

Návrh a realizace 3D počítačové hry - základní modely prostředí a interakce

The design and implementation of 3D computer game - basic
models and interaction

Ondřej Květ



Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2010/2011

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: Ondřej KVĚT

Osobní číslo: A08065

Studijní program: B 3902 Inženýrská informatika

Studijní obor: Informační a řídicí technologie

Téma práce: Návrh a realizace 3D počítačové hry – základní modely a interakce

Zásady pro vypracování:

1. Vypracujte literární rešerši na zadané téma.
2. Navrhněte základní mechaniku 3D počítačové hry a její pravidla.
3. Navrhněte design herního prostředí a objektů a vymodelujte je v Blenderu. Pro tyto modely vytvořte vhodné textury.
4. Zvolte vhodný fyzikální engine a implementujte jej do hry.
5. Pomocí zvukového engine irrKlang implementujte do hry ozvučení.
6. Navrhněte a vytvořte menu a uživatelské prostředí pro hru.



Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. ADAMS, Ernest. Fundamentals of Game Design. 2nd edition. [s.l.] : New Riders, 2010. 675 s. ISBN 9780321643377.
2. GEBHARDT, Nikolaus. Irrlicht Engine : A free open source 3d engine [online]. 2009 [cit. 2011-01-25]. Dostupné z WWW: [http://irrlicht.sourceforge.net/].
3. POKORNÝ, Pavel. Blender : naučte se 3D grafiku. 2. české. [s.l.] : BEN, 2009. 288 s. ISBN 978-80-7300-244-2.
4. THORN, Alan. Introduction to Game Programming with C++. [s.l.] : Wordware Publishing, 2007. 392 s. ISBN 9781598220322.
5. IrrEdit : A free realtime 3D world editor [online]. 2008 [cit. 2011-01-25]. Dostupné z WWW: [http://www.ambiera.com/irredit/index.html].
6. IrrKlang : An audio library for C++, C and .NET and high level 3D and 2D sound engine [online]. 2008 [cit. 2011-01-25]. Dostupné z WWW: [http://www.ambiera.com/irrklang/index.html].

Vedoucí bakalářské práce:

Ing. Pavel Pokorný, Ph.D.

Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce:

25. února 2011

Termín odevzdání bakalářské práce:

7. června 2011

Ve Zlíně dne 25. února 2011

prof. Ing. Vladimír Vašek, CSc.
děkan



prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Cílem teoretické části bakalářské práce je osvojení principů tvorby počítačových her v grafickém enginu Irrlicht se zaměřením hlavně na implementaci fyzikální a zvukové knihovny a tvorbu uživatelského rozhraní hry. V praktické části je to pak vytvoření jednoduché počítačové 3D hry zvoleného žánru ve spolupráci se studentem, který řeší téma „Návrh a realizace 3D počítačové hry - komplexní modely a jádro hry“.

Klíčová slova: Irrlicht, 3D engine, 3D počítačová hra, Newton Game dynamics, IrrEdit, IrrKlang, Blender

ABSTRACT

The goal of theoretical part of this work is to master the principles of creating computer games in the Irrlicht graphics engine, focusing mainly on implementation of sound and physical library and graphical user interface. The practical part consists of creating simple 3D computer game of chosen genre in collaboration with the work called „Design and implementation of 3D computer game - complex models and the core of the game“.

Keywords: Irrlicht, 3D engine, 3D computer game, Newton Game dynamics, IrrEdit, IrrKlang, Blender

Rád bych poděkoval vedoucímu své bakalářské práce, Ing. Pavlovi Pokornému, Ph.D. za rady a nápady v průběhu vzniku této práce. Další velký dík patří mým rodičům a sourozencům za podporu. Dále děkuji svým přátelům a spolužákům, kteří byli tak ochotní a testovali výsledek praktické části práce na svých počítačích, jejich dojmy a nápady mi mnohokrát pomohly. Největší vděk má moje přítelkyně Bára Bártová, která mi poskytla Múzu a velkou dávkou psychické opory.

Motto: „Týmová práce je zpravidla práce mnoha lidí pro jednoho člověka.“

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD.....	10
I TEORETICKÁ ČÁST	11
1 GRAFICKÝ ENGINE IRRLICHT	12
1.1 O ENGINU IRRLICHT.....	12
1.2 MOŽNOSTI A EFEKTY ENGINU IRRLICHT	13
1.3 PODPOROVANÉ FORMÁTY IRRLICHTU	14
1.4 JMENNÉ PROSTORY IRRLICHTU (NAMESPACES).....	15
1.5 DATOVÉ TYPY IRRLICHTU.....	16
1.5.1 Číselné a znakové datové typy:	16
1.5.2 Vektory a matice	16
1.5.3 Řetězce	17
1.5.4 Kontejnery	17
1.6 NEJDŮLEŽITĚJŠÍ ČÁSTI ENGINE IRRLICHT	17
1.6.1 IrrlichtDevice.....	17
1.6.2 Manažer scény Irrlichtu – ISceneManager	18
1.6.3 Uživatelské prostředí v Irrlichtu – IGUIEnvironment.....	18
1.6.4 Vykreslovací rozhraní Irrlichtu – IVideoDriver	19
1.6.5 Odchytávání vnějších událostí – IEventReceiver.....	19
1.7 VÝVOJ IRRLICHTU A JEHO AUTOŘI.....	19
1.8 IRRLICHT TOOL SET.....	22
2 3D EDITOR IRREDIT	23
2.1 O 3D EDITORU IRREDIT	23
2.1.1 Možnosti IrrEditu	24
2.1.2 Light map generátor	24
2.2 VÝVOJ IRREDITU.....	24
3 ZVUKOVÝ ENGINE IRRKLANG	26
3.1 O ZVUKOVÉM ENGINU IRRKLANG	26
3.2 PODPOROVANÉ FORMÁTY SOUBORŮ	27
3.3 EFEKTY A MOŽNOSTI ENGINU IRRKLANG.....	27
3.3.1 Podpora 3D zvuku	27
3.3.2 Zvukové efekty.....	27
3.4 POUŽITÍ ENGINU IRRKLANG.....	28
3.4.1 IrrKlangDevice	28
3.4.2 Přehrávání 2D zvuku	28
3.4.3 Přehrávání 3D zvuku	29
3.4.4 Zdroje zvuku	29
3.4.5 Další užitečné funkce enginu	29
4 FYZIKÁLNÍ ENGINE NEWTON GAME DYNAMICS	30

4.1	O FYZIKÁLNÍM ENGINU NEWTON	30
4.2	ZÁKLADY ENGINU NEWTON	30
4.2.1	NewtonWorld	30
4.2.2	Charakterizace objektu v enginu Newton	31
4.2.2.1	NewtonCollision	31
4.2.2.2	NewtonBody	33
4.2.3	Callback funkce	33
5	XML PARSER TINYXML	34
5.1	STRUČNĚ O XML	34
5.2	O XML ANALYZÁTORU TINYXML	34
5.2.1	Načtení souboru	35
5.2.2	Čtení hodnot	35
5.2.3	Zapisování hodnot	35
6	BLENDER	36
6.1	STRUČNĚ O BLENDERU	36
6.2	VÝVOJ BLENDERU	36
6.3	MOŽNOSTI BLENDERU	37
6.3.1	Uživatelské rozhraní	37
6.3.2	Modelování	37
6.3.3	Animace	37
6.3.4	Rendering	37
6.3.5	Tvorba her	38
6.3.6	Soubory	38
II	PRAKTICKÁ ČÁST	39
7	TVORBA POČÍTAČOVÉ HRY V ENGINU IRRLICHT	40
7.1	NÁVRH POČÍTAČOVÉ HRY A JEJÍ PRAVIDLA	40
7.1.1	Volba žánru hry	40
7.1.2	Grafický styl hry	40
7.1.3	Prostředí hry	41
7.1.4	Hlavní hrdina hry	42
7.1.5	Ovládání hry	42
7.1.6	Pravidla hry a postup hráče	43
7.1.7	Fyzika a její vliv na hratelnost	44
7.2	NÁVRH UŽIVATELSKÉHO ROZHRANÍ HRY	44
7.2.1	Hlavní menu hry	44
7.2.1.1	Návrh menu a požadavky	44
7.2.1.2	Struktura menu	45
7.2.1.3	Třída pro zapouzdření nastavení hry	45
7.2.1.4	Třída pro zapouzdření hlavního menu	47
7.2.2	Inventář	48
7.2.2.1	Návrh inventáře a požadavky	48
7.2.2.2	Vzhled a struktura inventáře	49
7.2.2.3	Třída pro charakterizaci předmětů	50
7.2.2.4	Třída pro herní inventář	52

7.2.3	Rozhraní pro rozhovory	53
7.2.3.1	Souborová struktura rozhovorů	53
7.2.3.2	Datová struktura rozhovorů	54
7.2.3.3	Třída pro objekty k navázání rozhovoru	56
7.3	GRAFICKÁ STRÁNKA HRY	57
7.3.1	Modelování	57
7.3.1.1	Modelovací techniky	57
7.3.2	Texturování	58
7.4	IMPLEMENTACE FYZIKÁLNÍHO ENGINU	60
7.4.1	Implementace fyziky – úroveň jádra hry	60
7.4.2	Implementace fyziky – úroveň objektů ve hře	61
7.4.2.1	Třída pro charakterizaci mapy	61
7.4.2.2	Třída pro charakterizaci fyzikálního objektu	62
7.5	OZVUČENÍ HRY	62
7.5.1	Hudba	63
7.5.2	Zvuky a efekty	64
7.6	TVORBA OBSAHU HRY	64
7.6.1	Tvorba úrovně v IrrEditu	64
7.6.2	Tvorba úkolů do hry	66
7.6.2.1	Příklad prvního úkolu hry	67
7.6.2.2	Skriptování úkolů	67
ZÁVĚR		68
ZÁVĚR V ANGLIČTINĚ		70
SEZNAM POUŽITÉ LITERATURY		72
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK		74
SEZNAM OBRÁZKŮ		75
SEZNAM TABULEK		76
SEZNAM PŘÍLOH		77

ÚVOD

Herní průmysl je dnes téměř stejně výdělečný, jako filmový. Programátorům pracujícím s objektovými jazyky se tak otevírá další cesta, jak se uplatnit. Kromě programátorů je při tvorbě počítačové hry také zapotřebí velká skupina lidí s různým zaměřením, od 2D či 3D grafiků přes designéry až po kodéry skriptovacích jazyků. Tvorba počítačových her tedy vyžaduje velké nasazení. Obsahuje totiž spoustu práce, která je pro finálního konzumenta neviditelná. Na kvalitním tvůrci her tedy je, aby při prvotním návrhu posoudil, které jeho části do výsledné aplikace implementovat a které ne.

Důsledkem velkého množství peněz v tomto odvětví je tedy také velká konkurence. Každým dnem vychází nespočet herních titulů různých žánrů a stylů, snažíc se ukousnout si z celkového koláče tohoto průmyslu svůj kousek. Ceny těchto titulů jsou velice variabilní, velký počet z nich je dokonce uvolněn zdarma. Uchytit se na tomto trhu není tedy vůbec jednoduché, tvůrce musí hráči nabídnout v porovnání s konkurencí určitou přidanou hodnotu. Základním předpokladem pro tvorbu dobré hry je však toto: hra musí především bavit. K dosažení tohoto cíle může tvůrce použít různých prostředků. Může hráči nabídnout výzvu, po jejímž překonání se dostaví pocit zadostiučinění, nebo třeba pěknou grafikou a příběhem může být nástrojem k úniku od reality, podobně jako knihy či filmy. Možnosti jsou téměř nekonečné.

V práci, jejíž literární rešerše je na následujících stránkách, jsem se zabýval pouze úzkým spektrem problematiky tvorby počítačových her, jako je ozvučení, implementace fyziky a kolizí nebo tvorba uživatelského rozhraní. Získaných poznatků jsem později použil v praktické části práce, kterou byla tvorba jednoduché počítačové hry společně s mým kolegou, který řešil přidružené téma. Jednotlivé aspekty tvorby byly předem rozděleny a každý tak dostal svůj díl práce na finálním projektu.

I. TEORETICKÁ ČÁST

1 GRAFICKÝ ENGINE IRRLICHT

1.1 O enginu Irrlicht

Irrlicht je 3D grafický realtime engine napsaný v jazyce C++. Je vybaven vysokoúrovňovým API pro tvorbu 3D i 2D aplikací, jako jsou hry nebo různé vizualizace. Je nezávislý na platformě, bez změny v kódu může běžet na Windows, Linuxu, OSX, Solaris nebo na platformách používajících SDL. Vysoká multiplatformnost engine umožňuje autorovi napsat aplikaci pouze jednou, a pak ji bez problému portovat na další platformy bez nutnosti změnit jediný řádek kódu. Název enginu pochází z německého slova irrlicht, které v češtině znamená bludička. [1], [10]



Obrázek 1 – logo enginu Irrlicht

Irrlicht engine je open source, uvolněn pod licencí Zlib a je zcela zdarma, pouze jména autora a enginu byla vyhrazena. Programátor tedy může ladit i opravit chyby přímo v knihovně, nebo do ní doplnit funkce, které jí chybí. Licence Zlib programátora nesvazuje povinností zveřejňovat své změny v engine a vytvořený kód lze využívat i ke komerčním účelům. V dokumentaci k výslednému projektu je ale poté nutno zmínit, že při tvorbě byla použita technologie Irrlicht. [1]

Irrlicht engine se snadno používá a má k dispozici kompletní a srozumitelnou dokumentaci. Jádro knihovny je sice napsané v C++, jsou ale k dispozici verze pro několik dalších programovacích jazyků: C#, Java, VisualBasic, Delphi apod. Irrlicht podporuje 6 různých renderovacích API:

- Direct3D 8.1
- Direct3D 9.0
- OpenGL 1.2–3.x
- Irrlicht software renderer
- Burningsvideo software renderer

- Null device

Programátor při používání Irrlichtu nemusí znát API, přes které engine grafiku vykresluje. Při inicializaci se určuje, které API bude mít při vykreslování přednost. Existují 3 důvody, proč se engine nezaměřuje jen na jedno vykreslovací API:

- Výkon: některé grafické adaptéry jsou optimalizovány pro Direct3D, jiné běží rychleji pod OpenGL. Je proto výhodnější nechat programátora zvolit, které API bude pro jeho použití lepší.
- Multiplatformost: Direct3D není z licenčních důvodů dostupný pro uživatele systémů na bázi Linuxu nebo Mac. Je tedy možné zvolit OpenGL nebo jeden ze software rendererů.
- Ovladače: dalším problémem, hlavně třeba u notebooků, mohou být ovladače. S použitím zastaralých ovladačů se mohou aplikace chovat nepředvídatelně nebo nemusí fungovat vůbec. Proto má programátor Irrlichtu na výběr. [1]

1.2 Možnosti a efekty engineu Irrlicht

Irrlicht standardně nabízí velké množství speciálních efektů. Kromě několika výjimek jsou všechny k dispozici při užití libovolného hardwarově akcelerovaného API, některé navíc i při softwarovém vykreslování. S novými verzemi se jejich množina pravidelně rozšiřuje:

- Animované vodní plochy, pixel a vertex shadery 1.1 až 3.0.
- Dynamická světla, dynamické stíny pomocí stencil bufferu, volumetrická světla.
- Mip-mapping terénu.
- Bump mapping, parallax mapping, normal mapping.
- Odlesky, transparency objektů, odrazy prostředí pomocí textury.
- Light mapy.
- Nastavitelné částicové efekty pro sníh, kouř, oheň, mlhu a další.
- Animované textury, billboardy.
- Antialiasing aplikující se na jednotlivé meshe.
- Filtrování textur (bilineární, trilineární, anizotropní).

1.3 Podporované formáty Irrlichtu

Irrlicht umí v základu číst z velkého množství běžně používaných formátů, některé dokáže i ukládat. Data se tedy nemusí zbytečně konvertovat, což ušetří spoustu času při vývoji. Vnitřní manažer zdrojů Irrlichtu se stará o všechna již načtená data, pokud tedy načtete stejný soubor vícekrát, manažer jej při dalším načtení přečte z vlastní cache namísto z disku.

V současnosti podporované formáty textur:

- JPEG File Interchange Format (Jpg) – čtení i zápis.
- Portable Network Graphics (Png) – čtení i zápis.
- TrueVision Targa (Tga) – čtení i zápis.
- Windows bitmapa (Bmp) – čtení i zápis.
- ZSoft Paintbrush (PCX) – čtení i zápis.
- Portable Pixmaps (Ppm) – čtení i zápis.
- Adobe Photoshop (PSD) – pouze čtení.
- Quake 2 textury (Wal) – pouze čtení.
- SGI truecolor textury (Rgb) – pouze čtení.

Podporované formáty animovaných meshů:

- B3D soubory (B3d) – pouze čtení.
- Microsoft DirectX (X) – pouze čtení.
- Milkshape (Ms3d) – pouze čtení.
- Quake 3 modely (MD3) – pouze čtení.
- Quake 2 modely (MD2) – pouze čtení.

Podporované formáty statických meshů:

- Irrlicht scény (IRR) – čtení i zápis.
- Irrlicht statické meshe (Irrmesh) – čtení i zápis.
- 3D Studio meshe (3ds) – pouze čtení.

- Alias Wavefront Maya (Obj) – čtení i zápis.
- Lightwave objekty (LWO) – pouze čtení.
- COLLADA 1.4 (XML, Dae) – čtení i zápis.
- Ogre meshe (Mesh) – pouze čtení.
- My3DTools 3 (My3D) – pouze čtení.
- Pulsar LMtools (Lmts) – pouze čtení.
- Quake 3 mapy (BSP) – pouze čtení.
- DeleD (DMF) – pouze čtení.
- FSRad oct (Oct) – pouze čtení.
- Cartography shop 4 (CSM) – pouze čtení.
- STL 3D soubory (STL) – čtení i zápis.
- PLY 3D soubory (Ply) – čtení i zápis. [1]

1.4 Jmenné prostory Irrlichtu (namespaces)

Irrlicht ve svém jádře využívá jmenné prostory. Slouží k logickému oddělení tříd podle jejich funkce. Níže si uvedeme jejich seznam:

- *irr* – zde se nachází celý engine včetně ostatních jmenných prostorů, definice číselných, znakových a výčtových typů engine.
- *irr::core* – seznam základních tříd pro používání engine (vektory, pole, seznamy, matice atd.) a jejich metody.
- *irr::gui* – třídy pro používání grafického prostředí aplikace (okna, tlačítka, edit boxy, dialogy apod.) a metody pro práci s nimi.
- *irr::io* – třídy pro vstupně/výstupní rozhraní, čtení a zápis do souborů, přístupy ke komprimovaným archivům.
- *irr::scene* – jmenný prostor pro management scény. Obsahuje načítání a práci s modely, vertexy, triangly a podobně.

- *irr::video* – obsahuje třídy pro přímý přístup ke grafickému ovladači přes zvolené vykreslovací API. Veškerý 3D i 2D rendering se provádí pomocí metod v tomto jmenném prostoru. [1]

1.5 Datové typy Irrlichtu

Irrlicht kvůli přenositelnosti na jiné platformy nepoužívá standardní datové typy. Ze stejného důvodu, ale třeba i kvůli vyšší rychlosti a efektivnosti Irrlicht nevyužívá ani STL knihovny. Definuje si vlastní výkonné a optimalizované kontejnery, jejichž používání je velice snadné. V dalších kapitolách si uvedeme příklady těch nejdůležitějších.

1.5.1 Číselné a znakové datové typy:

- *u32* – 32bitové celé číslo bez znaménka (podobně *u8* a *u16*).
- *s32* – 32bitové celé číslo se znaménkem (podobně *s8* a *s16*).
- *f32* – 32bitové číslo s plovoucí řádovou čárkou (podobně *f64*).
- *c8* – 8bitový znak bez znaménka.
- *s8* – 8bitový znak se znaménkem.

1.5.2 Vektory a matice

- *vector2d* – 2rozměrný vektor.
- *vector3d* – 3rozměrný vektor.
- *dimension2d* – určuje libovolný 2rozměrný prostor (čtverec, obdélník).
- *dimension3d* – určuje libovolný 3rozměrný prostor (krychle, kvádr).
- *CMatrix4* – 4x4 matice, používána většinou jako transformační matice objektů.
- *rect* – obdélník, používá se pro ohraničení objektů GUI rozhraní.
- *quaternion* – 4rozměrný vektor pro reprezentaci rotací.
- *triangle3d* – charakterizace čtyřstěnu.

1.5.3 Řetězce

- *stringc* – řetězec znaků (pole znaků *c8*).
- *stringw* – řetězec znaků rozšířené znakové sady (pole znaků *wchar_t*).

1.5.4 Kontejnery

- *array* – dynamicky alokované pole, podobné STL vektoru. Umožňuje některé další funkce jako lineární či binární vyhledávání, třídění apod.
- *list* – obousměrný seznam.
- *map* – stromová struktura. [1]

1.6 Nejdůležitější části engine Irrlicht

V následujících kapitolách si popíšeme nejdůležitější části engineu Irrlicht a práci s nimi.

1.6.1 IrrlichtDevice

IrrlichtDevice je kořenovým objektem ke všemu, co patří k engineu. Stará se o celé okno aplikace a jeho vlastnosti (rozlišení, barevná hloubka apod.), odchytávání kurzoru a všech vnějších událostí, časovačů atd. *IrrlichtDevice* se vytváří funkcí *createDevice()*, jejíž návratová hodnota je ukazatel na tento objekt, má 7 parametrů jdoucích vzestupně:

- *deviceType*: typ rendereru. Zde se pomocí jedné z hodnot výčtového typu zvolí, který renderer se použije.
- *windowSize*: rozlišení okna aplikace, zadává se pomocí typu *dimension2d*.
- *bits*: barevná hloubka na jeden pixel, pokud je aplikace spuštěna v okenním režimu, je tento parametr ignorován, zadává se pomocí libovolného celočíselného typu.
- *stencilbuffer*: povolení/zakázání stencil bufferu pro vykreslování stínů.
- *vsync*: povolení/zakázání vertikální synchronizace, i tato volba je ignorována při okenním režimu.
- *eventReceiver*: objekt pro odchytávání vnějších událostí (ovládání aplikace)

Na konci aplikace se musí zavolat metoda *drop()* objektu *IrrlichtDevice*. Při používání Irrlichtu je nutno při ukončení práce mazat všechny objekty, při jejichž vytváření byla použita funkce nebo metoda začínající na slovo „create“. Irrlicht má v sobě implementován jednoduchý čítač referencí, aby nedošlo k volání této metody pro objekty, které jsou ještě potřeba pro správný běh aplikace. Každá část engine, která zmíněný objekt ještě potřebuje, si totiž zavolá jeho metodu *drag()*, což zvýší jeho čítač referencí o jedničku. Objekt tedy může být smazán až tehdy, je-li jeho čítač referencí roven nule. [1]

1.6.2 Manažer scény Irrlichtu – *ISceneManager*

Manažer scény se v Irrlichtu stará o všechny uzly scény. Scéna je v engine Irrlicht znázorněna grafem, její obsah reprezentují uzly tohoto grafu. Uzel ve scéně může být například mesh, kamera, světlo nebo billboard. Každý uzel scény může mít svoje potomky, kteří se pohybují v relativních souřadnicích ke svému rodiči. Pokud například nastavím neviditelnost na jeden uzel, všichni jeho potomci budou mít tento parametr nastaven stejně. Vykreslení všech uzlů manažera scény se provádí voláním metody *drawAll()* na objektu *ISceneManager*.

Ukazatel na manažera scény získáme voláním metody *getSceneManager()* na *IrrlichtDevice*. [1]

1.6.3 Uživatelské prostředí v Irrlichtu – *IGUIEnvironment*

Grafické prostředí pro ovládání funguje v Irrlichtu na podobném principu, jako manažer scény. Grafické rozhraní je reprezentováno stromovou strukturou, všechny jeho prvky jsou uzly v této struktuře. Každý prvek grafického rozhraní může mít svoje potomky: posuvná lišta je potomek okna, obrázky jsou potomky tlačítek a podobně. Opět stejně jako u scénového manažera je pozice každého potomka relativní ke svému rodiči, pozice nejvyššího potomka ve stromu je relativní k základnímu oknu aplikace, o které se stará *IrrlichtDevice*. Vykreslení všech prvků uživatelského rozhraní se provádí voláním metody *drawAll()* na objektu *IGUIEnvironment*.

Ukazatel na ovládací prvek grafického rozhraní získáme voláním metody *getGUIEnvironment()* na *IrrlichtDevice*. [1]

1.6.4 Vykreslovací rozhraní Irrlichtu – IVideoDriver

Vykreslovací rozhraní Irrlichtu slouží k veškerému 3D i 2D renderingu a manipulaci s texturami. Je dokonce možné používat engine pouze za využití tohoto rozhraní, tedy bez použití scénového manažera. Ten je ale k práci značně přívětivější a obsahuje spoustu užitečných metod. Veškeré vykreslování grafiky v aplikaci musí být mezi voláními metod vykreslovacího rozhraní *beginScene()* a *endScene()*.

Ukazatel na vykreslovací rozhraní Irrlichtu se získá voláním metody *getVideoDriver()* na *IrrlichtDevice*. [1]

1.6.5 Odchytávání vnějších událostí – IEventReceiver

IEventReceiver je velmi důležitou částí Irrlicht engine, umožňuje programátorovi přístup k událostem, které vyvolávají vnější zdroje, jako jsou klávesnice, myš, nebo různé časovače.

Irrlicht vnější události obsluhuje s pomocí jednoduchého vnitřního event handleru. Jeho interface (virtuální třída) má název *IEventReceiver* a sama od sebe zpracovává pouze základní formy vstupů. Je tedy na samotném programátorovi, aby tuto třídu pomocí dědičnosti rozšířil o vlastní zpracování.

Obsluhovačů událostí je možné mít samozřejmě i více, stačí pro ně pouze vytvořit jednotlivé odvozené třídy a napsat jejich metody. Jejich připojení k Irrlichtu se provádí metodou *setEventReceiver()*, která se volá na objekt *IrrlichtDevice*. [1]

1.7 Vývoj Irrlichtu a jeho autoři

Irrlicht engine začal vyvíjet v roce 2002 rakouský programátor Nikolas Gebhardt. Sám navrhoval i vyvíjel prvotní verze knihovny. Postupně mu při vývoji pomáhalo stále více lidí a dnes na Irrlichtu pracuje 12 stálých členů s tím, že spousta práce je do projektu ještě dodávána externě. Níže si uvedeme několik prvních verzí engine a hlavní funkce, které v nich byly postupně přidávány. Změny v dalších verzích již nebyly tak markantní, nebudeme si je tedy zmiňovat. [1], [10]

- Irrlicht 0.1 (březen 2003)
 - První alfa verze engine zatím pouze pro Windows platformu.

- Renderování scény pomocí Direct3D 8.1. Vykreslovací API OpenGL ani software renderer ještě nebyly zcela implementovány a zatím podporovaly jen základní funkce.
- GUI prostředí a jeho systém událostí, dynamické světlo, průsvitnost objektů i management scény již byly hotovy.
- Načítání souborů nezávisle na platformě, čtení ZIP archivů.
- Irrlicht 0.1.5 (duben 2003)
 - Irrlicht je nově pod licencí Zlib
 - Vykreslování pomocí OpenGL plně implementováno.
 - Nově možnost vytvářet komplexní terény generátorem přímo v aplikaci.
 - Přidána FPS kamera s nastaveným systémem událostí a ovládáním podobným jako u akčních her z pohledu první osoby.
- Irrlicht 0.2 (květen 2003)
 - Implementována podpora skyboxů.
 - Přidány stíny pro animované meshe.
 - Nová třída pro časovač.
- Irrlicht 0.3 (červenec 2003)
 - Irrlicht engine nyní běží i na operačních systémech založených na Linuxu.
 - Engine je nově dobře použitelný i pro vykreslování 2D grafiky.
- Irrlicht 0.4 (září 2003)
 - Implementována základní detekce kolizí.
 - Částicový emitor je plně funkční.
 - Přidána možnost vykreslovat realistickou vodu animováním statického meshe.
- Irrlicht 0.5 (únor 2004)
 - Přidána plná podpora Direct3D 9.

- Nové GUI elementy jako edit boxy a tabulátory.
 - Přidána podpora mlhy.
- Irrlicht 0.6 (březen 2004)
 - Nově integrovaný vlastní XML parser irrXML.
- Irrlicht 0.7 (září 2004)
 - Implementována podpora Pixel a Vertex shaderů.
- Irrlicht 0.10 (květen 2005)
 - Přidána podpora parallax mappingu do enginu.
 - Nově také podpora renderingu do textury (pro použití například u simulace zrcadel a podobně).
 - Implementována podpora lesklých povrchů.
- Irrlicht 0.11 (červenec 2005)
 - Přidána podpora antialiasingu pro renderery Direct3D 8.1 a Direct3D 9.
- Irrlicht 0.12 (srpen 2005)
 - Podpora pixel a vertex shaderů 3.0.
 - Přidána možnost vykreslování více scén najednou (použitelné například u splitscreenu v závodních hrách).
- Irrlicht 1.1 (srpen 2006)
 - Implementována možnost ukládat screenshot přímo v rámci enginu, tato funkce funguje pod všemi renderery.
- Irrlicht 1.4 (říjen 2007)
 - Irrlicht má nově vlastní formát pro statické meshe (irmesh), přidána možnost přímo z aplikace do tohoto formátu ukládat.
- Irrlicht 1.5 (prosinec 2008)
 - Přidána základní podpora pro gamepady a joysticky.
 - Nově verze pro Windows Mobile 6.0.

- Irrlicht 1.6 (září 2009)
 - Začíná se pracovat na verzi Irrlicht engine pro XBOX.
- Irrlicht 1.7 (únor 2010)
 - Programátor může provádět úpravy mipmappingu a měnit nastavení kvality jednotlivých úrovní textur.

Vývoj Irrlichtu byl ve svých začátcích hodně překotný, prvních několik verzí bylo pouze alfa verzemi a jejich stabilita nebyla valná. V dnešní době se vývoj znatelně zpomaluje, nové funkce již téměř nepřibývají a spíše se pracuje na opravách chyb.

Aktuální stabilní verze Irrlichtu je verze 1.7.1, která byla vydána v únoru 2010. Nová verze 1.7.2 se v době psaní tohoto textu (duben 2011) už blíží, přinese ale pouze opravy několika menších chyb. [1]

1.8 Irrlicht tool set

Pro snadnější vývoj aplikací s použitím engine Irrlicht jeho tvůrci dále vyvíjejí bezplatné nástroje. Tyto nástroje a knihovny jsou na Irrlichtu zcela nezávislé, i když vycházejí z jeho logické struktury a jsou navrženy tak, aby byly s engine plně kompatibilní. Zde je stručný výčet a popis těchto nástrojů:

- Zvukový engine IrrKlang – Irrlicht jakožto grafický engine (ne herní) nemá v sobě přímo zakomponovány prostředky pro práci se zvukem, k dispozici je proto na tyto účely tento externí nástroj.
- IrrEdit – IrrEdit je 3D grafický editor scén, který ukládá scény do irr souborů, které jsou nativním formátem engine Irrlicht.
- IrrXML – IrrXML je open source XML parser napsaný pro jazyk C++. Analyzuje data v XML souborech a předává je aplikaci, je velmi jednoduchý k použití a velice rychlý. [1]

2 3D EDITOR IRREDIT

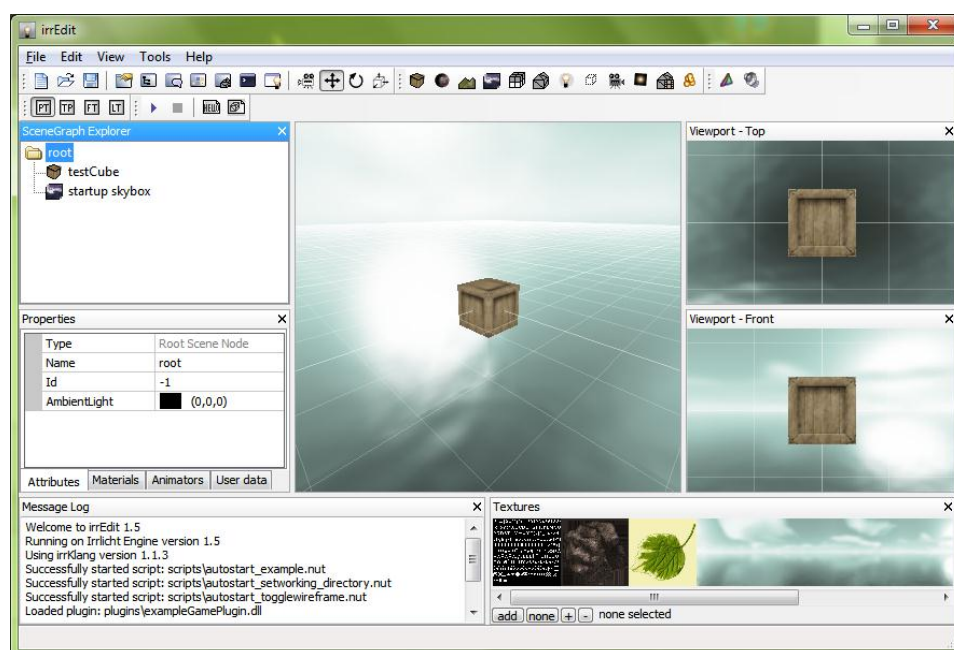
2.1 O 3D editoru IrrEdit

IrrEdit je editor scén pro grafický engine Irrlicht. Je k dispozici zcela zdarma i pro komerční účely, ale jeho zdrojové kódy otevřeny nejsou. IrrEdit pracuje s irr soubory a je možné jej používat jako 3D editor, návrhář částicových systémů, meshviewer a další. Dokáže importovat meshe ze všech formátů, které podporuje samotný engine, obsahuje jednoduchý, ale výkonný systém materiálů a umožňuje do scén přímo přidávat spoustu speciálních efektů. Vzhled uživatelského prostředí editoru je uveden na přiloženém obrázku (Obr. 3). [4]



Obrázek 2 – logo editoru IrrEdit

Scéna v IrrEditu funguje stejně jako v Irrlichtu, každý prvek je tedy uzal ve scénovém grafu, může mít své potomky a platí i pravidlo o relativním pozicování potomků: pokud se například v animaci pohybuje rodič, budou se pohybovat i všichni jeho potomci a podobně. [4]



Obrázek 3 – základní prostředí 3D grafického editoru IrrEdit

2.1.1 Možnosti IrrEditu

- Kvalitní radiosity light map generátor.
- Přidávání světel, skyboxů, billboardů a podporovaných meshů do scény.
- Editor scén, terénu a částicového systému.
- Jednoduchý animátor objektů.
- Skriptovací systém.
- Možnost napsání vlastního pluginu pro rozšíření editoru. [4]

2.1.2 Light map generátor

IrrEdit má rychlý vestavěný light map generátor. Tento generátor je užitečný pro počítačové hry nebo aplikace se statickou geometrií, které potřebují ostré stíny a statické osvětlení. Informace o světle a stínu jsou jednoduše uloženy přímo do textury.

Výpočet stínů pro scénu běžné velikosti je pro generátor otázkou několika sekund. Umožňuje také další funkce, jako je například průhlednost světla, podvzorkování okolního světla, nebo rozprostření barev. Lze takto vytvořit iluzi, že je scéna osvětlena pomocí globálního osvětlení. [4]

2.2 Vývoj IrrEditu

První verze IrrEditu vyšla 17. června 2006. V roce 2009 byl vývoj editoru ukončen a jeho tvůrci od té doby vyvíjejí velmi podobný, ovšem placený Irrlicht editor CopperCube. CopperCube má daleko více možností, ale je značně svázaný licencí: k dispozici je 14denní trial verze s tím, že po této době se některé funkce z programu odeberou. Naštěstí ale tyto funkce nejsou potřeba pro export irr souborů, a tak lze i po vypršení trial lhůty editor pro potřeby práce s Irrlicht scénami používat.

- IrrEdit 0.1 (rok 2006)
 - První vydání editoru.
- IrrEdit 0.2 – 0.6
 - Přidána možnost skriptování.

- Přidána podpora formátu x.
 - Soubory lze ukládat i s relativními cestami.
 - Light mapper je plně dokončen.
 - Přidána možnost podvzorkování pro light mapper.
 - Nové skriptovací funkce.
 - Přidáno globální osvětlení na vykreslení light mapperu pomocí radiozity.
- IrrEdit 0.7 – 0.7.1 (rok 2007)
 - IrrEdit nově podporuje pluginy.
 - Přidána 3D mřížka, možnost změnit barvu pozadí.
- IrrEdit 1.4 – 1.4.2 (rok 2008)
 - Přidán nový formát irrmesh, což je defaultní formát IrrEditu pro statické meshe.
 - IrrKlang zvukový engine je nově používán přímo v editoru.
- IrrEdit 1.5 (rok 2009)
 - Poslední verze IrrEditu. [4]

3 ZVUKOVÝ ENGINE IRRKLANG

3.1 O zvukovém enginu IrrKlang

IrrKlang je velice výkonné API pro přehrávání zvuků ve 2D i 3D aplikacích jako jsou hry, vizualizace, nebo multimediální aplikace. Původně byl psán v jazyce C++, dnes již jsou k dispozici verze pro jazyky rodiny .NET (C#, VisualBasic.NET, Delphi.NET atd.). K nekomerčnímu použití je IrrKlang k dispozici zcela zdarma, pro využití tohoto API pro komerční aplikace je zde možnost zakoupit verzi IrrKlang pro. Autorem IrrKlangu je stejný tým, který se podílel na tvorbě enginu Irrlicht, vedoucí týmu je opět rakouský programátor Nikolaus Gebhardt. [3]



Obrázek 4 – logo enginu IrrKlang

Zvukový engine IrrKlang je multiplatformní, podobně jako Irrlicht umožňuje programátorovi psát stejný kód pro různé platformy. Seznam platforem, na kterých je IrrKlang schopen pracovat včetně zvukových API, které ke svému výstupu může používat:

- Windows 98, ME, NT 4, 2000, XP, Vista, Windows 7
 - DirectSound 3.
 - DirectSound 8.
 - WinMM.
- Systémy na bázi Linuxu nebo všeobecně Unixu
 - ALSA.
- Mac OS X (architektura x86 i PowerPC)
 - CoreAudio. [3]

3.2 Podporované formáty souborů

V nejnovější verzi IrrKlang podporuje tyto formáty souborů:

- RIFF WAVE (wav).
- Ogg Vorbis (ogg).
- MPEG-1 Audio Layer 3 (mp3).
- Free Lossless Audio Codec (flac).
- Amiga Modules (mod).
- Impulse Tracker (it).
- Screem Tracker 3 (s3d).
- Fast Tracker 2 (xm).

Pokud by však tento výčet formátů nedostačoval, je velice jednoduché jejich výčet rozšířit pomocí pluginů. Podpora pro přehrávání formátu mp3 je již přímo v enginu z důvodu licenčních omezení tohoto formátu implementována pouze pomocí pluginu, který je však ve stažitelné verzi již nainstalován. [3]

3.3 Efekty a možnosti enginu IrrKlang

3.3.1 Podpora 3D zvuku

IrrKlang má vestavěnou podporu pro 3D zvuk pro všechny platformy a zvukové ovladače. Původně byla navržena pro použití ve 3D hrách, a tak je velmi efektivní a spotřebovává málo procesorového času. [3]

3.3.2 Zvukové efekty

Všechny zvukové efekty jsou k dispozici pouze při použití DirectSound8 zvukového výstupu. Fungují pro přehrávání v režimu 2D i 3D. Jejich výčet je následující:

- Zkreslení zvuku.
- Ozvěna zvuku.
- Zasekávání zvuku.

- Parametrický ekvalizér.
- Chorálový zvuk. [3]

3.4 Použití engineu IrrKlang

Zvukový engine IrrKlang je velice jednoduchý k použití. Filosofie jeho návrhu je navíc velice podobná Irrlichtu. API opět podporuje jmenné prostory podobně jako Irrlicht, celý engine je zabalený do jmenného prostoru *iirklang*. [3]

3.4.1 IrrKlangDevice

IrrKlangDevice je podobně jako *IrrlichtDevice* kořenovým prvkem všeho, co API obsahuje. Je to interface pro veškerou práci s engineem, pomocí kterého se přehrávají všechny zvuky.

IrrKlangDevice se nejčastěji vytváří voláním funkce *createIrrKlangDevice()*. Parametry této funkce se nastavuje použitý zvukový ovladač, ID zařízení a různé další volby. Pokud se však programátor nechce těmito nastaveními zabírat, engine za něj automaticky vybere nejlepší možné řešení v závislosti na platformě, kterou používá. ID zařízení je mimo jiné důležité tehdy, používá-li autor více IrrKlang zařízení současně. Lze takto velmi jednoduše separovat například zvukové efekty a hudbu v aplikaci. [3]

3.4.2 Přehrávání 2D zvuku

K přehrávání zvuku se používá metoda *play2D()*, která se volá na objekt *IrrKlangDevice*. Tato metoda má parametry, kterými programátor může nastavit opakované přehrávání nebo opožděný start přehrávání. Dalším parametrem lze nastavit, zda tato metoda bude vracet prázdný ukazatel, nebo zda vrátí ukazatel na přehrávaný zvuk.

Obecně platí, že chceme-li s přehrávaným zvukem dále pracovat, musíme si od této metody nechat vrátit jeho ukazatel, v opačném případě totiž není způsob, jak k tomuto objektu přistoupit. Posledním parametrem této metody autor určí, zda se na tento zvuk budou aplikovat speciální efekty, nebo ne. [3]

3.4.3 Přehrávání 3D zvuku

Pro přehrávání prostorového zvuku se v IrrKlang API používá metoda *play3D()*. Tato metoda funguje obdobně, jako metoda *play2D()*, jediným rozdílem je parametr *position*, kterým se zadává souřadnice zdroje zvuku v 3D prostoru.

Aby však mělo prostorové přehrávání vůbec smysl, potřebuje engine ještě informaci o pozici posluchače. Ta se zadává pomocí metody *setListenerPosition()*. Tato metoda se opět volá na objektu *IrrKlangDevice*, jejími parametry jsou pozice posluchače, vektor otočení posluchače a vektor rychlosti pohybu posluchače, který se používá pro počítání Dopplerova jevu. [3]

3.4.4 Zdroje zvuku

IrrKlang engine může při načítání souborů používat tzv. zdroje zvuku. Je to v podstatě ukazatel na zvukovou stopu (například zvukový soubor nebo třeba stream). Zdroj zvuku je v API charakterizován třídou *ISoundSource*.

Každému zdroji lze pak v aplikaci nastavit spoustu defaultních parametrů, které se použijí při každém přehrávání tohoto zdroje. Programátor si tak může ušetřit spoustu práce a informace o jednotlivých zvucích aplikace mohou tak být uloženy přímo ve zvukových zdrojích. [3]

3.4.5 Další užitečné funkce enginu

Pro zjednodušení práce s API nabízí IrrKlang ještě spoustu užitečných metod. Všechny se podobně jako *play2D()* a *play3D()* volají na objekt zařízení enginu, tedy *IrrKlangDevice*.

Metoda *setSoundVolume()* nastavuje hlasitost všech přehrávaných zvuků. API sice umožňuje nastavit hlasitost každého zvuku zvlášť, tato metoda však umožňuje nastavit velice rychle všechny naráz. Další užitečnou metodou je *stopAllSounds()*, která jak již název napovídá, zastaví všechny právě přehrávané zvuky. [3]

4 FYZIKÁLNÍ ENGINE NEWTON GAME DYNAMICS

4.1 O fyzikálním enginu Newton

Newton Game Dynamics (dále jen Newton) je snadno integrovatelný fyzikální engine pro simulaci fyziky prostředí v reálném čase. Je multiplatformní, existují jeho verze pro Windows, Linux i Mac, nově jsou nabízeny i verze pro iPhone a iPod Touch. API je šířeno pod licencí Zlib a od února roku 2011 je nově i open source.



Obrázek 5 – logo enginu Newton

Dokumentace k enginu je k dispozici ve formě wiki modulu, který je součástí oficiálních stránek projektu. Knihovna je velice jednoduchá a obsahuje jen málo funkcí, díky tomu je však snad k použití, rychlá a velice stabilní.

Autorem enginu je dvojice programátorů původem ze spojených států: Julio Jerez a Alain Suero. Aktuální verze enginu (duben 2011) má označení 2.33, je optimalizována pro více jádrové procesory, pracuje se také na využívání výpočtů pomocí grafických procesorů. [11]

4.2 Základy enginu Newton

4.2.1 NewtonWorld

NewtonWorld je základní rozhraní knihovny Newton, slouží k identifikaci simulovaného fyzikálního světa. Ukazatel na *NewtonWorld* vrací funkce *NewtonCreate()*, která je v aktuální verzi bezparametrická. Tento ukazatel je potřeba uložit, protože při jakékoliv další práci s knihovnou je potřebný. Je dokonce možné, aby aplikace používala více těchto „světů“, v tom případě stačí jednoduše tuto funkci zavolat vícekrát, ve většině situací bude ale programátorovi stačit jeden. Na konci aplikace je nutno zavolat funkci *NewtonDestroy()*, která jako jediný parametr bere ukazatel na *NewtonWorld*, tato funkce ruší veškeré působení enginu a uvolňuje paměť. [11]

4.2.2 Charakterizace objektu v enginu Newton

Objekt, který má být součástí fyzikální simulace v enginu v něm také musí být charakterizován. To se dělá pomocí dvou datových struktur: *NewtonCollision* a *NewtonBody*. [11]

4.2.2.1 *NewtonCollision*

NewtonCollision reprezentuje tvar objektu. Nepočítá tedy s jeho hmotností nebo jejím rozdělení, to je záležitostí objektu *NewtonBody*. S tímto rozdělení je tedy možné vytvořit objekt, který bude mít tvar například koule, ale rozdělení hmotnosti a točivé momenty bude mít jako krychle.

Newton podporuje širokou škálu kolizních primitiv, dokonce umožňuje programátorovi definovat vlastní kolizní primitiva.

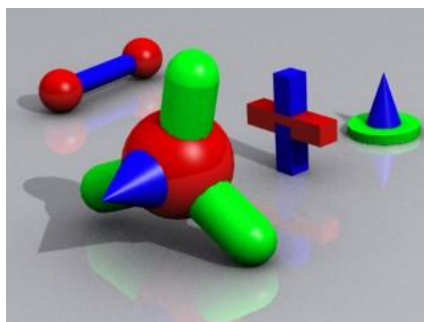
- *Box* – standartní kvádrová primitiva, vytváří se funkcí *NewtonCreateBox()*.
- *Sphere* – kolizní primitiva pro tvary koule nebo obecně elipsoidy, k vytvoření se volá funkce *NewtonCreateSphere()*.
- *Capsule* – tvar kapsle (Obr. 6), jejíž délka musí být větší než její průměr, v opačném případě se tyto velikosti jednoduše otočí. Tvoří se voláním funkce *NewtonCreateCapsule()*.



Obrázek 6 – primitiva tvaru kapsle

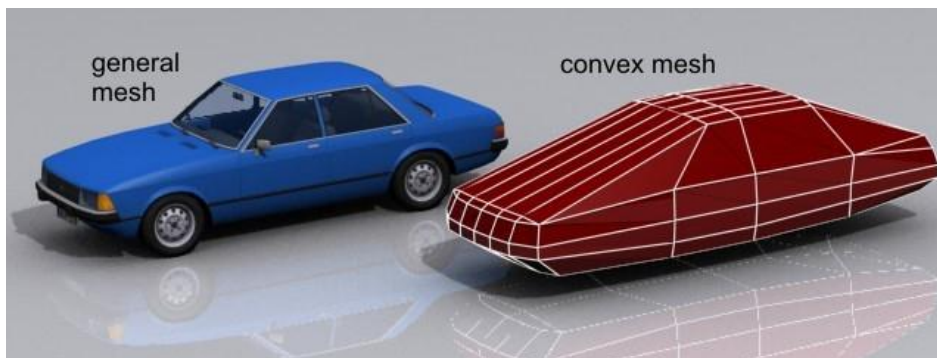
- *Cylinder* – primitiva pro charakterizaci válcových objektů, vytváří se pomocí funkce *NewtonCreateCylinder()*.

- *Cone* – primitiva použitelná pro kuželovité objekty, výška kužele musí být větší nebo stejně velká jako průměr jeho podstavy, v opačném případě se hodnoty otáčí. Tato primitiva se vytváří pomocí funkce *NewtonCreateCone()*.
- *Null primitive* – používá se pro charakterizaci objektů, které nekolidují s ostatními. K vytvoření se volá funkce *NewtonCreateNull()*.
- *Compound* – kontejner, který drží pole primitiv. Ta jsou vůči sobě vždy určitým způsobem otočena nebo posunuta, ilustrace je uvedena na obrázku (Obr. 7). Toto složené primitivum získáme voláním funkce *NewtonCreateCompoundCollision()*.



Obrázek 7 – složená primitiva

- *ConvexHull* – *ConvexHull* jsou primitiva pro charakterizaci skupiny vertexů, můžou být použity pro obecné znázornění libovolného meshe (Obr. 8). Maximální počet vertexů k vytvoření tohoto primitiva není limitován, z důvodů zachování vysokého výkonu je však nutné jejich počet držet na minimu. Minimální počet vertexů pro výpočet *ConvexHull* je 4. K vytvoření se volá funkce *NewtonCreateConvexHull()*. [11]



Obrázek 8 – charakterizace meshe pomocí primitiva *ConvexHull*

4.2.2.2 *NewtonBody*

Jak už bylo výše naznačeno, *NewtonBody* je datová struktura charakterizující hmotnost, rozložení hmotnosti, těžiště nebo také třeba rotaci.

K vytvoření *NewtonBody* se volá funkce *NewtonCreateBody()*, která jako parametr přebírá ukazatel na *NewtonWorld*, ukazatel na kolizní charakter objektu matici (4x4 prvky) reprezentující jeho rotaci a pozici.

K další práci s těmito objekty knihovna obsahuje množství dalších funkcí:

- *NewtonBodySetCentreOfMass()* – nastavení těžiště objektu.
- *NewtonBodySetMassMatrix()* – nastavení hmotnostní matice, ta charakterizuje hmotnost a její rozložení v objektu.
- *NewtonBodySetOmega()* – nastavení úhlové rychlosti objektu.
- *NewtonBodySetTransformCallback()* – parametrem je ukazatel na funkci, která odchytává transformaci objektu (callback funkce), více o těchto funkcích později.
- *NewtonBodySetForceAndTorqueCallback()* – další callback funkce, tato objektu nastavuje síly, které na něj působí (například gravitace).
- *NewtonBodySetUserData()* – touto funkcí newton propojuje vykreslovací engine se svou simulační částí, parametrem této funkce může být například model, jehož chování simulujeme. [11]

4.2.3 Callback funkce

Volání callback funkcí je způsob, kterým Newton dává objektům vědět, jakým způsobem a kdy se mají transformovat (rotovat, posunovat a podobně). Zjištění, kterou callback funkci je potřeba volat, se provádí volání funkce *NewtonUpdate()*, tuto funkci je v aplikaci potřeba volat alespoň 100x za sekundu, velmi důležitá je taky pravidelnost volání, která je pro přesnost simulace nezbytná. [11]

5 XML PARSER TINYXML

5.1 Stručně o XML

XML (eXtensible Markup Language) je obecný značkovací jazyk, který je vyvíjen a standardizován konsorciem W3C. Je odvozeninou staršího jazyka SGML. Význam jazyka je v uchovávání obsahu dat, ne v jeho formě. Výhoda XML spočívá v tom, že kromě samotného textu nese i informaci o jeho významu. Jazyk je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů, u kterých definuje strukturu a obsah jednotlivých částí. Vzhled dokumentu může být definována pomocí kaskádových stylů (CSS). [7], [8]

XML je strukturou velice blízké HTML, má však mnohem přísnější pravidla:

- celý dokument musí být uzavřen mezi nějaký počáteční a ukončovací tag.
- všechny tagy musí být párové, výjimku tvoří prázdné elementy, které nemají žádný obsah.
- jména všech elementů a atributů jsou citlivá na velikost písmen.
- standardní znaková sada je v XML dokumentech Unicode (ISO 10646).
- předpokládá se, že XML dokument bude uložen v kódování UTF-8, pokud v dokumentu použijeme jiné kódování, musíme to uvést v XML deklaraci. [7]

5.2 O XML analyzátoru TinyXML

TinyXML je jednoduchá, malá knihovna napsaná v jazyce C++, která slouží k analyzování XML jazyka. Dokáže XML soubory číst, tak i do nich jednoduše zapisovat, respektuje přitom zavedený Document Object Model (DOM). Knihovna je velice malá, obsahuje pouze 2 hlavičkové a 4 zdrojové soubory. S TinyXML má programátor na výběr, zda chce nebo nechce ve svém projektu využívat STL knihovny: zvolí si tak prostou definici před začátkem kompilování knihovny. [9]

TinyXML čte XML dokument logicky a strukturovaně. Každý prvek je tedy rodičem všech prvků, které jsou do něj vloženy, stejně tak jako všech svých atributů. Přístup k tagům i jejich atributům je tak velice snadný.

5.2.1 Načtení souboru

Pro načtení souboru, ze kterého chceme XML data číst, musíme nejprve vytvořit objekt třídy *TiXmlDocument*, ten ve svém konstruktoru vyžaduje cestu k tomuto souboru. Poté už stačí zavolat metodu tohoto objektu *LoadFile()*, která ve svém ukazateli může navíc přebírat typ kódování uložený pomocí výčtového typu, tento parametr má však nastavenou default hodnotu, je proto nepovinný. Metoda *LoadFile()* vrací hodnotu typu boolean, lze tak jednoduše testovat, bylo-li načtení souboru úspěšné. Po načtení souboru již můžeme používat všechny funkce knihovny: zapisování i čtení. [9]

5.2.2 Čtení hodnot

Vlastní čtení hodnot se provádí voláním metod na objekty jednotlivých elementů. TinyXML umí hodnotu přečíst jako STD *string* nebo celé číslo. Například volání metody *Value()* na objekt typu atribut vrátí text uložený v těle atributu (uzavření uvozovkami) ve formátu STD *string*. Metoda *IntValue()* volaná na stejný typ objektu vrátí stejný text, ovšem konvertovaný na *integer*. [9]

5.2.3 Zapisování hodnot

Zapisování hodnot probíhá velice podobně, jako jejich čtení. Metody fungují na stejném principu a mají i stejný název, jen začínají slůvkem „set“. [9]

Chceme-li kupříkladu uložit do atributu některého prvku celé číslo, zavoláme nejprve na objekt tohoto prvku metodu *FirstAttribute()*, která nám vrátí první atribut. Pokud chceme číst z některého z dalších atributů, pomocí metod *NextSiblingElement()* volaných na tyto atributy se v jejich rámci můžeme přesouvat dále. Tímto způsobem si tedy vybereme žádaný atribut našeho prvku. Nastavení jeho hodnoty provedeme metodou *SetIntValue()*, která jako parametr přebírá hodnotu celého čísla, které chceme do souboru uložit.

6 BLENDER

6.1 Stručně o Blenderu

Blender je software pro 3D modelování, tvorbu her a rendering. Je založen na grafické knihovně OpenGL, díky tomu je k dispozici pro velké množství platforem jako je Windows, Linux, Irix, Sun Solaris, FreeBSD nebo Mac OS. Blender je zcela zdarma pod licencí GNU a má otevřené zdrojové kódy, uživatel si tedy může program upravit podle libosti tak, aby vyhovoval právě jeho požadavkům. [6], [12]



Obrázek 9 – logo Blenderu

Největší výhodou Blenderu je vysoký poměr kvality a ceny. Licence konkurenčních komerčních programů totiž obvykle stojí desetitisíce až statisíce korun. Další výhodou je jeho malá velikost a hardwarová náročnost: 32bitová verze instalátoru aktuální verze má velikost přibližně 20MB. Jeho rychlost pochopitelně závisí na konfiguraci počítače, nejdůležitějšími komponentami pro výkon Blenderu je pak grafický adaptér a procesor. [6]

6.2 Vývoj Blenderu

Blender patří mezi nejmladší programy pro 3D grafiku, jeho vývoj započala holandská animační společnost NeoGeo v roce 1995. Po třech letech vývoje vznikla z původní společnosti nová – Not a Number (NaN), ta ale v roce 2001 zkrachovala poté, co se pokusila prosadit komerční verzi produktu Blender Publisher.

Od března roku 2002 Blender vyvíjí nezisková organizace Blender Foundation, která v polovině října téhož roku zakoupila práva k dřívějším verzím programu, a tak mohl být Blender uvolněn jako open source. Od verze 2.26 (únor roku 2003) se tak na vývoji Blenderu podílejí programátoři z celého světa. Aktuální stabilní verze Blenderu je 2.57a,

která nově přinesla kompletní přepracování a modernizaci uživatelského prostředí. [6], [12]

6.3 Možnosti Blenderu

Stručný popis několika nejdůležitějších funkcí Blenderu.

6.3.1 Uživatelské rozhraní

- Uživatelské rozhraní Blenderu je plně nastavitelné, lze vytvořit libovolný počet nepřekrývajících se oken.
- Zabudovaný textový editor, který může sloužit pouze ke psaní poznámek, nebo také k programování Python skriptů.

6.3.2 Modelování

- Práce s různými 3D i 2D objekty: mesh, nurbs plochy, Beziérový a B-spline křivky.
- Editační režim pro práci s mesh objekty na úrovni vertexů, ploch a hran.
- Široká škála editačních nástrojů jako např.: *extrude*, *bevel*, *cut*, *warp* nebo *spin*.

6.3.3 Animace

- Deformační kosti (skeletony) s podporou dopředné i inverzní kinematiky.
- Editor nelineárních animací, editor pro animaci postav.
- Podpora zvuku a nástrojů k synchronizaci obrazu se zvukem.
- Částicové efekty společně s možností použití různých deformátorů: vítr, gravitace, detekce kolizí a další.

6.3.4 Rendering

- Možnost instalace externích rendererů.
- Oversampling, motion blur, environment mapy, halo, radiosity, ambient occlusion.
- UV editor s několika různými metodami pro *unwrap* objektů.

6.3.5 Tvorba her

- Grafický editor pro naprogramování logiky aplikace.
- Detekce kolizí a simulace dynamiky.
- Podpora všech OpenGL světelných modelů včetně průhlednosti nebo animovatelných a reflexivních textur.

6.3.6 Soubory

- Vlastní formát blenderu, který všechna data ukládá do jediného souboru s příponou blend.
- Čte/zapisuje formáty tga, jpg, png, Iris, SGI Movie, iff, avi, tiff, psd, mov.
- Nativní podpora exportu a importu mnoha formátů, jejichž množina je dále rozšiřitelný pomocí Python skriptů. [6]

PRAKTICKÁ ČÁST

7 TVORBA POČÍTAČOVÉ HRY V ENGINEU IRRLICHT

Praktická část této práce se skládá z vytvoření jednoduché 3D počítačové hry (dále jen hry) v engineu Irrlicht s využitím poznatků získaných při tvorbě teoretické části. Se studentem, který řešil druhou část této praktické práce, jsme si rozdělili úkoly podle témat, které jsme vypracovávali. Moje úlohy tedy měly být tvorba uživatelského rozhraní hry, implementace fyzikálního engineu a tvorba ozvučení. Společně jsme ale prvně provedli návrh této hry a její základní pravidla.

7.1 Návrh počítačové hry a její pravidla

7.1.1 Volba žánru hry

První rozhodnutí, které bylo potřeba při návrhu hry udělat, byl výběr žánru hry. Naše původní vize byla taková, že hra nesmí být v žádném případě násilného charakteru. Dále jsme do hry chtěli nějakým způsobem zakomponovat logické hádanky a rozhovory. Tato omezení nám náš žánr v podstatě definovala: adventura.

Adventura je žánr počítačových her, kde hlavní hrdina (jeden nebo i více) řeší rozličné úkoly, což mu umožňuje postupovat k cíli. Hry mají obvykle velice propracovaný příběh, který bývá plný zvratů a překvapení. Mnoho adventur umožňuje hráči sbírat předměty a vzájemně je kombinovat k pozdějšímu využití ve specifických situacích hry. Tento typ her také často obsahují složité dialogy se spoustou postav. Na výsledku těchto rozhovorů může záviset další postup ve hře.

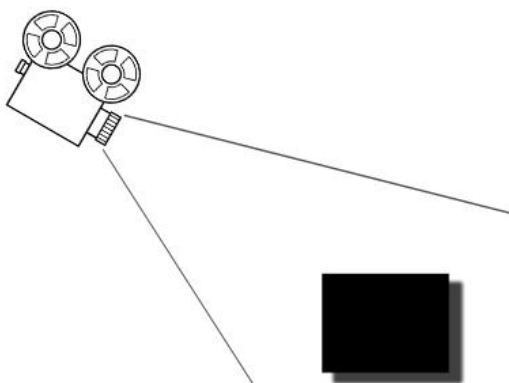
7.1.2 Grafický styl hry

Ačkoliv Irrlicht má implementované funkce pro práci s 2D (i když jen značně omezené), jeho plná síla se projeví až při vykreslování grafiky se třemi rozměry. Pro vývoj hry jsme se tedy rozhodli pro 3D grafiku.

S grafickou stylizací úzce souvisí i pohled kamery. Při použití 3D grafiky má programátor na výběr z nespočetně úhlů a pohledů. To, jakým způsobem herní kamera scénu snímá, je však pro hráče velice důležité. V dnešní době je v 3D hrách hráči často dána volba úhlu či přiblížení kamery. Přílišné omezení totiž může hráče často frustrovat, pokud často nevidí tu část scény, kterou by zrovna vidět chtěl. Na druhou stranu ale i přílišná volnost nastavení

kamery může být na škodu: hráč je pak zaměstnán rotací či přiblížením kamery a nemůže se zcela soustředit na hru.

Pro naše účely jsme tedy zvolili pevnou kameru zobrazující scénu, hráč má však možnost si zvolit několik dalších úhlů, pokud by mu ten základní nestačil. Další možnost je pak přiblížení pohledu kamery. Kamera tedy snímá hlavního hrdinu shora pod úhlem asi 30° , ilustrace je uvedena na obrázku (Obr. 10). To hráči umožňuje mít přehled o velké části scény, a zároveň je dobře viditelná hloubka 3D prostoru.



Obrázek 10 – ilustrace pohledu kamery

Uprostřed zobrazení herní kamery je vždy hlavní hrdina. Kamera tedy snímá jeho pohyb a hráč má vždy přehled, kde hlavní hrdina je. Tento pohled je vypůjčen z 2D izometrických her, dnes se však již velmi často využívá v 3D hrách u akčních her na hrdiny či strategiích.

7.1.3 Prostředí hry

Tvorba 3D i 2D grafiky je velice časově náročná, z úsporných důvodů mělo být prostředí hry značně omezené. Nevýhodou jsou malé možnosti pohybu pro hráče, na druhou stranu však tvorba grafiky zabere znatelně méně času.

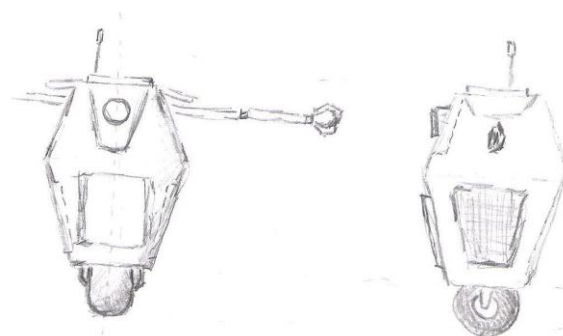
Hra se odehrává v ne příliš vzdálené budoucnosti. Toto časové období umožňuje ve hře výskyt moderních, či dokonce ještě nevyvinutých technologií. Neuvrhne však hráče do naprosto neznámého prostředí, kde by k pochopení funkčních pravidel prostředí musel vyvinout přílišné soustředění: hra má totiž především bavit.

Jako vhodné prostředí byla tedy zvolena vesmírná loď. Umožňuje totiž tvůrci omezit pohyb hráče, toto omezení je navíc i logické. Tato loď ovšem na rozdíl od vesmírných plavidel naší doby umožňuje delší pobyt ve vesmíru pro větší posádku. Je tedy vybavena

generátory gravitace (smyšlená technologie pro tvorbu zemské gravitace ve vzdáleném vesmíru).

7.1.4 Hlavní hrdina hry

Každá adventura má jednoho či více hlavních hrdinů, který doprovází hráče příběhem. Pro naši hru jsme zvolili hrdinu jednoho. Modelování a hlavně animace lidských postav je práce velice náročná, bez použití moderních technik jako je motion capturing (systém digitalizace pohybu na živých hercích) jsou výsledky i po hodinách práce stále nepřesvědčivé. Z toho důvodu jsme se rozhodli pro hrdinu mechanického, které nám animační práci nanejvýš zjednodušil: robota (Obr. 11).



Obrázek 11 – náčrtek hlavního hrdiny hry

Modelování i animace robota není nijak složitá. Po několika málo hodinách práce lze dosáhnout velmi pěkných výsledků. Další výhodou je pak to, že animací nemusí být tolik. Pohybuje-li se robot například na kolech, animace pro otáčení postavy lze pro zjednodušení zcela vyřadit.

7.1.5 Ovládání hry

Další velice důležitou částí návrhu hry je systém ovládání. Pro naše účely jsme zvolili tzv. point & click systém, tedy v překladu „ukaz a klikni“. V tomto systému hráč hru ovládá pomocí myši, kurzorem jí zadá pozici a kliknutím levého tlačítka dá hrdinovi příkaz, aby se na danou pozici přesunul. Tento systém je mezi adventurami velice často používán, dokonce se podle něj definuje vlastní dílčí žánr: point & click adventury.

Komunikace s prostředím se pak ve hře zadává s pomocí pravého tlačítka myši. Takto hráč může kupříkladu zavést rozhovor s jiným robotem či počítačem tím, že na něj jednoduše klikne.

Pokud chce hráč kameru přiblížit či oddálit, má k tomu k dispozici rolovací kolečko myši. Tento ovládací prvek je součástí všech moderních myší a proto není třeba tuto činnost odchyťovat za pomoci klávesnice. I kdyby však náhodou hráč neměl k dispozici myš s kolečkem, může ji zcela ignorovat. Tato funkce není k úspěšnému postupu hrou nijak vyžadována, slouží pouze jako možnost pro hráče podívat se na herní objekty z větší blízkosti. Přesná definice ovládání hry je v následující tabulce (Tab. 1). Ovládání na úrovni menu či inventáře je pak řešeno pouze pomocí levého tlačítka myši.

Tabulka 1 – ovládání hry

Levé tlačítko myši	Pohyb robota (pokud je to možné)
Pravé tlačítko myši	Interakce s okolím (seber, komunikuj)
Kolečko myši	Přibližování/oddalování kamery
I	Inventář zobrazit/skrýt
F1, F2, F3, F4	Změna kamery
ESC	hlavní menu hry

7.1.6 Pravidla hry a postup hráče

Pravidla každé dobré hry nesmí být příliš komplexní, aby si je hráč rychle osvojil a dokázal se do hry vžít, ale zároveň musí hráči nabídnout výzvu. Pro jednoduchou implementaci jsme tak zvolili pravidla velice jednoduše:

- Hlavní hrdina (tedy hráč) má k dispozici inventář, ve kterém může přenášet najednou až 6 předmětů.
- V inventáři je možno předměty vzájemně kombinovat (pokud to jde), někdy je tato kombinace nezbytná.
- Sebrané předměty lze použít, tato činnost hráče posouvá do dalších částí hry.
- Hlavní hrdina má možnost vést dialogy s ostatními roboty či počítači, výsledky těchto dialogů jsou často podmínkou pro další postup ve hře.
- V dialogu má hráč většinou na výběr z vícera možností, každou odpovědí se mu otvírá další část rozhovoru.

Pokud se tedy hráč dostane do problému, jeho řešení může být pouze dvojího druhu: v určitém dialogu musí hráč sestavit správnou kombinaci odpovědí, nebo je řešením sběr a používání předmětů.

7.1.7 Fyzika a její vliv na hratelnost

Dostatečná variabilita prostředí je dnes téměř podmínkou k vytvoření kvalitní hry. Proto jsme se při návrhu hry rozhodli pro implementaci fyzikálního enginu. Ten nám velkým dílem dopomohl snížit staticnost herního prostředí. Pomocí fyzikální knihovny lze také snadno vyřešit problémy s kolizemi hlavního hrdiny a ostatními objekty.

Skutečný vliv na hratelnost ovšem fyzikální engine nemá velký. Je ale možné některé důležité předměty skrýt za bedny či barely a nechat na hráči, aby je tělem hrdiny odsunul. V původním návrhu se počítalo s několika hádankami, které by fyzikální engine využily lépe, do finálního projektu se však z časových důvodů nedostaly.

7.2 Návrh uživatelského rozhraní hry

Uživatelské rozhraní slouží ke komunikaci hráče a hry. Umožňuje tak uživateli dávat hře příkazy, jakým způsobem se bude hra dál ubírat. Hře naopak umožňuje hráči zobrazovat výsledky jeho snažení.

Uživatelské rozhraní naší hry se skládá z hlavního menu, inventáře a prostředí pro rozhovory. Všechny tyto části jsou na sobě vzájemně zcela nezávislé a jsou objektově zapouzdřeny. K jejich práci tak není potřeba znát jejich vnitřní funkčnost, ale pouze umět používat jejich metody.

7.2.1 Hlavní menu hry

S pomocí hlavního menu hráč může hru spustit či ukončit, nebo procházet její nastavení a měnit je. Hlavní menu je charakterizováno třídou, jeho prvky jsou privátní (i veřejné) prvky této třídy a jsou to objekty tříd spadajících pod Irrlicht GUI rozhraní.

7.2.1.1 Návrh menu a požadavky

Pro návrh menu jsme měli několik požadavků, jejich posloupnost si rozepíšeme v bodech:

- Nezávislost na rozlišení: menu by se mělo zobrazovat vystředěné v jakémkoliv podporovaném rozlišení.
- Vykreslení menu bez používání externích grafických knihoven, tedy pouze za použití možností pro uživatelské prostředí v Irrlichtu.

- Přehlednost, jednoduchost, slušná grafická úroveň a dobrá čitelnost textu.

7.2.1.2 Struktura menu

Struktura menu je z velké části závislá na tom, jakým způsobem funguje struktura uživatelského rozhraní v Irrlichtu. Jak jsme si již v teoretické části zmínili, Irrlicht řadí jednotlivé prvky uživatelského rozhraní do rodičovských vazeb. Potomek má tedy své souřadnice závislé na svém předkovi a ten zase na svém atp. Toho jsme při tvorbě menu využili.

Nejvyšším prvkem v stromové struktuře je okno menu. Je charakterizováno třídou *IGUIWindow*. Tento prvek používám pouze k zapouzdření všech ostatních, žádné další funkce okna v Irrlichtu nejsou využity. Dalším prvkem je pak obrázek pozadí, objekt *IGUIImage*. Velikost obrázku na pozadí menu je konstantní, proto je také menu pod všemi rozlišeními stejně velké.

Obrázek na pozadí je pak rodičovským prvkem pro všechny ostatní objekty menu. Jeho prvním potomkem je obrázek loga. Dalšími potomky jsou pak všechny ovládací prvky menu: tlačítka (*IGUIButton*), check boxy (*IGUICheckBox*) a posuvníky (*IGUIScrollBar*).

7.2.1.3 Třída pro zapouzdření nastavení hry

Definice třídy zapouzdřující všechna nastavení hry se nachází v hlavičkovém souboru *settings_class.h* a její název je *settings_node*. Tato třída slouží k ukládání a načítání nastavení hry do XML souboru, jehož struktura je vyobrazena na přiloženém obrázku (Obr. 12). Všechna tato data se pak pomocí jednoduchých metod z této třídy získávají.

```
1 |<?xml version="1.0" ?>
2 |<root>
3 |   <resolution width="1680" height="1050" depth="32" />
4 |   <water_quality value="1" />
5 |   <sound_volume value="4" />
6 |   <music_volume value="5" />
7 |   <fullscreen value="1" />
8 |</root>
9 |
```

Obrázek 12 – struktura souboru pro ukládání nastavení

Seznam hodnot, které tato třída ukládá (v závorce jsou uvedeny datové typy):

- Rozlišení obrazovky (*dimension2d*): hodnota rozlišení hlavního herního okna.
- Barevná hloubka (*u32*): hloubka zobrazovaných barev v bitech, Irrlicht umožňuje pouze 16 nebo 32.
- Hlasitost hudby (*f32*): hlasitost přehrávané hudby ve hře, rozmezí nastavení je od 0 (hudba je zcela vypnuta) do 1 (hlasitost hudby je na nejvyšší hodnotě).
- Hlasitost zvuků (*f32*): hlasitost efektů a dalších zvuků ve hře (také zvuky menu), tato hodnota je zcela nezávislá na hlasitosti hudby, rozmezí je zvoleno stejně.
- Antialiasing (*bool*): volba, zda se při vykreslování scény má používat antialiasing.
- Celá obrazovka (*bool*): volba, zda se má hlavní okno hry roztáhnout na celou obrazovku.

Popis některých metod, které třída pro nastavení obsahuje (v závorce jsou uvedeny parametry a návratové hodnoty):

- Konstruktor (cesta k XML souboru nastavení, nevrací nic): konstruktor načítá data uložená v souboru a ukládá je do datové struktury v třídě. Hodnoty uložené pod typem s plovoucí řádovou čárkou jsou v souboru uloženy jako celá čísla, stará se tedy o jejich převod (v souboru je hodnota uložena v rozmezí 0 – 10, provede tedy prosté dělení a přetypuje). *Boolean* hodnoty XML parser také nezná, v souboru jsou tedy uloženy jako celé číslo (0, 1).
- Metody pro získávání dat (bez parametrů, vrací hodnoty uložené v nastavení): jména těchto funkcí začínají slovíčkem „get“. Pro každou hodnotu nastavení je vytvořena jedna. Kupříkladu metoda pro získání rozlišení má název *get_resolution()* a vrací rozlišení ve formátu *dimension2d*.
- Metody pro ukládání dat do datové struktury (parametry ve formátech jednotlivých nastavení, nevrací nic): tyto metody začínají slovem „set“ a fungují obdobně jako ty pro získávání dat.
- Metoda pro uložení dat do souboru (bez parametru, nevrací nic): uloží data do stejného souboru, ze kterého konstruktor četl. Zároveň se stará o zpětné přetypování všech dat před uložením.

7.2.1.4 Třída pro zapouzdření hlavního menu

Třída charakterizující menu je definována v hlavičkovém souboru *menu_class.h* a její název je *menu_node*. Uvedu stručný seznam prvků této třídy:

- Ukazatel pro práci s Irrlicht GUI rozhraním (*IGUIEnvironment*), ukazatel pro Irrlicht video driver (*IVideoDriver*) na práci s obrázky.
- Rozlišení herní obrazovky (datový typ *dimension2d*).
- GUI prvky pro okno, obrázky loga a pozadí menu, tlačítka, posuvníky a check boxy.
- Objekt charakterizující nastavení hry.

Metody používané pro práci s hlavním menu (metody jsou s výjimkou konstruktoru bez parametrů a nevrací žádné hodnoty, v závorkách jsou uvedeny přesné názvy metod):

- Konstruktor: jako parametr přijímá ukazatele na objekt nastavení a Irrlicht zařízení. Uvnitř metody se vytváří všechny GUI prvky, které zůstanou až do vypnutí celé aplikace (menu se zavřením pouze skryje).
- Zobrazení menu (*show_menu*): metoda pro zobrazení menu, volá se kupříkladu při stisknutí klávesy escape. Tato metoda zobrazuje skryté prvky menu.
- Zavření menu (*close_menu*): tato metoda menu skryje, pokud je zobrazeno.
- Skrytí všech tlačítek menu (*erase_menu*): tato metoda skryje všechny tlačítka menu, volá se při přechodu z jednoho menu do druhého.
- Zobrazení hlavního menu (*main_menu*): tato metoda zobrazuje tlačítka pro hlavní menu.
- Zobrazení menu nastavení (*options_menu*): metoda zobrazí všechny ovládací prvky pro menu nastavení.



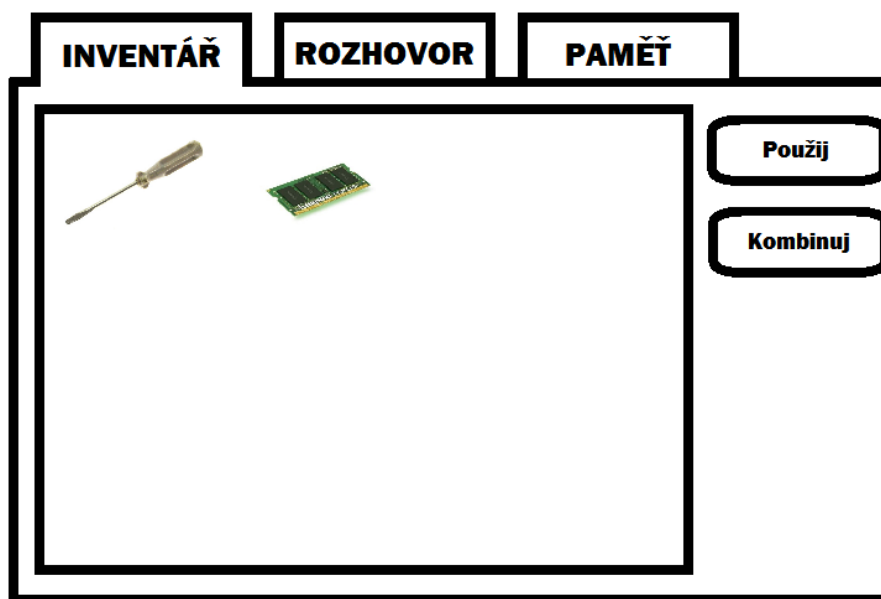
Obrázek 13 – finální verze hlavního menu

7.2.2 Inventář

Inventář ve hře slouží jako katalog sebraných předmětů. Je možné zde předměty kombinovat a používat. Inventář je podobně jako hlavní menu charakterizován třídou, která zapouzdřuje všechny prvky GUI i množinu sebraných předmětů.

7.2.2.1 Návrh inventáře a požadavky

V prvotním návrhu pro inventář se počítalo s jediným komplexním uživatelským prostředím pro inventář i ovládání rozhovorů dohromady (Obr. 14). Tyto prvky měli být k dispozici pod záložkami, poslední záložka měla charakterizovat jakousi paměť hlavního hrdiny, kde měli být zapsány aktuální úkoly. Z tohoto složitého návrhu ovšem později sešlo.



Obrázek 14 – prvotní návrh herního inventáře

V dalším návrhu jsme se tedy spokojili pouze s jednoduchým inventářem, další prvky hry měly mít vlastní rozhraní. Naše požadavky byly tedy následující:

- Podobně jako u menu, nezávislost na rozlišení obrazovky a vykreslování bez použití externích knihoven.
- Z omezení daných GUI rozhraním Irrlichtu musel být omezen počet sebraných předmětů.
- Charakterizace předmětu pomocí obrázku a jména.
- Možnost označení jednoho či dvou předmětů a následné použití či zkombinování.

7.2.2.2 Vzhled a struktura inventáře

Struktura inventáře je velice podobná struktuře hlavního menu, proto si jej popíšeme jen stručně. Hlavním rodičovským prvkem všech ostatních GUI objektů je tedy také okno (*IGUIWindow*). Jeho prvním potomkem je obrázek pozadí. Tento obrázek je roztáhnutý na celou šířku i výšku okna. Aby byly další prvky inventáře pod obrázkem na pozadí vidět, musí být tedy jeho potomci a ne potomci okna (v případě překrývání dává uživatelské rozhraní Irrlichtu přednost tomu prvku, který se vykresluje jako poslední).



Obrázek 15 – konečná podoba inventáře

Obrázek má pak 8 potomků: 6 velkých tlačítek, které charakterizují jednotlivé sebrané předměty a 2 menší tlačítka pro kombinaci a použití předmětu. Velká tlačítka mají pak jako potomka nastaven obrázek, který jednotlivé předměty charakterizuje a ještě menší část grafiky, které zobrazuje označení předmětu.

Na obrázku (Obr. 15) je znázorněna konečná podoba herního inventáře. Vrchní tři tlačítka znázorňují sebrané předměty, spodní velká tlačítka jsou prázdná místa v inventáři (slotů je tedy v inventáři celkem šest). Dvě spodní tlačítka slouží k použití či kombinaci předmětů, pokud není označen právě jeden předmět, tlačítko pro použití zůstává neaktivní. Na tlačítko pro kombinaci předmětů lze pak kliknout jen tehdy, jsou-li označeny dva předměty, které lze kombinovat. Zelené ohraničení u tlačítka v pravém horním rohu tedy označuje, že je tento prvek vybrán.

7.2.2.3 Třída pro charakterizaci předmětů

Než se zaměříme na samotnou třídu pro inventář, musíme si nejprve osvětlit, jakým způsobem je ve hře charakterizován předmět, který může hráč ve hře sebrat. Tato třída je definována v hlavičkovém souboru *pickable_class.h*, její název je pak *pickable_node*.

Tyto předměty nemají ve fyzikálním modelu naší hry svoji reprezentaci, snažíme se tak zabránit situacím, které by hráči mohli znemožnit další postup hry: kupříkladu hráč by

mohl omylem tento předmět odsunout mimo svůj vlastní dosah a potom by nebyl schopen jej sebrat a tedy i použít. Níže si uvedeme stručný seznam datové struktury této třídy (v závorce jsou uvedeny datové typy):

- Animovaný uzel scény (*IAnimatedMeshSceneNode*): tento prvek charakterizuje předmět ve hře, má v sobě uloženou informaci o aktuální pozici prvku, všechny textury a model.
- Jméno objektu (*stringw*): jednotlivé předměty se od sebe odlišují právě jménem, které musí být jedinečné.
- Kombinační identifikační číslo (*u32*): udává identifikátor, se kterými objekty je tento předmět kombinovatelný, 0 značí, že tento předmět kombinovat nelze.
- Místo k použití (*vector3df*): charakterizuje pozici na mapě, na které lze předmět použít.
- Jméno odvozených předmětů (*stringw*): název předmětů, které se získají po kombinaci tohoto předmětu (kombinace dvou předmětů tyto předměty smaže a vytvoří jeden výsledný předmět).
- Místo k použití odvozených předmětů (*vector3df*): pozice, kde lze použít odvozené předměty z tohoto předmětu (jeho potomky po kombinaci).
- Obrázek charakterizující objekt (*ITexture*): malý obrázek (200x200px), který zobrazuje podobu předmětu (obrázek vytípnut jako render modelu v Blenderu).

Seznam některých užitečných metod pro práci s předměty (v závorkách jsou uváděny potřebné parametry):

- Konstruktor (ukazatel na Irrlicht zařízení, animovaný uzel scény, jména a pozice předmětu i jeho potomků): ukládá všechna data pro předmět a načítá obrázek.
- „Set“ a „get“ metody (dle významu): obdobně jako u dřívějších tříd se těmito metodami načítají či zapisují data do privátních prvků třídy.
- Přetížení operátoru porovnávání (`==`): tato metoda je nezbytná pro lineární vyhledávání předmětů, které jsou umístěny v dynamických polích. Porovnává se podle jmen předmětu, která jsou jedinečná.

7.2.2.4 Třída pro herní inventář

Třída pro inventář je definována v souboru *inventory_class.h* a jmenuje se *inventory_node*. Má za úkol starat se o ukládání sebraných předmětů v jednotné struktuře a poskytovat metody pro další práci s těmito předměty. Uvedeme si stručný seznam prvků této třídy (v závorce jsou uváděny použité datové typy):

- Dynamické pole sebraných předmětů (kontejner *array* pro *pickable_node*): počet předmětů, které může hráč sebrat, je omezen na 6. V tomto poli jsou všechny uloženy a přistupuje se k nim pomocí jednoduchých celočíselných identifikátorů.
- Dynamické pole označených předmětů (kontejner *array* pro *pickable_node*): v tomto kontejneru se ukládají označené předměty, je omezen na maximální počet dvou prvků. Pomocí označování hráč určuje, kterých předmětů se další jeho volby týkají.
- Rozlišení obrazovky (*dimension2d*): tato hodnota je důležitá pro vystředění inventáře na střed obrazovky při jakémkoliv rozlišení.
- Prvky uživatelského rozhraní inventáře: podobně jako u menu jsou zde uloženy ukazatele všech GUI prvků inventáře.

Výčet některých metod pro práci s inventářem (v závorkách jsou parametry těchto metod):

- Zobraz inventář (*void* metoda): tato metoda zobrazí inventář, spouští se po odchycení klávesy „I“. Podobně jako menu je inventář vytvořený nastálo a při zavření se pouze skryje.
- Zavři inventář (bez parametrů): tato metoda inventář opět skryje.
- Přidej předmět (ukazatel na objekt třídy *pickable_node*): přidá do kontejneru předmětů tento předmět. Probíhá zde také kontrola, jestli již počet sebraných předmětů nepřekročil limit. Tato metoda se volá po pravém kliknutí na předmět, pokud je hrdina dostatečně blízko.
- Zahod' předmět (identifikátor předmětu): odstraní vybraný předmět z kontejneru. Ostatní předměty se dle možností posunou. Volá se například po úspěšném provedení kombinace.

- Vyber předmět (identifikátor): přesune zvolený předmět mezi vybrané. Probíhá kontrola, zda vybraných předmětů není moc. Pokud ano, předmět se nevybere.
- Zjištění kombinovatelnosti (bez parametrů): vrací *bool* hodnotu, jsou-li dva vybrané předměty kombinovatelné (porovnává jejich identifikátory kombinovatelnosti). Pokud nejsou vybrány 2 předměty, vrací nepravdu.
- Použij předmět (pozice hlavního hrdiny ve *vector3df*): Použije jeden vybraný předmět. Porovnává, jestli se robot nachází v okolí, ve kterém lze předmět použít. Vrací *bool* hodnotu, zda proces proběhnul.
- Kombinuj předměty (bez parametrů): zkombinuje dva vybrané předměty. Zahodí původní, a vytvoří nový předmět a nastaví jeho vlastnosti.

7.2.3 Rozhraní pro rozhovory

Rozhovory jsou ve hře vyobrazeny pomocí uživatelského prostředí. Hráč před sebou vidí výpis předchozího rozhovoru a má na výběr z několika možností, které mu rozhovor poskytuje. Tyto možnosti jsou charakterizovány tlačítky, po kliknutí na zvolené tlačítko se dialog posune do dalšího kroku.

Rozhovor lze ve hře inicializovat pravým kliknutím na předmět, který je hráči označen. Dialog ovšem nemusí sloužit pouze pro samotné rozhovory, může sloužit v podstatě k libovolnému strukturálnímu vyobrazení informace pro hráče.

7.2.3.1 Souborová struktura rozhovorů

Rozhovory jsou ve hře načítány pomocí XML souborů. Každý předmět, se kterým lze rozhovor ve hře zavést, má tedy svůj jeden nebo i více těchto souborů, ve kterých jsou data pro rozhovor uložena. Tyto soubory jsou umístěny v adresáři *dialogues*, který se nachází u hry.

Na obrázku (Obr. 16) je znázorněna struktura těchto souborů. Uvnitř tagů text se nachází jednotlivé uzly rozhovoru. Každý uzel se skládá z textu, který se zobrazí jako reakce na předešlou volbu (atribut *value* tagu *text*) a ze svých vlastních voleb (tagy *option*). Tento tag má důležitý atribut *link*, který značí, na který další uzel rozhovoru tato volba směřuje. Všechny uzly i volby jsou identifikovány pomocí atributu *id*.

Každý rozhovor tedy začíná přístupem na uzel 0 a dále zobrazí hráči možnosti, které tento uzel u sebe má. Po zvolení volby se rozhovor přesune do dalšího uzlu, jehož identifikátor se shoduje se směrovacím identifikátorem zvolené volby. Opět se pak zobrazí text uzlu atd. Rozhovor je ukončen tehdy, klikne-li uživatel na možnost s textem „END“. V tomto případě nezáleží na směrovacím identifikátoru této možnosti a rozhovor se ukončí.

```
1 <?xml version="1.0"?>
2
3 <root outcome="2">
4   <text id="0" value="Testovací dialog, zvolte svou volbu: ">
5     <option id="0" link="1">Volba 1</option>
6     <option id="1" link="2">Volba 2</option>
7     <option id="2" link="0">END</option>
8   </text>
9   <text id="1" value="Vybral jste volbu 1">
10    <option id="0" link="0">END</option>
11  </text>
12  <text id="2" value="Vybral jste volbu 2">
13    <option id="0" link="0">END</option>
14  </text>
15 </root>
```

Obrázek 16 – struktura XML souborů pro ukládání rozhovorů

Další důležitou informací uloženou v souboru je identifikátor výsledku rozhovoru, označeném jako atribut kořenového tagu *outcome*. Pomocí tohoto prvku se ve hře rozhoduje, byl-li rozhovor úspěšný. Identifikátor označuje uzel, kterým hráč musí projít, aby splnil podmínku rozhovoru.

Při psaní rozhovorů je tak velice důležité dbát na správnou posloupnost identifikačních čísel a správném provázání jednotlivých uzlů a voleb. Mohlo by se totiž snadno potom stát, že se hráč zacyklí v části rozhovoru, ze které již není úniku.

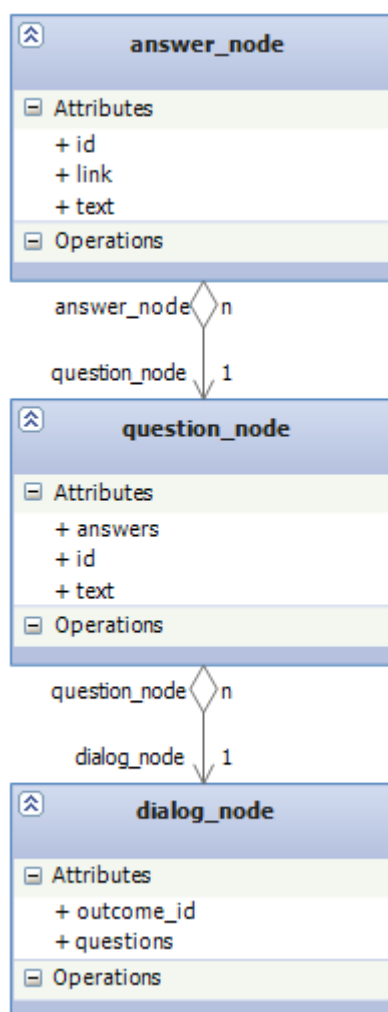
7.2.3.2 Datová struktura rozhovorů

Po načtení dialogu ze souboru se tento uloží do datové struktury. Ta je charakterizována třemi třídami, jejichž vzájemný vztah je vyobrazen na obrázku (Obr. 17).

Třída *answer_node* charakterizuje jednotlivé volby, má v sobě uložen svůj identifikátor i identifikátor směrovací (oba ve formátu *u32*), a také samotný text, který se zobrazuje na tlačítkách voleb (datový typ *stringc*).

Další v pořadí je třída *question_node*, která je označením pro jednotlivé uzly rozhovoru. Šipka k ní směřující dolů označuje agregaci 1:N, to znamená, že každý uzel může mít N

voleb. Třída má privátně uložen identifikátor uzlu (*u32*), text uzlu (*stringc*) a dynamické pole svých voleb (*array*).



Obrázek 17 – UML diagram

datové struktury rozhovorů

Poslední třída je *dialog_node*, která zapouzdřuje celý rozhovor. Opět je zde agregace 1:N vůči předchozí třídě. Datová část této třídy je identifikátor výsledku rozhovoru (*u32*) a pole všech uzlů rozhovoru (*array*).

Třída *dialog_node* se také stará o grafické rozhraní rozhovoru, podobně jako u menu a inventáře má tedy v sobě uloženy ukazatele na jednotlivé grafické prvky. Všechna funkčnost je tedy velice podobná a podrobněji si ji popisovat nebudeme.

7.2.3.3 Třída pro objekty k navázání rozhovoru

Tato třída charakterizuje objekty, na něž lze kliknout pravým tlačítkem a navázat tak rozhovor. Třída je definována v hlavičkovém souboru *talkable_class.h* a jmenuje se *talkable_node*. Opět tedy uvedu stručný seznam nejdůležitějších prvků této třídy (v závorce jsou použité datové typy):

- Animovaný uzel scény (*IAnimatedMeshSceneNode*): charakterizuje polohu, model objektu i jeho textury.
- Fyzikální reprezentace objektu (*NewtonCollision*, *NewtonBody*): tyto objekty na rozdíl od předmětů už ve fyzikální knihovně reprezentovány jsou, ale pouze staticky (kolize existují, na předměty však neúčinkují žádné síly).



Obrázek 18 – finální podoba dialogu

- Cesta k souboru s rozhovorem (ukazatel na *const char*): rozhraní rozhovorů funguje trochu jinak než menu a inventář: při startu každého rozhovoru se vytvoří nový, konec rozhovoru pak volá jeho destruktorku. Tato funkčnost je v aplikaci zavedena z důvodu, aby šel rozhovor pro stejný objekt snadno měnit (jednoduchou změnou této cesty).

- Povolení rozhovoru (*boolean*): pomocí této proměnné je možné zakázat zobrazení rozhovoru po kliknutí. Používáme tak kupříkladu v případě, že je objekt neaktivní a hráč musí k jeho aktivaci něco udělat.
- Jméno objektu (*stringw*): jedinečný identifikátor objektu.

7.3 Grafická stránka hry

Tvorba grafiky do hry si vzala asi třetinu naší práce. Při navrhování byl zvolen i design grafických prvků: chtěli jsme realisticky vypadající hru s pěknými modely i texturami. Jelikož ani jeden z nás nepovažuje sám sebe za umělce, může se zdát grafická podoba naší hry poněkud strohá. Je to ovšem důsledek toho, že tvorbu grafiky měli na starosti nezkušení programátoři. Tvorba grafiky se ve zkratce skládala ze dvou činností: modelování a texturování.

7.3.1 Modelování

Při tvorbě modelů jsme se často inspirovali předměty v našem okolí, stejně tak i některými sci-fi filmy či seriály. Na počátku každého modelu byl předběžný návrh, který byl nakreslen tužkou na papír, v některých, jednodušších případech jeho podoba zůstala pouze v hlavě autora.

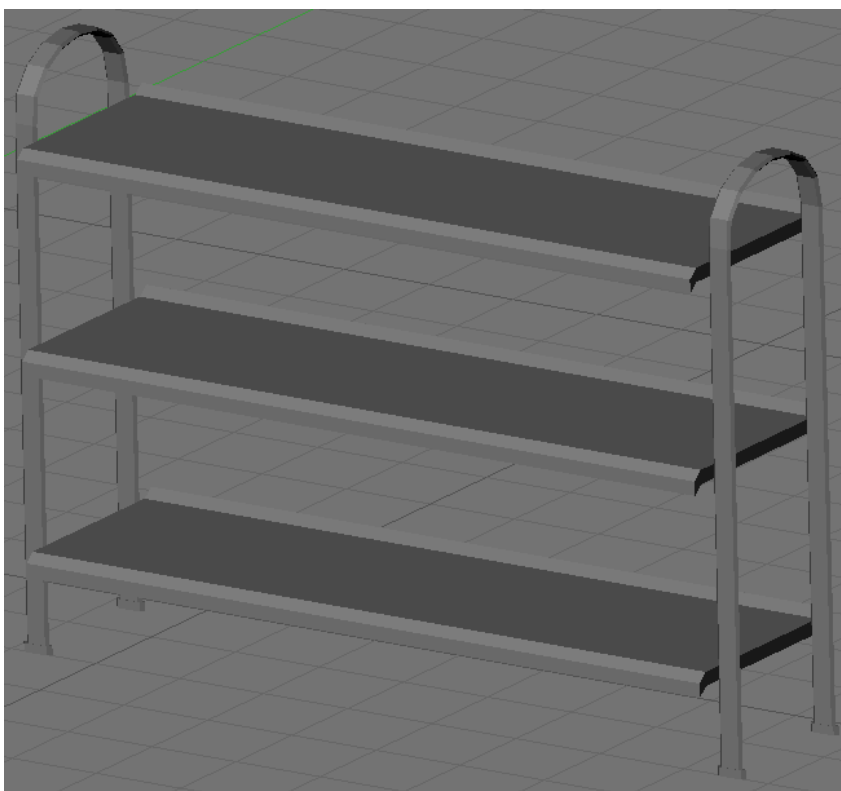
Po důsledném promyšlení návrhu tedy přišlo na řadu samotné modelování. Veškeré práce jsme prováděli v programu Blender, který jsem stručně popsal v teoretické části.

7.3.1.1 Modelovací techniky

Protože jsme dopředu nevěděli, jak moc detailní modely nám Irrlicht vykreslí při zachování stabilního framerate, byly naše první modely velice jednoduché. Základním tvarem pro většinu mých modelů tak byla Blenderovská primitiva (nejčastěji krychle, koule a kužel). Tyto základní tvary jsem potom pomocí funkcí tažení (*extrude*), rozdělení (*subdivide*), zaoblování (*bevel*) nebo rozřezávání (*knife*) upravoval do jejich finálních podob.

Další častou technikou při modelování bylo využívání křivek. Pracovali jsme s křivkami Bezierovými, protože jejich možnosti nám vyhovovali nejvíce. Pomocí tažení či tvarování je možné dosáhnout složitějších tvarů, než s pomocí primitiv. Přesto se i tyto tvary daly „osekat“ na nízký počet vertexů například pomocí funkce *decimate*.

Na obrázku (Obr. 19) je ukázka modelu police do naší hry. Každý jednotlivý regál má základ v krychli (*cube*), jenž je pomocí několikrát aplikovaného vytažení (*extrude*) vyvedena do výsledné podoby. Rámová konstrukce po stranách je pak vytvořena pomocí Bezierovy křivky a její následné transformace na mesh. Tento konkrétní model má pouze 228 vertexů, přesto vypadá docela dobře.



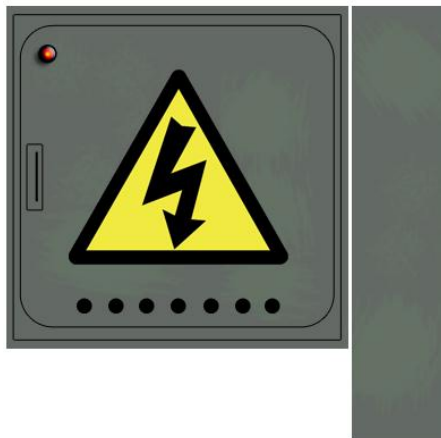
Obrázek 19 – jednoduchý model police vytvořený v Blenderu

7.3.2 Texturování

Tvorba textur je další důležitou částí práce. Kvalitní textura dokáže zakrýt nedokonalosti modelu. Při texturování byla použita metoda UV mapování, což je nejflexibilnější způsob mapování 2D textury na 3D objekt. Při tomto procesu vezmeme trojrozměrnou plochu objektu a „rozbalíme“ ji na dvojrozměrný obraz, na kterém vytvoříme texturu v jiném programu. Následně tento 2D obraz nanese jako UV texturu zpátky na 3D objekt.

Na obrázku (Obr. 20) je příklad textury, kterou může být model potažen. V tomto případě se jedná o jednoduchý model tvaru kvádra. Větší čtverec reprezentuje hlavní část objektu, která je nejvíce vidět. Části objektu, které jsou ve většině případů kameře skryty (v našem případě zbývající stěny kvádra) mohou být tedy velice zjednodušeny a znázorněny

například pouze jednou barvou. Všimněme si, že zadní část kvádru není na textuře vůbec znázorněna. Je to z toho důvodu, že tento objekt bude vždy těsně u zdi a zadní strana tedy může být potažená například bílou částí textury dole.



Obrázek 20 – ukázka textury

Pro tvorbu samotných textur byly použity programy Adobe Photoshop (zkušební 30 denní verze) a Gimp. Samotný postup nanášení textur byl tedy následující:



Obrázek 21 – výsledný objekt s texturou

- Rozřezání modelu na menší spolu související části: tato část je velice důležitá. Musí se správně zvolit hrany, které budou objekt rozdělovat (v Blenderu pomocí příkazu *mark seam*).
- Rozbalení objektu (*unwrap*): roztáhnutí povrchu 3D modelu na 2D.
- Export rozbaleného objektu (*save UV face layout*).

- Editace textury v programu pro práci s rastrovou grafikou (Adobe Photoshop či Gimp)
- Import obrázku a jeho aplikace jako textura.
- Export modelu do formátu, který Irrlicht podporuje (nejčastěji 3ds).

Na dalším obrázku (Obr. 21) je pak znázorněna podoba výsledného objektu s potaženou texturou.

7.4 Implementace fyzikálního engine

Pro naše účely jsme jako vhodný fyzikální engine zvolili Newton Game Dynamics. Je snadný k použití a na internetu existuje mnoho tutoriálů, jak jej úspěšně provázat s Irrlicht engine. Jeho popis jsem uvedl v teoretické části této práce.

Fyzikální knihovna je ve hře implementována v několika úrovních. První a nejdůležitější částí je jádro hry, ve kterém je vytvořený svět fyzikálního světa, který nám celou simulaci zapouzdřuje. V hlavní smyčce hry pak musí probíhat pravidelná aktualizace fyzikálního engine.

Další úroveň implementace je pak u samotných objektů ve hře. Tyto objekty se sami starají o své fyzikální reprezentace a poskytují fyzikální knihovně všechna potřebná data pro interakci s nimi (hmotnosti, síly potřebné k převrácení, těžiště apod.). Obě tyto úrovně tedy v následujících kapitolách popíšu.

7.4.1 Implementace fyziky – úroveň jádra hry

Na této úrovni se provádí samotná inicializace knihovny a nastavení některých základních požadavků, které dále rozepíši v bodech:

- Inicializace engine: zavoláme funkci *NewtonCreate()*, která vytvoří fyzikální svět. Tato funkce vrátí ukazatel na Newton zařízení.
- Nastavení základních vlastností světa: například nastavení frikčního modelu, detailu výpočtů kolizí atd.
- Nastavení hranic fyzikálního světa: tato hranice udává knihovně prostor, ve kterém může své výpočty provádět (pokud se objekt dostane mimo tuto hranici, žádná síla

už na něj nepůsobí). Toto nastavení se provádí voláním funkce *NewtonSetWorldSize()*.

- Aktualizace enginu: provádí se každý desátý vykreslený snímek, tedy v ideálním případě 6x za sekundu (framerate je pomocí vertikální synchronizace omezen na maximální hodnotu 60fps). Provádí se voláním funkce *NewtonUpdate()*.

7.4.2 Implementace fyziky – úroveň objektů ve hře

Objektů pro charakterizaci ve fyzikálním světě je hned několik, liší se podle použití. Níže si uvedeme dva základní, všechny ostatní jsou pak odvozeniny těchto dvou.

7.4.2.1 Třída pro charakterizaci mapy

Mapou je myšlen statický objekt, na který se neaplikuje síla v žádném směru. Tento objekt má kolizní model vytvořený pomocí *NewtonTreeCollision*, každý vertex meshe je tedy započítán. Pomocí tohoto objektu vytváříme jednotlivé části úrovně, jako jsou zdi a podlahy.

Třída pro charakterizaci mapy je definována v hlavičkovém souboru *physic_map_class* a jmenuje se *physic_map_node*. Třída obsahuje tato data (seznam není kompletní, jsou vybrány pouze důležité položky, v závorce je uveden typ dat):

- Animovaný uzel scény (*IAnimatedMeshSceneNode*): tento prvek charakterizuje předmět ve hře, má v sobě uloženou informaci o aktuální pozici prvku, všechny textury a model.
- Kolizní model objektu (*NewtonCollision*): jak bylo zmíněno výše, kolize u tohoto typu objektu se charakterizuje pomocí *NewtonTreeCollision*.
- Rigid body objektu (*NewtonBody*): objekt reprezentovaný fyzikální knihovnou.
- Důležité ukazatele pro práci s Irrlichtem a Newtonem.

Třída neobsahuje žádné důležité metody, popíšeme si tedy pouze konstruktor. Ten přebírá jako své parametry ukazatele pro práci s Irrlichtem i Newtonem a ukazatel na uzel scény. V cyklu je pak propočítán každý vertex modelu tohoto uzlu a přidán do kolizního modelu objektu.

7.4.2.2 Třída pro charakterizaci fyzikálního objektu

Fyzikální objekt je jakýkoliv uzel scény, který je reprezentován ve fyzikální knihovně. Kolizní model pro tento objekt může být vytvořen dvěma způsoby: *convex hull* nebo *bounding box*. *Convex hull* je pro detailnější reprezentaci objektu, je také ovšem výpočetně náročnější. *Bounding box* reprezentuje kvádrové ohraničení objektu. Pomocí těchto objektů vytváříme pohyblivé části mapy, jako jsou různé krabice či barely.

Třída pro všechny fyzikální objekty je definována v souboru *physic_object_class* a má jméno *physic_object_node*. Seznam obsažených dat je stejný jako u třídy *physic_map_node*.

Nejdůležitější metodou tohoto objektu je tedy zase konstruktor (parametry má stejné jako konstruktor pro objekt mapy, navíc obsahuje *bool* parametr, kterým programátor rozhoduje, jakým způsobem se bude počítat kolizní model a hmotnost objektu). Konstruktor ve svém těle počítá *convex hull* kolizní model, z jehož tvaru dále zjistí přibližné těžiště objektu.

Tato třída ovšem není jediná, která reprezentuje fyzikální objekty (další je například třída pro charakterizaci hlavního hrdiny). Všechny ostatní však fungují na naprosto stejném principu a nebudu se jimi tedy zabývat.

7.5 Ozvučení hry

Jako zvukový engine hry byl zvolen IrrKlang pro svou jednoduchost a jeho snadné použití v Irrlichtu. Jeho stručný popis je uveden v teoretické části této práce. Ozvučení je ve hře implementováno pomocí dvou na sobě nezávislých tříd. První se stará o hudební doprovod ke hře, druhá reprezentuje všechny ostatní zvuky. Toto nezávislé rozdělení je provedeno na úrovni IrrKlang zařízení. Každé zařízení tak může mít svou globálně nastavenou hlasitost.

Veškeré hudba i zvuky použité v této hře byly staženy z internetové stránky The Freesound Project [13]. Je to databáze zvuků uvolněných pod licencí Creative Commons, můžeme tedy volně používat a sdílet tyto soubory, nemůžeme je však měnit.

7.5.1 Hudba

Hudební doprovod ve hře obstarává třída *music_node*, její definice se nachází v hlavičkovém souboru *music_class*. Tato třída není ve hře reprezentována žádným viditelným objektem, slouží pouze k přehrávání hudby a nastavení hlasitosti pomocí metod.

Důležitá data, která třída pro charakterizaci hudby obsahuje (v závorce jsou uvedeny datové typy):

- Ukazatel na IrrKlang zařízení (*ISoundEngine*): tento je potřeba pro jakoukoliv práci s knihovnou.
- Dynamické pole zdrojů pro zvuky (kontejner *array* pro data typu *ISoundSource*): zde se ukládají veškeré hudební stopy hry. Třída k nim pak může jednoduše přistupovat a přehrávat je.
- Počet zvukových stop (*u32*): celé číslo reprezentující počet písní, které se budou z adresáře načítat.
- Identifikátor právě přehrávané písničky (*s32*): identifikátor určuje zdroj z dynamického pole, který je právě přehráván. Pokud není přehráván žádný, je roven -1.

Soupis některých důležitých metod pro práci se třídou *music_node* (v závorkách je uveden seznam atributů a návratové hodnoty těchto funkcí):

- Konstruktor (ukazatel na IrrKlang zařízení, nevrací nic): provádí načítání zdrojů hudby, které hledá v adresáři *music*. Zároveň nastaví identifikátor právě přehrávané písničky na -1.
- Přehraj píseň (identifikátor písně, *bool* hodnota opakování, nevrací nic): *boolean* hodnota uvedená v parametrech určuje, zda bude přehrávání opakované.
- Přehraj písně (identifikátor první písně, nevrací nic): přehrává všechnu hudbu v kolekci, dokud není zastaven. Postupuje pomocí inkrementace identifikátoru písně.
- Zastav hudbu (bez parametrů, nevrací žádnou hodnotu): přeruší všechny právě přehrávané písničky.

- Nastav hlasitost (*f32*): nastaví hlasitost veškeré přehrávané hudby.

7.5.2 Zvuky a efekty

Doprovodné ozvučení ve hře obstarává třída *sound_effects_node*, její definice je v hlavičkovém souboru *sound_effects_class*. Tato třída reprezentuje veškeré ostatní ozvučení hry, jako jsou zvuky menu či zvukové efekty ve hře.

Třída *sound_effects_node* je svou strukturou velice podobná třídě *music_node*, má však k dispozici dvě kolekce. První kolekce reprezentuje zdroje pro ozvučení menu, které jsou přehrávány klasickým způsobem. Druhá kolekce reprezentuje zvukové efekty, které jsou přehrávány trojrozměrně (tedy hlasitost přehrávání je závislá na poloze zdroje). Pojděme se ještě blíže podívat na metody této třídy, které nabízejí několik dalších funkcí (stejně jako v minulém případě jsou v závorkách uvedeny parametry a návratové hodnoty):

- Aktualizace polohy posluchače (poloha ve formátu *vector3df*, nevrací žádnou hodnotu): tato metoda je volána v hlavní smyčce programu (asi 6x za sekundu) a nastavuje posluchače zvukového zařízení na stejnou polohu, na které se nachází hlavní hrdina hry.
- Přehraj zvukový efekt (identifikátor zvuku ve formátu *u32*, *bool* opakování, pozice přehrávání zvuku ve *vector3df*): metoda pro přehrávání 3D zvuku, jako parametr přebírá vektor pozice, ve kterém se nachází zdroj zvuku.

7.6 Tvorba obsahu hry

Do této chvíle jsme řešili pouze práci na jádře hry. To je pro vývoj hry velice důležité, ale žádný skutečný obsah nepřinese. Pro naše účely jsme do naší hry chtěli vytvořit první úroveň s jednoduchým tutoriálem. Jednotlivé části levelu jsme modelovali v Blenderu a poté exportovali do formátů 3ds (pokud šlo o modely statické) nebo md2 (pokud šlo o animované modely).

7.6.1 Tvorba úrovně v IrrEditu

IrrEdit je freeware editor pro engine Irrlicht, jehož stručný popis je uveden v teoretické části této práce. Umožňuje vytváření Irrlicht scén, které ukládá do nativního formátu tohoto

enginu (irr). Jeho možnosti ovšem nejsou příliš široké, spoustu věcí tedy musí programátor psát přímo do aplikace.



Obrázek 22 – tvorba úrovně v editoru IrrEdit

Tvorba úrovně v IrrEditu tedy probíhala takto, ilustrace na obrázku (Obr. 22):

1. Import meshe – každý objekt ve hře byl načten jako animovaný mesh. Není pak třeba rozlišovat animované objekty od statických, pro Irrlicht jsou všechny stejné. Veškeré modely se nacházejí v adresáři *models* ve složce hry.
2. Načtení textury – používané formáty textur byly jpg a png. Všechny textury ve hře jsou v adresáři *textures*.
3. Aplikování textur na příslušné modely – všechny textury byly mapovány pomocí UV mapování ještě před exportem z Blenderu. Po použití textury na model je tedy ihned vše správně nastaveno.
4. Nastavení materiálu – pro průhledné či jinak výjimečné objekty je na úrovni editoru ještě nastaven materiál.
5. Nastavení velikosti a polohy objektu – modely jsou často vyexportovány tak, že vůči ostatním v poměru proporcí nesedí. Je tedy nutné jejich velikosti upravit. Dále je pak nutné nastavit polohu jednotlivých objektů, tvůrce pak má přehled o sestavení celé úrovně.

Na úrovni editoru lze tedy sestavit celý level, nelze však poté přidat jednotlivým objektům jejich funkčnosti. Tato práce je ponechána skriptovacímu systému hry. Každý uzel scény lze přímo v editoru charakterizovat dvěma identifikátory, číslem a jménem, přičemž ani jeden z nich nemusí být unikátní. Pro identifikaci objektů jsme si tedy zvolili jméno, které je ve formátu *stringw*.

Pomocí jmen objektů tedy charakterizujeme, jaký význam ve hře budou mít. Lze takto přesně vymezit například objekty, se kterými lze navázat rozhovor, předměty, které jdou sebrat nebo objekty, který žádný účel jednoduše nemají a slouží jen k vyplnění scény. Toto rozlišování je součástí komplexnějšího sestavovacího systému, který napsal můj kolega. Jeho úkolem je načíst mapu a dynamicky vytvořit a přiřadit objekty, které na ní najde. V tabulce (Tab. 2) jsou uvedeny některé speciální názvy objektů, které mají vymezené zvláštní funkčnosti.

Tabulka 2 – specifické názvy uzlů ve scéně

Klíčové jméno uzlu	Popis funkčnosti
<i>hero</i>	Postava hlavního hrdiny
<i>physic_border</i>	Vymezení fyzikální hranice, tyto objekty musí být dva a společně charakterizují kvádr fyzikálního ohrazení
<i>point-to-go</i>	Animovaný ukazatel pro pohyb hrdiny
<i>level</i>	Statické části úrovně
<i>spawn-steam</i>	Objekt vymezující generování páry
<i>spawn-fire</i>	Objekt vymezující generování ohně
<i>sl111</i>	Bod pro světlo bílé
<i>sl100</i>	Bod pro světlo červené
<i>sl010</i>	Bod pro světlo zelené
<i>sl001</i>	Bod pro světlo modré
<i>sl110</i>	Bod pro světlo žluté
<i>sl101</i>	Bod pro světlo purpurové
<i>sl011</i>	Bod pro světlo azurové
<i>pickable</i>	Sebratelný předmět, za pomlčkou je vždy uveden název
<i>talkable</i>	Předmět pro komunikaci, za pomlčkou je vždy uveden název
<i>static</i>	Objekt bez hmotnosti (nelze jimi posunout, ale kolidují s ostatními)

7.6.2 Tvorba úkolů do hry

Jak jsem již dříve předdesílal, možností není v naší hře mnoho a proto ani úkoly nemohou být příliš variabilní. Jediná možnost řešení problému je buďto pomocí rozhovoru, nebo

použití předmětu na správném místě. Přesto se nám s kolegou povedlo poskládat úvodní sadu úkolů do první úrovně, jejichž sestavení určitou podobu variability obsahuje.

7.6.2.1 *Příklad prvního úkolu hry*

Pro příklad tedy uvedu první sekvenci úkolů. Na počátku se hráč (robot) nachází v lodním skladišti, místě, kde se nakládá a vykládá zařízení. V rozhovoru s blízkým počítačem se dozví úvod do příběhu a je postaven před problém, jak se z této místnosti dostat. V dalším rozhovoru vyjde najevo, že místnost má dveře zablokovány z důvodu nedostatku kyslíku. Jelikož má první úroveň také charakter tutoriálu, dostane hráč radu jak atmosféru v místnosti dostat do správného stavu. Použitím páčidla uvolní uzávěr přísunu kyslíku do počítače, který vypouští vzduch do místnosti. Následně pomocí rozhovoru s tímto počítačem nastaví správný poměr kyslíku a kysličníku uhličitého ve vzduchu.

Další překážkou je však přístup k počítači, který provádí autorizaci otevírání dveří. Tento počítač totiž neobsahuje obvod, který umožňuje komunikaci s roboty. Hrdina tedy musí tuto komunikační konzoli nalézt. Po prozkoumání úrovně ji nalezne v části mapy, do které nemá přístup (nižší patro, které je zatopeno vodou). Pomocí rozhovoru s čerpadlem tedy nejprve vodu odčerpá, poté také pomocí rozhovoru s mechanickou paží získá potřebný předmět. Po připojení komunikační konzole jsou tedy dveře hráči odemčeny. K jejich otevření však stále nedojde, protože se na lodi porouchal generátor gravitace. Až po restartu tohoto generátoru je cesta pro hlavního hrdinu otevřena dále.

7.6.2.2 *Skriptování úkolů*

Naše aplikace nemá žádný jednotný skriptovací systém, je tedy třeba sekvenci úkolů do hry zapisovat jinak. Děje se tak na úrovni odchyťování vnějších událostí (event handler). Každé ukončení rozhovoru či použití předmětu zde tak může mít za následek téměř libovolné změny ve hře. Postup hráče hrou je tedy na této úrovni signalizován pomocí stavových booleovských proměnných, jejichž změna vyvolá reakci hry.

ZÁVĚR

Grafický engine Irrlicht je nástroj, který lze využít pro tvorbu 2D i 3D počítačových her. Není to nástroj dokonalý, je ovšem otevřený změnám a poskytován zcela zdarma. Pouze pomocí Irrlichtu však kvalitní hru vytvořit nelze. K dosažení interaktivity a kompletnosti je nutno implementovat ještě vhodný fyzikální a zvukový engine. Výběrem a implementací těchto knihoven jsem se zabýval v praktické části této práce, stejně jako tvorbou grafiky a uživatelského rozhraní. V teoretické části této bakalářské práce jsem se podrobně seznámil s používanými technologiemi při vývoji (Irrlicht, fyzikální engine Newton, zvuková knihovna IrrKlang a editor scén IrrEdit) a popisem některých programů, které jsme při tvorbě této práce využívali (Blender).

Tvorba komplexní počítačové hry není v reálném čase práce pro dva programátory. Hry se profesionálně vyvíjejí ve velkých týmech a každý člen má přísně vyhraněné pole působnosti. Tohle si musí každý tvůrce hry již při návrhu ujasnit a nenechat se strhnout plánováním nespílitelného. Dobrý návrh hry je tedy velice důležitý. Naše aplikace proto byla již od návrhu zamýšlena jako velice jednoduchá hra, ovšem s jasně danými pravidly. Přesto jsme později museli spoustu věcí z návrhu vypustit a spokojit se s jistými omezeními, času prostě nebylo dostatek.

Při výběru vhodné fyzikální knihovny jsme se nejvíce soustředili na kvalitní dokumentaci a jednoduchost implementace. Naše požadavky na samotné vlastnosti enginu (možnosti simulace a podobně) byly vcelku zanedbatelné, pro náš projekt jsme si vystačili jen se základními funkcemi, které mohla nabídnout kterýkoli engine. Po koketování s mnoha různými knihovnami jsme se nakonec spokojili s Newton Game Dynamics, který všechny naše požadavky splnil. Jeho implementace byla relativně snadná a dopomohlo nám při ní velké množství tutoriálů dostupných na Internetu. Oproti tomu výběr zvukové knihovny byl mnohem jednodušší. IrrKlang je svou strukturou Irrlichtu velice podobný a pracuje se s ním naprosto stejným způsobem. Jedinou jeho nevýhodou je jeho licence, v našem případě jsme však o komerčním využití projektu vůbec neuvažovali, proto se IrrKlang jevil jako nejlepší volba.

Poslední částí mé práce byla tvorba uživatelského rozhraní do hry. To se skládalo z menu, inventáře a systému dialogů. Při jeho tvorbě jsem se spokojil s nástroji grafického uživatelského rozhraní přímo v enginu Irrlicht. Jeho možnosti sice byly mnohdy omezené,

pro naše úmysly však dostačovaly. Pro komplexnější řešení uživatelského rozhraní má navíc tvůrce možnost implementovat do engine vlastní grafické struktury, jsou podporovány různé pluginy či skiny.

K dosažení výsledku praktické části této práce jsem se musel naučit velkou spoustu nových poznatků: naučil jsem se obstojně pracovat s Irrlichtem i s jeho editorem. Osvojil jsem si práci s fyzikálním engine a objevil úskalí jeho implementace do jádra hry. Navrhoval jsem složité datové struktury a tvořil grafické uživatelské rozhraní. Nejdůležitějšími novými poznatky však byla práce v týmu na společném projektu, komunikace a jednotné vytyčování cílů. Dalším krokem při rozvíjení našeho projektu by rozhodně musel být kvalitní skriptovací systém. Zároveň by se musel zdokonalit způsob načítání jednotlivých úrovní a všechna data pro jednotlivé úrovně by se dynamicky načítala z XML souboru. Po implementaci těchto funkcí přímo na úrovni jádra hry by se velmi zjednodušil způsob vytváření nových úrovní.

ZÁVĚR V ANGLIČTINĚ

Irrlicht graphics engine is a tool which can be used for 2D and 3D computer games. It's not perfect, but it is open source and provided for free. However, Irrlicht itself is not a guarantee of game quality. It is necessary to implement appropriate physical and sound engine to achieve interactivity and completeness. In the practical part of this work I have dealt with selection and implementation of these libraries, and graphics and user interface as well. In the theoretical part I have focused on the technologies used in development (Irrlicht, Newton physics engine, sound library IrrKlang and scene editor IrrEdit) and a description of some programs which we used while making this project (Blender).

Creating a complex computer game in real time is not work for two programmers. Games are professionally developed in large teams, and each member has a strict part. Every game designer must make this clear before he starts and assure himself he does not plan anything unrealistic. Good game design is really important. Therefore, our application was intended to be a very simple game, but with a clear set of rules. However, we had to skip some things from the original design later. There was simply not enough time.

While choosing a suitable physical library, we focused most on quality documentation and simplicity of implementation. Our requirements on properties of engine (simulation possibilities, etc.) are quite negligible for our project. We did just fine with basic functions which could offer any engine. After toying with many different libraries, we finally settled for Newton Game Dynamics, which met all our requirements. Its implementation was relatively easy and a large number of tutorials available on the Internet were really helpful. In contrast, the choice of sound library was much easier. IrrKlang has very similar structure as Irrlicht and working with both is exactly the same. Its only disadvantage is its license, in our case, however, we didn't wonder about commercial use of this project and irrKlang seemed like the best option.

The last part of my work was to create a user interface to the game. It is composed of menu, inventory and dialog system. In this work, I settled with a graphical user interface tools in the Irrlicht engine. Its options were often limited to our intentions, however, it was sufficient. The user interface also has the possibility to implement the author's own graphics structure for a more comprehensive solution, various plugins and skins are supported.

I had to learn a lot of great insights to achieve practical results of this work: I learned to work fairly well with Irrlicht and IrrEdit. I learned to work with physics engine as well and discovered the pitfalls of its implementation into the game core. I designed a complex data structures and created a graphical user interface. The most important new findings, however, was the team work on a joint project, communication and common goal-setting. The next step in developing our project definitely must be a good scripting system. It also would have to improve the loading of levels and all data for each level should be dynamically loaded from the XML file. Implementing these functions directly to the core of the game would greatly simplify the creating of new levels.

SEZNAM POUŽITÉ LITERATURY

- [1] GEBHARDT, Nikolaus. Irrlicht Engine : A free open source 3d engine [online]. 2009 [cit. 2011-05-75]. Dostupné z WWW: <<http://irrlicht.sourceforge.net/>>.
- [2] THORN, Alan. Introduction to Game Programming with C++. [s.l.] : Wordware Publishing, 2007. 392 s. ISBN 9781598220322.
- [3] IrrKlang : An audio library for C++, C# and .NET and high level 3D and 2D sound engine [online]. 2008 [cit. 2011-05-27]. Dostupné z WWW: <<http://www.ambiera.com/IrrKlang/index.html>>.
- [4] IrrEdit : A free realtime 3D world editor [online]. 2008 [cit. 2011-05-27]. Dostupné z WWW: <<http://www.ambiera.com/IrrEdit/index.html>>.
- [5] ADAMS, Ernest. Fundamentals of Game Design. 2nd edition. [s.l.] : New Riders, 2010. 675 s. ISBN 9780321643377.
- [6] POKORNÝ, Pavel. Blender : naučte se 3D grafiku. 2. české. [s.l.] : BEN, 2009. 288 s. ISBN 978-80-7300-244-2.
- [7] KOSEK, Jiří. Kosek : Téměř vše o WWW [online]. 1999, 28. května 1999 [cit. 2011-04-23]. Úvod do XML. Dostupné z WWW: <<http://www.kosek.cz/clanky/xml/xml-uvod.html>>.
- [8] W3schools [online]. c2011 [cit. 2011-04-23]. Introduction to XML. Dostupné z WWW: <http://www.w3schools.com/xml/xml_what_is.asp>.
- [9] THOMASON, Lee; BERQUIN, Yves; ELLERTON, Andrew. Grinninglizard : TinyXml [online]. 2010 [cit. 2011-04-23]. TinyXml Documentation. Dostupné z WWW: <<http://www.grinninglizard.com/tinyxmldocs/index.html>>.
- [10] GEBHARDT, Nikolaus. Irrlicht3d [online]. 2005, 21-04-11 [cit. 2011-04-24]. Dostupné z WWW: <<http://www.irrlicht3d.org/>>.
- [11] JEREZ, Julio; SUERO, Alain. Newton Game Dynamics : Open-Source Physics Engine [online]. c2000 [cit. 2011-04-24]. Dostupné z WWW: <<http://newtondynamics.com>>.
- [12] Blender Foundation. Blender [online]. c2011 [cit. 2011-04-24]. Dostupné z WWW: <<http://www.blender.org/>>.

[13] The Freesound Project [online]. 2005 [cit. 2011-05-09]. Dostupné z WWW: <<http://www.freesound.org/>>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ADT	Abstraktní datový typ
API	Rozhraní pro programování aplikací
Checkbox	Prvek grafického uživatelského rozhraní, který umožňuje uživateli provádět výběr ze dvou možností (ano/ne)
Direct3D	Rozhraní pro práci s 3D grafikou pouze pro platformu Windows
Driver	Ovladač zařízení
Engine	Soubor knihoven, který reprezentuje jádro hry, nebo programu
GUI	Grafické uživatelské rozhraní
Kontejner	Soubor dat uložených v určité vzájemné relaci
Level	Úroveň hry
Mesh	Model ve formě sítě
MipMapping	Metoda, která zrychluje práci grafických karet tím, že vytváří několik verzí jedné textury v různých rozlišeních
OpenGL	Multiplatformní rozhraní pro tvorbu aplikací počítačové grafiky
Plugin	Software, který nepracuje samostatně, ale jako doplňkový modul jiné aplikace a rozšiřuje tak její funkčnost (zásuvný modul)
Rendering	Tvorba reálného obrazu na základě počítačového modelu
STL	Součást standartní knihovny jazyka C++, obsahuje velké množství užitečných datových struktur a algoritmů
Tag	Klíčové slovo nebo termín, kterému je přidělena informace
Triangl	Plocha trojúhelníkového tvaru
Vertex	Bod v prostoru, je definován souřadnicemi (x, y, z)
Void	Datový typ jazyka C++ (a mnoha dalších), značí funkci bez návratové hodnoty

SEZNAM OBRÁZKŮ

Obrázek 1 – logo engine Irrlicht	12
Obrázek 2 – logo editoru IrrEdit	23
Obrázek 3 – základní prostředí 3D grafického editoru IrrEdit	23
Obrázek 4 – logo engine IrrKlang	26
Obrázek 5 – logo engine Newton	30
Obrázek 6 – primitiva tvaru kapsle	31
Obrázek 7 – složená primitiva	32
Obrázek 8 – charakterizace meshe pomocí primitiva ConvexHull	32
Obrázek 9 – logo Blenderu	36
Obrázek 10 – ilustrace pohledu kamery	41
Obrázek 11 – náčrtek hlavního hrdiny hry	42
Obrázek 12 – struktura souboru pro ukládání nastavení	45
Obrázek 13 – finální verze hlavního menu	48
Obrázek 14 – prvotní návrh herního inventáře	49
Obrázek 15 – konečná podoba inventáře	50
Obrázek 16 – struktura XML souborů pro ukládání rozhovorů	54
Obrázek 17 – UML diagram	55
Obrázek 18 – finální podoba dialogu	56
Obrázek 19 – jednoduchý model police vytvořený v Blenderu	58
Obrázek 20 – ukázka textury	59
Obrázek 21 – výsledný objekt s texturou	59
Obrázek 22 – tvorba úrovně v editoru IrrEdit	65

SEZNAM TABULEK

Tabulka 1 – ovládání hry	43
Tabulka 2 – specifické názvy uzlů ve scéně	66

SEZNAM PŘÍLOH

- P I CD s finální verzí hry, zdrojovými kódy, bakalářskou prací a ukázkami modelů a textur
- P II Adresářová struktura hry
- P III Doprovozné obrázky a screenshoty z finální verze hry

PŘÍLOHA P I: CD S POSLEDNÍ VERZÍ HRY, ZDROJOVÝMI KÓDY, BAKALÁŘSKOU PRACÍ A UKÁZKAMI MODELŮ A TEXTUR

- /Práce: text bakalářské práce ve formátu pdf
- /Frodu: výsledná spustitelná hra s přiloženými knihovnamí (spustitelný soubor project_frodu.exe)
- /Zdrojové kódy: všechny zdrojové kódy práce
- /Návod: návod na hraní přiložené hry ve formátu pdf
- /Modely: modely objektů ve hře ve formátu blend

PŘÍLOHA P II: ADRESÁŘOVÁ STRUKTURA HRY

- /dialogues: xml soubory se sepsanými rozhovory
- /irr_meshes: modely ve formátu irrmesh (pro potřeby IrrEditu)
- /levels: jednotlivé úrovně hry (formáty irr)
- /models: použité modely (formáty 3ds a md2)
- /music: archiv hudby (formát mp3)
- /pictures: 2D grafika do hry (různé obrázky a podobně)
- /sounds: herní zvuky a efekty (formát wav)
- /textures: textury do hry (formáty png a jpg)

PŘÍLOHA P III: DOPROVODNÉ OBRÁZKY A SCREENSHOTY Z FINÁLNÍ VERZE HRY



