* pro *DIGITOV*. Je tedy nutné mít možnost *DIGITOV* jako modifikaci pro hru nainstalovat zvlášť. Protože tvorba v *SDK* má za předpoklad přítomnost této modifikace, nesmí být možné konfigurovat *DIGITOV* pro *SDK* samostatně bez provedení instalace *DIGITOVu*. Konfigurace *SDK* pro *DIGITOV* je realizována v souboru *GameConfig.txt*, který obsahuje všechny instalační cesty. Instalace musí zahrnovat i automatickou úpravu tohoto souboru.

Aby uživatel nemusel spouštět pouze prázdnou hru, před spuštěním by měl mít možnost vybrat si soubor mapy, která bude automaticky načtena po spuštění hry bez konzolového příkazu *map <jméno_mapy>*.

Je možné stahovat různé zdroje, jako modely, textury, *prefaby* atd. z internetu z různých komunitních stránek. Pro snadné kopírování bude program nabízet otevření složek *DIGITOVu* i *VHE*. Uživatel je tak nemusí hledat v průzkumníku.

Aktualizace balíčku *DIGITOV* obnáší vyhledání cesty k modifikaci a ruční kopírování souborů. Program by tedy měl umět tyto soubory nakopírovat do složky s modifikací. Nejlepší řešení by bylo provést kontrolu na aktualizace z internetu. V případě přítomnosti nového balíčku by měl program tyto stáhnout a nainstalovat. K tomu využije instalační cesty, které se při instalaci uložily do konfiguračního souboru.

O umístění e-booku na internetu zatím nebylo rozhodlo a ještě ani nezačaly práce na internetových stránkách. I tak se však počítá s možností otevřít internetové odkazy, které povedou alespoň na provizorní webovou stránku s možností stažení instalátoru a na které budou umístěny informace o aktuální verzi a aktualizační balíky. Překlady do jiných jazyků se budou řešit buď úpravou zdrojového kódu, nebo umístěním všech řetězců do externího souboru.

Po aktualizaci *SteamPipe* se v nabídce distribuční služby *Steam* neukazuje modifikace *DIGITOV* a není ji tak možné pohodlně spustit. Zároveň neexistuje spouštěč *Source SDK Launcher*, přes který se spouštěl *Valve Hammer Editor*. Instalátor by tedy měl zároveň sloužit jako spouštěč.

6.2 Struktura složek a soubor GameConfig.txt

Pokud uživatel nezvolí jinak nebo nemá jinou distribuci *HL2* než z edice *Orange Box* nebo ze služby *Steam*, jsou veškeré potřebné soubory ve složce *Steam/SteamApps/common/Half-Life 2* a *Steam/SteamApps/common/Source SDK Base 2013 Singleplayer*. Složka *Steam* se ve výchozím nastavení nachází ve složce *Program Files*, případně *Program Files (x86)* pokud počítač používá 64-bitovou verzi systému *Windows*. V registrech jsou složky zapsány v klíčích:

- složka *Half-Life 2*:
  - `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\Steam App 420\InstallLocation` v případě *Win 7 x64* a *Win 8 x64*
  - `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\ Steam App 420\InstallLocation` v případě *Win Vista x32*
- složka *Valve Hammer Editoru*:
  - `HKEY_CURRENT_USER\Software\Valve\Hammer\General\Directory`

Ve složce se hrou *Half-Life 2* jsou další složky s jednotlivými modifikacemi, z nichž jedna se jmenuje *ep2* a obsahuje rozšíření *Episode Two*. Dále se zde nachází soubor *hl2.exe*, jehož spouštěcí argument *–game <složka>* spustí navíc i rozšíření (modifikaci) obsažené v dané složce. *DIGITOV* je tedy spouštěn argumentem *–game digitov*. Struktura uvnitř této složky s modifikací je popsána kapitole 2.1.7.

Valve Hammer Editor se spolu se složkou *prefabs* (vkládatelné objekty), souborem *GameConfig.txt* a dalšími softwarovými nástroji je umístěn ve složce *Steam/SteamApps/common/Source SDK Base 2013 Singleplayer/bin*. Instalátor bude používat soubor *GameConfigInsert.txt* (viz Výpis 6.1), jehož obsah bude do původního souboru přidán současně se záměnou všech výskytů řetězců *HL2EP2DIR* a *SourceSDKDIR* budou nahrazeny skutečnými cestami, čímž dojde ke konfiguraci *VHE*.

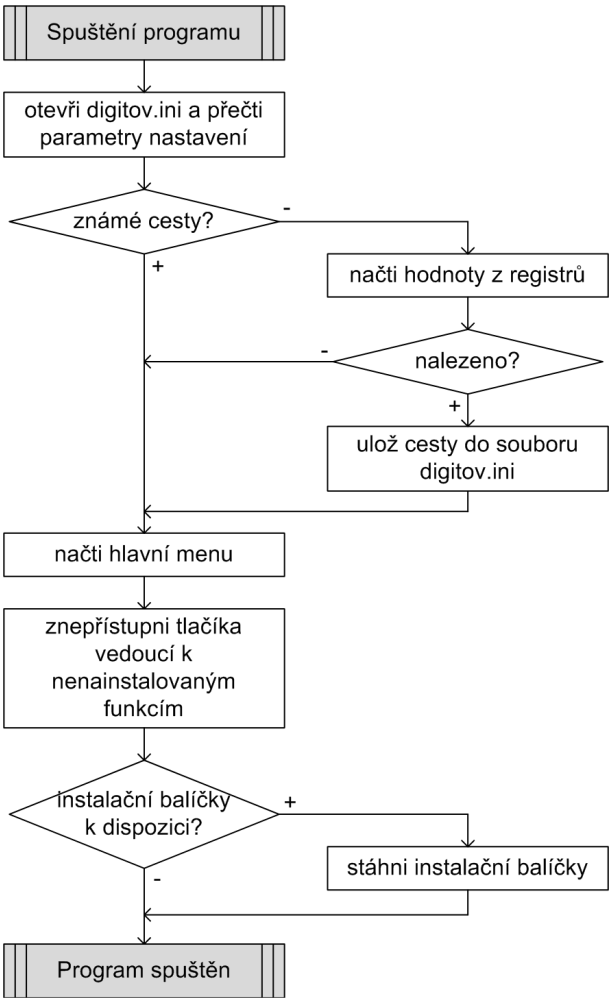
```
"DIGITOV"
{
	"GameDir"		"HL2EP2DIR\DIGITOV"
	"Hammer"
	{
		"GameData0"		"SourceSDKDIR\halflife2.fgd"
		"TextureFormat"			"5"
		"MapFormat"		"4"
		"DefaultTextureScale"		"0.250000"
		"DefaultLightmapScale"		"16"
		"GameExe"		"HL2EP2DIR\hl2.exe"
		"DefaultSolidEntity"		"func_detail"
		"DefaultPointEntity"		"info_player_start"
		"BSP"		"SourceSDKDIR\vbsp.exe"
		"Vis"		"SourceSDKDIR\vvis.exe"
		"Light"		"SourceSDKDIR\vrad.exe"
		"GameExeDir"		"HL2EP2DIR"
		"MapDir"		"HL2EP2DIR\DIGITOV\mapsrc"
		"BSPDir"		"HL2EP2DIR\DIGITOV\maps"
		"CordonTexture"			"tools\toolsskybox"
		"MaterialExcludeCount"		"0"
	}
}
```

Výpis 6.1. Obsah souboru *GameConfigInsert.txt*.

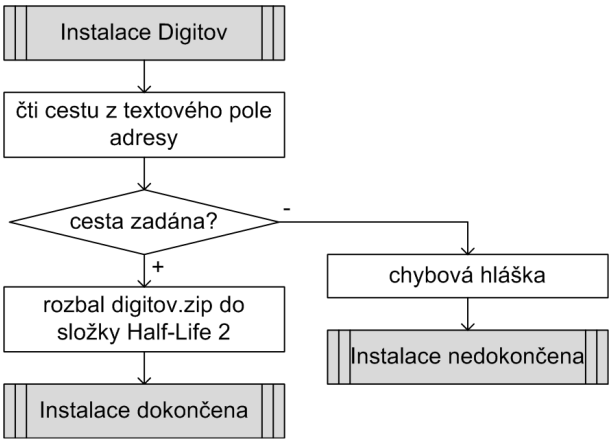
6.3 Návrhy algoritmů

Při prvním spuštění jsou vyhledávány klíče obsahující cestu ke hře *Half-Life 2* a *Valve Hammer Editoru* v registru systému *Windows* a pokud budou nalezeny, budou zapsány do souboru *DIGITOV.ini*. Tento soubor obsahuje informaci o aktuální verzi balíčku *DIGITOV* a informace o tom, zda je nainstalovaný *DIGITOV* a zda je *Source SDK* konfigurované k práci s ním. Algoritmus (Obr. 6.1) probíhá kromě prvního spuštění pokaždé, dokud nejsou nalezeny cesty ke hře a editoru nebo dokud instalace neproběhla a instalační cesty nejsou známé.

Z hlavního okna programu bude možné spustit hru a vybrat si mapu ze seznamu souborů *BSP*, který je načten ze složky *DIGITOV\maps*. Rovněž bude možné spustit *VHE* a otevřít složky se zdroji *DIGITOVu* (tj. *DIGITOV\maps*, *models* a *materials*), dále otevřít e-book nebo webové stránky *DIGITOVu* (tyto dvě funkce budou plně implementovány až v případě rozhodnutí umístit výukový e-book na internet a až budou webové stránky funkční). Jedna z funkcí bude kontrola aktualizací a otevření obrazovky s konfigurací instalace, kde bude možné nainstalovat nebo odinstalovat *DIGITOV*. Instalace *DIGITOVu* je jednoduchá, v podstatě jde jen o rozbalení jediného *ZIP* souboru do správné složky (viz Obr. 6.2).

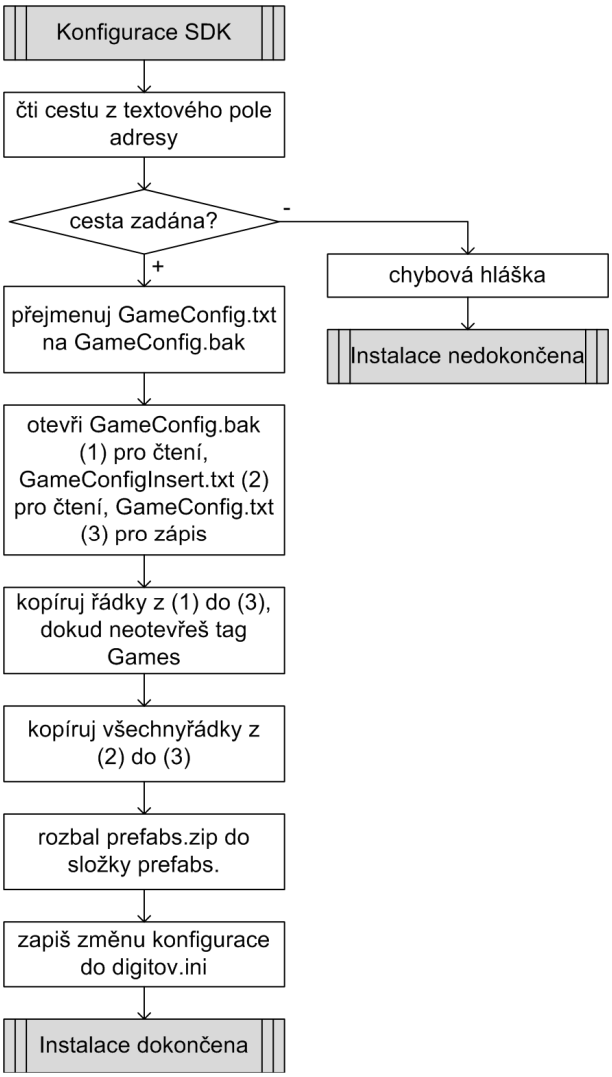


Obr. 6.1. Algoritmus spouštění programu.



Obr. 6.2. Algoritmus instalace *DIGITOVu*.

Složitější je konfigurace *Source SDK*. Je nutné upravit soubor *GameConfig.txt* tak, že v něm musí být změněny instalační složky. To je provedeno jeho zálohováním a nakopírováním obsahu *GameConfigInsert.txt* dodávaného s instalátorem, kde jsou části řetězců se složkami nahrazeny cestami. Například všechny výskyty řetězce *PathHL2EP2* budou nahrazeny za cestu k *Half-Life 2* ,viz Obr. 6.3.



Obr. 6.3. Algoritmus konfigurace *Source SDK*.

Odinstalace *DIGITOVu* spočívá v odstranění jeho složky. Protože nemusí být žádoucí smazat již vytvořené a zkompilované mapy, bude složka *maps* zálohována a nakopírována zpět na původní místo. Odstranění konfigurace *Source SDK* spočívá v odstranění složek s *prefaby* a nahrazení souboru *GameConfig.txt* zálohovaným souborem *GameConfig.bak.*

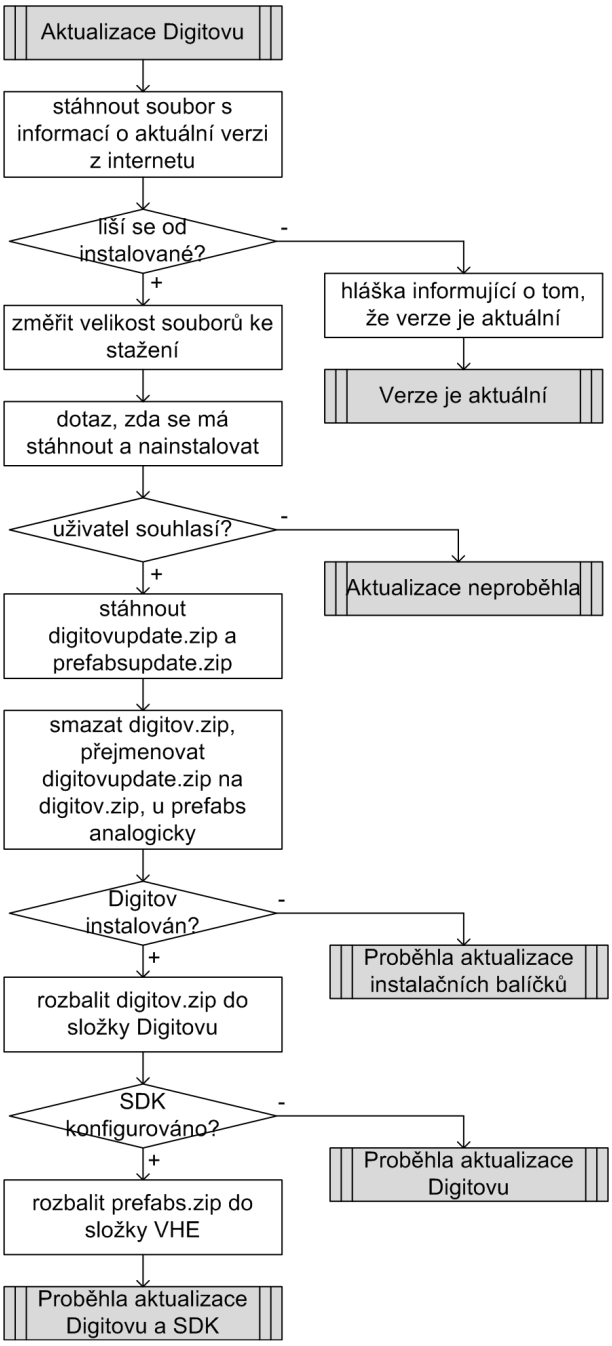
Zjištění aktuálnosti balíčku *DIGITOV* lze provést stažením malého textového souboru z internetu a otevřením. V tomto souboru je obsažen řetězec označující verzi, která je k dispozici ke stažení. Pokud se bude shodovat s verzí v lokálním konfiguračním souboru, pak je instalovaná verze aktuální. Pokud ne, uživateli bude nabídnuto automatické stažení aktualizačních *ZIP* souborů a jejich automatické rozbalení na správné místo. Tento proces znázorňuje algoritmus na Obr. 6.4.

6.4 Realizace programu

Pro realizaci programu byl zvolen programovací jazyk Visual Basic ve vývojovém prostředí *Microsoft Visual Studio 2010 Express*, ve kterém bude vyvinuta *Windows formulářová aplikace* využívající standardní komponenty, grafiku a styly aktuální verze operačního systému *Windows* na uživatelově počítači.

Všechny potřebné metody až na metody pro práci se *ZIP* soubory obsahuje framework *.NET 4.5*, který je instalován společně s vývojovým prostředím. Chybějící metody pro rozbalení

ZIP archivů poskytuje volně dostupná knihovna *DotNetZip* [29]. Pro spuštění instalátoru/spouštěče uživatel potřebuje framework *.NET verze 4.0* nebo novější. Knihovna *DotNetZip* je k instalátoru přiložena.



Obr. 6.4. Aktualizace *DIGITOVu*

Výpisy jednotlivých zdrojových kódů v práci nejsou uvedeny z důvodu, že zde není využito nějaké převratné nebo nové řešení. Celá složka projektu vytvořená vývojovým prostředím a s ní veškeré zdrojové kódy, je však k dispozici v CD příloze této práce.

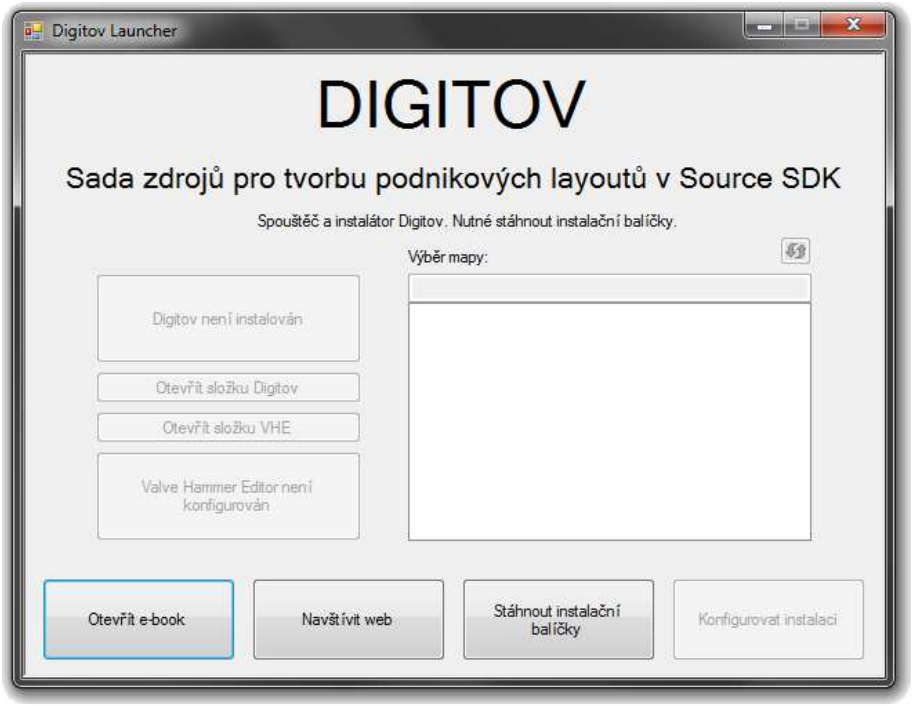
Program plní veškeré navržené funkce, ale přesto se počítá s možností, že budou nějaké další přidány dle potřeby. Po zaregistrování domény pro webové stránky, umístění aktualizačních souborů a souboru s verzí *DIGITOVu* bude program přepsán pro nové umístění a bude

k dispozici ke stažení. Prozatím plní úlohu internetového úložiště osobní univerzitní stránka autora [30].

6.5 Návod k použití programu

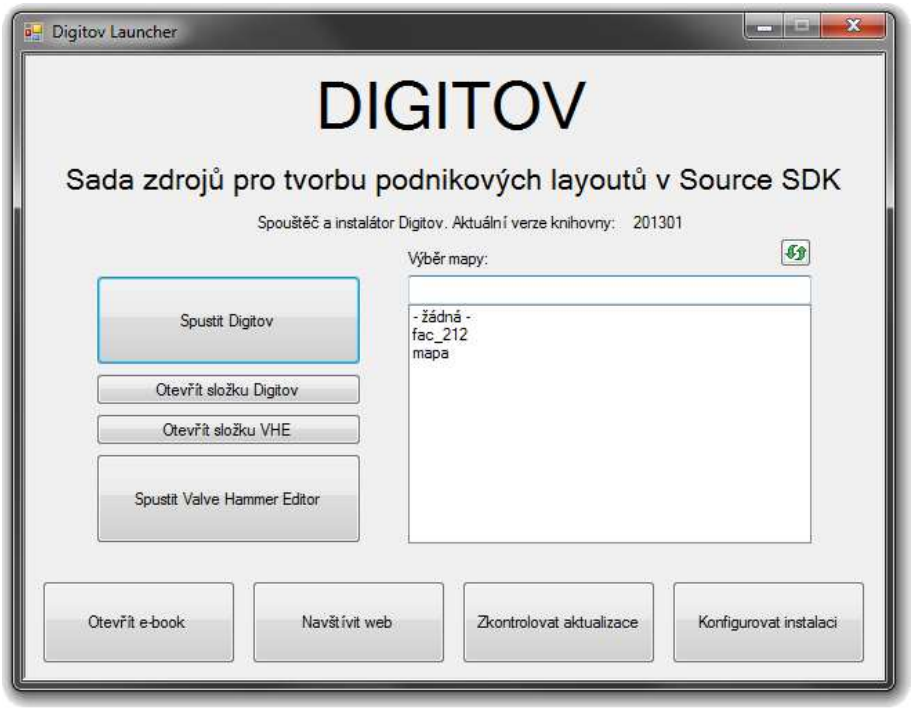
Po rozbalení instalačního balíčku je program spouštěn jediným *EXE* souborem, *DIGITOVLauncher.exe*, ve složce. K jeho úspěšnému spuštění je třeba mít instalovaný *.NET Framework 4.0* nebo novější. Program dále využívá knihovnu *DotNetZip* [29], která je k programu přiložena.

Předpokladem funkce *DIGITOVu* je samozřejmě nutné mít nainstalovanou hru *Half-Life 2: Episode Two* pro prohlížení vytvořených virtuálních prostředí, pro jejich tvorbu navíc ještě *Source SDK Base 2013 Singleplayer*. Program vyžaduje zápis těchto dvou položek do registru systému *Windows*, proto je nutné před jeho použitím spustit hru *Half-Life 2: Episode Two* a *Valve Hammer Editor*.



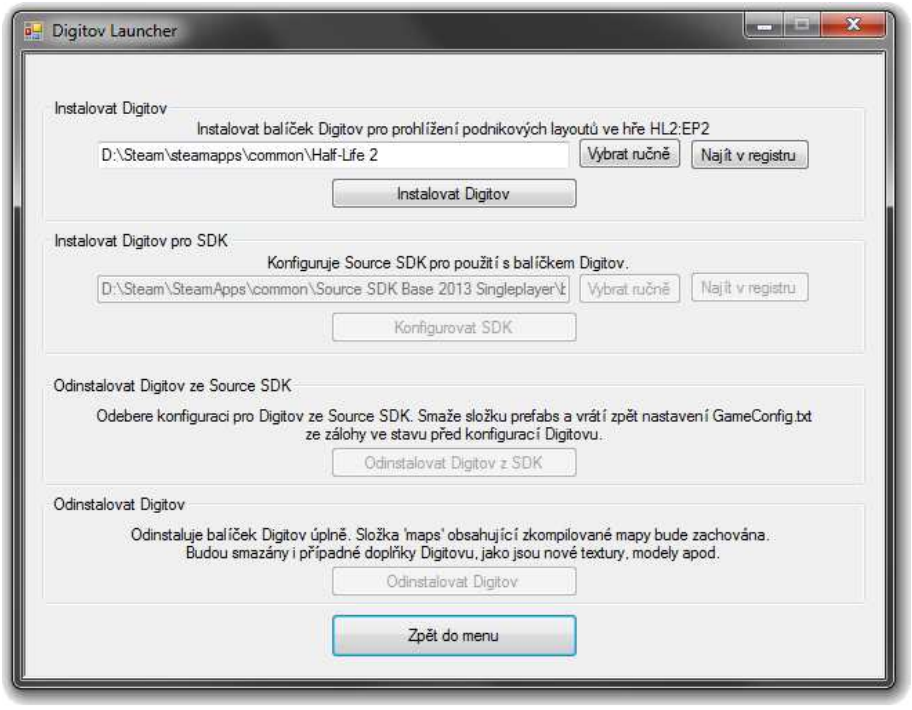
Obr. 6.5. Okno programu *DIGITOV Launcher* bez instalačních balíčků *DIGITOVu*.

Při prvním spuštění (Obr. 6.5) program vyhledá v registru *Windows* instalační cesty. Pokud nejsou nalezeny, program tuto informaci zaznamená do souboru *DIGITOV.ini* a při dalším spuštění se o jejich vyhledání pokusí znovu. Pokud uživatel stáhnul verzi instalátoru bez instalačních balíčků, je nutné je po prvním spuštění stáhnout z internetu klepnutím na tlačítko *„Stáhnout instalační balíčky“*. Pokud uživatel opatřil verzi s instalačními balíčky, nebo tyto byly již staženy, je zpřístupněna možnost *„Konfigurovat instalaci“* (viz Obr. 6.6). Zde je možné postupně nainstalovat *DIGITOV* a nakonfigurovat jej pro *Source SDK*. Instalační cesty jsou vyplněny automaticky, uživatel je ale může ručně zvolit, případně se pokusit znovu o jejich vyhledání za pomoci tlačítka *„Najít v registru“ (*Obr. 6.7*)*.



Obr. 6.6. Okno programu po nainstalování balíčku *DIGITOV*.

Na této obrazovce je možné obojí odinstalovat. Všechna tlačítka jsou postupně zpřístupněna nebo znepřístupněna tak, aby bylo zachováno správné pořadí. Nejdříve je nutné instalovat *DIGITOV* a pak až konfigurovat *Source SDK*. Naopak, při odinstalaci je nutné nejdříve odstranit konfiguraci *Source SDK* a pak až teprve *DIGITOV*.

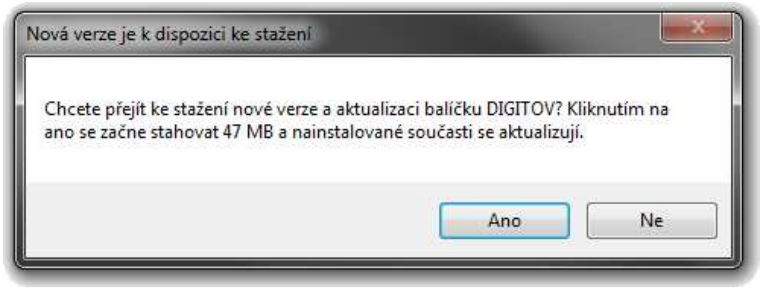


Obr. 6.7. Konfigurační okno programu.

Po klepnutí na tlačítko *„Zpět do menu“* se uživatel dostane zpět do hlavního menu, kde má nyní zpřístupněné možnosti spuštění *HL2:EP2* s *DIGITOVem* i *VHE*. Před spuštěním hry je možné vybrat mapu ze seznamu, ve kterém je možné volit z obsahu složky *DIGITOV\maps*. Tlačítkem s ikonou aktualizace dojde k načtení tohoto seznamu znovu.

Pro případ, že by uživatel potřeboval otevřít složky se zdroji *DIGITOVu*, použije tlačítka *„Otevřít složku DIGITOV“* nebo *„Otevřít složku VHE“*. Složky budou otevřeny v průzkumníku *Windows*. Dvě tlačítka vlevo dole slouží k otevření e-booku nebo webových stránek *DIGITOVu*.

Tlačítko *„Zkontrolovat aktualizace“* slouží ke stažení aktuálních instalačních balíčku a ověření, zda jsou k dispozici nové aktuální. Pokud ano, po potvrzení souhlasu (Obr. 6.8) uživatelem dojde ke stažení těchto balíčků a jejich instalaci.



Obr. 6.8. Potvrzení stažení a instalace aktualizace.

6.6 Shrnutí

Program byl pojmenován *DIGITOV Launcher*, slouží ke stažení a instalaci *DIGITOVu* a jeho spouštění uživatelsky příjemným způsobem. Je možné jej použít i k vyhledání aktualizacích a jejich instalaci.

Program funguje v souladu s požadavky a plní všechny funkce. Na řadě je odladění různých chyb během používání na různých počítačích s různým způsobem instalace *HL2:EP2* i *Source SDK*.

Program je prozatím v alfaverzi a je možné jej provizorně stáhnout z autorovy osobní stránky [30] a používat bez omezení. Autor nezaručuje dlouhodobé uložení a funkčnost na této adrese od okamžiku, kdy bude mít *DIGITOV* oficiální stránky a dojde tak k aktualizaci *URL* adres v programu.

7 Animace v Source Engine

Doposud chybějící využití *Source Engine* jakožto výukového nástroje bylo to, že nebyly k dispozici vhodné nástroje, jak dělat animace výrobních strojů při jejich funkci – tedy při obrábění. Bylo vytvořeno několik pokusných animací, přičemž všechny narážely na nedostatky vyplývající z vybavenosti *Source Engine* nebo jeho chybovosti. To se však podařilo vyřešit úpravou zdrojového kódu dle na internetu dostupných návodů a jeho kompilaci. Tato kapitola popisuje cestu k tvorbě animace soustružení.

7.1 Vhodné třídy entit pro řízení pohybu

Nejjednodušším případem je pohyb vřetene soustruhu, který je možné realizovat v podstatě jedině pomocí *brushové* entity *func_rotating*. Tato entita zajišťuje otáčení *brushe* dle libovolné osy, která je rovnoběžná se třemi osami souřadného systému. Znamená to, že celý stroj nemůže být libovolně natočen, ale jeho natočení se musí podřídit právě pohybu vřetene. Vřeteno jako takové může být realizováno modelem z *brushů* vytvořeným přímo ve *VHE* nebo pomocí externího modelu vloženého entitou třídy *prop_dynamic* s pohybovou vazbou (*parent*) na instanci *func_rotating*.

Ubíraný materiál by bylo teoreticky možné reprezentovat entitami *func_breakable*, které se při daném poškození (*damage*) mohou rozbít – v podstatě odstranit *brush*, nad kterým jsou definovány. Těchto entit by však muselo být příliš velké množství a navíc v hrubé síti souřadnicového systému *Source Engine*, takže by estetické vlastnosti ani nebyly příliš dobré. Autor proto volil jinou cestu. Tou je využití *brushové entity func_door*, jejímž vhodným použitím je možné posouvat s jednotlivými *brushi* a simulovat tak úběr materiálu. Bohužel by pravděpodobně nebylo (snadno) možné realizovat jakýkoli obecný tvar výsledného produktu.

Při pokusech s pohybem suportů a nože se autor setkal s potížemi vyplývající z nutnosti libovolně nastavitelného pohybu ve dvou osách. Ne všechny entity fungují dobře v případě, že jsou pohybově navázané na jinou entitu. To se týká konkrétně entit *func_tracktrain* a *func_movelinear*, které se pro tento případ jako jediné z entit, které jsou k dispozici, hodí. Tyto entity však fungují dobře, pokud definují prvotní pohyb a ostatní entity jsou navázané na ně. *Source Engine* naštěstí umožňuje bez problémů vázat entitu třídy *func_door*.

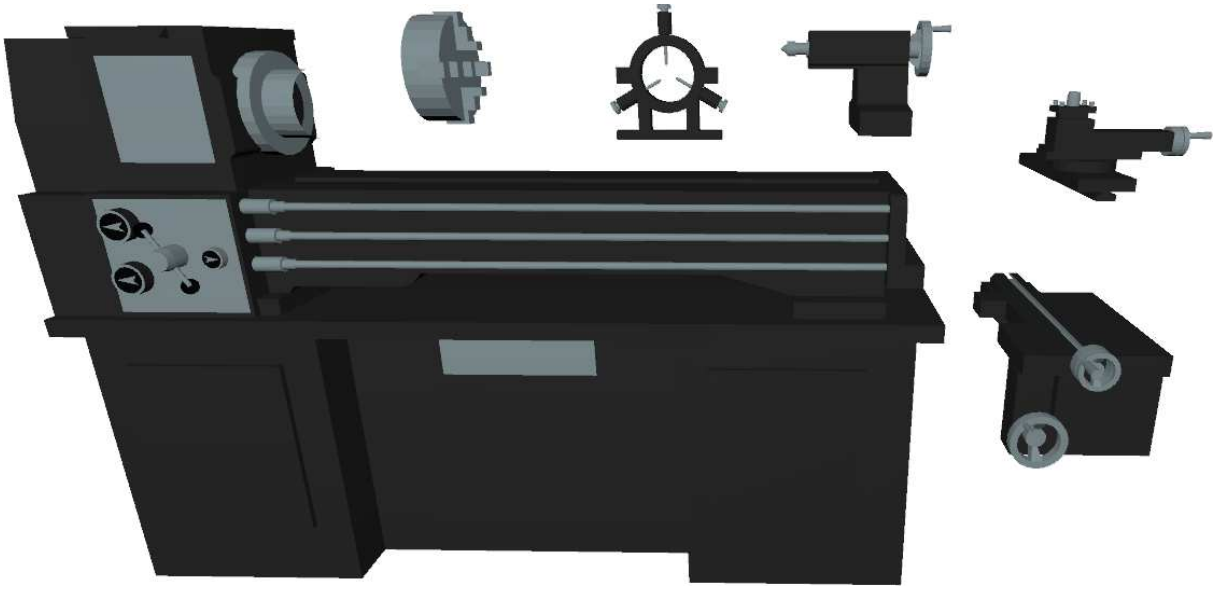
Třídy entit *func_tracktrain* a *func_movelinear* nesou velmi zřejmou chybu spočívající v tom, že pokud je jedna z těchto entit navázána na kteroukoli jinou entitu, nedojde ke správnému výpočtu souřadnic dítěte (závislé entity, která je pohybově vázána na druhou – rodiče). Správný výpočet spočívá v součtu absolutních souřadnic rodiče a ofsetu dítěte, engine ale tyto souřadnice evidentně počítá jinak. Pro jeden stroj na mapě se toto dá jaksi nouzově kompenzovat umístěním dítěte do počátku souřadné soustavy. V případě, že jsou použity entity třídy *func_tracktrain* včetně bodových entit třídy *path_track* definujících trajektorii jejího pohybu. Tato chyba je v případě entity fu*nc_movelinear* zmíněna i v oficiální příručce na *Valve Developer Community* [31].

Pro velmi jednoduché tvary je možné použít jako dítě entitu třídy *func_door*, která se může pohybovat mezi dvěma krajními polohami, nemůže však být zastavena mezi nimi. V následující kapitole je popsána tvorba třech způsobů za využití entit popsaných v této kapitole.

Modely *MDL* suportů a dalších pohyblivých dílů stroje jsou realizovány entitou třídy *prop_dynamic*, která je vázána na jednotlivé funkční entity řídící pohyb.

7.2 Tři varianty tvorby animace

Na začátku bylo nutné získat modely jednotlivých částí soustruhu. To bylo provedeno převedením modelu soustruhu značky *Jet* z *3D Warehouse* [32] a jeho standardní úpravou pomocí *3dS MAX 2011*. Model byl při převodu rozdělen na jednotlivé pohyblivé části. Kromě vřetena nebyly jednotlivé části vystředěny do středu souřadného systému a to z toho důvodu, že několik entity třídy *prop_static* nebo *prop_dynamic* ve stejném místě umístí všechny části přesně. V modelu *MDL* je posun ve vlastním souřadném systému zachován. Tento model byl v rámci složky machines vložen do vlastní složky component_lathe, aby se zdůraznilo, že se jedná o model stroje a jeho části (viz Obr. 7.1)



Obr. 7.1. Model soustruhu *Jet* [32] převedený do formátu *MDL* rozložený na jednotlivé komponenty.

Po zhotovení komponentního modelu se projevila hrubost souřadnicové mříže *Source Engine*, protože nebylo možné přesně umístit jeho otáčející části. Nepřesnost polohy se projeví v házení, navíc není možné výrobek modelovat přímo ve *VHE* a bylo by nutné tvořit *MDL* modely, což by bylo pracné a výrazně by ztížilo tvorbu animací v budoucnu. Řešením, které ovšem poněkud omezí použití takových modelů, je vytvořit zvětšený model komponentního soustruhu, se kterým je možné snadno pracovat. Komponentní model soustruhu byl znovu zkompilován do složky *macro_models\component_lathe* s přibližně pětinásobným zvětšením, které je zadáno v *QC* souboru řídícím průběh kompilace.

Následující podkapitoly popisují funkci jednotlivých tří animací, jejichž zdrojový kód je obsažen v CD příloze diplomové práce.

7.2.1 Animace první

První vytvořenou animací bylo soustružení válcového osazení za využití dvou entit třídy *func_tracktrain* navázaných na sebe s tím, že dítě muselo být umístěno do počátku souřadného systému. Jedna tato entita slouží pro ovládání příčného suportu – tedy ovládání přísuvu nože. Ta je závislá na druhé entitě stejné třídy, která řídí posuv. Při posuvu tedy dochází k pohybu obou suportů, při přísuvu pouze k pohybu příčného suportu, což odpovídá kinematice skutečného soustruhu.

Pohyby jsou řízeny entitami *path_track*, které vysílají řídící výstupy při dotyku počátku entity *func_tracktrain*. Každý pohyb nástroje má svou vlastní trajektorii mezi dvěma *path_track*, které na sebe pevně navazují. Tato třída může vysílat výstupy při průjezdu entity

func_tracktrain, takže je možné velmi snadno řídit tyto pohyby. Například při dosažení určité hodnoty vzdálenosti nože od osy otáčení obrobku dojde k zastavení přísuvu (výstup *Stop* na dítě *func_tracktrain*) a k zapnutí posuvu (výstup *StartForward* na rodiče *func_tracktrain*), případně ještě na ovládání otáčení vřetene nebo k zahájení ubývání materiálu při řezu (v tomto případě *Open* na *func_door*).

Dítě a všechny jeho *path_track* musely být umístěny do počátku souřadné soustavy mapy, což by komplikovalo tvorbu v případě více strojů na jedné mapě. Výhodou je však možnost v podstatě jakéhokoli pohybu nástroje a snadné stavby řídících výstupů. Nevýhodou je velmi viditelná nepřesnost těchto pohybů. Není možné přesně řídit pohyb nože a vyskytují se odchylky. Tvorba animace je navíc díky tomu velmi náročná a znamená spoustu pokusných běhů mapy a jemného nastavování polohy jednotlivých entit až k dosažení požadovaného výsledku. Z tohoto důvodu bude od této metody nejspíše upuštěno.

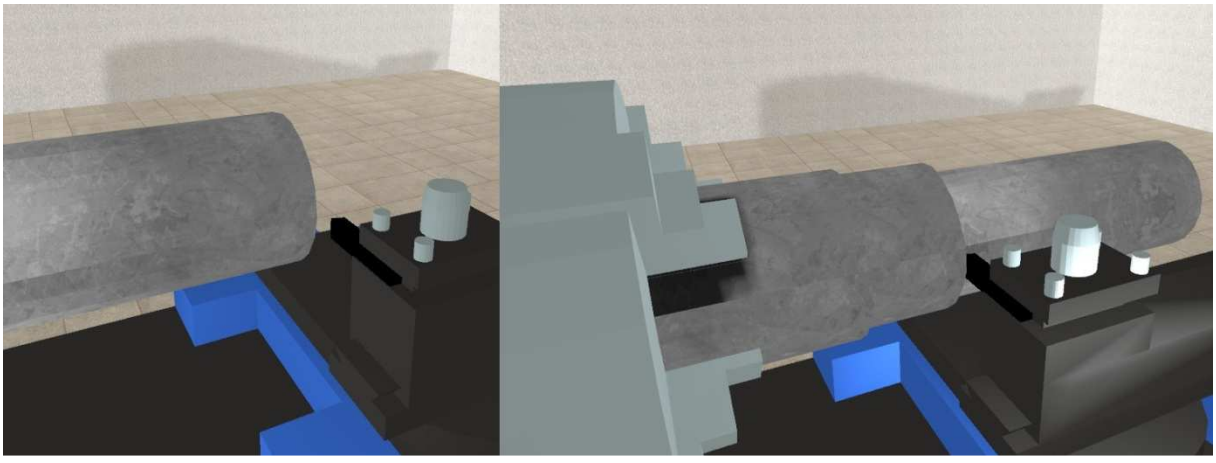
Animace přestala být kompatibilní po opravě dynamických knihoven obsahující třídy entit, která je popsána v následujících kapitolách.

7.2.2 Animace druhá

Vazba dvou entit *func_movelinear* by se potýkala se stejnými problémy jako entita *func_tracktrain*. Tato entita nefunguje správně, pokud je zúčastněna v pohybové vazbě jako dítě. Z tohoto důvodu je závislá entita třídy nikoli *func_movelinear*, ale *func_door*. Rozdíl mezi těmito dvěma entitami je, že pohyb entity třídy *func_door* není možné zastavit v libovolném místě mezi dvěma krajními body jejího pohybu.

Touto metodou byla vytvořena stejná animace soustružení osazené hřídele jako v případě první metody. Přísuv jakožto hlavní pohyb (rodič) je realizován entitou třídy *func_movelinear*, jejíž možností je přesně měnit vzdálenost nože od osy obrábění. Vedlejší pohyb (dítě) je realizován entitou třídy *func_door* a má pevnou délku. Posuv je rovněž přesný, ale není možné jej jakkoli prodlužovat nebo zkracovat.

Všechny pohyby jsou řízeny entitou *logic_relay*, která obsahuje seznam výstupů jako posunutí příčného suportu na danou vzdálenost od osy obrábění (výstup *SetPosition X* na rodiče *func_movelinear*, kde *X* je číslo v intervalu <0;1> reprezentující relativní polohu mezi krajními polohami této entity), otevření a zavření dveří reprezentující posuv při řezání a vracení nástroje (výstup *Open* a *Close* na dítě *func_door*).



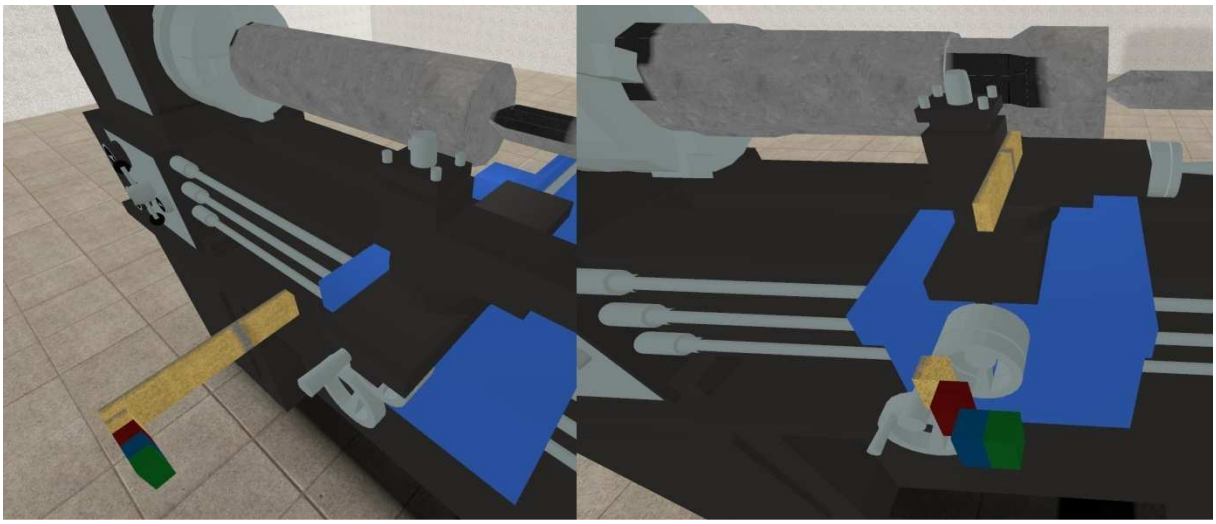
Obr. 7.2. Ukázka animace soustruhu.

Tento způsob tvorby animace se vyznačuje snadnou a intuitivní realizací, avšak s poněkud složitějším řízením, kdy všechny výstupy obstarává jediná entita třídy *logic_relay* a neobejde se bez několika výpočtů. Je třeba počítat časy jednotlivých pohybů a z toho odvodit, ve

kterém časovém okamžiku od spuštění této entity je třeba spustit jednotlivé pohyby. Pro větší množství pohybů je možné dělit tyto výstupy mezi více entit třídy *logic_relay*. Nevýhodou je pevně daný posuv stroje, naproti tomu je možné tyto pohyby nadefinovat přesně, na rozdíl od první metody, viz Obr. 7.2.

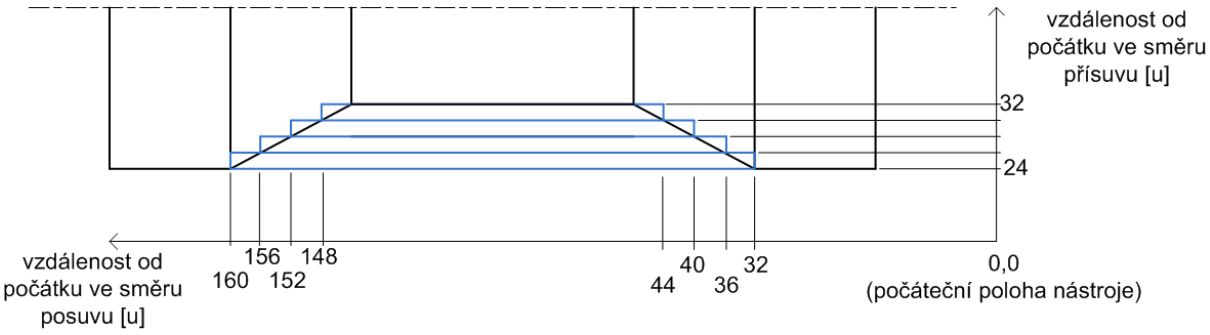
7.2.3 Animace třetí

Tato metoda se snaží kompenzovat nevýhody předchozích dvou řešení. *Func_movelinear* je tentokrát použita pro řízení posuvu a přísuv řídí několik entit *func_door* o malé velikosti tak, že jejich správnou kombinací je možné přesně umístit nástroj, jak je viditelné na Obr. 7.3. Barevné bloky vznášející se ve vzduchu jsou entity třídy *func_door* sériově navázané na sebe. Vhodnou kombinací jejich pohybu je možné vysouvat příčný suport do požadovaného místa. Na obrázku jsou vlevo jsou všechny bloky v nulové poloze, vpravo v nastavení pro soustružení druhé ze čtyř vrstev (dle schématu na Obr. 7.4).



Obr. 7.3. Animace soustruhu třetí metodou. Barevné bloky volně v prostoru jsou entity třídy *func_door*.

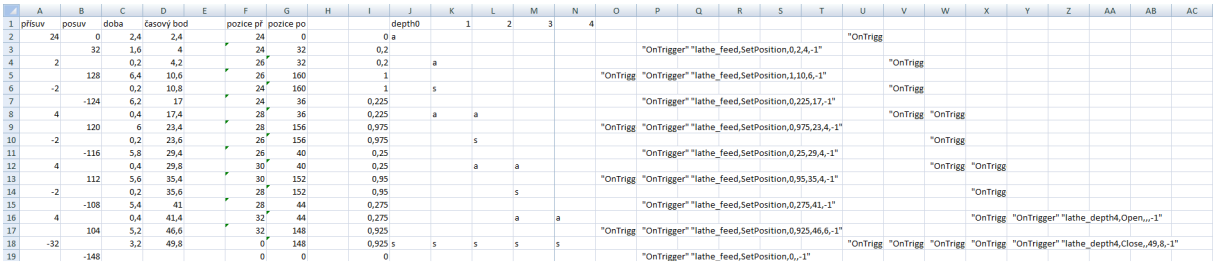
Touto metodou šla realizovat animace soustružení, kdy byla délka posuvu měněna v průběhu obrábění. Bylo však nutné provést několik výpočtů a již velmi složitě řídit pohyby jednotlivých entit jedinou instancí entity třídy *logic_relay*. Výsledkem je však přesná animace se složitějšími tvary, dle Obr. 7.4.



Obr. 7.4. Schéma soustružení pro tvorbu výstupů z entity *logic_auto*. Černě je znázorněn výsledný tvar, modře entity třídy *func_door*, které budou odsunuty, čímž je znázorněn úbytek materiálu.

Protože zápis výstupů by byl složitější než vytvořit algoritmus, který by výstupy zapsal, byl pro tyto účely za pomoci *MS Excel* vytvořen generátor kódu výstupů z entity (viz Obr. 7.5),

který je možno nakopírovat přímo do zdrojového kódu mapy. Tento soubor .xlsx je v CD příloze této práce a Výpis 7.1 výsledného kódu následuje (). Tento soubor byl účelově vytvořen pro tento případ, nicméně jeho úpravou by bylo možné řešit i jiné úlohy.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1	přísuv	posuv	doba	časový bod		pozice př	pozice po			depth0	1	2	3	4															
2	24	0	2,4	2,4		24	0		0	a											"OnTrigg								
3		32	1,6	4		24	32		0,2							"OnTrigger" "lathe_feed,SetPosition,0,2,4,-1"													
4	2		0,2	4,2		26	32		0,2	a												"OnTrigg							
5		128	6,4	10,6		26	160		1						"OnTrigg	"OnTrigger" "lathe_feed,SetPosition,1,10,6,-1"													
6	-2		0,2	10,8		24	160		1	s												"OnTrigg							
7		-124	6,2	17		24	36		0,225							"OnTrigger" "lathe_feed,SetPosition,0,225,17,-1"													
8	4		0,4	17,4		28	36		0,225	a	a											"OnTrigg	"OnTrigg						
9		120	6	23,4		28	156		0,975						"OnTrigg	"OnTrigger" "lathe_feed,SetPosition,0,975,23,4,-1"													
10	-2		0,2	23,6		26	156		0,975		s												"OnTrigg						
11		-116	5,8	29,4		26	40		0,25							"OnTrigger" "lathe_feed,SetPosition,0,25,29,4,-1"													
12	4		0,4	29,8		30	40		0,25		a	a											"OnTrigg	"OnTrigg					
13		112	5,6	35,4		30	152		0,95						"OnTrigg	"OnTrigger" "lathe_feed,SetPosition,0,95,35,4,-1"													
14	-2		0,2	35,6		28	152		0,95				s											"OnTrigg					
15		-108	5,4	41		28	44		0,275							"OnTrigger" "lathe_feed,SetPosition,0,275,41,-1"													
16	4		0,4	41,4		32	44		0,275				a	a										"OnTrigg	"OnTrigger" "lathe_depth4,Open,,,-1"				
17		104	5,2	46,6		32	148		0,925						"OnTrigg	"OnTrigger" "lathe_feed,SetPosition,0,925,46,6,-1"													
18	-32		3,2	49,8		0	148		0,925	s	s	s	s	s							"OnTrigg	"OnTrigg	"OnTrigg	"OnTrigg	"OnTrigger" "lathe_depth4,Close,,49,8,-1"				
19		-148				0	0		0							"OnTrigger" "lathe_feed,SetPosition,0,,-1"													

Obr. 7.5. Aplikace v *MS Excel* pro algoritmickou tvorbu kódu výstupů entity *logic_auto*.

```
"OnTrigger" "lathe_depth0,Open,,2.4,-1"
"OnTrigger" "lathe_feed,SetPosition,0.2,4,-1"
"OnTrigger" "lathe_depth1,Open,,,-1"
"OnTrigger" "lathe_cut1,Open,,10.6,-1"
"OnTrigger" "lathe_feed,SetPosition,1,10.6,-1"
"OnTrigger" "lathe_depth1,Close,,10.8,-1"
"OnTrigger" "lathe_feed,SetPosition,0.225,17,-1"
"OnTrigger" "lathe_depth1,Open,,,-1"
"OnTrigger" "lathe_depth2,Open,,,-1"
"OnTrigger" "lathe_cut2,Open,,23.4,-1"
"OnTrigger" "lathe_feed,SetPosition,0.975,23.4,-1"
"OnTrigger" "lathe_depth2,Close,,23.6,-1"
"OnTrigger" "lathe_feed,SetPosition,0.25,29.4,-1"
"OnTrigger" "lathe_depth2,Open,,,-1"
"OnTrigger" "lathe_depth3,Open,,,-1"
"OnTrigger" "lathe_cut3,Open,,35.4,-1"
"OnTrigger" "lathe_feed,SetPosition,0.95,35.4,-1"
"OnTrigger" "lathe_depth3,Close,,35.6,-1"
"OnTrigger" "lathe_feed,SetPosition,0.275,41,-1"
"OnTrigger" "lathe_depth3,Open,,,-1"
"OnTrigger" "lathe_depth4,Open,,,-1"
"OnTrigger" "lathe_cut4,Open,,46.6,-1"
"OnTrigger" "lathe_feed,SetPosition,0.925,46.6,-1"
"OnTrigger" "lathe_depth0,Close,,49.8,-1"
"OnTrigger" "lathe_depth1,Close,,49.8,-1"
"OnTrigger" "lathe_depth2,Close,,49.8,-1"
"OnTrigger" "lathe_depth3,Close,,49.8,-1"
"OnTrigger" "lathe_depth4,Close,,49.8,-1"
"OnTrigger" "lathe_feed,SetPosition,0,,-1"
```

Výpis 7.1. Výstupy entity *logic_auto* pro řízení animace vytvořené třetím způsobem.

Animace funguje dobře, pohyby jsou přesné, ale autor si nedokáže představit jeho použití při tvorbě rozsáhlých hal se spoustou strojů, kdy by tvorba takovýchto animací značně prodloužila čas tvorby celé mapy.

7.2.4 Oprava kódu entity func_movelinear

Při tvorbě map pro *Source Engine* se autor často potýká s problémy vyplývajícími z různých nedostatků a chyb dodaného buildu *DLL* knihoven. V tomto případě se jedná o chybu ve třídě *func_movelinear* znemožňující její účast v pohybové vazbě na straně dítěte. Na [33] je k dispozici návod pro opravu kódu této entity. K aplikaci takovéto opravy je nutné vytvořit nový build souborů dynamických knihoven *Server.dll* a *Client.dll*, které mimo jiného tyto třídy obsahují.

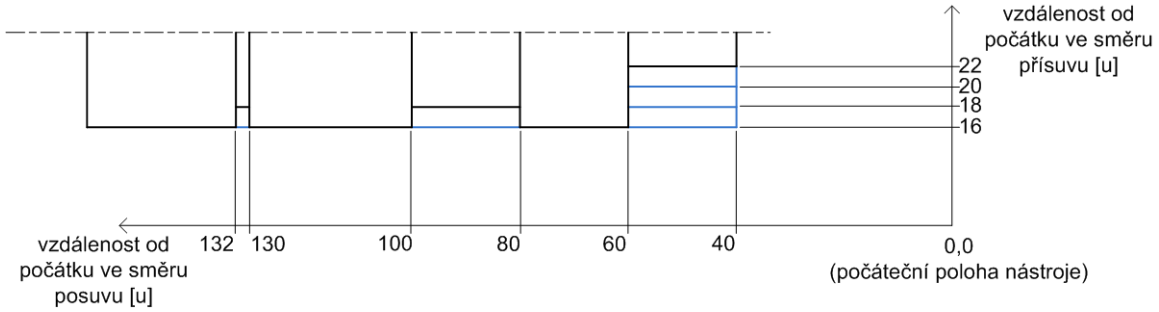
Zdrojový kód minulých verzí těchto knihoven z edice *Orange Box* a ve verzi *Source SDK Base 2009* se autorovi nikdy nepovedl zkompilovat. Tyto starší verze ke kompilaci vyžadovaly *Microsoft Visual C++ 2005*. Zdrojové kódy *Source SDK Base 2013* nejsou již součástí *Source SDK*, ale *Valve* je poskytuje přes server *GitHub.com* [34]. Tato verze vyžaduje *Microsoft Visual C++ 2010* [35] s aktualizací *Service Pack 1* [36].

V této aktuální verzi vývojového prostředí je možné bez problému kompilovat nové *DLL* knihovny a po použití návodu [33] byla opravena třídy entity *func_movelinear*. Po otestování výsledných *DLL* souborů entita funguje při použití pohybové vazby *parent* správně jako rodič i jako dítě.

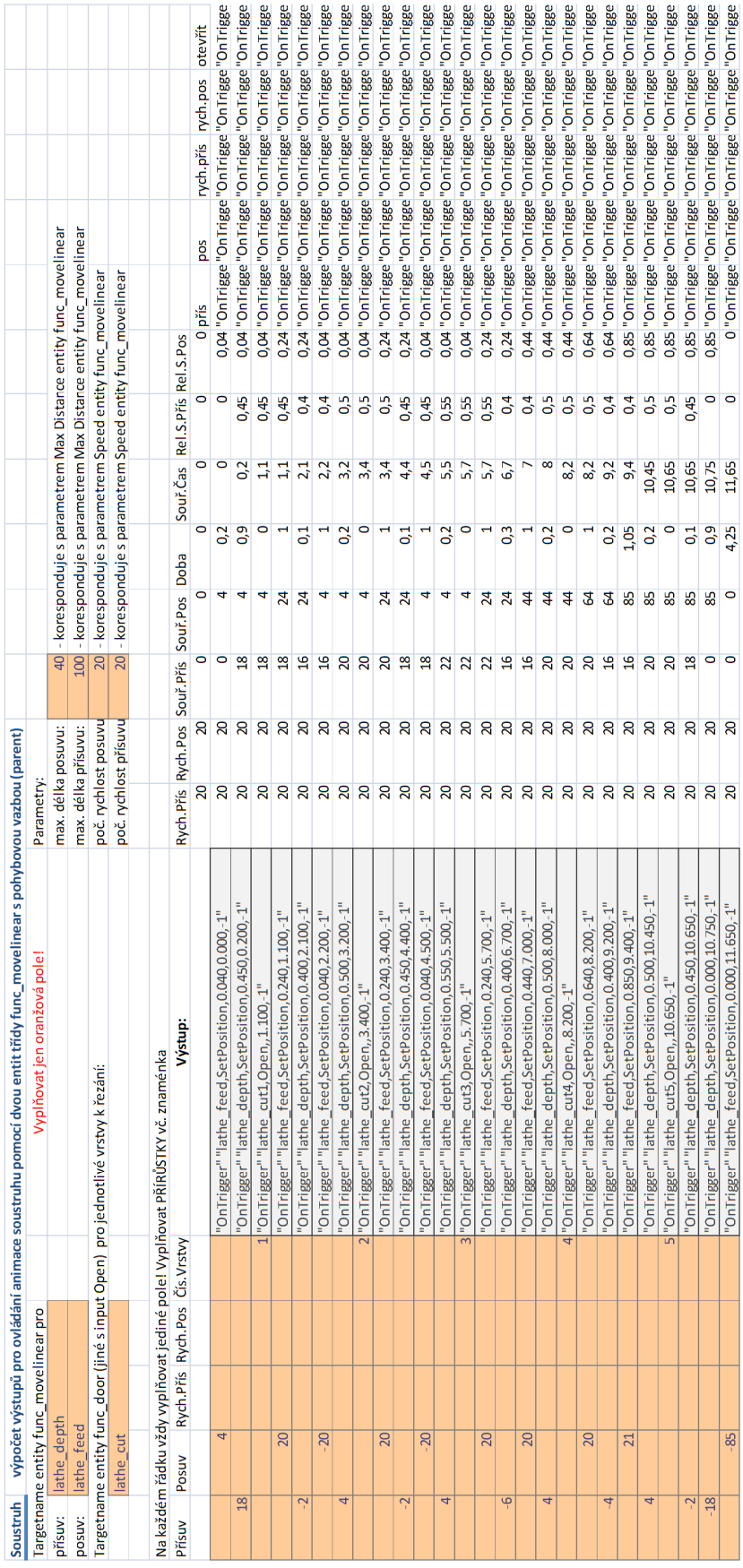
7.2.5 Finální animace

Po opravě třídy *func_movelinear* je konečně možné bez větších problémů vytvořit animaci soustružení, která je přesná a není nutné složitě obcházet chyby v enginu. Entita třídy *func_movelinear* nyní může být pohybově závislá na jiné entitě – tedy být ve vazbě typu *parent* coby dítě.

Po zkušenostech s generováním řídících výstupů pohybu se autor rozhodl začít obecně použitelným programem v *MS Excel*. V tomto programu jsou vyplněny názvy obou entit *func_movelinear* a začátky názvů entit, které symbolizují odřezávání jednotlivých vrstev v jednotlivých krocích obrábění. Dále jsou vyplněny délky drah pro posuv i přísuv a počáteční rychlosti. Vyplňují se výhradně oranžová pole jednotlivými přírůstky poloh, novými absolutními hodnotami rychlostí nebo koncovkami názvů pro odřezávanou vrstvu. Na každém řádku je možné vyplnit vždy pouze jednu z těchto položek. Program vypíše kód určený k ručnímu nakopírování přímo do zdrojového kódu mapy na správné místo – do tagu *connections* entity, která bude řídit celou animaci, nejlépe třídy *logic_relay*. Náhled této aplikace je na Obr. 7.7. Tato tabulka byla vyplněna dle skici na Obr. 7.6.

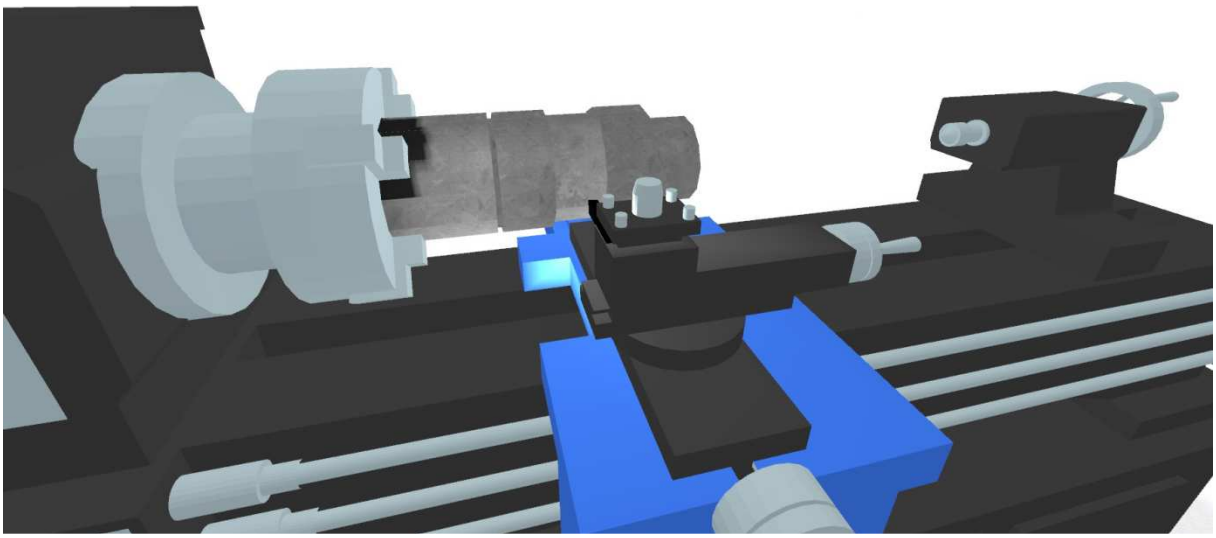


Obr. 7.6. Schéma soustružení pro animaci soustružení pomocí dvou entit třídy *func_movelinear*.



Obr. 7.7. Program v *MS Excel* pro genrování výstupů pro dvojici navázaných entit třídy *func_movelinear*.

Tvorba této animace trvala přibližně půl hodiny od vytvoření skici a schématu na papíře přes vyplnění tabulky v *MS Excel*, realizace této animace ve *VHE* a odladění chyb (ukázka na Obr. 7.8). Nyní je tedy k dispozici způsob, jak rychle a poměrně snadno tvořit animace s přesnými pohyby. Soubor se zdrojovým kódem mapy je v CD příloze této práce, výsledná animace bude v jiné, estetičtější podobě součástí balíčku *DIGITOV* jako demonstrační mapa.



Obr. 7.8. Ukázka z animace soustruhu vytvořené za pomoci dvou entit *func_movelinear*.

7.3 Shrnutí

Tvorba animací znázorňující soustružení v *Source Engine* je možná, ale v případě modelování obrobků z *brushů* nese jistá omezení v obráběných tvarech. Tvorba je nicméně velmi rychlá a animace jsou esteticky poměrně kvalitní. Je zde však vidět, že *Source Engine* nebyl k tomuto účelu primárně určen a musela být dle návodu [33] opravena třída entity *func_movelinear*, která se po této opravě vyloženě hodí k účelům tvorby animace pro ovládání víceosých pohybů.

Pro snadnou tvorbu animací autor doporučuje začít skicou na papíře s vyznačením počáteční polohy nástroje, délek a rozměrů jednotlivých segmentů, z které je možné snadno vyčíst přírůstky polohy příčného a podélného suportu a na základě toho vyplnit tabulku v *MS Excel*, který vypočítá řídící výstupy pro jejich pohyby, které jsou následně ručně zkopírovány na správné místo ve zdrojovém kódu mapy.

8 Cíl dalšího vývoje

DIGITOV je sice již v dnešní době velmi dobře využitelný, nicméně s dalším rozšiřováním se počítá. Jedná se především o přidání nových textur, modelů a objektů do knihovny a plánuje se i přímý zásah do zdrojového kódu za účelem vytvoření nových entit, které by zavedly proměnné a podobně. Ve vývoji je automatizovaný překladač layoutu vytvořeného v programu *VisTable Touch* do formátu *VMF*, tedy zdrojového kódu mapy. Nutné je vyvinout nový instalátor kompatibilní s novým systémem souborů a *Source SDK 2013*. Počítá se i se zveřejněním knihovny a s tím související tvorbou webových stránek.

8.1 Nové textury, modely, objekty

Nejdůležitější rozšíření spolu s novým instalátorem je doplnění knihoven o nové prvky. Především jde o modely strojů, vozidel a nábytku, jak dílenského, tak kancelářského. Nové modely i textury by měly být k dispozici ve více barevných variantách. Chybí textury některých běžně používaných povrchů v budovách. Počítá se i s vytvořením nových interaktivních objektů ve formě *prefabů*.

Nové modely nebudou z důvodu vysoké pracnosti modelovány, ale převáděny z existujících zdrojů katedry nebo otevřených knihoven, například knihovny modelů *SketchUp* [20], samozřejmě s uvedením zdroje přímo v dokumentaci k *DIGITOVu*.

8.2 Zkoumání možností engine

Source Engine má poměrně velký potenciál i v současném stavu. Ne všechny možnosti však již byly odhaleny a počítá se s objevením nových interaktivních nebo vizuálních prvků. Vyloučen není přímý zásah do kódu a tvorba buildu přímo na míru *DIGITOVu*, který by obsahoval nové entity nesoucí proměnné, podmínky a další užitečné prvky pro tvorbu algoritmů přímo v mapě.

8.3 Webové stránky a distribuce DIGITOVu

DIGITOV je téměř zralý pro distribuci ostatním světovým univerzitám a jiným zájemcům o nekomerční nebo výukové využití, které je v souladu s licencí *Source Engine* a *Half-Life 2*. Je zde velký potenciál o navázání meziuniverzitní spolupráce nejen v této oblasti, ale i v jiných oborech.

K tomuto účelu bude třeba vytvořit webové stránky poskytující informace v češtině, angličtině a dalších jazycích. Stránky budou obsahovat návod k instalaci a četné odkazy na stránky *Valve Developer Community* [24], která poskytuje návod k použití *Source SDK* v angličtině a některých dalších jazycích. K tomu by měly být přeloženy pro *DIGITOV* specifické kapitoly. Zvažuje se možnost uvolnění e-booku Tvorba virtuálních prostředí v *Source SDK*, ale pouze v češtině.

8.4 Publikované články

Projekt zkoumání a vytváření nových možností modelování layoutů za použití počítačových her byl již publikovaný v konferenci na Krétě [26], celý článek je uveden v příloze II této práce. Publikace je uvedena v databázích *SpringerLink* a *Scopus*. Druhá publikace je uveřejněna v čísle časopisu *MM Science Journal* [27] z prosince 2013.

Autor projekt DIGITOV prezentoval na soutěži SVOČ na Univerzitě Tomáše Bati ve Zlíně, kde vyhrál první místo [37]. Dále je jeden článek o DIGITOVu uveden ve sborníku soutěže SVOČ Fakulty strojní Západočeské univerzity v Plzni [38].

9 Hodnocení

Výsledky práce přinášejí další možnosti využití *DIGITOVu*, především v oblasti algoritmizované tvorby layoutů a vizualizace výrobních procesů ve strojírenství.

DIGITOV jako takový přináší nekonvenční a unikátní vývojové prostředí, které je v mnoha ohledech schopné konkurovat profesionálním nástrojům, jejichž pořízení stojí na akademickou sféru příliš, řádově sto tisíc eur. Takovými nástroji jsou například *3DVIA Studio* nebo *IC.IDO* [3][39].

Algoritmizovaný převod layoutů ušetří práci řádově v hodinách při tvorbě layoutu, který byl již vytvořen ve *VisTable Touch*. Převod je sice algoritmizovaný částečně, nicméně dokáže zajistit nejpracnější část práce, a tedy přepočítávání rozměrů a umístění z metrických jednotek na *units* a přesné umisťování jednotlivých objektů. Konkrétní doba ušetřená touto činností však závisí na složitosti layoutu a schopnostech tvůrce.

Instalátor rovněž vede k ušetření práce. Ruční instalace *DIGITOVu* je poměrně složitá, i když jedna instalace trvá přibližně deset minut čistého času. V případě instalace na 13 počítačích tak dojde k ušetření asi dvou hodin práce. Dílčí časové úspory přináší i snadné spouštění hotových podnikových modelů a jejich veřejné prezentace budou důstojnější, protože nebude nutné spouštět konzoli *DIGITOVu* a teprve z ní pomocí příkazu *map* pouštět jednotlivé modely.

Uživatel může používat *DIGITOV*, pokud vlastní počítačovou hry *Half–Life 2: Episode Two*. Tuto hru je možné pořídit v současné době za €5,99 z distribuční služby *Steam*.

10 Závěr

Práce shrnula potřebné znalosti o *Source Engine* z technického hlediska a umožnila tak další vývoj, rozšíření a využití *DIGITOVu*. Byl vyvinut konvertor layoutů z *VisTable Touch do Source SDK*, který umožňuje značné ušetření práce při tvorbě nových layoutů. Dále byl vyvinut instalátor a spouštěč *DIGITOV*u. Tento program je možné používat i k distribuci a aktualizaci *DIGITOV*u. V poslední řadě byl zkompilován nový build dynamických knihoven *Server.dll* a *Client.dll*, které nesly chybu znemožňující tvorbu animací v *Source Engine*. Následně byla vytvořena animace soustružení.

Práce shrnula důležité teoretické poznatky, na základě kterých bude *DIGITOV* dále vyvíjen. Jedná se především o to, jak a v jakých formátech *Source SDK* a *Source Engine* uchovávají data. Na základě pochopení zdrojových kódu mapy se nabízí možnost vytvořit nástroj pro algoritmickou tvorbu mapy podle layoutu vytvořeného v jiném programu, například *VisTable Touch*. Zdrojový kód prototypu tohoto programu je uveden v příloze I.

Do budoucna byly stanoveny další cíle, kam by se vývoj *DIGITOVu* měl ubírat: doplnění knihoven, tvorba webových stránek o projektu, tvorba nového instalátoru. Nově zjištěné poznatky a nově vytvořené prvky budou ve formě návodu k použití zdokumentovány v novém vydání e-booku Tvorba virtuálních prostředí v *Source SDK*.

Projekt tvorby *DIGITOVu* a *DIGITOV* jako takový byl jako alternativní možnost tvorby podnikových layoutů za použití počítačových her již publikován [26][27]. Jeden z těchto článků je uveden v příloze II.

11 Literatura

[1] *Virtual training 'puts' real in realistic environment.* [2013-12-01] Dostupné z: <http://www.army.mil/article/97582/>.
[2] HOŘEJŠÍ, P., GÖRNET, T., KURKIN, O. *VYZTYMPDP: Virtuální realita a DP, e-book.* Plzeň: ZČU-KPV, 2012. ISBM 978-80-87539-07-1.
[3] *3DVIA Virtools*. [2013-12-01] Dostupné z: <http://a2.media.3ds.com/products/3dvia/3dvia-virtools/>.
[4] Polcar, Jiří. *Využití Source Engine pro vizualizaci a interakci v prostředí digitální továrny a tvorba studijních podkladů*. Plzeň, 2012. Bakalářská práce (Bc.). Západočeská univerzita v Plzni, Fakulta strojní. Vedoucí práce Petr Hořejší.
[5] *iD Tech – Wikipedia, the free encyclopaedia*. [2013-12-02] Dostupné z: <http://en.wikipedia.org/wiki/Id_Tech>.
[6] *Quake (video game) – Wikipedia, the free encyclopaedia*. [2013-12-02] Dostupné z: <http://en.wikipedia.org/wiki/Quake_(video_game)>.
[7] *Quake II – Wikipedia, the free encyclopaedia*. [2013-12-02] Dostupné z: <http://en.wikipedia.org/wiki/Quake_II>.
[8] *Half-Life (series) – Wikipedia, the free encyclopaedia*. [2013-12-02] Dostupné z: <http://en.wikipedia.org/wiki/Half-Life_(series)>.
[9] *John Carmack's Blog – Archive*. [2013-12-02] Dostupné z: <http://armadilloaerospace.com/n.x/johnc/recent%20updates/archive?news_id=290>.
[10] *VPhysics – Valve Developer Community*. [2013-12-02] Dostupné z: <https://developer.valvesoftware.com/wiki/VPhysics>.
[11] *Source BSP File Format – Valve Developer Community*. [2013-12-02] Dostupné z: <https://developer.valvesoftware.com/wiki/Source_BSP_File_Format>.
[12] *BSPZIP – Valve Developer Community*. [2013-12-02] Dostupné z: <https://developer.valvesoftware.com/wiki/BSPZIP>.
[13] *Decompiling maps – Valve Developer Community*. [2013-12-02] Dostupné z: <https://developer.valvesoftware.com/wiki/Decompiling_Maps>.
[14] Hořejší, P., Polcar, J. *Tvorba virtuálních prostředí v Source SDK*. ZČU FST, 2012.
[15] *Valve Hammer Editor – Half-Life Wiki*. [2013-12-02] Dostupné z: <http://half-life.wikia.com/wiki/Valve_Hammer_Editor>.
[16] *Nem's Tools [VTFLib - About]*. [2013-12-02] Dostupné z: <http://nemesis.thewavelength.net/index.php?p=40>.
[17] *Autodesk Students Community | Free Students Download on Login*. [2013-12-02] Dostupné z: <http://www.autodesk.com/education/student-software>.
[18] *wunderboy.org hl2/source sdk*. [2013-12-02] Dostupné z: <http://www.wunderboy.org/sourceapps.php#max_smd>.
[19] *Updated SMD exporter with support for skin modifier and bone weights*. [2013-12-02] Dostupné z: <http://www.chaosincarnate.net/cannonfodder/3dsmax.php>.
[20] *Galerie 3D objektů*. [2013-12-02] Dostupné z: <http://sketchup.google.com/3dwarehouse/?hl=cs>.
[21] *wunderboy.org guistudiomdl 2.x/source – gui wrapper for the source studiomdl compiler*. [2013-12-02] Dostupné z: <http://www.wunderboy.org/apps/guistudiomdl2.php>.
[22] *Source SDK 2013 – Valve Developer Community*. [2013-12-02] Dostupné z: <https://developer.valvesoftware.com/wiki/Source_SDK_2013>.
[23] *Visual Studio*. [2013-12-02] Dostupné z: <http://www.visualstudio.com/cs-cz>.
[24] *Valve Developer Community*. [2013-12-02] Dostupné z: <https://developer.valvesoftware.com/>.
[25] *<http://home.zcu.cz/~jpolcar/digitov/>*
[26] Horejsi, P., Polcar, J.: *Implementation of Source Engine for Virtual Tours in Manufacturing Factories, 9th International Symposium*, ISVC 2013, Rethymnon, Crete, Greece, July 29-31, 2013. Proceedings, Part II, pp 737-746, DOI: 10.1007/978-3-642-41939-3_72, series: 8034, Print ISBN: 978-3-642-41938-6, Online ISBN: 978-3-642-41939-3, ISSN: 0302-9743, Springer Berlin Heidelberg
[27] Horejsi, P., Polcar, J.: *AN UNCONVENTIONAL SOFTWARE ENVIRONMENT FOR FACTORY LAYOUT DESIGN AND AUTOMATED CONVERTER*, MM Science Journal, December 2013

[28] *Geometrické transformace v GIS.* [2014-03-31] Dostupné z: <http://geomatika.kma.zcu.cz/studium/ugi/referaty/05/GeometrickeTransformace/index.html>
[29] *DotNetZip Library.* [2014-03-31] Dostupné z: <http://dotnetzip.codeplex.com/>
[30] <http://home.zcu.cz/~jpolcar/DIGITOV>
[31] *Func_Movelinear – Valve Developer Community.* [2014-03-31] Dostupné z: <https://developer.valvesoftware.com/wiki/Func_movelinear>
[32] *Jet Model BDB-1304 Lathe – 3D Warehouse.* [2014-03-31] Dostupné z: <https://3dwarehouse.sketchup.com/model.html?id=7d39371696d5ec4f601bea3dca268229>
[33] *CfuncMoveLinear ParentingFix – Valve Developer Community.* [2014-04-1] Dostupné z: <https://developer.valvesoftware.com/wiki/CFuncMoveLinear_ParentingFix>
[34] *ValveSoftware / source-sdk-2013. GitHub.* [2014-04-1] Dostupné z: <https://github.com/ValveSoftware/source-sdk-2013>
[35] *Přehled produktů ke stažení – Microsfot Visual Studio 2010 Express.* [2014-04-1] Dostupné z: <http://www.visualstudio.com/downloads/download-visual-studio-vs#DownloadFamilies_4>
[36] *Download Microsoft Visual Studio Service Pack 1 (Installer) from Official Microsoft Download Center.* [2014-04-1] Dostupné z: <http://www.microsoft.com/en-us/download/details.aspx?id=23691>
[37] *Studentská vědecká a odborná činnost 2014, Univerzita Tomáše Bati ve Zlíně.* [2014-05-12] Dostupné z: <http://www.utb.cz/fame/o-fakulte/studentska-vedecka-a-odborna-cinnost-svoc-2014>
[38] SVOČ 2014. [2014-05-12] Dostupné z: <http://old.fst.zcu.cz/_files_web_FST/_SP_FST%28SVOC%29/_2014/_sbornik/Index.htm>
[39] *IC.IDO | ESI Group* – Virtual Product Engineering software and services. [2014-04-12] Dostupné z: <https://www.esi-group.com/software-services/virtual-reality/icido>

Příloha I

Výpis zdrojového kódu programu VIS2VMF

```
//vis2vmf.cpp
#include <iostream>
#include <fstream>
#include <string>
#include <conio.h>
#include <math.h>

using namespace std;
bool debug_rezim = false;



string odstran_priponu_ciselnou(string radek)
{
	bool vsechna_cisla = true;
	int pozice_tecky = radek.length(), dokud_for;
	if (radek.length()-5 <= 0) return radek;   //dodelat, kdyz je radek prilis kratky
	for (int i=radek.length()-1 ; i>=radek.length()-5 ; i--)
	{
		if ((radek[i] <= 48)||(radek[i] >= 57)) vsechna_cisla=false;
		if ((radek[i] == '.')&&(vsechna_cisla = true)) pozice_tecky=i;
	}
	radek.resize(pozice_tecky);
return radek;
}

string odstran_priponu_souboru(string radek)
{
	bool vsechna_cisla = true;
	int pozice_tecky = radek.length();
	for (int i=radek.length()-1 ; i>=radek.length()-5 ; i--)
	{
		if ((radek[i] <= 65)||(radek[i] >= 90)||(radek[i] <= 97)||(radek[i] >= 122))
vsechna_cisla=false;
		if ((radek[i] == '.')&&(vsechna_cisla = true)) pozice_tecky=i;
	}
	radek.resize(pozice_tecky);
return radek;
}

//prepise zdrojovy soubor ve formatu CSV do citelnejsi formy, DODELAT: jmeno vstupniho souboru
dle parametru
void uprav_zdrojovy_soubor()
{
	ifstream IN;
	ofstream OUT;
	string radek;
	int pocet_mezer, j=0;
	char novy_radek[512];
	char spatny_znak = 0; //ASCII kod spatneho znaku
	bool mezery = false;   //promenna mezery je indikatorem toho, jestli je formatovani
spatne, tj. kazdy druhy znak je ASCII 0 (NUL).

	IN.open("zdrojCSV", ios::in);
	OUT.open("temp.txt", ios::out);

	getline(IN,radek);
	//zjistuje se, zda prvni radek obsahuje hlavicku souboru nezavisle na tom, zda je chyba
ve formatovani a pripadne se rovnou nacte dalsi radek
	if ((((radek[0] == 'N' && radek[2] == 'a') && radek [4] == 'm') && radek[6] == 'e') ||
(((radek[0]=='N'   &&   radek[1]=='a')   &&   radek[2]=='m')   &&   radek[3]=='e')            )
getline(IN,radek);

	//Nyni se zjistuje, zda vstupni soubor obsahuje chybu ve formatovani, kdy kazdy druhy
znak je 0 (NUL).
	for (int i = 0; i <= radek.length() - 1 ; i++)
	{
		if (radek[i] == spatny_znak) pocet_mezer++;
		if (pocet_mezer > 10)  //pokud je techto spatnych znaku vice nez 10, muzeme
spolehlive tvrdit, ze jde o soubor se spatnym formatovanim
		{
			mezery = true;
			break;
		}
	}
```

```
        while(true) //konec cyklu je dle vlastni podminky
        {
                //Opraveni formatovani
                //znaky z promenne radek budou prekopirovany do promenne novy_radek, ve ktere
bude chyba formatovani osetrena
                j=0;
                if (mezery == true)
                {
                        for (int i=0; i <= radek.length() - 1; i++)
                        {
                                if (radek[i] != spatny_znak) //kopirovat kazdy znak, ktery neni
NUL
                                {
                                        novy_radek[j] = radek[i];
                                        j++;
                                }
                        }
                        novy_radek[j] = '\0';
                }
                else
                {
                        for (int i=0; i <= radek.length() - 1; i++)
                        {
                                novy_radek[j] = radek[i];
                                j++;
                        }
                        novy_radek[j] = '\0';
                }
                //Nahrazeni mezer podtrzitky a stredniku mezerami
                for (int i = 0; i <= strlen(novy_radek) - 1; i++) //vymeni mezery za podtrzitka
a stredniky za mezery
                {
                        if (novy_radek[i] == ' ')
                        {
                                novy_radek[i]='_';
                        }
                        else if (novy_radek[i] == ';')
                        {
                                novy_radek[i]=' ';
                        }
                }
                OUT << novy_radek << "\n";
                getline(IN,radek);  //getline je na konci v cyklu, aby vyhodil error pro
podminku ve while; proto je jeden getline i pred cyklem, aby nacetl prvni radek
                if (radek.length() < 10) break; //pokud bude delka radku mensi nez 10, urcite
jsme na konci souboru a prestaneme jej tedy cist
        }
        IN.close();
        OUT.close();
}


//Nacteni DAT souboru dle jeho jmena ze zdrojoveho textu a vystup jednoho znaku, ktery bude
ridit druh zapisovaneho objektu
//      Vystup:
//       E - neexistujici data
//       M - pro model do prop_static
//       N - pro NPC
//       B - pro brush
char precti_DAT(string& soubor, double& kx, double& ky, double& kz, double& kfi)
{
    ifstream DAT;
        string dat_soubor, typ = "alias"; //typ alias odkaze na jiny DAT soubor, ktery bude
nacten misto tohoto, vhodne pro hromadnou upravu
        while (typ == "alias")
        {
                dat_soubor = "data\\" + soubor +".dat";
                DAT.open(dat_soubor.c_str(), ios::in);
                if  (!DAT.is_open())  return  'E';  //preruseni  procedury  a  navrat  E  pro
neexistenci souboru
                DAT >> soubor >> typ >> kx >> ky >> kz >> kfi;
                DAT.close();
                if (typ == "model") return 'M'; else if (typ == "npc") return 'N'; else return
'B';
```

```
	}
}

//Pokud z funkce precti_DAT je vystupem znak M, pak se provede zapis entity tridy prop_static s pouzitym modelem dle DAT souboru
void zapis_model(ofstream& OUT, string soubor, int x,int y,int z,int fi, int id)
{
	if (debug_rezim == true) cout << "ZAPIS: prop_static";
	OUT << "entity\n{\n";
	OUT << "	\"id\" \"" << id << "\"\n";
	OUT << "	\"classname\" \"prop_static\"\n";
	OUT << "	\"angles\" \"0 " << fi << " 0\"\n";
	OUT << "	\"model\" \"" << soubor << "\"\n";
	OUT << "	\"skin\" \"0\"\n";
	OUT << "	\"solid\" \"6\"\n";
	OUT << "	\"origin\" \"" << x << " " << y << " " << z << "\"\n";
	OUT << "}\n";
}

//Pokud z funkce precti_DAT je vystupem znak N, pak se provede zapis entity tridy npc_citizen s pouzitym modelem dle DAT souboru
void zapis_npc(ofstream& OUT, string soubor, int x,int y,int z,int fi, int id)
{
	if (debug_rezim == true) cout << "ZAPIS: npc_citizen";
	OUT << "entity\n{\n";
	OUT << "	\"id\" \"" << id << "\"\n";
	OUT << "	\"classname\" \"npc_citizen\"\n";
	OUT << "	\"angles\" \"0 " << fi << " 0\"\n";
	OUT << "	\"model\" \"" << soubor << "\"\n";
	OUT << "	\"origin\" \"" << x << " " << y << " " << z << "\"\n";
	OUT << "	\"AlwaysTransition\" \"No\"\n";
	OUT << "	\"ammoamount\" \"1\"\n";
	OUT << "	\"ammosupply\" \"SMG1\"\n";
	OUT << "	\"citizentype\" \"Default\"\n";
	OUT << "	\"DontPickupWeapons\" \"No\"\n";
	OUT << "	\"expressiontype\" \"2\"\n";
	OUT << "	\"GameEndAlly\" \"No\"\n";
	OUT << "	\"physdamagescale\" \"1.0\"\n";
	OUT << "	\"renderamt\" \"255\"\n";
	OUT << "	\"rendercolor\" \"255 255 255\"\n";
	OUT << "	\"spawnflags\" \"1327622\"\n";
	OUT << "}\n";
}

//Pokud z funkce precti_DAT je vystupem znak N, pak se provede zapis entity tridy npc_citizen s pouzitym modelem dle DAT souboru
void zapis_brush(ofstream& OUT, string soubor, double px, double py, double pz, double pfi, double lx, double ly, double lz, int id, int side_id) //POZOR - tohle vsechno nejsou int ale double! krome id a side_id!
{
	OUT.precision(4);                                                    //POZN:      parametr prevest do INI souboru
	double s = sin(pfi/180*3.14159265359), c = cos(pfi/180*3.14159265359);
	double bod[8][3],vektor[4]; //bod - 8 vrcholu brushe a jejich tri souradnice; vektor - tri souradnice [0 1 2] a jeho norma [3]; souradnice vzdy [012]=[xyz]
	int cislo_bodu[6][3]; //pro kazdy brush je 6 sten definovano pomoci tri bodu (hodnoty 0 az 7 pro promennou bod)
	//s = sin(pfi/180*3.14159265359); //vyzkouset, jestli to funguje takhle!!!
	//c = cos(pfi/180*3.14159265359);
	bod[0][0] = bod[4][0] = px - lx/2*c + ly/2*s; bod[0][1] = bod[4][1] = py - ly/2*c - lx/2*s; bod[0][2] = bod[1][2] = bod[2][2] = bod[3][2] = pz;         //Vypocet souradnic vsech osmi vrcholu brushe
	bod[1][0] = bod[5][0] = px + lx/2*c + ly/2*s; bod[1][1] = bod[5][1] = py - ly/2*c + lx/2*s; bod[4][2] = bod[5][2] = bod[6][2] = bod[7][2] = pz + lz;
	bod[2][0] = bod[6][0] = px + lx/2*c - ly/2*s; bod[2][1] = bod[6][1] = py + ly/2*c + lx/2*s;
	bod[3][0] = bod[7][0] = px - lx/2*c - ly/2*s; bod[3][1] = bod[7][1] = py + ly/2*c - lx/2*s;
	cislo_bodu[0][0] = 4; cislo_bodu[0][1] = 7; cislo_bodu[0][2] = 6; //Urcuje, ktere tri body (druhy parametr) definuji tu kterou rovinu (prvni parametr).
	cislo_bodu[1][0] = 3; cislo_bodu[1][1] = 0; cislo_bodu[1][2] = 1; //Brushe jsou urceny ohranicenim rovinami.
	cislo_bodu[2][0] = 0; cislo_bodu[2][1] = 3; cislo_bodu[2][2] = 7; //Protoze zalezi na poradi bodu i sten, byly vylusteny z VMF souboru z Hammeru.
	cislo_bodu[3][0] = 2; cislo_bodu[3][1] = 1; cislo_bodu[3][2] = 5;
	cislo_bodu[4][0] = 3; cislo_bodu[4][1] = 2; cislo_bodu[4][2] = 6;
```

```
	cislo_bodu[5][0] = 1; cislo_bodu[5][1] = 0; cislo_bodu[5][2] = 4;

	//Uvozeni brushe (solid)
	OUT << "   solid\n";
	OUT << "   {\n";
	OUT << "      \"id\" \"" << id << "\"\n";


	for (int i=0; i <= 5; i++)
	{
		OUT << "      side\n"; //Uvozeni steny (side)
		OUT << "      {\n";

		OUT << "         \"id\" \"" << side_id << "\"\n";
		OUT << "         \"plane\" \""; //Uvozeni roviny (plane)
		//Cyklus zapise tri body urcujici rovinu
		for (int j=0; j <=2; j++)
		{
			OUT << "("; //Uvozeni jednoho bodu
			//Cyklus zapise tri souradnice bodu
			for (int k=0; k <=2; k++)
			{
				OUT << (int) bod[cislo_bodu[i][j]][k];
				if (k != 2) OUT << " ";
			}
			OUT << ")";  //Uzavreni jednoho bodu
			if (j != 2) OUT << " ";
		}
		OUT << "\"\n"; //Uzavreni plane
		OUT << "         \"material\" \"" << soubor << "\"\n";
		//Zapis U-axis - vektoru urcujici U souradnici v UV mape
		//Vypocet vektoru
		vektor[3] = 0; //Vynulovani normy vektoru
		for (int j=0; j <= 2; j++) //cyklus vypocet 3 souradnic vektoru definujiciho u-
axis textury
		{
			vektor[j]   =   bod[cislo_bodu[i][1]][j]   -   bod[cislo_bodu[i][0]][j];
//Vektor V je urcen jako rozdil bodu [n][0] - [n][2]
			vektor[3] = vektor[3] + vektor[j]*vektor[j];  //Pythahorova veta
		}
		vektor[3] = sqrt(vektor[3]); //Norma vektoru
		//Zapis vektoru do souboru
		OUT << "         \"uaxis\" \"[";
		for (int j=0; j <= 2; j++) //Cyklus pro zapis tri souradnic vektoru U
		{
			vektor[j] = vektor[j]/vektor[3]; //Normovani vektoru
			OUT << fixed << vektor[j] << " ";
		}
		OUT << "0] 0.25\"\n"; //Vychozi meritko pro textury je 0.25

		//To same pro V-axis
		vektor[3] = 0;
		for (int j=0; j <= 2; j++)
		{
			vektor[j] = bod[cislo_bodu[i][1]][j] - bod[cislo_bodu[i][2]][j]; //pro
vektor v to bude i0 a i2
			vektor[3] = vektor[3] + vektor[j]*vektor[j];
		}
		vektor[3] = sqrt(vektor[3]);
		OUT << "         \"vaxis\" \"[";
		for (int j=0; j <= 2; j++)
		{
			vektor[j] = vektor[j]/vektor[3];
			OUT << fixed << vektor[j] << " ";
		}
		OUT << "0] 0.25\"\n";
		//Konec zapisu V-axis

		OUT << "         \"rotation\" \"0\"\n";
		OUT << "         \"lightmapscale\" \"16\"\n";
		OUT << "         \"smoothing_groups\" \"0\"\n";

		OUT << "      }\n"; //Uzavira side
		side_id++;
	}
	OUT << "   }\n"; //Uzavira solid
```

```
	if (debug_rezim == true) cout << "ZAPIS: brush";
}

void main()
{
	uprav_zdrojovy_soubor(); //Vytvori ze zdrojoveho souboru ve formatu CSV oddeleneho stredniky soubor temp.txt oddeleny mezerami (puvodni mezery jsou nahrazeny _)
	//exit(1);
	ifstream IN;
	ofstream OUT;
	IN.open("temp.txt", ios::in);
	OUT.open("mapaVMF", ios::out);

	string soubor, mapa;
	double px, py, pz, kx, ky, kz, lx, ly, lz, pfi, kfi, area;
	int side_id,id, x, y, z, fi;
	double meritko = 0.052;                                          //do INI souboru
	char typ;
	id = 1;
	side_id = 1;
	bool obsahuje_brush = false;
	string vychozi_textura = "digitov/podlaha_dlazdice2modralesk";  //do INI souboru
	bool psat_nenalezene_jako_brushe = true;                                    //do INI souboru
	bool odstranit_temp = false;
	//do INI souboru


	//Ve VMF souboru jsou brushe uvozeny pod world, musi se tedy napsat jako prvni
	IN >> soubor >> px >> py >> pz >> lx >> ly >> lz >> pfi >> area;
	while(!IN.fail())
	{
		if (debug_rezim == true) cout << soubor << " " << px << " " << py << " " << pz << " " << lx << " " << ly << " " << lz << " " << pfi; //
		soubor = odstran_priponu_ciselnou(soubor); //Odstani ciselnou priponu z VisTable
		typ = precti_DAT(soubor, kx, ky, kz, kfi); //Vrati typ a parametry z DAT souboru
		if (debug_rezim == true) cout<<" "<<typ<<"    ";
	//	if (typ == 'B') {kx = 0; ky = 0; kz = 0; kfi = 0;}; //overit tuto podminku - predpokladem je, ze brush (B) nepotrebuje korekci, prozatim to tady necham, muze se preci jen hodit
	//	px = px + kx;
	//	py = py + ky;
	//	pz = pz + kz; //meritka jsou az u volani procedury zapisu brushe
	//	pfi = pfi + kfi;  // korekce - prepocet souradnic z mm na units, uhel je stejna jednotka
		if (typ == 'B' || ((typ =='E')&&(psat_nenalezene_jako_brushe)==true))
		{
			//Je zjisteno, ze se budou psat brushe a je tedy nutne zapsat jejich uvod
			if (obsahuje_brush == false)
			{
				OUT << "world\n"; //Uvozeni world
				OUT << "{\n";
				OUT << "   \"id\" \"" << id << "\"\n";
				OUT << "   \"mapversion\" \"5\"\n";
				OUT << "   \"classname\" \"worldspawn\"\n";
				OUT << "   \"detailmaterial\" \"detail/detailsprites\"\n";
				OUT << "   \"detailvbsp\" \"detail.vbsp\"\n";
				OUT << "   \"maxpropscreenwidth\" \"-1\"\n";
				OUT << "   \"skyname\" \"sky_day01_01\"\n";
				id++; //id uz je pouzito pro world, musi byt unikatni
				obsahuje_brush = true;
			}
			if (typ == 'E') soubor = vychozi_textura;
			if (abs(lx) < 52)
			{
				lx = 52; //minimalni rozmer brushe, asi bude lepsi se na to zeptat v INI soboru
				soubor   =   "digitov/podlaha_dlazdice2cervenalesk";   //vychozi textura pro tenke
			}
			if (abs(ly) < 52)
			{
				ly = 52;
```

```
                                soubor    =    "digitov/podlaha_dlazdice2cervenalesk";    //vychozi
textura pro tenke
                        }
                        if (abs(lz) < 52)
                        {
                                lz = 52;
                                soubor    =    "digitov/podlaha_dlazdice2cervenalesk";    //vychozi
textura pro tenke
                        }

        zapis_brush(OUT,soubor,px*meritko,py*meritko,pz*meritko,pfi,lx*meritko,ly*meritko,lz*me
ritko,id,side_id);
                        side_id = side_id + 6;
                        id++;
                }

                IN >> soubor >> px >> py >> pz >> lx >> ly >> lz >> pfi >> area;
                if (debug_rezim == true) cout << "\n";
        }

        if (obsahuje_brush == true) OUT << "}\n"; //Uzavreni world


        IN.clear();
        IN.seekg(0, ios::beg); //vrati se na zacatek vstupniho souboru
        if (debug_rezim == true) cout << "Zapis brushu ukoncen, nyni se zapisi entity\n";

        //Pokracuje zapis bodovych entit
        IN >> soubor >> px >> py >> pz >> lx >> ly >> lz >> pfi >> area;
        while(!IN.fail())
        {
                if (debug_rezim == true) cout << soubor << " " << px << " " << py << " " << pz
<< " " << lx << " " << ly << " " << lz << " " << pfi;
                soubor = odstran_priponu_ciselnou(soubor);
                typ = precti_DAT(soubor, kx, ky, kz, kfi);
                if (debug_rezim == true) cout<<" "<<typ<<"    ";
                x = px*meritko + kx;
                y = py*meritko + ky;
                z = pz*meritko + kz;
                fi = pfi + kfi;  // korekce - prepocet souradnic z mm na units, uhel je stejna
jednotka
                if  (typ  ==  'M')  zapis_model(OUT,soubor,x,y,z,fi,id);  else  if  (typ  ==  'N')
zapis_npc(OUT,soubor,x,y,z,fi,id); else id--; //pokud se nic nezapise, musi se snizit id zpet
o 1
                id += 1;       // id parametr ve VMF souboru
                IN >> soubor >> px >> py >> pz >> lx >> ly >> lz >> pfi >> area;
                if (debug_rezim == true) cout << "\n";
        }

        IN.close(); OUT.close();
        //getch();
        if (odstranit_temp == true) remove("temp.txt");
}
```

Příloha II

Implementation of Source Engine for Virtual Tours in Manufacturing Factories

Implementation of Source Engine for Virtual Tours in Manufacturing Factories

P. Horejsi, J. Polcar

University of West Bohemia, Univerzitni 8, 306 14 Plzen

Abstract. When we discuss virtual reality as a medium, there are two main genres: virtual tour and virtual training. This paper deals with the use of virtual reality for interactive virtual tours in manufacturing factories. We developed a brand new software **package/library DIGITOV** which serves as an aid for constructing new virtual interactive manufacturing factory models using Source Engine. This graphic engine is known for its use in the Half-Life 2 computer game [1]. One of the general advantages of adapting this engine is the public availability of the powerful development software tools included in the Source SDK. While adapting the engine to a new environment it is necessary to prepare a library consisting of many new 3D models, textures, sounds, choreographed scenes etc. The package includes for instance: machines, products, parts of a manufacturing line, specific sounds etc.

1 Introduction

We can observe a visible trend of penetration of virtual reality into everyday praxis. Many industrial corporations have their own concept of a digital factory: all the aspects of manufacturing are digitally verified on digital mock-ups before physical manufacturing. This approach brings significant cost savings. Virtual reality is used mainly in the validation phase; for example, verifying the design and functionality of a digital prototype, the driving properties of a physically non-existing car, ergonomics of a workplace etc. We will focus on the validation of manufacturing companies' layout and employee training. Using the further described virtual tours it is possible to visualize the whole factory including administration space, validate the perspective for working activities, have a virtual discussion with guides, etc. Generally it is possible to perform many more interactive actions in a customizable environment. All the virtual tours can be practically made using stereoscopic projection in a CAVE (Computer Aided Virtual Environment) using a haptic controller (see Fig. 1). There is a possibility of using supported DirectX stereoscopy with the Razor Hydra controller.

There are several software tools for modeling digital factories with the possibility of adding models from a 3D library to user composition. Two of today's major solutions are the Dassault Systeme DELMIA and Siemens Tecnomatix. These can be used to perform a large number of analyses [2]. From these two packs the full virtual tour in CAVE can be provided by Teamcenter VisMockUp software but with limited interaction. The other suitable tool is VisTable which natively supports a virtual tour but without the possibility of interacting with the virtual world.

All the universal software tools for virtual environment development are relatively expensive. This is why we have been searching for a way to use a widely spread environment for enterprise and factory design. From the practical side: we were also looking for a modular possibility in order to develop a universal learning tool for a virtual world design course for industrial engineers. After researching more possibilities we have chosen to modify Source Engine. Its single non-commercial license costs about €10.

Fig. 1. Target single-screen CAVE with IS-900 tracking device

2 Serious Games

Our system (see below) uses the principles of serious gaming. This term relates to specific interactive simulations where the main reason is not only to entertain, but also to educate the user or engage him/her in solving particular problems. A certain amount of fun while playing is required, though. One of the first applications of serious gaming came from the army. There is for example a free game by America's Army for training soldiers and advertising [3]. Another technically simpler game is for example Food Force. It was developed by the United Nations and informs about famine in the world [4].

Serious games, like any other computer game, can be developed from scratch using conventional programming environments or with the aid of so called 'sandbox' programs. These programs can be used to accelerate the development of virtual environments with the possibility of implementing interactive elements. 2D or 3D objects can usually be imported from standard formats, the interactive elements are implemented by block programming or scripting. Some examples of stand-alone sandbox solutions are Thinking Worlds, WinterMute Engine, Unity3D or VirTools (with native supports of multicluster visualization in CAVE). Another possibility is the use of game editors made by game developers simultaneously with the engine (which is the case described in this paper).

Adaptation of standard game engines for serious games is very frequent. For example, CryEngine has been successfully adapted for training Dubai's S.W.A.T police commandos [5], the US Army [6] and even as a surgery simulator [6]. Source Engine was used for example for a restaurant hygiene simulator [8] and other serious games.

3 Source Engine and Source SDK

Another implementation described here uses **Source Engine**, which is a modular game engine for PC, Linux, Mac OS, PS3 and Xbox. Source Engine was released in 2004 for the computer game Counter-Strike: Source, and then for Half-Life 2 one year later. Source is a high-quality graphics core with simulated physical systems support using the Havok engine. Source engine

is based on the DirectX architecture with the possibility of High Dynamic Range. More features such as multicore rendering support, Hardware Facial Animation, "Soft" particles, etc. have recently been added [9].

The virtual environment is built by a **map** (or level). The virtual world can be comprised of more interconnected maps. All particular environments are limited to a user-defined enclosed volume which is composed of **brushes**. Brushes are basic 3D objects that represent walls, floors, ceilings, cliffs, terrain, etc. The world details: e.g. furniture, humans, trees or sometimes even whole buildings (with esthetic functionality only) are represented by **3D models** (static, dynamic and physical props). Brushes are objects of lower detail than 3D models and are modeled and edited directly in Source SDK enclosed tools. On the other hand, 3D models are modeled using complex 3D editing tools like 3DS MAX, Blender or XSI SoftImage and then imported to a Source based map. All items and NPCs (Non-Player Characters) are represented by detailed 3D models. Functional parts of maps are implemented by so-called **entities** (doors, lifts, switches, light control, any mathematical or physical logic, etc.).

Every virtual environment developed in Source Engine is physically stored in a specific directory structure which includes for instance these components (directories):

- Maps – all maps are stored here (BSP - Binary Space Partitioning format)
- Materials – contains all textures (VTF – Valve texture File) including a material description file in text format (VMF – Valve Material File)
- Models – all the detailed 3D models (MDL format). Only special textures from Materials directory can be used for models.
- Scenes – scripted scenes data (choreography) created by the FacePoser tool which is included in the Source SDK
- Sound – sounds in WAV format

This structure can be compiled into a single .gfc archive. There is also a possibility to inherit data from other modifications (adapted Source Engine packs) or Source powered games. In other words, it is possible to use all previously designed models, textures, NPCs, etc.

Components of the virtual environments in Source Engine are created and edited using the Source SDK which is a powerful development package tool which can be downloaded via the Steam distribution system by the owner of a valid Source engine game (like Half Life 2). This environment includes a possibility of new modification (so called 'mods') development. Within or without this modification a new map can be developed. There is also a tool for new map creation called **Hammer Editor**. Software tool FacePoser can be used for creating choreographed scenes including face mimics and lips to sound synchronization. There are a lot of other tools included, usually file format converters, such as image to VTF (Valve Texture File) and many more. The source code of the engine is included, so the possibilities are virtually unlimited.

4 Reference Model

First a reference model was created in order to validate the possibility of developing such complex models in the given engine.

Fig. 2. Manufacturing layout of a reference model (in Siemens NX)

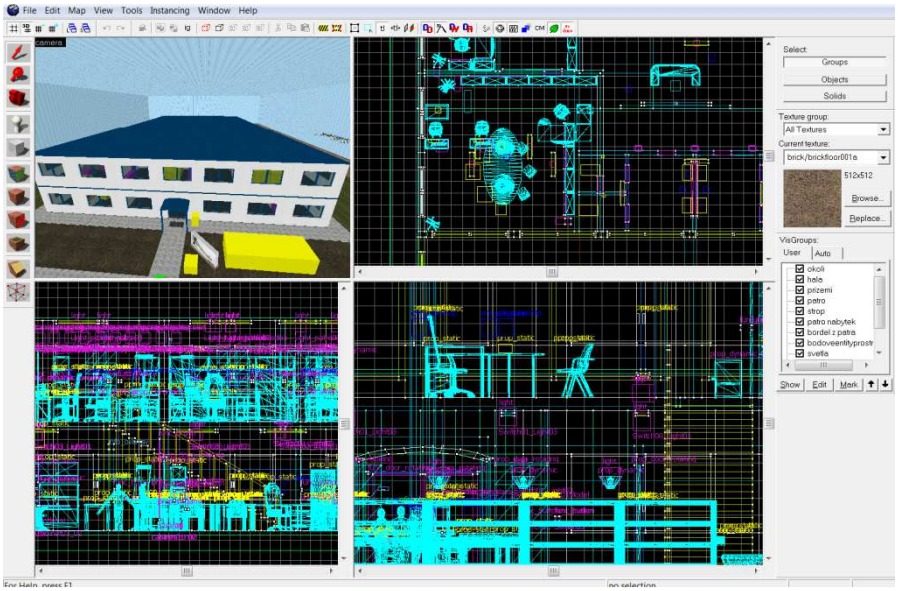


Fig. 3. Example of the development of the first reference model

An extensive virtual re-engineering project was used as an ideological base. In this project each step of the referential product (RC car) production planning was digitally composed starting with a raw design and ending with assembly line simulation optimization. All the production data including 2D and 3D layout from the Technomatix software package were available (such as 3D layout in Fig. 2). The assembly line was re-created using the Source SDK tools. The project output data also helped with construction of other non-manufacturing facilities (offices, canteen, toilets, exterior etc.). Simple interactive features were implemented, such as switchable lights, doors, ladders, etc. More complex interactivity was also added: e.g. the conveyor belts control, interaction with the operators and choreographed scenes with the virtual actors (mentors) in general.

During the development (see Fig. 3) it was revealed that the number and type of models and textures included in the original core was not sufficient. 3D models of workshop and office furniture, machines, IT equipment, etc, were missing. Based on the original gaming purpose, it is clear that the original includes models and textures which are supposed to immerse the

user in a thrilling atmosphere, for example the furniture is sometimes half-broken and the walls are dirty, ergo they were not suitable to be used for our case.

5 DIGITOV package

The evaluation of the development, implementation and validation process of the referential model raised the need for new content. So as to be able to effectively create new virtual models of manufacturing factories, it was necessary to produce a brand new library comprising 3D models, textures, interactive features and so called in Source SDK **pr**

efabs (logically grouped brushes, entities and 3D models). Some of these were created while working on the reference model. In order to maximize the modularity of the "building set", a lot more of such components were needed to be added. The DIGITOV modification for the computer game Half-Life 2: Episode Two started to take form. This is not a standard game modification, like a new story, but a collection of components for developers of virtual enterprise environments. The package includes an automatic installer developed in Delphi. There is just one requirement for the DIGITOV package to work: to own a user license for the Half-Life 2: Episode Two computer game, preferably in the Orange Box edition.

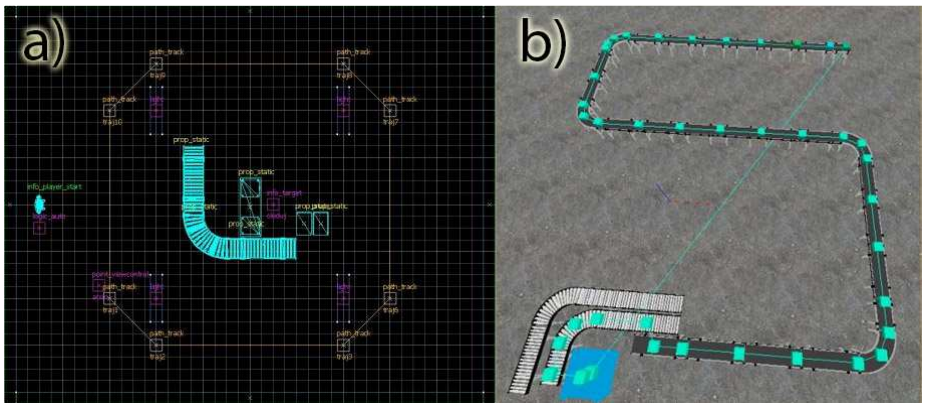


Fig. 4. a) conveyor belt curve prefab, b) conveyor belt constructed from prefabs in Hammer Editor

The DIGITOV currently contains approximately 150 new 3D models, more than 50 textures for models, more than 50 textures for maps and more than 25 static and dynamic prefabs. Static prefabs represent usually concrete or brick blocks with/without windows/doors, from which a raw factory layout could be composed. Dynamic prefabs are sets of prepared interactive aggregations. One example of dynamic prefabs from the DIGITOV package could be particular active parts of a conveyor belt (start/end, straight, curve, T-shape part, etc.), where besides the model and track path are also defined points where the product model is transformed (assembled), where the belt can be connected to another part of the belt, where the operator performs interactions etc. (see Fig. 4)

The package also contains various predefined choreographed scenes with virtual employees like various types of greetings, mentoring, presentations and more. These can be assembled into arbitrary sequences in the level editor. The original virtual actors' clothes were "virtually cleaned and ironed" (see Fig. 5-a). All these choreographic scenes were prepared using the FacePoser tool, where the particular face mimics and speech is synchronized and composed into replicas in a timeline (see Fig. 5-b). For English language Microsoft Speech SDK can be used for automatic mimic and voice synchronization. For other languages time consuming manual synchronization has to be done.

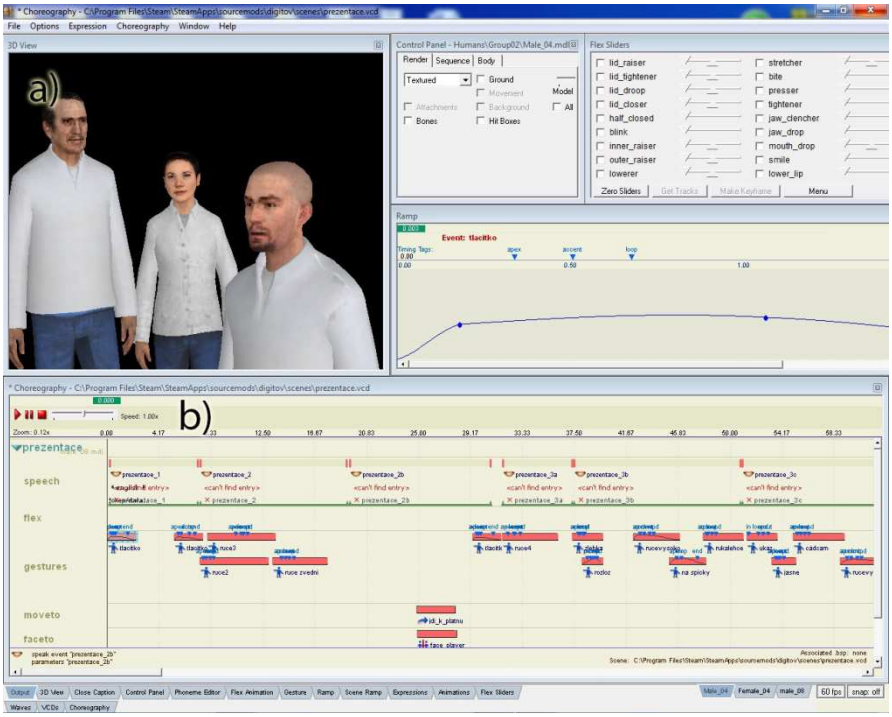


Fig. 5. a) Re-skinned NPC employees in Face Poser, b) Action editor

The DIGITOV package is installed in the modifications folder. While launching the Source SDK tool, the DIGITOV mod can be simply selected from the menu instead of the default Source Engine 2009, then all the additional options are available in the Valve Hammer Editor.

6 An example of DIGITOV package implementation

Let us show one of the virtual enterprise models which was designed with the aid of the DIGITOV package from a user point of view.

The first person camera perspective view of the model is shown just after the virtual environment has been loaded. The control system help is displayed immediately after loading the model. The user has a complete mouse and keyboard controlled freedom of movement. The right mouse key serves for displaying labels or hints - mainly for interactive elements (like offices, persons, products, etc.). This feature is very useful for instance for key elements of conveyor belt inspection.

While taking a tour in one of the models the user follows a storyboard (see Fig. 6):

Fig. 6. Storyboard

a) External view of the manufacturing enterprise. It is possible to examine the parking lot, climb up on to the roof. Let us take a look inside,
b) where the user is welcomed. A virtual lady will take us to the presentation room,
c) where the lecturer will show us some basics of the production in this factory (this presentation is more than 10 minutes long, but can be skipped)
d) then the user is brought to the manufacturing section. Each particular section of the manufacturing process can be inspected.
e) The user can also be familiarized with the functions of every single administrative office in the administrative part of building.
f) The manufacturing section can be observed from a bird's-eye view.

7 Validation, conclusion and results

The developed package is an alternative to other enterprise design software. It is a convenient tool not only for students, but can be used also for commercial use (a valid commercial license needs to be purchased by an enterprise). It can be used for instance to show the factory to new employees, who are recruited in another part of the world or for common visualization of not yet existing enterprise buildings for many reasons.

A manual and an education e-book (with more than 200 pages) were written alongside development. 14 authors spent 3 years developing DIGITOV and the e-book.

The ready-to-use package has been used for making 15 complex virtual enterprise models; two of them were models of real factories, validated by its staff; two more models of residential houses and a partial model of a university building. More reusable assets were designed and added to the package during the development of the models.

In comparison with large commercial tools for factory design like Delmia, Tecnomatix or Autodesk Factory CAD the proposed package does not offer the option to perform analyses of material flow, capacity tasks, ergonomics task, etc. but the DIGITOV package has advantages in better visualization, solution of training tasks and user interactivity implementation with low-cost. Industrial engineers who were approached reacted very positively to the presentation power and high fidelity of simulation. DIGITOV offers an assets library which can compete with factory design software packages.

Right now with the DIGITOV package and a little experience a fully interactive factory design can be produced in a few hours. For example, in comparison with DELMIA, the same factory layout was developed in half the time.

The DIGITOV package successfully supports the lessons of the “Digital enterprise and virtual reality” subject (so far for two years). The students are shown the basics of developing virtual worlds. The package is also being used for lessons at the “Summer school of virtual reality” for high school students.

The environments can be viewed stereoscopically on a 3D monitor or in CAVE. The virtual world can be controlled using a keyboard and a mouse or with a special Razor Hydra controller 9, where a sensor monitors the position of the special controllers in both hands via a magnetic field. The right location has to be found while using this controller in CAVE because of possible interference issues.

Although the DIGITOV package represents a powerful alternative tool for making mock-ups of virtual enterprises, there are still some ideas for improvement, such as:

- Dynamic animations in some models are missing.
- The map must be run from the console using the “map” command – a GUI launcher would be suitable.
- Interactive height selection of the player in order to verify the working positions perspective for different operator heights.

One of these recently corrected imperfections was the low count of textures. Volunteers searched the internet for usable free textures. Then a texture service pack was released containing more than 1000. Another recent validated potency is the possibility of adding more remote host users into a map and then to arrange off-distance briefings in a virtual factory.

The main point of the subsequent research is to develop a convertor, which will automatically convert a factory model from VisTable software into DIGITOV ready maps.

Acknowledgements

This paper was prepared with support of the Internal Science Foundation of the University of West Bohemia SGS–2012-063.

References

1. Source Engine games: http://www.giantbomb.com/source-engine/3015-751/games/ [06/2013]
2. Kurkin, O., Januska, M., Product Life Cycle in Digital factory, In: KNOWLEDGE MANAGEMENT AND INNOVATION: A BUSINESS COMPETITIVE EDGE PERSPECTIVE, VOLS 1-3, 15th International-Business-Information-Management-Association, Cairo, EGYPT, NOV 06-07, 2010, ISBN: 978-0-9821489-4-5 (2010) 1881-1886
3. America's Army: http://www.americasarmy.com/ [06/2013]
4. Food Force 2: https://code.google.com/p/foodforce/ [06/2013]
5. Dubai police training: http://tbreak.com/megamers/16428/features/dubai-police-uses-cry-engine-3-for-swat-training/ [06/2013]
6. Intelligence Electronic Warfare Tactical Proficiency Trainer: http://www.peostri.army.mil/PRODUCTS/IEWTPT [06/2013]
7. Virtual Surgery Far Cry Demo: http://www.youtube.com/watch?v=QF0yyfhchvg [06/2013]
8. Mac Namee, Brian et al.: Serious Gordon: using serious games to teach food safety in the kitchen. 9th. International Conference on Computer Games: AI, Animation, Mobile, Educational and Serious Games CGAMES06 (2006)
9. Wikipedia: Source(game engine): http://en.wikipedia.org/wiki/Source_(game_engine) [06/2013]
10. Razer Hydra: http://www.razerzone.com/gaming-controllers/razer-hydra [06/2013]
11. Kurkin, O., Simon, M., Optimization of Layout Using Discrete Event Simulation, In: BUSINESS TRANSFORMATION THROUGH INNOVATION AND KNOWLEDGE MANAGEMENT: AN ACADEMIC PERSPECTIVE, VOLS 1-4, 14th International-Business-Information-Management-Association Conference, Instanbul, TURKEY, JUN 23-24, 2010, ISBN: 978-0-9821489-3-8