Irrlicht engine
Grafické rozhranie a práca zo súbormi

Irrlicht engine
Graphic interface and file operations

Jaroslav Coufalík

Bakalářská práce
2008

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

*** nascannované zadání str. 1 ***

*** nascannované zadání str. 2 **

ABSTRAKT

Cieľom mojej práce je podrobne sa zoznámiť s Irrlicht enginom a zamerať sa najmä na objekty a funkcie grafického rozhrania a prácu so súbormi. V praktickej časti potom vytvoriť komplexnejší program v spolupráci so študentom, ktorý rieši téma „Irrlicht engine – podpora grafických technológií a management scény.“ Navrhnutý program bude napísaný v C/C++.

Klíčová slova: Irrlicht engine, graficé rozhranie, práca so súbormi

ABSTRACT

My goal of this bachelor thesis is closely execute with Irrlicht engine and focus mainly on objects and functions of graphical interface and work with files. In practical part create a complex program cooperate with student who solve topic „Irrlicht engine – graphic technology support and management scene“. The program will be written in C/C++.

Keywords: Irrlicht engine, file operations

Chcel by som poďakovať Ing. Pavlu Pokornému, PhD. za vedenie pri tvorbe práce.

Ďakujem za pripomienky, ochotu a čas, ktorý mi venoval.

Prohlašuji, že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků, je-li to uvolněno na základě licenční smlouvy, budu uveden jako spoluautor.

Ve Zlíně …….

Podpis diplomanta

OBSAH

ÚVOD

V dnešnej dobe sa človek stále častejšie stretáva s počítačovými technológiami, ktoré sa veľmi rýchlo vyvíjajú a počítačová grafika zaznamenala v poslednej dobre veľký pokrok. Ten bol spôsobený najmä novým hardwarom, v podobe grafických akcelerátorov. Začali sa stále viac objavovať programi využívajúce 3D grafiku. V dnešnej dobe už dokážeme vytvárať vskutku realistické simulácie.

Základom každej aplikácie, ktorá pracuje s priestorovou grafikou, je nejaký grafický engine. Grafický engine je vývojová knižnica definujúca dané aplikačné rozhranie, ktoré je určené pre vývoj grafických programov. Dnes už existuje nespočetne veľa enginov, ktoré sú buď komplexnejšieho charakteru, alebo sa zameriavajú len na určitú oblasť. Newton engine sa napríklad spoločne s ODE (Open Dynamics Engine) orientujú na fyziku, OGRE sa zameriava na 3D vykreslovanie grafiky spolu Irrlicht enginom. [8]

Zo stálym zlepšovaním grafických akcelerátorov vzrastá kvalita grafických efektov, vznikajú stále nové a lepšie efekty a algoritmy. Jednou zo základných vlastností enginu, je možnosť neustáleho rozširovania o nové a nové algoritmy. Tým je zaistený neustály vývoj a rozširovanie enginov. Existuje veľa firiem zameraných na vývoj 3D enginov a ich distribúciou.

Irrlicht engine je jedným z predstaviteľov voľne šíriteľných 3D enginov. Spolu s ďalšími prídavnými programami tvorí ucelený systém pre tvorbu aplikácií.

I. TEORETICKÁ ČASŤ

1 IRRLICHT ENGINE

Irrlicht engine je rýchly a mocný engine, pomocou ktorého je možné veľmi ľahko a rýchlo vytvoriť 2D a 3D hry, je efektívny pri vytváraní akýchkoľvek grafických aplikácií. Najväčšou výhodou Irrlicht enginu je, že je to Open source, teda úplne zadarmo a ponúka mnoho ďalších programov, ktoré spolupracujú s enginom a sú tiež Open source (IrrEdit). Obsahuje vlastné grafické užívateľské rozhranie, ktoré je možné upravovať do podoby podľa vlastných predstáv. Nechýbajú mu funkcie pre prácu so súbormi - vstup a výstup zo súborov. Dokáže importovať z nezpočetne veľa grafických formátov a umožňuje ďalej prácu s nimi a dokáže ich načítať priamo za behu.

Irrlicht využíva Direct3D, OpenGL render. V irrlichte sú implementované aj vlastné softwarové rendery. Jeden je orientovaný na rýchlosť a druhý na kvalitu. Oba sú pritom absolútne nezávislé na platforme a grafických ovládačoch. Jednou z najväčších výhod pri tom je, že programátor vôbec nemusí vedieť aké API bude použité, pretože je Irrlicht absolútne abstraktným enginom.

Podporuje veľké množstvo efektov, s ktorými nie je pozadu a neustále le rozširovaný o stále nové. Ich použitie je pritom jednoduché, stačí ak ich programátor jednoducho zapne, alebo vypne. Medzi implementované efekty patrí napr.: realistický vodný povrch, skybox, hmla, priesvitné objekty, bump mapping, dynamické svetlá, animácia textúr a iné.

Programátorov určite poteší rýchla a jednoduchá implementovaná detekcia kolízií, optimalizovaná matematika a priame čítanie z komprimovaných súborov (.zip). Umožňuje pŕacu v Microsoft Visual Studio, Dev-C++, Codewarrior, Code::Blocks, a ďalšimi. Je napísaný v jazyku c++ a je dostupný pre .NET jazyky, ako sú napr. C#, VisualBasic a Delphi.NET, a použiteľný na viacerých platformách Windows 98 – Vista (vrátane 64bitových verzií), Linux, MacOS, Sun Solaris/SPARC [1].

1.1 Zoznam názvosloví Irrlichtu

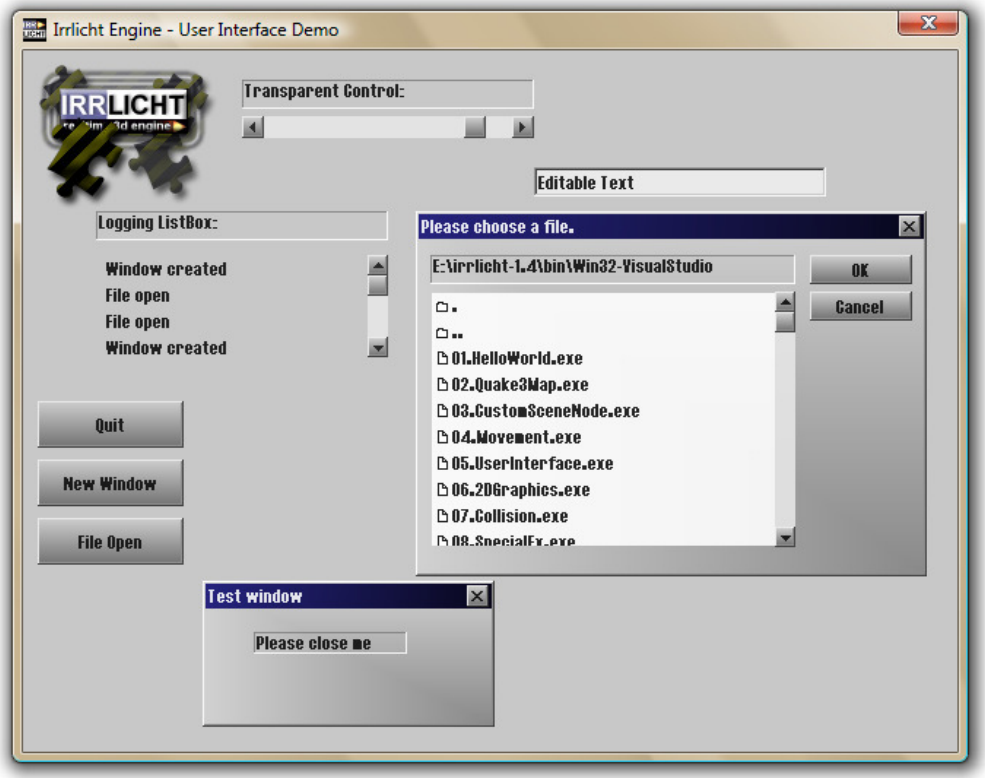
Štruktúra Irrlichtu sa skladá z niekoľkých častí, ktoré majú vlastné názvoslovie a zameriavajú sa na určitú oblasť enginu[12].

- **irr** – je základné názvoslovie, pod ktoré patrí všetko, čo sa týka enginu
- **irr::core** – pod týmto názvoslovím nájdeme všetky základné triedy ako vektory, polia, zoznamy a podobne.
- **irr::gui** – tu sa nachádzajú triedy na vytvorenie grafického užívateľského rozhrania
- **irr::io** – názvoslovie poskytujúce rozhranie pre vstup a výstup
- **irr::scene** – celé riadenie scény sa tu nachádza
- **irr::video** – obsahuje triedy pre prácu s video driverom

2 IRR::GUI - NÁZVOSLOVIE PRE GRAFICKÉ ROZHRANIE

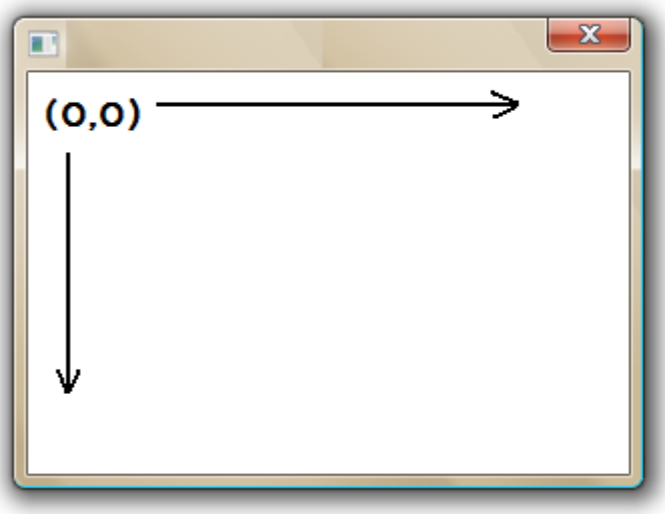
Grafické užívateľské rozhranie je rozhranie, ktoré umožňuje užívateľovi ovládať počítač. Ide vlastne o druh komunikácie medzi počítačom a človekom. V dnešnej podobe sa najčastejšie vyskytuje v podobe okien a tlačítok a iných interaktívnych prvkov, ktoré slúžia na zobrazovanie informácií alebo ako vstupy od užívateľa. Väčšinou sú umiestnené v dvoj rozmernom priestore obrazovky [7].

V hrách sa toto rozhranie používa najmä na vytvorenie menu a na niektoré časti ovládania hry.



Obrázek 1: Ukážka Irrlicht GUI rozhrania

Irrlicht Engine ponúka veľmi efektívnu prácu s prvkami z GUI. Vytváranie je ľahké, rýchle a ponúka veľa možností úprav. Je dôležité poznať súradnicový systém pre GUI než sa značnú popisovať jednotlivé prvky. V Irrlichte je rovnaký ako pre Windows. Počiatok oboch súradníc sa teda nachádza v pravom hornom rohu obrazovky[10].



Obrázek 2: Súradnicový systém pre irr::gui

2.1 IGUIEnvironment –trieda GUI prostredia

Základom celého grafického rozhrania v irrlichte je trieda irr::gui::IGUIEnvironment. Tá funguje ako správca GUI v irrlichte. Prostredníctvom tejto triedy môžeme vykresliť jednotlivé prvky grafického rozhrania. Funkcia dokáže GUI prvky načítať, vymazať, priradiť nový skin, font atď[12].

Niektoré metódy triedy:

- addButton(...) – pridá tlačítko, ako parametre nastavujeme vlastnosti tlačítka
- addWindow(...) – pridá okno, parmetre určujú vlastnosti okna
- addImage(...) – vložíme obrázok do prostredia
- addGUIElement(...) – pridáme prvok z GUI
- addMessageBox(...) – vloží okno správ
- addListBox(...) – pridá prvok List box
- hasFocus(IGUIElement* element) – vráti, či je prvok zameraný
- removeFocus(IGUIElement* element) – odstráni zameranie z prvku
- clear() – vymaže všetky tlačítka z prostredia
- drawAll() – vykreslí všetky prvky GUI
- setSkin(IGUISkin* skin) – vráti pointer na aktívny skin
- getVideoDriver() – vráti pointer aktuálneho video driveru
- getRootElement() – získame ukazateľ na hlavný prvok

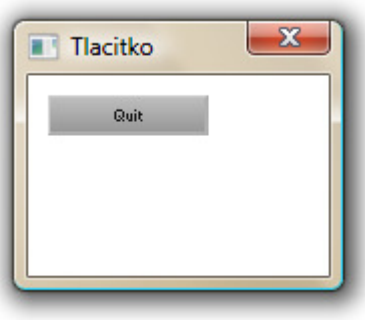
2.2 IGUIElement – základný prvok GUI rozhrania

Trieda irr::gui::IGUIElement reprezentuje základný prvok GUI rozhrania, z ktoŕeho vychádzajú všetky ostatné prvky rozhrania a dedia všetky jeho vlastnosti. Môžeme ho pokladať za rodiča všetkých GUI prvkov v Irrlichte [12].

Niektoré metódy triedy irr::IGUIElement:

- `addChild(IGUIElement* child)` – pridá prvku dieťa
- `removeChild(IGUIElement* child)` – vymaže dieťa
- `draw()` – vykreslí prvok a všetky jeho deti
- `setID()` a `getID()`- nastaví a vráti ID daného prvku
- `setEnabled(bool enable)` – nastaví, či je prvok aktívny
- `setRelativePosition(core::rect<s32>)` – nastaví relatívnu pozíciu. Relatívna pozícia znamená pozíciu vzhľadom k jeho rodiča
- `setVisible(bool visible)` – určuje, či je prvok viditeľný alebo nie

2.3 IGUIButton - tlačítko



Obrázek 3: Ukážka tlačítka

Pri vytváraní tlačítka začneme vytvorením ukazateľa a následne jeho vložením do triedy `irr::gui::IGUIEnvironment`, ktorá ho v sebe uchová a neskôr vykreslí [10].

Vytvorenie tlačítka:

```
IGUIButton* tlacitko;
Tlacitko = environment->addButton(const core::rect<s32>&
                                  rectangle,
                                  IGUIElement* parent,
                                  s32 id,
                                  const wchar_t* text
                                  )
```

Parametre funkcie irr::gui::IGUIEnvironment::addButton():

- `core::rect<32>(x1,y1,x2,y2)` – je obdĺžnik, ktorý určuje tvar a umiestnenie tlačítka. Súradnice x1 a y2 nám pri tom udávajú ľavý horný roh tohto obdĺžnika a x2 y2 pravý dolný roh
- `IGUIElement* parent` – je parameter, do ktorého predávame ukazateľ rodičovského prvku
- `s32 id` – každý prvok má identifikačné číslo
- `const wchat_t* text` – je text na tlačítku

Tlačítku môžeme ďalej pridať do pozadia ľubovoľný obrázok pomocou metódy `setImage()`, môžeme nastaviť, či sa má vykresliť rám tlačítka, na to nám slúži metóda `setDrawBorder()` alebo mu priradiť obrázok, ktorý sa vykreslí pri jeho stlačení – `setPressedImage()`.

Užitočnou metódou tejto triedy je metóda `setUseAlphaChannel()`, pomocou ktorej nastavujeme tlačítku alfa kanál. Teda, či tlačítko bude používať priesvitnú farbu alebo nie.

```
tlacitko->setImage(video::ITexture* image);
tlacitko->setDrawBorder(false);
tlacitko->setUseAplhaChannel(true);
```

2.4 IGUIStaticText - statický text

Obrázek 4: Ukážka statického textu

Na zobrazenie statického textu slúži trieda `irr::gui::IGUIStaticText`. Farbu textu nastavíme pomocou metódou:

- `serOverrideColor(video::SColor color)`

a farbu pozadia:

- `setBackgroundColor(video::SColor color).`

Triedu `irr::video::SColor` môžeme definovať konstruktorom:

- `SColor::SColor(s32 alpha, s32 red, s32 green, s32 blue).`

V `irr::gui` nie je problém taktiež priradiť textu vlastný font, ktorý si vytvoríme a použiť pre každý statický text iný font. To umožňuje metóda:

- `setOverrideFont(IGUIFont* font).`

Font musí byť uložený ako obrázok. Pre prevod z bežného fontu môžeme použiť program, ktorý je priložený k enginu („IrrFontTool“).

Ďalej sa dá podľa potreby text zarovnávať podľa okrajov štvorca, ktorý sme zadefinovali pri vytváraní statického textu.

Kód pre vytvorenie statického textu:

```
IGUIStaticText* text;
  text = env->addStaticText(L"Hello World! This is a static
               text!",rect<int>(10,10,200,50), true, true,
               0, 123, True);
  text->setOverrideFont(font);
  text->setBackgroundColor(video::SColor(128,0,0,255));
  text->setOverrideColor(video::SColor(255,255,255,0));
  text->setTextAlignment(EGUIA_CENTER, EGUIA_CENTER);
```

Parametre metódy `irr::gui::IGUIEnvironment::addStaticText()`:

- `const wchar_t* text` – je text, ktorý chceme vykresliť
- `const core::rect<s32>& rectangle` – obdĺžnik, v ktorom sa bude text nachádzať
- `bool border` – nastavuje, či bude mať text 3D ohraničenie
- `bool wordWrap` – povoľuje multiline, teda viac riadkový text
- `IGUIElement* parent` – určuje rodiča
- `s32 id` – je id číslo prvku
- `bool fillBackground` – určuje, či bude pozadie vyplnené

2.5 IGUIImage – obrázok

Obrázek 5: Ukážka obrázku

Obrázok má oproti predchádzajúcim prvkom jeden zásadný rozdiel pri vytváraní, a to, že jeho pozíciu nastavíme pomocou ľavého horného roh, a nie definovaním obdĺžnika. Jeho šírka a dĺžka je potom závislá od rozmerov textúry.

Medzi podporovanými vlastnosťami obrázku je priehľadnosť, ktorá sa dá buď povoliť alebo zakázať metódou `setUseAplhaChannel(bool use)`. Pri vytváraní obrázku máme na výber. Buď priradíme obrázku textúru hneď pri pridávaní do environmentu, alebo ju môžeme priradiť neskôr pomocou metódy `setImage(video::ITexture* image)`.

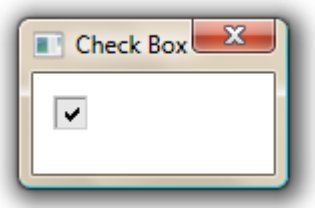
Parametre metódy `irr::gui::IGUIEnvironment::addImage()`:

- `video::ITexture *image` – textúra obrázku
- `core::position2d<s32>` - udáva pozíciu pravého horného rohu obrázku

Kód pre vytovrenie obrázku:

```
IGUIImage *image;
Image = env->addImage(driver->getTexture("irrlichtlogo2.png"),
                      core::position2d<s32>(80,50),
                      bool useAlphaChannel = true,
                      IGUIElemnent* parent,
                      s32 id
                      );
imgIrrlichtLogo->setScaleImage(true);
```

2.6 IGUICheckBox – zaškrtávacie políčko



Obrázek 6: Check Box

Trieda `irr::gui::ICheckBox` má dve dôležité metódy a to `isChecked()`, ktorá vracia booleanovskú premennú, či je Chceck box zaškrtnutý alebo nie. Druhou metódou je `setChecked(bool chcecked)`, ktorá nastavuje, či sa má zobraziť zaškrtnuté alebo prázdne [12].

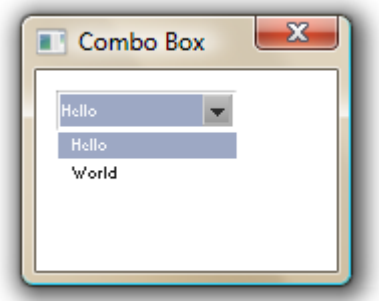
Kód pre vytvorenie Check boxu:

```
IGUICheckBox *checkbox;
chceckbox = environment->addChceckBox(false,
          core::rect<s32>(10,10,30,30), 0, 122);
```

Parametre metódy `irr::gui::IGUIEnvironment::addCheckBox()`:

- `bool checked` – parameter, ktorý určuje, či bude pri vytvorení políčko zaškrtnuté
- `const core::rect<s32>& rectangle` – obdĺžnik chceck boxu
- `IGUIElement *parent` – pointer na rodiča
- `s32 id` – ID prvku

2.7 IGUIComboBox – rozbalovací zoznam



Obrázek 7: Combo box

Na výber zo zoznamu nám slúži combo box, ktorý v irr::gui samozrejme nechýba a nájdeme ho pod triedou `irr::gui::IGUIComboBox`. Do zoznamu môžeme pridávať, mazať, získať označený prvok a nastaviť, ktorý sa ako označený ukáže.

Do zoznamu pridáme prvok pomocou metódy `addItem(const wchar_t* text)`. Vstupným parametrom je text, ktorý chceme pridať do zoznamu. Prvky sa pridávajú za sebou podľa poradia pridávania a zároveň sa im priraďuje id postupne od 0.

Zoznam vymažeme metódou `clear()`. Ak ale chcem vymazať len jeden konkrétny prvok, tak použijeme metódu `removeItem(s32 id)`, kde ako vstupný parameter uvedieme ID prvku.

Označený prvok získame funkciou `getSelected()`, ktorá vráti ID označeného prvku a v prípade, že žiadny prvok nebol označený, tak vráti hodnotu -1. Ak chceme zistiť aký prvok sa nachádza na nejakej pozícii, použijeme metódu `getItem(s32 id)`, pričom celkový počet prvkov zoznamu nám vráti funkcia `getItemCount()`.

Trieda nám umožňuje určiť, ktorý prvok bude označený pri vytvorení combo boxu a to pomocou funkcie `setSelected(s32 id)` a do parametru predáme ID zvoleného prvku [12].

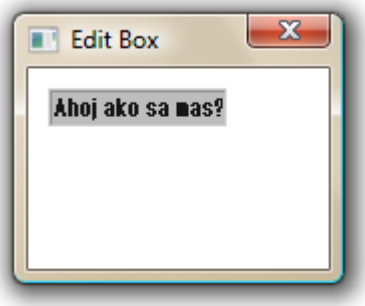
Parametre funkcie `irr::gui::IGUIEnvironment::addComboBox()`:

- `const core::rect<s32>` - prvým parametrom je obdĺžnik, v ktorý bude zobrazovať prvky zo zonamu
- `IGUIElement *parent` – do druhého parametru predáme pointer na rodiča, v prípade. že nechceme, aby mal rodiča, predáme 0
- `s32 id` – identifikačné číslo prvku

Kód pre vytvorenie Combo boxu:

```
IGUIComboBox *combo;
combo = environment-
>addComboBox(core::rect<s32>(10,10,100,40), 0, 143)
combo->addItem(L"Hello");
combo->getSelected();
```

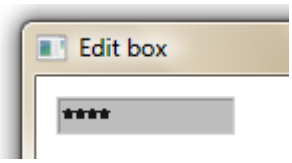
2.8 IGUIEditBox – pole pre editovanie textu



Obrázek 8: Edit box

Asi najzákladnejším prvkom každého GUI je výpis a editácia textu na obrazovke. `Irr::gui` ponúka v triede `IGUIEditBox` možnosť vykresliť text, ktorému môžeme nastaviť pomocou metódy `setDrawBorder(bool border)` vykreslenie 3D obĺžnika. Edit box podporuje skratky ako ctrl+X, ctrl+V, ctrl+C, shift+Left, shift+Right, Home, End a ďalšie [10].

Medzi vlastnosti Edit boxu patrí „PasswordBox“ (políčko pre heslo), na to aby sme ho ale mohli aktivovať funkciou `setPasswordBox(bool passwdBox, wchat_t passwdChar=L"*")`, musíme vypnúť funkciou `setMultiLine(bool enable)` vlastnosť viac riadkov a taktiež ešte rozdeľovanie slov metódou `setWordWrap(bool enable)`. V druhom parametre metódy `setPasswdBox()` nastavujeme znak, ktorý chceme vypisovať namiesto zobrazovania písmen v Edit boxe pri písaní.



Obrázek 9: PasswordBox

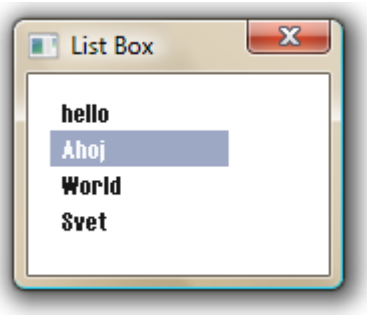
Edit boxu môžeme určiť koľko znakov sa bude dať do neho napísať metódou `setMax()`, textu priraďiť farbu, font, zarovnávanie a ďalšie funkcie na prácu s textom. Parametre metódy `irr::gui::IGUIEnvironment::addEditBox()`:

- `const wchar_t* text` – text, ktorý sa zobrazí v Edit boxe
- `const core::rect<s32>& rectangle` – obdĺžnik ohraničujúci text
- `bool border` – nastavuje 3D obĺžnik
- `IGUIElement *parent` – pointer na rodiča
- `s32 id` – identifikačné číslo

Ukážka vytvorenia Password Boxu:

```
IGUIEditBox *editbox;
editbox = env->addEditBox(L"Ahoj",
core::rect<s32>(10,10,100,30), true, 0, 123);
editbox->setMultiLine(false);
editbox->setWordWrap(false);
editbox->setPasswordBox(true, L'*');
```

2.9 IGUIListBox – zoznam textu



Obrázek 10: List Box

Ďalší z prvkov, ktorý pracuje s textom je List box. Je podobný ako combo box a má aj podobné vlastnosti. Textu list boxu môžeme meniť font, farbu, nastaviť, či sa má vykresľovať 3D pozadie atd.

Text do zoznamu pridáme metódou `addItem(const wchat_t *text)` a metódou `removeItem(s32 id)` ho odtiaľ zasa odstrániť. Ak chceme odstrániť naraz všetky prvky zoznamu, použijeme funkciu `remove()`.

Kód pre vytvorenie list boxu:

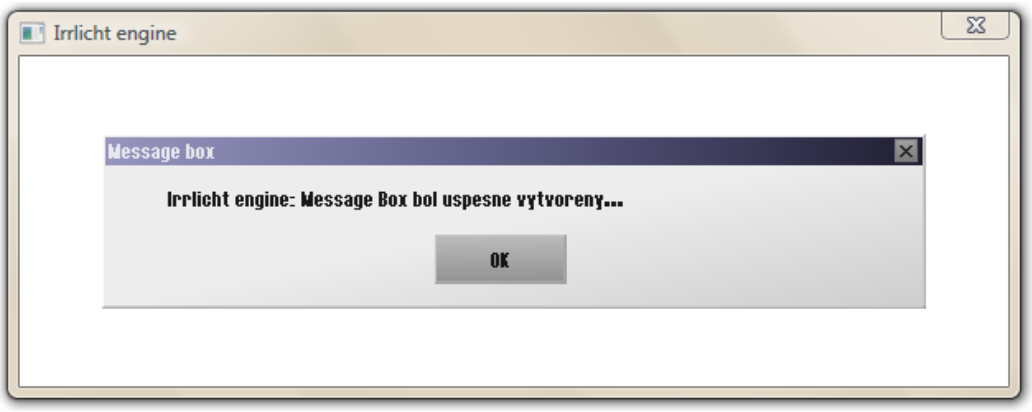
```
IGUIListBox *listbox;
listbox = env->addListBox(core::rect<s32>(10,10,100,100), 0,
123, true);
listbox->addItem(L"hello");
listbox->addItem(L"Ahoj");
```

Označený text získame pomocou metódy `getSelected()`, ktorá nám vráti index označeného prvku a tak isto nastavujeme aj text, ktorý chceme, aby bol označený, metódou `setSelected(s32 id)` [10].

2.10 IGUIWindow - Okno správ

Okno správ (Message Box) som zámerne oddelil od obyčajného okna, ktoré som priradil k Tab a Tab Control, pretože okno podľa mňa patrí skôr k Tab na zlučovanie rôznych prvkov GUI.

Okno správ je, ako vidíme na obrázku, dobrá metóda, ako zobrazovať užívateľovi dôležité správy, napríklad chyby a pod [12].



Obrázek 11: Okno správ

Vytvorenie je veľmi jednoduché, stačí len zavolať metódu triedy `irr::gui::IGUIEnvironment`:

- `irr::gui::IGUIEnvironment::addMessageBox(const wchar_t *caption, const wchar_t *text, bool modal, s32 flags, IGUIElement *parent, s32 id).`

Parametre funkcie:

- `const wchar_t *caption` – je text zobrazený v záhlaví okna
- `const wchar_t *text` – je text zobrazený v okne
- `bool modal` – nastaví. či bude možné manipulovať s ostatnými prvkami GUI, kým bude zobrazené okno spŕav
- `s32 flag` – je enum, `EMESSAGE_BOX_FLAG` ktorý určuje, aké tlačítko chceme pridať do okna správ
- `IGUIElement* parent` – rodič
- `s32 id` – ID prvku

Typ enum `EMESSAGE_BOX_FLAG` obsahuje:

- `EMBF_OK`
- `EMBF_CANCEL`
- `EMBF_YES`
- `EMBF_NO`

Kód pre vytvorenie okna správ:

```
IGUIWindow *msgBox;
msgBox = env->addMessageBox(L"Message box", L"Irrlicht engine:
Message Box bol uspesne vytvoreny...", true, EMBF_OK, 0, 123);
```

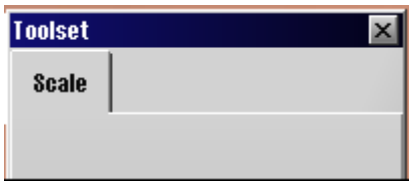
2.11 IGUIWindow, IGUITabontrol, Tab - Okno, Tab a Tab control

Obrázek 12: Samostatné okno

Kód pre vytvorenie okna:

```
IGUIWindow *okno;
okno = env->addWindow(rect<s32>(10,10,300,200),
false,L"Okno",0,5000);
```

Okno, spoločne s Tab nám slúži na zlučovanie jednotlivých prvkov GUI a umožňuje tak spoločnú manipuláciu s vybranými prvkami. Po vytvorení okna, vytvoríme Tab Control, do ktorého potom môžeme vložiť viac Tabov, ktoré budú obsahovať vybrané prvky. Prepínať medzi prvkami nám dovoľujú záložky, ktoré sa vytvoria spolu s Tabom [11].

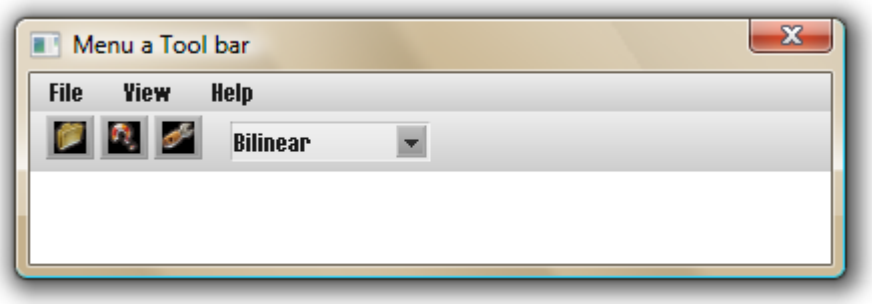


Obrázek 13: TabControl, Tab a Okno

Ukážka kódu na vytvorenie Tab a umiestnenie do Tab controlu:

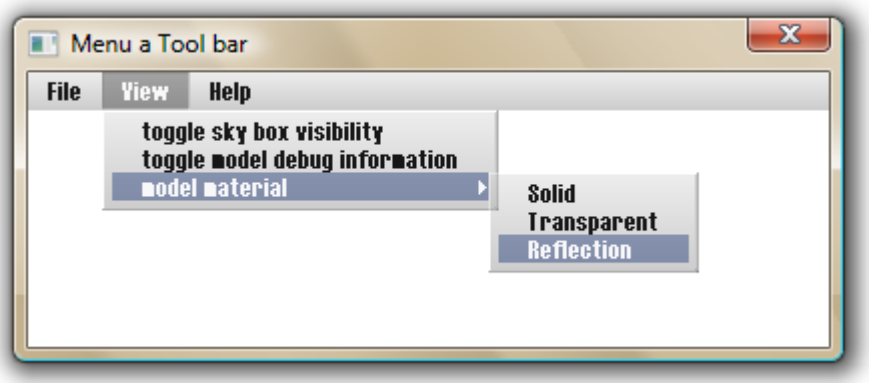
```
IGUITabControl* tblc = env-
>addTabControl(rect<s32>(0,30,100,10),
                        okno,false,false);
IGUITab* tb1 = tblc->addTab(L"Scale");
```

2.12 IGUIMenu, IGUIToolBar - Menu a Tool bar



Obrázek 14: Menu a Tool bar

Menu vytvoríme metódou `irr::gui::IGUIEnvironment::addMenu()`, ktorej nemusíme predať žiadny parameter. Inak má iba dva parametre, a to pointer na rodiča, a ID. Ak nepredáme rodiča, tak sa menu vytvorí priamo v environmente a pridá sa na horný okraj obrazovky. Prvky do menu sa pridávajú taktiež veľmi jednoducho, stačí zavolať metódu `IGUIContextMenu::addItem()`, tej predáme ako parametre názov, a boolean premennú, či bude mať submenu alebo nie. Prvkom menu sa priraďuje identifikačné číslo od 0. Podľa neho potom rozoznávame prvky menu a môžeme im neskôr priradiť ďalšie prvky [11].



Obrázek 15: Menu a submenu

Toolbar sa vykreslí po vytvorení v hornom okraji svojho rodiča, takže ak nepredáme Toolbaru žiadneho, vykreslí sa pod vytvorené menu, ako môžeme vidieť na obrázku 24. Metódou `addToolBar()` predáme prvok triede `irr::gui::IGUIEnvironmet`. Tool bar má v podstate jedinú vlastnú metódu a tou je `addButton()`, ktorej do parametrov predávame obrázok, ID, a alfa kanál.

2.13 irr::IEventReceiver - Správca udalostí

Irrlicht engine obsahuje aj správcu udalostí tzv. event receiver, ktorý sa stará o obsluhu udalostí v GUI. Programátor si ho môže upraviť podľa vlastných predstáv. Celá obsluha sa riadi v metóde `OnEvent(const SEvent& event)`. Tá sa dokáže reagovať na udalosti, ktoré nastanú. Jednotlivé udalosti rozoznáva pomocou enum typu `EEVENT_TYPE` [12].
Ten obsahuje:

- `EET_GUI_EVENT` – udalosť spôsobená prvkami z GUI
- `EET_MOUSE_INPUT_EVENT` – udalosť spôsobená myšou
- `EET_KEY_INPUT_EVENT` – udalosť spôsobená stlačením klávesnice
- `EET_LOG_TEXT_EVENT`
- `EET_USER_EVENT`

Udalosti spôsobené prvkami z GUI sú obsiahnuté v enum `type EGUI_EVENT_TYPE`.
Prvky z enumu `irr::gui::EGUI_EVENT_TYPE`:

- `EGET_BUTTON_CLICKED` – stlačenie tlačítka
- `EGET_LISTBOX_CHANGED` – zmena v list boxe
- `EGET_MENU_ITEM_SELECTED` – bol vybraný prvok z menu
- `EGET_ELEMENT_FOCUSED` – prvok je zameraný

Aby nám správca udalostí mohol reagovať na vzniknuté udalosti, musíme jeho pointer predať metódou `irr::IrrlichtDevice->setEventReceiver(receiver)`.

3 PRÁCA ZO SÚBORMI

Irrlicht engine podporuje množstvo formátov, ktoré môžu byť nahrané do enginu priamo za behu. Pritom nie je problém doprogramovať ďalší loader a použiť ho bez rekompilácie enginu.

3.1 Podporované textúry

Textúry sa používajú na „obalenie“ modelov, dávajú im konečný vzhľad a upravujú ich do reálnejšej podoby. Pre vytvorenie a úpravu textúr existuje nekonečne veľa aplikácií, napr.: GIMP, Adobe Photoshop, Malování vo Windowse, a mnoho ďalších a používajú ich všetky 3D programy, ktoré vytvárajú nejaké modely. [1]

Medzi podporované formáty textúr patria:

- Adobe Photoshop (.psd)
- JPEG File Interchange Format (.jpg)
- Portable Network Graphics (.png)
- Truevision Targa (.tga)
- Windows Bitmap (.bmp)
- Zsoft Paintbrush (.pcx)

3.1.1 JPEG (.jpg)

JPEG je formát súboru, ktorý sa používa najmä na uchovávanie počítačových obrázkov vo fotorealistickej kvalite s hladkými prechodmi v tóne a farbe. Používa pritom metódu strátovej kompresie, ktorá funguje v tomto prípade oveľa efektívnejšie ako niektoré bezstrátové metódy, pričom poskytuje stále veľmi dobrú kvalitu a vysokú úroveň kompresie.[13]

3.1.2 Portable Network Graphics (.png)

Je to grafický formát, určený pre bezstrátovú kompresiu rastrovej grafiky. Ponúka podporu 24 bitovej farebnej hĺbky. Naviac obsahuje osem bitovú priehľadnosť, nazývaný tiež alfa kanál. Z toho vyplýva, že obrázok môže byť v rôznych miestach inak priesvitný. Nevýhodou oproti JPEG je väčšia veľkosť.[14]

3.1.3 Načítanie textúr

Na načítanie textúr slúži funkcia triedy `irr::video::IVideoDriver`:

- `getTexture(const c8* filename)`

Táto funkcia vracia pointer na vloženú textúru.

3.2 Podporované 3D modely

Objekty vložené do scény sú vytvorené pomocou modelov, ktoré sú vymodelované v externých programov špecializovaných na modelovanie 3D objektov. V dnešnej dobe existuje veľa takýchto programov, niektoré sú komerčné, iné sú uvoľnené ako Open source. Každá aplikácia pritom preferuje iný formát, ktorý jej najviac vyhovuje, väčšinou však má vytvorený formát vlastný. Irrlicht obsahuje veľa loaderov, takže dokáže načítať veľkú časť týchto objektov.

Medzi podporované formáty 3D modelov patria:

- 3D Studio meshes (.3ds) -
- B3D files (.b3d)
- Alias Wavefront Maya (.obj)
- COLLADA (.xml, .dae)
- DeleD (.dmf)
- FSRad oct (.oct)
- Irrlicht scenes (.irr)
- Irrlicht static meshes (.irrmesh)
- Microsoft DirectX (.x) (binary & text)
- Milkshape (.ms3d)
- My3DTools 3 (.my3D)
- OGRE meshes (.mesh)
- Pulsar LMTools (.lmts)
- Quake 3 levels (.bsp)
- Quake 2 models (.md2)
- STL 3D files (.stl)

3.2.1 B3D formát súboru (.b3d) - súbor na uloženie modelu

B3D je formát súboru pre ukladanie modelov vytvorený pre engine Blitz3D. Dokáže uchovať kostrovú animáciu modelu. Uchováva odkaz na textúry modelu, ktoré si Irrlicht vďaka implementovanému loaderu dokáže sám nahrať.

3.2.2 Quake 3 levels (.bsp) a Quake 2 models (.md2)

Quake 3 je populárna hra od IDSoftware. Quake 3 mapy nie sú skutočne animované mapy, ale len veľká kopa statickej geometrie s pripojenými materiálmi. Pre nahratie mapy, sa musí najprv otvoriť .pk3 formát, ktorý v sebe obsahuje .bsp mapu. Ide vlastne o komprimovaný súbor. Quake 2 modely sú animované modely, ktoré sa používajú v hre Quake a je možné ich implementovať do Irrlichtu.

3.2.3 COLLADA (.xml, .dae)

COLLADA (Collaborate Design Activity) je formát založený na štruktúre XML, použiteľný pre výmenu dát, 3D scény medzi aplikáciami (MAYA, 3dsMax). COLLADA sú vlastne súbory XML, identifikované príponou .dae (digital asset exchange). Popisuje kompletne 3D scénu vrátane shaderov, fyzikálneho modelu scény, zvuky a podobne. Nevýhodou je pomalé načítanie dát. Veľkou výhodou je ale podpora veľkého počtu aplikácií[15].

3.2.4 DirectX

Je to binárny formát firmy Microsoft. Má implementovanú podporu načítania. Podporuje animácie. A je široko používaný v komerčných aj nekomerčných programov. (3ds Max, CInema 4D, Maya)[16].

3.2.5 Načítanie 3D modelov

Načítanie všetkých podporovaných formátov 3d modelov, bez ohľadu na ich typ formátu sa používa funkcia triedy `irr::scene::ISceneManager`:

- `getMesh(const c8 *filename)`

Táto funkcia vracia ukazateľ na animovateľný mesh. Ak by sme chceli vymazať a načítať mesh znovu, použijeme funkciu removeMesh();

Ukážka kódu na načítanie 3D modelu:

```
IAnimatedMesh* mesh = smgr->getMesh(
                              "../../media/sydney.md2");
```

4 IRR::IO – NÁZVOSLOVIE PRE VSTUP A VÝSTUP

Názvoslovie irr::io ponúka rozhranie pre vstup a výstup. Dokáže prečítať súbor, vytvoriť súbor, otvoriť archív, pracovať s xml a iné [12].

Triedy názvoslovia:

- IAttributeExchangingObject
- IAttributes – poskytuje všeobecné rozhranie pre atribúty a ich hodnoty a možnosť ich meniť
- IFileList – zoznam všetkých súborov v adresári
- IFileReadCallBack
- IFileSystem – zpráva súborov
- IIrrXMLReader – slúži na čítanie z XML súboru
- IReadFile – rozhranie pre prístup k súboru
- IWriteFile – rozhranie umožňujúce zapisovanie do súboru
- IXMLBase – prázdna trieda používaná ako otcovská pre IrrXMLReader triedu
- IXMLWriter – slúži na zapisovanie do XML súboru
- SAttributeReadWriteOptions

4.1 IFileSystem – zpráva súborov

Táto trieda spravuje súbory a umožňuje prístup k ním. Táto trieda sa využíva najmä pre načítanie skomprimovaných súborov. Umožňuje to skomprimovať vstupné súbory, ktoré bude program používať a tým zmenšiť aj konečnú veľkosť programu.

Niektoré metódy triedy:

- addZipFileArchive(…) – pridá zip archív do systému súborov
- createFileList () – vytvorí zoznam súborov a zložiek v aktívnom adresári a vráti ho
- createXMLReaderUTF8 (IReadFile *file) – vytvorí XML reader a vráti všetky načítané reťazce jako ASCII/UTF-8 znaky
- getFileDir (const core::stringc &filename) – vráti názov zložky v ktorej sa súbor nachádza
- getAbsolutePath (const core::stringc &filename) – zmení relatívnu adresu súboru na absolútnu adresu

4.2 XML

Je to obecný značkovací jazyk, ktorý umožňuje vytváranie konkrétnych značkovacích jazykov pre rôzne účely a široké spektrum dát. Je určený predovšetkým pre výmenu dát medzi aplikáciami. XML dokument je text, vždy Unicode, u nás zvyčajne kódovaný ako UTF-8, ale sú prístupné aj iné kódovania. Je silno závislý na štruktúre. Aby bol dokument považovaný za správne štruktúrovaný, musí mať najmenej nasledujúce vlastnosti:

- Musí mať práve jeden koreňový (root) element.
- Neprázdne elementy musia byť ohraničené štartovacou a ukončovacou značkou
- Prázdne elementy môžu byť označené tagom „prázdny element“
- Všetky hodnoty vlastností musia byť uzavreté v úvozovkách
- Elementy môžu byť vnorené, ale nesmú sa prekrývať

XML dokument sa skladá z uzlov a ich vlastností. Uzol môže byť buď párový alebo nepárový. Ďalej môže obsahovať komentáre a obyčajný text, ktorý je zvyčajne umiestnený vo vnútri párového ulzu. Dokumentu by nemala chýbať ani hlavička, ktorá nesie informáciu o verzie, prípadne kódovaní dokumentu [17].

Ukážka kódu XML súboru:

```
<?xml version="1.0" encoding="UTF-8"?>
<dokument>
     <titulok>Jednoduchý názov </titulok>
     <ovladac value=“1“ />
</dokument>
```

4.3 IrrXMLReader

Ako prvý je dobré spomenúť `irr::io::IrrXMLReader`, pretože sa častejšie používa čítanie, ako zapisovanie. Reader, čiže čítač, má všetky potrebné funkcie, pre prácu s XML, ako je napríklad, `getNodeName()`, ktorá vracia názov uzlu. Ďalej veľmi často používaná metóda `getAttributeValue()`. Tá vracia hodnotu vlastnosti uzlu. Návratová hodnota vlastnosti (attributu) je pri tejto metóde typ string. Ak však chceme, aby návratová hodnota bola iného typu, napr. float alebo int, máme na výber hneď z niekoľkých ďalších metód, pričom každá vracia hodnotu iného typu. Pre typ float existuje metóda `getAttributeValueAsFloat()`, pre Integer je to `getAttributeValueAsInt()`. Všetky tieto metódy majú jeden vstupný parameter, a tým je názov vlastnosti. Ak však

chceme použiť radšej ID, tak je možné miesto mena nahradiť indexom, ktorý začína 0. Počet vlastností prvku pritom zistíme použitím metódy `getAttributeCount()`, a meno konkrétnej vlastnosti `getAttributeName(s32 id)`.

V triede `irr::io::IrrXMLWriter` existuje ešte jedna metóda na získanie hodnoty, a tou je `getAttributeValueSafe(meno)`, ktorá vracia ako návratový typ `string` a ak vlastnosť uzlu neexistuje, tak vráti prázdny `string` („“). Text z XML dokáže prečítať metóda `getNodeData()` [12].

Najdôležitejšou metódou tejto triedy je určite metóda read(), ktorá prechádza celý XML súbor postupne element po elemnentu až po posledný. Keď príde nakoniec, vráti hodnotu `false`. Metódou `getNodeType()` dostaneme typ uzlu, na ktorom sa práve nachádzame. Typy uzlov sú definované v enum typu `irr::io::EXML_NODE` a obsahuje tieto prvky:

- `EXN_NONE` – žiaden uzol.
- `EXN_ELEMENT` – XML prvok ako `<foo>`.
- `EXN_ELEMENT_END` – ukončovací prvok `</foo>`.
- `EXN_TEXT` – text vo vnútri prvku: `<foo> text </foo>`.
- `EXN_COMMENT` – komentár: `<!-- komentár -->`.
- `EXN_CDATA` – XML cdata úsek: `<![CDATA[ nejaké CDATA ]]>`.
- `EXN_UNKNOWN` – neznámy prvok

Po ukončení práce s XML súborom, treba XML parser vymazať príkazom `drop()`. Časť kódu na vytvorenie XMLReaderu a jednoduchého cyklu na prechádzanie uzlov:

```
irr::io::IXMLReaderUTF8 *xml;
xml = Device->getFileSystem()->createXMLReaderUTF8(file);
while(m_xml && m_xml->read()){
      switch(m_xml->getNodeType()){
            case io::EXN_ELEMENT:
                  break;
      }
 }
```

4.4 IrrXMLWriter

Na zapisovanie do súborov XML nám slúži trieda `irr::io::IrrXMLWriter`. Pomocou nej dokážeme vytvoriť XML dokument veľmi efektívne.

XML uzol vytvoríme metódou `writeElement()`. Prvým parametrom je názov uzlu a za ním nasleduje booleanovská premenná, ktorou oznamujeme, či je prvok prázdny alebo nie, potom nasledujú jeho vlastnosti s ich hodnotami. Tie môžeme predať jednotlivo, alebo pomocou pola [12].

IrrXML ešte dokáže vytvoriť XML komentáre pomocou metódy `writeComment()`. Ako parameter predáme typ `const wchar_t`. Ďalej dokáže zapísať hlavičku (`writeXMLHeader()`), skok na ďalší riadok (`writeLineBreak()`), vypísať text (`writeText()`), a zatvárací element (`writeClosingTag()`) ako napr. </foo>.

Ukážka kódu na vytvorenie XML súboru:

```
irr::io::IXMLWriter xml;
xml = Device->getFileSystem()->createXMLWriter(file);
xml->writeXMLHeader();
xml->writeLineBreak ();
xml->writeElement(name.c_str(), true, attrName.c_str(),
attrValue.c_str());
```

II. PRAKTICKÁ ČASŤ

5 VYTVORENIE APLIKÁCIE S IRRLICHT ENGINOM

Praktická časť tejto práce sa skladá z vytvorení aplikácie v jazyku C/C++ za použití Irrlicht enginu s využitím poznatkov z teoretickej časti. So študentom, ktorý riešil tému – podpora grafických technológií a management scény, sme sa rozhodli, že vytvoríme jednoduchú 3D hru. Rozdelili sme si úlohy, podľa tém, ktoré sme spracovávali. Mojou úlohou teda bolo použiť GUI rozhranie a využiť poznatky pri práci so súbormi.

Názvoslovie irr::gui som použil na vytvorenie herného menu, kde sa dajú nastavovať vlastnosti hry, ako rozlíšenie. Irr::io som potom použil na prácu s XML, kde sme zo súborov tohto formátu načítavali dáta do hry.

5.1 Vytvorenie GUI prvkov menu

Pre menu som si vytvoril vlastnú triedu menu, ktorá sa stará o celé jeho fungovanie. Obsahuje funkcie pre vytvorenie jednotlivých prvkov. Tie som rozdelil na dve časti:

- Prvou sú prvky úvodnej obrazovky, ktoré obsahujú tlačíka „Play“, „Quit“, a „Settings“. Tieto prvky sa vykreslia pri spustení, ostanú vykreslené počas celého trvania menu a nemajú žiadny rodičovský prvok. Všetky patria elementu „m_tbMain“. Je to prvok typu `irr::gui::IGUITab`.
- Druhou časťou sú prvky nastavenia. Ich spoločným rodičom, je okno `IGUIWindow *m_wndSettings`. To sa zobrazí po stlačení tlačítka Settings a obsahuje ďalšie prvky, ktoré slúžia pre obsluhu nastavenia. Nastavuje sa iba rozlíšenie a bitová hĺbka obrazu. Ďalej okno informuje o ovládaní celej hry.

Jediným prvkom, ktorý nemá žiadneho rodiča, je obrázok v pozadí, pretože ten ostáva stále na svojom mieste, nemanipuluje sa s ním.

Pri vytváraní vlastného menu som použil tieto GUI prvky:

- `IGUIButton`
- `IGUIWindow`
- `IGUIControlTab`
- `IGUITab`
- `IGUIImage`

Obrázek 16: Ukážka menu

5.2 Prvky hlavnej časti menu

Prvky hlavnej časti menu patria spoločnému prvku. Je ním prvok IGUITab, ktorý ich spojuje a umožňuje tým lepšiu manipuláciu pri umiestňovaní na ploche. Najprv som si ale definoval premenné, ktoré určovali umiestnenie tohto Tab, skôr ako som ho vytvoril.

Kód nadefinovaných rozmerov:

```
s32 main_upper_x = mDevProps.screenWidth / 20;
s32 main_upper_y = mDevProps.screenHeight / 2;
s32 main_bottom_x = mDevProps.screenWidth / 4;
s32 main_bottom_y = ((mDevProps.screenHeight/4) *3);
```

Mojím zámerom bolo, aby tlačítka ostali v rovnakom umiestnení aj keď sa zmení rozlíšenie.

Kód na vytvorenie m_tbMain:

```
m_tbMain = m_env->addTab(core::rect<s32>(main upper_x,
                         main_upper_y, main_bottom_x,
                         main_bottom_y), 0);
```

5.2.1 Tlačítka

Tlačítka hlavnej časti menu sú vytvorené pomocou triedy `IGUIButton`. Pridaním do environmentu a predaním jeho ukazatela som vytvoril tlačítko „btnQuit“. Pri vytváraní som použil funkciu `addButton()`, ktorá patrí triede `irr::gui::IGUIEnvironment`. Nastavený obdĺžnik má nulovú veľkosť, pretože jej skutočné umiestnenie a veľkosť som prispôsobil podľa jeho rodičovského prvku, ktorým je „m_tbMain“.

Obrázek 17: Tlačítko

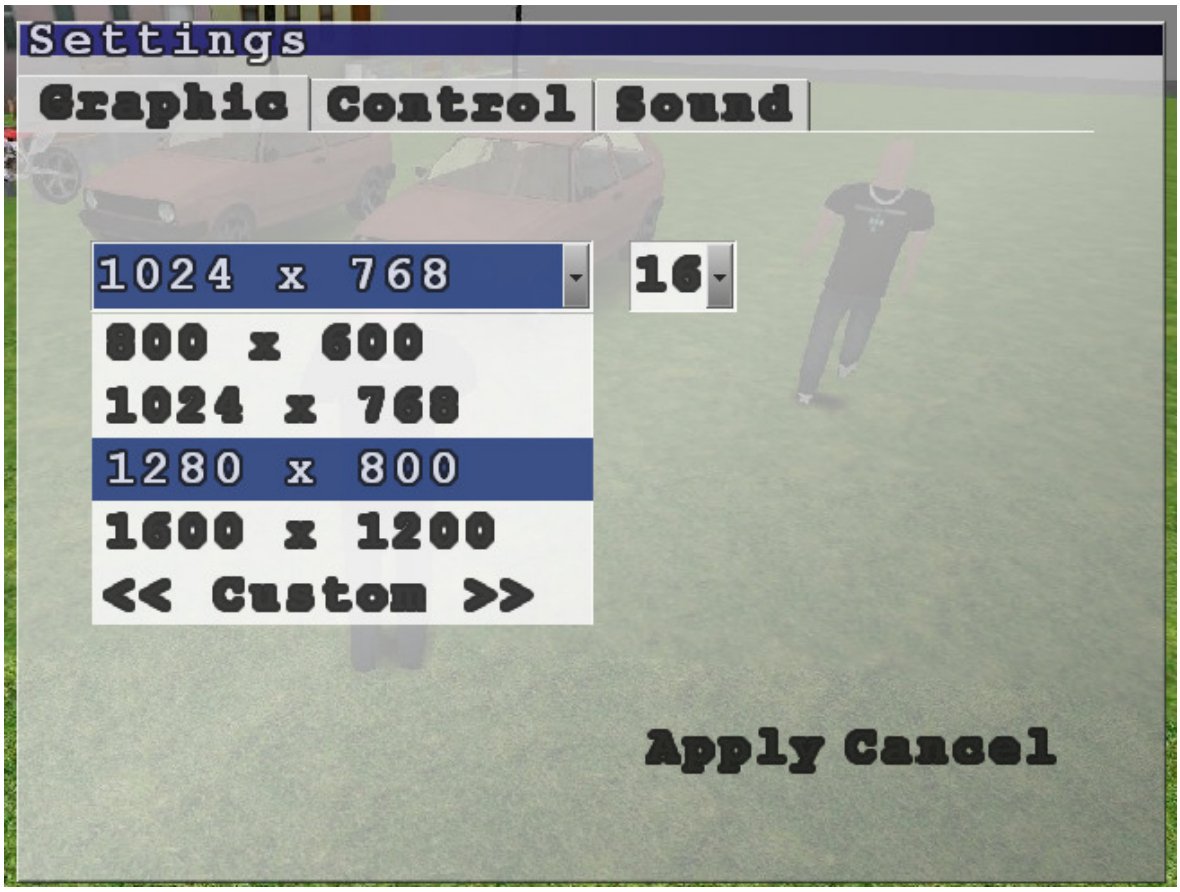
Na obrázku 31 vidíme jedno z tlačítok, konkrétne je to tlačítko „Start“. Má nastavené, aby používal alfa kanál a je mu pridaná relatívna pozícia vzhľadom na jeho rodiča. Ďalej má nastavené vykreslenie okraja na hodnotu false, čiže má zakázané vykresľovať okraj a pozadie tlačítka, preto tlačítko vyzerá ako nápis.

Obrázek 18: Ukážky tlačítok

Podobné vlastnosti majú aj ostatné dve tlačítka hlavnej časti menu. Ich funkcie sú popísané v kapitole 3.2 Funkcie menu.

5.2.2 Prvky časti nastavenia

Ako som už spomínal, všetky prvky druhej časti patria oknu m_wndSettings, vďaka ktorému môžu byť schované, pokým sa nestlačí tlačítko Settings a potom sa naraz ukázať.



Obr. 31 Okno časti nastavenie

Najprv som vytvoril okno vytvoril pomocou funkcie `addWindow()` okno, ktorému som nastavil rozmery 640 na 480. ID som mu priradil 5000, a názov som zvolil „Settings“.

V menu je viac Tabov, a na ich spájanie a kontrolu sa používa Tab Control, ten vytvorí záložky, pomocou ktorých je potom možné pohybovať medzi Tabmi. Tab Control vytvorím zavolaním metódy `IGUIEnvironment::addTabControl`. Potom jej pridáme do parametrov rozmery, pointer na rodiča, ktorým je v tomto prípade okno `m_wndSettings`. Ďalej som mu zakázal vykreslenie pozadia a 3D okraju. ID som v tomto prípade nevypĺňal, pretože s Tab Controlom nebudem ďalej manipulovať.

Vytvorenie TabControlu:

```
IGUITabControl* tblSettings = m_env;
m_env->addTabControl(rect<s32>(0,30,600,480), m_wndSettings,
                     false, false);
```

Do Tab Controlu patrí niekoľko Tabov, ktoré v sebe obsahujú prvky. Tab vytvoríme zavolaním metódy `irr::gui::IGUIEnvironment::addTab()`. Jediným parametrom v tomto prípade je len názov, ktorý sa potom ukáže na záložke.
Vytvorenie Tab:

```
IGUITab* tbGraphic = tblSettings->addTab(L"Graphic");
IGUITab* tbControl = tblSettings->addTab(L"Control");
```

Prvý Tab má názov tbGraphic, v záložke Tab Controlu pomenovaný ako „Graphic“ a spája dokopy dva prvky pre nastavenie rozlíšenia a bitovej hĺbky. Pre nastavenie rozlíšenia je použitý combo box, ktorý obsahuje 5prvkov. Combo box vytvoríme metódou `irr::gui::IGUIEnvironment::addComboBox()`. Jeho rodičom je nastavený Tab s názvom tbGraphic, ktorý sme predtým vytvorili. Jednotlivé prvky do combo boxu priradíme metódou `irr::gui::IGUIComboBox::addItem()` a ako parameter predáme názov. Každému názvu sa pri jeho vytvorní pridá ID.

Vytvorenie combo boxu:

```
m_cmbResolution = m_env->
        addComboBox(rect<s32>(40,60,320,100),
                    tbGraphic, 108);
  s = "800 x 600";    m_cmbResolution->addItem(s.c_str());
  s = "1024 x 768";   m_cmbResolution->addItem(s.c_str());
  s = "1280 x 800";   m_cmbResolution->addItem(s.c_str());
  s = "1600 x 1200";  m_cmbResolution->addItem(s.c_str());
  s = "<< Custom >>"; m_cmbResolution->addItem(s.c_str());
```

Na nastavenie bitovej hĺbky som použil taktiž combo box, ktorému som ale priradil iba dva prvky 16 a 32.

```
m_cmbBitperPixel = m_env->
   addComboBox(rect<s32>(340,60,400,100),
                         tbGraphic, 111);
  s = "16";   m_cmbBitperPixel->addItem(s.c_str());
  s = "32";   m_cmbBitperPixel->addItem(s.c_str());
```

6 NAČÍTANIE A ULOŽENIE DÁT DO XML

Pretože je práca s dátami na pomocou XML súborov veľmi efektívna, vybraným dátam, ktoré bolo potrebné uložiť na disk a taktiež ich rýchlo z disku získať, sme vytvorili štruktúry, ktoré sa potom pomocou XML parseru mohli buď zapísať alebo prečítať. Pre tieto dáta sú vytvorené štruktúry, ktorá ich v sebe nesie. Táto štruktúra sa potom predáva ako parameter funkciám pre prácu s XML súbormi.

6.1 Načítanie a zapísanie dát do XML

Na uloženie nastavení, ako napr. rozlíšenie, je vytvorená štruktúra s názvom deviceProperties, ktorá v sebe nesie premenné, ktoré sme potrebovali uložiť, aby sme ich mohli pri opätovnom spustení hry načítať.

DeviceProperties obsahuje premenné typu:

- `irr::s32 screenWidth` – je šírka rozlíšenia
- `irr::s32 screenHeight` – výška rozlíšenia
- `irr::s32 bitsperpixel` – bitová hĺbka

Tie sa po spustení pri vytváraní menu načítajú z disku, v ktorom sa dajú neskôr jednoducho zmeniť. Tento XML súbor obsahuje jeden koreňový uzol, ktorý je pomenovaný „config“, v ňom sú vnorené uzly, ktoré nesú potrebné vlastnosti. Názvy sú zhodné s premennými, kvôli väčšej prehľadnosti, názov vlastností je „value“.

Obsah XML súboru na uloženie vlastností pre vytvorenie „device“:

```
<?xml version="1.0"?>
<config>
     <screenWidth value="800" />
     <screenHeight value="600" />
     <BitsperPixel value="0" />
</config>
```

Funkcia na načítanie tohto súboru má názov:

- **void** `parseDeviceData(IrrlichtDevice *device, , const c8* file, deviceProperties &props)`

Parametre funkcie:

- `IrrlichtDevice *device` – je pointer na device hry, pomocou neho vytvoríme vo funkcii IrrXMLReader
- `const c8* file` – je cesta k súboru, ktorý chceme prečítať, v tomto prípade je to súbor s názvom „device.xml“
- `deviceProperties &props` – je odkaz na adresu štruktúry, ktorá nesie vlastnosti, do ktorých sa bude zapisovať

Vo funkcii sa vytvorí nový lokálna premenná typu `irr::io::IXMLReaderUTF8` s názvom xml a pomocou funkcie `read()` a cyklu `while()` sa prechádza celý XML súbor. Metódou `getNodeName()` sa získa názov uzlu a porovná so žiadanými. Potom sa jeho vlastnosti pomocou metódy `getAttributeValueAsInt(„value“)` načítajú do premenných štruktúry `deviceProperties`.

Za zapisovanie do XML pre vlastnosti „devicu“ slúži funkcia:

- void writeXMLDevice(IrrlichtDevice *Device, const c8 *file, deviceProperties &props)

Parametre funkcie:

- `IrrlichtDevice *device` – je pointer na device hry, pomocou neho vytvoríme vo funkcii IXMLWriter
- `const c8* file` – je cesta k súboru, do ktorého chceme zapísať dáta, v tomto prípade je to súbor s názvom „device.xml“, teda prepíšeme XML súbor, ktorý slúži aj na získanie dát
- `deviceProperties &props` – je odkaz na adresu štruktúry, ktorá nesie vlastnosti, ktoré chceme zapísať

Funkcia najprv vytvorí IXMLWriter a predá ho pointeru xml, ktorý je rovnakého typu. Pri vytváraní je použitá funkcia createXMLWriter(`const c8* file`). XML súbor by mal obsahovať hlavičku, ktorá sa vytvorí metódou `writeXMLHeader()`. Hlavička, ktorá sa vytvorí bude vyzerať nasledovne:

- `<?xml version="1.0"?>`

Za hlavičkou nasleduje hlavný uzol „config“. Ten je vytvorený metódou `writeElement(„config“, false)`. Prvý parameter označuje názov uzlu, druhý odkazuje, že uzol nebude mať žiadne vlastnosti. Ďalšie uzly sú vytvorené rovnakou metódou, ale ako druhý parameter maj nastavený názov vlastnosti a tretím je hodnota

vlastnosti. Názov vlastnosti je v tomto prípade „value“ a do hodnoty sa zapisuje hodnota danej premennej zo štruktúry `deviceProperties`.

Niektoré časti kódu funkcie `writeXMLDevice`:

```
irr::io::IXMLWriter *xml = Device->getFileSystem()->
                           createXMLWriter(file);
xml->writeXMLHeader();
 xml->writeLineBreak ();
 name = "screenWidth";
            attrValue = props.screenWidth;
            xml->writeElement(name.c_str(), true,
                  attrName.c_str(), attrValue.c_str());
            xml->writeLineBreak ();
```

Pre autá sme vytvorili štruktúru carProperties. Tá v sebe nesie informácie ako cestu k meshu, textúre, o rozložení kolies auta, o sile motora, o jeho dĺžke, výške a o ďalších vlastnostiach, ktoré sú treba pri vytvorení auta:

6.2 Načítanie dát pre autá

Informácie o aute sme potom uložili v XML súbore s názvom „car.xml“. Tento súbor slúži len na získavanie dát pri vytvorení áut a umožňuje vložiť do hry nové auto, bez zásahu do kódu, pretože obsahuje všetky potrebné informácie k jeho vytvoreniu.

Kód XML súboru s názvom „car.xml“:

Premenné štruktúry:

- `f32 suspensionLength;`
- `s32 suspensionSpring;`
- `s32 suspensionShock;`
- `s32 engineTorque;`
- `s32 brakePower;`
- `s32 mass;`
- `float length;`
- `float width;`
- `float height;`
- `short chassisMatID;`

- `short windowMatID;`
- `vector3df tirePos1;`
- `vector3df tirePos2;`
- `vector3df tirePos3;`
- `vector3df tirePos4;`
- `f32 scale;`
- `vector3df tireScale;`
- `float inertia[3];`
- `string<c8> wheelMesh;`
- `string<c8> wheelTex;`

Funkcia na prečítanie tohto súboru má podobnú štruktúru ako funkcia na čítanie z „device.xml“. Jediným rozdielom je štruktúra carProperties, ktorá má oveľa viac premenných, pretože na vytvorenie auta v scéne je potreba veľa informácií.

Funkcia na čítanie zo súboru „car.xml“:

- `void parseCarData(IrrlichtDevice *Device, const c8* file, carProperties &props)`

Parametre tejto funkcie:

- `IrrlichtDevice *Device` – potrebný pre vytvorenie IXMLReaderu
- `const c8* file` – cesta k súboru „car.xml“
- `carProperties &props` – štruktúra vlastností auta

6.3 Načítanie dát pre dynamické vkladanie objektov

Ďalším súborom, z ktorého treba získať dáta, je súbor „objects.xml“. Ten v sebe uchováva informácie o „dynamických“ objektov, ktoré nie sú do scény vložené napevno. Ich počet, umiestnenie a tvar je uložený v XML, takže sú tieto parametre nezávislé od kódu a do scény sa môžu pridávať nové objekty bez potreby nejako meniť skompilovaný kód. Tieto objekty však môžu byť iba statické, bez animácií.

Pretože ich počet nie je napevno daný a môže byť ľubovoľnej veľkosti, nepredáva sa do funkcie, ktorá tento súbor číta štruktúra ale dynamické pole štruktúr.

Štruktúra, ktorá nesie informácie o jednom objektu má názov `dynObjProperties` a má následujúce premenné:

- `vector3df size` – je to vektor, ktorý určuje veľkosť objektu
- `float mass` – je hmotnosť objektu
- `string<c8> meshPath` – je cesta k meshu
- `string<c8> shape` – je tvar objektu
- `vector3df pos` – je vektor, který určuje pozíciu v scéne
- `vector3df rot` – je vektor určujúci rotáciu objektu

Funkcia na čítanie dát z tohto súboru má názov:

- `void parseXML(IrrlichtDevice *Device, const c8* file, array<dynObjProperties> &poleVlastnosti)`

Parametre tejto funkcie:

- `IrrlichtDevice *Device` – potrebný pre vytvorenie IXMLReaderu
- `const c8* file` – cesta k súboru „objects.xml“
- `array<dynObjProperties> &poleVlastnosti` – pole štruktúr `dynObjProperties`

Štruktúra tejto funkcie má opäť rovnaký princíp ako predchádzajúce funkcie na čítanie XML súborov, kde sa v cykle prechádzajú všetky uzly a zapisujú sa do štruktúry hodnoty ich vlastností. Objektov, ktoré takto vkladáme je veľa a viaceré sú podobné, a jediná vlastnosť, ktorá sa im veľa krát mení je pozícia s rotáciov. Napr. smetiaky, ktoré sú takto vložené do scény, majú rovnakú veľkosť, a cestu k meshu. Jediné, čo sa im mení je pozícia, pri lavičkách sa k pozícii pridáva aj rotácia, pritom rovnakých smetiakov a lavičiek je v scéne niekoľko. Aby sa do XML nemuseli písať stále tie isté vlastnosti objektov, ktoré sa nemenia, uspôsobila sa podľa toho funkcia na načítanie. Teraz stačí pre určitý typ objektov napísať na začiatok ich statické vlastnosti a potom už len zapisovať vlastnosti v ktorých je zmena. Pri prečítaní uzlu so zmenenou pozíciou sa do pola štruktúr uloží celá štruktúra so všetkými vlastnosťami.

Kúsok kódu zo súboru „objects.xml“:

```
<?xml version="1.0"?>
<cube1>
 <properties>
   <size x="1.5" y="1.5" z="1.5"/>
   <mass value="20"/>
   <meshPath value="data/common/cube1.b3d"/>
   <shape value="box">
 </properties>
 <cube>
  <position x="-12" y="0.759" z="-11.4"/>
  <rotation x="0" y="0" z="0"/>
 </cube>
 <cube>
  <position x="-12" y="0.759" z="-9"/>
  <rotation x="0" y="0" z="0"/>
 </cube>
```

ZÁVER

Cieľom tejto práce bolo zoznámiť sa s Irrlicht enginom a zamerať sa najmä na objekty a funkcie grafického rozhrania a práce so súbormi. Na základe získaných znalostí, so študentom, ktorý rieši téma Irrlicht engine – podpora grafických technológii a managementu scény, vytvoriť návrh komplexnejšieho programu využívajúceho tento engine.

S uvedenými časťami enginu som sa podrobne zoznámil a popísal ich v teoretickej časti práce. V druhej kapitole som popísal grafické rozhranie vybraného enginu. Niektorým prvkom som venoval väčšiu pozornosť, pretože som ich pokladal za dôležitejšie a používal som ich aj v praktickej časti pri tvorbe zvoleného programu.

Prácu so súbormi som rozpísal v tretej a štvrtej kapitole. Zameral som sa na podporované formáty, a na ich načítanie do scény. Pretože formát XML používame v praktickej časti dosť často, je mu venovaná väčšia časť kapitoly.

V praktickej časti, ktorá začína piatou kapitolou, vytvorením jednoduchého herného menu, kde popisujem, prácu s GUI, použitie jednotlivých prvkov a funkcie GUI. Zmeny, ktoré sa nastavujú v menu sa ukladajú pomocou XML súboru. Ukladanie som však popisoval v ďalšej kapitole, ktorá je zameraná na vstupy a výstupy.

V šiestej kapitole som nakoniec zhrnul využitie z poznatkov z časti teórie, ktorá je zameraná na prácu so súbormi. Vo zvolenom programe sa používa XML formát na získavanie dát pre vložené objekty v scéne, preto som podrobne opísal tieto funkcie na čítanie a výpis z tohto formátu.

RESULT

Ambition of my work was to familiarize with Irrlicht engine and focus on objects and functions of graphic user interface and file operations. Then cooperate with student who solve topic „Irrlicht engine – graphic technology support and management scene“ and create complex program.

Those engine parts that are noted I have described in detail in the theoretical part. In the second chapter I have described graphical interface selected engine. Some elements I have devoted most attention, because I considered it important, and I have used them also in the practical part in the development of the chosen program.

The file operations I have described in the third and fourth chapter. I focused on the supported formats, and getting them into the scene. Because XML format is used in the practical part often enough, I devoted the greater part of chapters for this format.

In the practical part, which begins with the fifth chapter, I started with describing of creating a simple game menu with use of various GUI elements and functions GUI. The amendments, which are set in the menu are saving to disk using XML file. I have described it in another chapter, which is focused on inputs and outputs.

In the sixth chapter, I finally summed up the use of the knowledge of the theory, which is designed to work with files. In the selected program is used XML format for data acquisition for embedded objects in the scene, so I described read-write functios of this format.

ZOZNAM POUŽITEJ LITERATÚRY

[1] Irrlicht engine – a free open source 3d engine [online].2007, [cit. 2008-01-22]. Dostupné z WWW: <http:// irrlicht.sourceforge.net/>

[2] FINNEY, Kenneth C. 3D game programming all in one. [s.l.] : Premier press, 2002. s. ISBN 1-59200-136-X.

[3] TEIXEIRA DE SOUSA, Bruno Miguel. Game programming all in one. [s.l.] : Premier press, 2002.992 s.ISBN 1-931841-23-3.

[4] MORRISON, Michael. Naučte se programovat počítačové hry za 24 hodin. 1. vyd. Brno : Computer Press, 2004. 241 s. ISBN 80-251-0371-4.

[5] LIBERTY, Jesse. Naučtese C++ za 21 dní. Brno : Computer Press, 2007. 796 s. ISBN 80-251-1583-1.

[6] GEBHARDT, Nikolaus. Irrlicht Engine : features [online]. c2003-2005 [cit. 2008-04-13]. Dostupný z WWW: <http://irrlicht.sourceforge.net/features.html>.

[7] Graphical user interface : Wikipedia, the free encyclopedia [online]. 2008 [cit. 2008-03-12]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Graphical_user_interface>.

[8] Game Engine : Wikipedia, the free encyclopedia [online]. [2008] [cit. 2008-04-12]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Game_engine>.

[9] Irrlicht Engine wiki : Irrlicht Engine wiki [online]. 2008 [cit. 2008-04-21]. Dostupný z WWW: <http://www.irrlicht3d.org/wiki/>.

[10] Irrlicht Engine : Tutorial: User Interface [online]. 2003-2005 [cit. 2008-04-20]. Dostupný z WWW: <http://irrlicht.sourceforge.net/tut005.html>.

[11] Irrlicht Engine : Tutorial: Mesh viewer [online]. 2003-2005 [cit. 2008-04-20]. Dostupný z WWW: <http://irrlicht.sourceforge.net/tut009.html>.

[12] Irrlicht Engine : Namespace Index [online]. 2003-2007 [cit. 2008-04-20]. Dostupný z WWW: <http://irrlicht.sourceforge.net/docu/namespaces.html>.

[13] JPEG : WIkipedia, the free encyclopedia [online]. 2008 [cit. 2008-03-12]. Dostupný z WWW: <http://en.wikipedia.org/wiki/JPEG>.

[14] PNG : WIkipedia, the free encyclopedia [online]. 2008 [cit. 2008-03-12]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Png>.

[15] COLLADA [online]. 2007 [cit. 2008-04-15]. Dostupný z WWW: <http://www.collada.org/mediawiki/index.php/Main_Page>.

[16] DirectX : Wikipedia, the free encyclopedia [online]. 2008 [cit. 2008-03-12]. Dostupný z WWW: <http://en.wikipedia.org/wiki/DirectX>.

[17] XML : Wikipedia, the free encyclopedia [online]. 2008 [cit. 2008-03-15]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Extensible_Markup_Language>.

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

GUI Grafické uživateľské rozhranie (graphical user interface).

XML Rozširovateľný značkovací jazyk (eXtensible Markup Language).

ID Identifikačné číslo.

UTF-8 Zpôsob kódovania reťazcov (UCS Transformation format)

ZOZNAM OBRÁZKOV

SEZNAM PŘÍLOH

PI – DVD

PŘÍLOHA P I: DVD

Adresárová štruktúra DVD:

/praca – v tejto zložke je uložená vypracovaná bakalárska práca

/hra – tu sa bude nachádzať zdrojový kód s hrou

/prirucka – vložená bude príručka podrobného popisu