Editor pro tvorbu 3D výukových simulátorů

Martin Vavrek

České vysoké učení technické

1.5.2014

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu 60 Zákona č. 121/2000Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 1.5.2014.....

Abstract

This diploma thesis deals with the creation of learning environments in 3D simulators. The aim of this work is to facilitate the development of such simulators and create a tool that makes it possible. This tool takes the form of 3D scenes editor and for the purpose of implementation was chosen the Microsoft Silverlight technology. Editor functionality is demonstrated on a simple resuscitation scenario of the patient.

Abstrakt

Táto diplomová práca sa zaoberá tvorbou výukových simulátorov v prostredí 3D. Cieľom práce je uľahčiť tvorbu takýchto simulátorov a vytvoriť nástroj, ktorý to umožní. Tento nástroj má formu editora 3D scén a pre implementáciu je zvolená technológia Microsoft Silverlight. Možnosti editoru sú demonštrované na jednoduchom scenári resuscitácie pacienta.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

V súčasnosti je virtuálna realita považovaná za jedno z najviac sa rozvíjajúcich odvetví informatiky. Ak si odmyslíme VR pomocou špeciálnych nástrojov, okuliarí a zariadení, tak jednou z najdostupnejších foriem VR je 3D počítačová hra. I takéto hry nám vedia ponúknuť pri súčasnom hardwari dostatočne pohlcujúci zážitok, ktorý priblíži simulovanú realitu veľmi presne. Ukazuje sa, že počítačová hra nemusí byť primárne použitá len pre zábavu a tak vzniká iniciatíva vyrábať simulačné hry nazývané serious games (výukové hry).

Tieto hry môžu byť použité ako medicínske simulátory, ktorými sa táto práca zaoberá. Cieľom tejto práce je vyskúšať a popísať tvorbu takéhoto simulátoru pomocou vhodného nástroja a tento nástroj vyrobiť. Tento nástroj bude mať formu 2D/3D editora. V práci popíšeme všetky úskalia spojené s tvorbou takejto simulácie a potrebných nástrojov.

V úvode si popíšeme možnosti medicínskych simulácii, ktoré spojíme s krátkou rešeršou hotových medicínskych hier. V ďalšej časti si rozoberieme možné technológie pre implementáciu simulácií a tvorbu spomínaného nástroja.

Keďže táto práca bude nadväzovať na tvorbu simulácií v laboratóriu biokybernetiky na 1.Lekárskej Fakulte UK, tak si v krátkosti popíšeme tvorbu 2D simulačných aplikácií pomocou nástrojov vyvinutých v tomto laboratóriu.

V kapitole Implementácia si popíšeme potrebné časti 3D herného enginu, ktorý sme museli vyrobiť a ďalej si popíšeme časti a funkcionalitu vytvoreného editora.

V posledných kapitolách ozrejmíme tvorbu scenáru vo vytvorenom editore. Výsledok úsilia budeme demonštrovať na príklade jednoduchého scenára kardiopulmunálnej resuscitácie pacienta.

V Závere práce zhodnotíme výsledky a prínos tejto práce. Zároveň bude diskutovaná vhodnosť a budúcnosť použitých technológii.

Kapitola 2

Virtuálne prostredie a výuka

V poslednej dobe sa často v spojení s výukou v školách hovorí o veľkej budúcnosti výukových hier (serious games) a aplikácií. Z herného odvetvia je to momentálne jedna z jeho najrýchlejšie sa vyvíjajúcich častí.

Vážne hry by sme mohli definovať ako hry, ktorých primárnym účelom nie je zábava. Tieto hry umožnia užívateľovi zažiť situácie, ktoré v reále nie je možné jednoducho zrealizovať z rôznych príčin, alebo opakovať danú situáciu pokiaľ ju užívateľ dostatočne nezvládne. Dôvodov môže byť mnoho, či už nebezpečnosť danej úlohy, cena, časové, alebo iné nároky. So súčasným rozvojom PC hardwaru je zároveň možné zdokonaľovať tieto hry, aby sa priblížili realite čo najviac. Veľkou výhodou je zlepšovanie efektivity pochopenia danej problematiky užívateľom oproti konvenčným metódam výuky, toto zlepšenie dokáže byť aj niekoľkonásobné [1].

Tieto hry môžu byť použité v rôznych odvetviach ako je armáda, vzdelávanie, firemné použitie a zdravotná starostlivosť. Táto diplomová práca sa konkrétne bude zaoberať zdravotnou starostlivosťou.

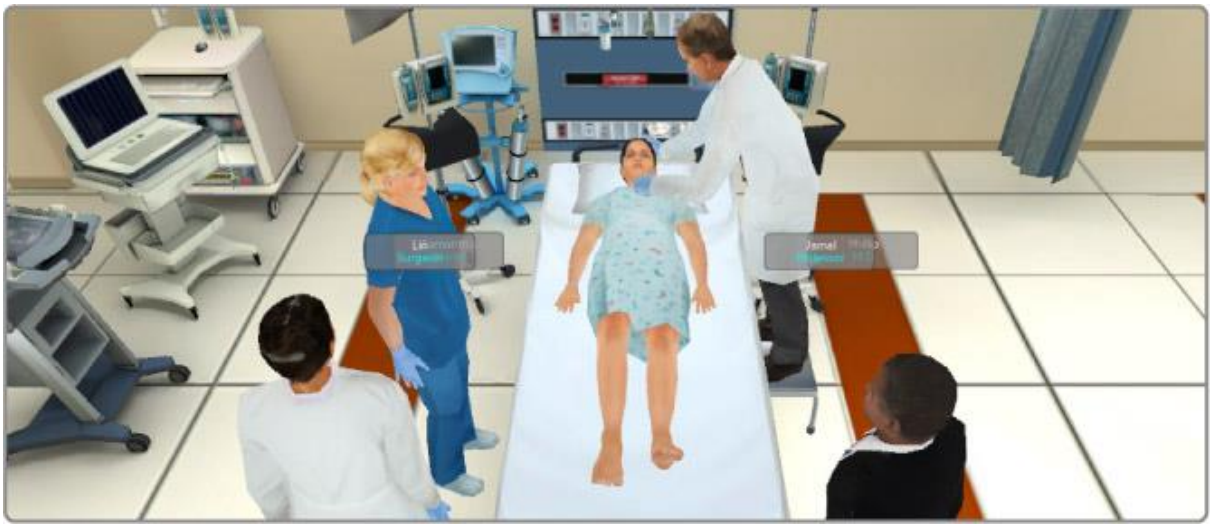
Toto odvetvie vážnych hier je nazvané ako vážne hry pre zdravotnú starostlivosť[1] a v poslednej dobe vzniká v tejto oblasti množstvo štúdií. Dôležitú úlohu u týchto hier hraje aj použiteľnosť. Použiteľnosť môže byť chápaná ako miera kvality interakcie, ktorú nám nejaký softvér ponúka. Pozostáva z atribútov ako sú: jednoduchosť učenia, efektivita, jednoduchosť zapamätania si [2].

Tieto hry patria do skupiny medicínskych simulátorov. Ukazuje sa, že tréningové programy pomocou týchto a iných simulátorov pomáhajú lekárom zlepšiť ich výkon,

[1] Serious Games for healthcare

čas na rozhodnutie v kritických situáciách, a zapamätanie si štandardných postupov. Tieto simulátory nám umožňujú skrátiť dobu výuky medika [1].

Pre ukážku by som rád uviedol kooperačnú hru z medicínskeho prostredia CliniSpace [3]. Jej cieľom je vzdelávanie lekárov a ich schopností navzájom spolupracovať. CliniSpace bola vytvorená pomocou technológie Unity 3D, ktorá bude spomenutá ďalšej kapitole ako možný nástroj. Autori v hre využili fyziologického modelu pacienta, ktorý reaguje na podnety z okolia a úkony lekára. Cena tejto hry je 9999$ za rok predplatného desiatich licencií.



Obrázok 1 - CliniSpace

Ďalším odvetvím lekárskych simulácií je použitie robotických figurín. V týchto figurínach sa nachádzajú robotické časti a senzory, ktoré simulujú chovanie reálneho pacienta.

V závere by sa dalo ešte uvažovať o miere reality, ktorú takéto výukové hry môžu ponúknuť. Bohužiaľ súčasné technológie nám neumožňujú úplne všetko. Ak by sme chceli trénovať lekárovu zručnosť v prípade nejakého zákroku, tak figuríny ľudského tela sú oveľa vhodnejšie ako 3D simulácia, ktorá nevie poskytnúť v niektorých prípadoch dostatočnú spätnú väzbu. Preto je nutné si stanoviť splniteľné ciele, ktoré môžeme od takejto hry požadovať.

2.1 Cieľ práce

V tejto diplomovej práci sme sa v spolupráci s laboratóriom biokybernetiky na lekárskej fakulte Univerzity Karlovej (ďalej len laboratórium biokybernetiky) snažili o zjednodušenie vizualizácie a pochopenia matematických modelov popisujúcich fungovanie ľudského tela pomocou 3D. Vizualizácia bude mať formu počítačovej hry, kde hráč (budúci lekár) bude jednoducho ovládať 3D prostredie a snažiť sa dosiahnuť požadovaný cieľ (napríklad zachrániť pacienta). Táto hra bude spustiteľná vo webovom prehliadači.

Výsledkom tejto práce by mal byť nástroj, ktorý zaučenému užívateľovi pomôže vytvoriť spomínanú hru a uľahčí jej tvorbu. Tento nástroj by mal umožniť definovanie 3D scény v ktorej sa bude daný scenár odohrávať, tak ako definovanie deja tohto scenára pomocou vizuálneho vyjadrenia. 3D scéna bude ovládaná pomocou klikania na jednotlivé objekty. Interaktívne objekty budú zvýraznené pomocou popiskov. Kliknutím na tieto predmety sa zobrazí menu s možnosťami daného predmetu v 3D. Nástroj bude umožňovať jednoduché načítanie 3D modelov a ich správu. Bude schopný editovať parametre objektov umiestnených v scéne.

V tejto práci sa zároveň pokúsime o prepojenie existujúcich techonológií, ktoré vznikli v laboratóriu biokybernetiky s 3D prostredím. Takto dôjde k zachovaniu väčšiny workflow a bude rozšírené o ďalšie role s ich úlohami, ktoré sa budú podieľať na tvorbe 3D simulátoru. Súčasný proces tvorby simulátorov v laborátoriu biokybernetiky si detailnejšie popíšeme v ďalších kapitolách.

Kapitola 3

Dostupné technológie

3.1 Silverlight

Technológia Microsoft Silverlight v poslednej verzii (5.0) obsahuje knižnice herného enginu XNA, ktorá umožňuje tvorbu 3D aplikácii. Tieto aplikácie môžu byť spustiteľné vo webovom prehliadači. Táto technológia pôvodne vznikla ako nástroj pre tvorbu biznis aplikácií vo webovom prostredí. Silverlight v sebe kombinuje text, vektorovú a bitmapovú grafiku, video a 3D. Preto môžeme povedať, že obsahuje robustný základ pre tvorbu nášho nástroja.

V prípade tejto práce bolo výhodou možné znovu použitie 2D aplikácií vytvorených pre túto platformu v laboratóriu biokyberentiky. Tieto aplikácie by sa dali použiť ako vrstva nad 3D časťou našej hry, ale aj ako textúry na objektoch v 3D scéne. Príklad môže byť EKG krivka na defibrilátore v scéne. Výzor týchto 2D aplikácii je možné aj spolu s animáciami testovať a navrhovať v Microsoft Blend. Microsoft Blend by sa dal nazvať vizuálnym editorom grafických rozhraní nadefinovaných pomocou jazyka XAML.

Silverlight je do istej miery multi platformný, keďže aplikácie vytvorené na tejto platforme sú spustiteľné na Windows a na OSX a pre staršie verzie SL existuje aj open sourcová implementácia pod menom Moonligt (bohužiaľ tá ostala vo verzii, ktorá dokázala prehrať aplikácie len zo Silverlight 3). Čo sa týka podpory OSX, tak tá neobsahuje časť pre 3D. Vysvetlením je použitie DirectX, ktoré na OSX nie je dostupné.

Momentálne je posledná verzia Silverlight dva roky stará a autor tejto práce nepredpokladá ďalšie zvyšovane verzie tohto produktu. Pred dvomi rokmi sa očakávalo práve rozšírenie SL o lepšiu podporu 3D, čo sa asi už nestane. Momentálne sú niektoré časti 3D knižníc len v Silverlight toolkite, čo je open source rozšírenie Silverlightu.

Ďalšími nevýhodami tohto riešenia sú zastaralosť a obmedzenosť verzie XNA, ktorú Silverlight obsahuje. XNA má v desktopovej verzii možnosť v dvoch profiloch: HiDef a Reach [4]. Tieto profily sa líšia v obmedzeniach počtu vertexov vo vertex bufferi, verziou shader modelu, maximálnou veľkosťou textúr, atď. Bohužial, v Silverlighte je povolený len profil Reach, ktorý vznikol hlavne pre tvorbu hier na low endový hardware. To znamená, že v prípade 3D na Siverlighte sme obmedzení často viac týmto profilom ako konečným hardwarom.

Výhodou pre užívateľa by sa dala nazvať bezpečnosť Silverlightu. Microsoft obmedzí volanie systémových knižníc, a čo sa týka 3D, tak užívateľ musí v nastaveniach 3D povoliť pre každú webovú aplikáciu zvlášť. Microsoft zároveň obmedzil volania pre grafickú kartu v rámci rozhrania XNA. Preto väčšina grafických štruktúr z XNA má metódy pre nastavenie dát, ale následne sa tieto dáta nedajú získať späť. Ako príklad uvediem textúru: celú scénu si môžeme vykresliť do textúry, ale k dátam, z tejto textúry sa v našej aplikácii bohužiaľ nedostaneme. Túto problematiku rozoberiem hlbšie v kapitole **Implementácia**.

3.2 Unity 3D

Unity 3D vzniklo v roku 2005 ako nástroj pre podporu tvorby 3D hier pre OSX. Unity 3D by sa dalo rozdeliť na 2 časti. Na jadro herného enginu, napísaného v jazyku C++ a na skriptovaciu časť, ktorá vychádza z jazyka C#. Táto časť sa kompiluje do CL (Common language) a je spúšťaná na virtuálnom stroji Mono, ktorý je open sourcovou implementáciou virtuálneho stroja Microsoft .NET [5].

Toto riešenie má nevýhodu v nemožnosti editovania a zobrazenia kódu jadra. Výhodou je ale výkon, ktorý C++ ponúka [6].

Unity 3D ponúka vlastné IDE v podobe 3D editora, ktorého vzhľad a chovanie je rozšíriteľné práve pomocou C#. Tento editor spĺňa parametre WISWYG[2], čo znamená, že nám uľahčí nastavenie scény a okamžite vidíme, ako bude scéna vyzerať. Umožňuje

[2] Z anglického „What you see is what you get"

nám v dobe návrhu hry prehrávať animácie, meniť parametre objektov v scéne a podobne.

Výhodou použitia C++ a Mono je multiplatformnosť. Pomocou Unity 3D vznikajú hry pre Windows, OSX, Linux, ale aj pre mobilné platformy iOS, Android a Windows Phone. Preto je možné vytvoriť hru pre väčšinu zariadení na trhu. Daň za tieto možnosti je cena za licencie. Unity 3D je dostupná v dvoch verziach: Unity 3D Basic a Unity 3D Pro. Unity 3D Basic je zadarmo, avšak ponúka dostatočne veľa funkcionality na vytvorenie hry pre spomínané desktopové platformy. V prípade, že chceme používať pokročilejšie funkcie tak cena Unity 3D Pro je v súčasnosti 1500$. V prípade potreby deploymentu s pokročilými funkciami pre mobilné zariadenia sa cena navýši o 1500$ za každú ďalšiu mobilnú platformu.

Tento nástroj je veľkou posilou pre herných vývojárov, ktorí chcú minimalizovať cenu a čas vývoja ich produktu, a zároveň ho mať v perfektnej kvalite. Vývojári zároveň nemusia začínať na zelenej lúke.

Nástroj ako celok ponúka množstvo možností a funkcií. Za hlavné z nich považujem jednoduchú správu obsahu hry, ako sú 3D modely (vrátane animácií), zvuky, obrázky, videá a podobne. Ďalej integráciu fyzikálneho enginu PhysX od firmy Nvidia, ktorý je samozrejme optimalizovaný. Potom je to možnosť používať pokročilé rendrovacie techniky ako sú realtime tiene, svetelné mapy na pár klikov.

Okolo Unity 3D sa vytvorila od jeho počiatkov obrovská komunita. Unity 3D má komunitný asset store, na ktorom je k dispozícii za rozumné ceny množstvo rozšírení, assetov, a pluginov. Keď nejaká funkcionalita nie je zahrnutá priamo v Unity, tak je veľká pravdepodobnosť, že sa bude dať dokúpiť ako plugin zo spomínaného asset storu.

3.3 Ďalšie možnosti

Ďalšou technológiou, ktorú v krátkosti spomenieme je WebGL. Jedná sa o Javascriptové API pre zobrazovanie 3D grafiky vo webových prehliadačoch. Malo by ísť o natívne zobrazenie 3D, ale nie všetky moderné prehliadače umožňujú použitie WebGL. Táto technológia nie je široko používaná a preto by muselo dôjsť k vlastnej

implementácií niektorých potrebných funkcií ako sú načítania modelov animácií v jazyku Javascript.

3.4 Porovnanie

V dobe písania tejto práce by bol jasný víťaz Unity 3D, ale bohužiaľ prvé rozhodnutia a časti kódu vznikli pred dvoma rokmi, keď bola do Silverlightu vkladaná veľká nadej a jednou z hlavných požiadaviek bola znovupoužiteľnosť hotových častí 2D aplikácii a časti kódu. Výsledkom tejto práce našťastie nemusí byť hra v špičkovej grafickej kvalite, a tak by mal byť Silverlight dostačujúcou platformou.

Ďalej bolo pravdepodobné, že v aplikácii budú práve 2D prvky, ktoré Unity natívne nepodporuje. A to by bolo zobrazenie grafov a vektorovej grafiky.

V súčasnosti by bol vývoj 3D časti aplikácie oveľa jednoduchší v Unity, ako v Silverlighte. Vývoj 2D časti samotnej hry by bol samozrejme zložitejší, ale v prípade obmedzenia sa na jednoduché grafy a bitmapovú grafiku nenáročný. Tak isto by sme mohli použiť niektoré hotové pluginy z Unity AssetStoru.

Keďže je XNA iba základný framework a prakticky dokáže načítať a zobraziť 3D a 2D obsah na obrazovke, tak bude nutné nad touto platformou vytvoriť robustnejší framework, ktorý je možné nazvať aj herným enginom a nad týmto enginom bude nutné vytvoriť spomínaný nástroj (editor). Výhodou je plná kontrola a možnosť prispôsobiť 3D engine tak, aby fungoval dobre s predchádzajúcou prácou v laborátoriu biokybernetiky. To bolo hlavným faktorom pre rozhodnutie použiť **Silverlight**.

Kapitola 4

Simulátory založené na Modelica a framework Bodylight

Táto práca nadväzuje na činnosť laboratória biokybernetiky na 1.LF UK a v tejto časti popíšeme fungovanie, workflow a tvorenie výukových simulátorov bez podpory 3D pomocou technológií vyvinutých v tomto laboratóriu.

Základom simulátorov je model vytvorený v jazyku Modelica. Tento model sa následne prekladá do jazyka C# a následne pomocou frameworku Bodylight je možné tieto modely spúšťať, nastavovať im parametre a prezentovať v prostredí Microsoft Silverlight.

4.1 Jazyk Modelica

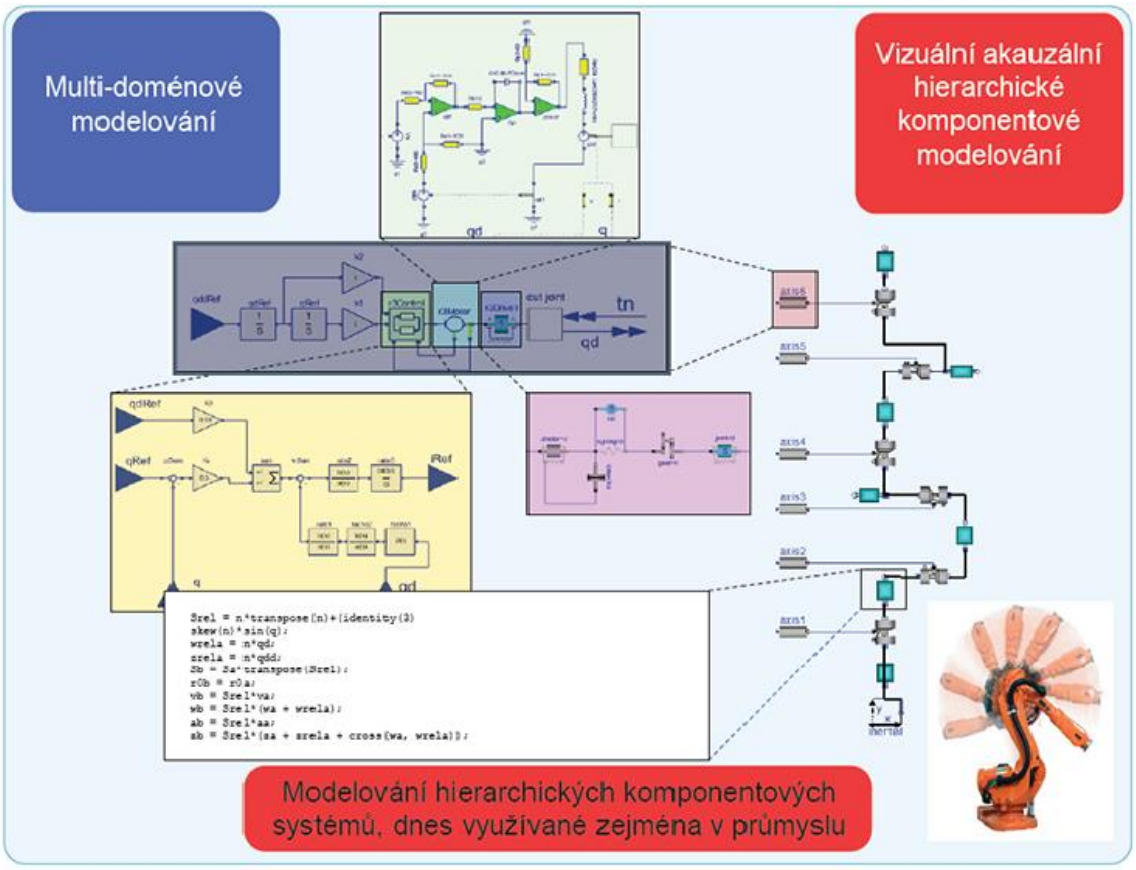
Modelica je štandardizovaný objektovo orientovaný, deklaratívny modelovací jazyk pre komponentové modelovanie komplexných systémov obsahujúcich komponenty z rôznych fyzikálnych domén. Modelica uľahčuje a zrýchľuje modelovanie komplexných fyzikálnych systémov obsahujúcich napr. elektrické, elektronické, hydraulické, tepelné, chemické a ďalšie procesne orientované komponenty.

Modely sú popísané pomocou diferenciálnych, algebrických a diskrétnych rovníc. Komponenty v modeloch majú hierarchickú štruktúru (viď. Obrázok 2) a najnižšia úroveň je vyjadrená textovou podobou pomocou rovníc. Tieto komponenty sú medzi sebou prepájané a majú aj svoju vizuálnu podobu. Každá komponenta je inštanciou nejakej modelikovej triedy.

Tento jazyk podporuje niekoľko komerčných nástrojov. Nutné je spomenúť nástroj Dymola, ktorého autori stoja za vývojom jazyka Modelica[3]. Tieto nástroje obsahujú

[3] Dymola bol pôvodne názov jazyka, z ktorého jazyk Modelica vychádzal

zároveň grafický editor jazyka Modelica. Tento editor prehľadne zobrazuje komponenty, z ktorých sa model skladá. Takto je možné povedať, že štruktúra modelu zobrazuje lepšie štruktúru modelovanej reality, než postup výpočtu modelu. [7].

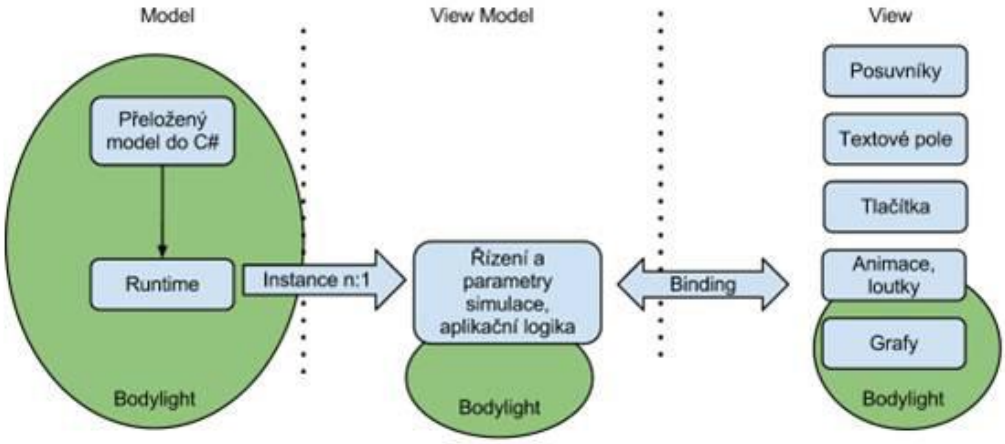


Obrázok 2 - Jazyk modelica podporuje grafické aj textové vyjadrenie modelu, tento spôsob uľahčuje orientaciu v modelu [4]

4.2 Framework Bodylight

Framework Bodylight je vyvíjaný v laboratóriu biokybernetiky už niekoľko rokov. Obsahuje nástroje pre prevod modelov z jazyka Modelica do prostredia .NET. Zároveň obsahuje prostredie pre spúšťanie týchto modelov (runtime) a ich riešič (solver).

Tento framework uľahčuje tvorbu výukových aplikácií a ich prepojenie s fyziologickými modelmi [8].



Obrázok 3 - Architektúra frameworku Bodylight [8]

Vďaka tejto technológii je možné spúšťať komplexne fyziologické modely priamo v prostredí .NET. To umožňuje prezentovanie takýchto modelov v technológii Silverlight, pre ktorú obsahuje tento framework knižnice umožňujúce efektívne prepojenie modelov s prezentačnou vrstvou. Takto môžeme pomocou modelu ovládať animácie a výstupy z modelu prezentovať ako grafy. Framework umožňuje jednoducho meniť vstupy modelu a pre užívateľa ich prezentovať v podobe tlačidiel, zaškrtávatiek, textových polí, alebo posuvníkov.

4.3 Postup tvorby simulátoru

Vytvorenie takéhoto simulátoru nie je jednoduché a od pôvodného nápadu po zrealizovanie je potrebných niekoľko krokov. V prvom rade je potrebné mať nápad na to, čo a ako chceme demonštrovať. Tento krok môžeme nazvať tvorbou scenáru. V našom prípade autor scenáru je najčastejšie pedagóg, doktor, alebo profesor, ktorý danej problematike rozumie.

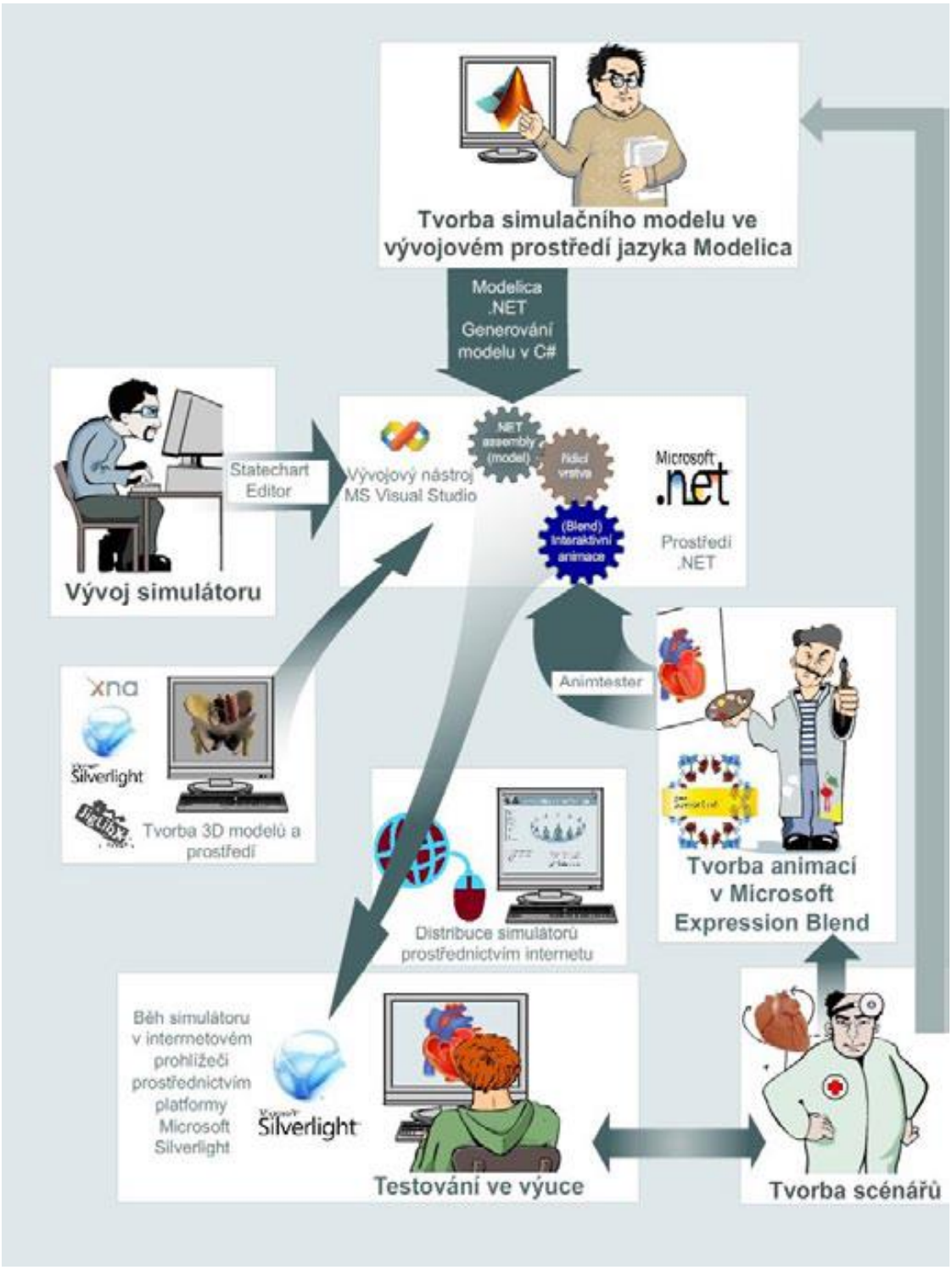
Autor scenára musí mať zároveň dostatočne presnú predstavu ako bude výsledná výuková aplikácia vyzerať a fungovať. Z týchto požiadaviek je následne možné zistiť aké parametre modelu a aké jeho výstupy budeme potrebovať.

Tento scenár je nutné prekonzultovať s tvorcom modelu. Ten sa následne snaží v jazyku Modelica navrhnúť model. Čo znamená nájsť vyjadrenie jeho rovníc a parametre tak, aby model pre rôzne vstupy vracal výsledky korešpondujúce so simulovaným systémom. V momente, keď sme s modelom spokojní na scénu prichádza programátor.

Ten následne nechá automaticky preložiť model do jazyka C#. K tomuto modelu vytvorí adekvátny viewmodel. V tomto viewmodely vystaví vstupné a výstupné parametre, ktoré budú potrebné vo výslednej aplikácii. Tieto parametre sú najčastejšie premenné typov boolean, reálne čísla a sady bodov, v prípade výstupu pre graf.

Nezávisle na práci programátora pracuje grafik, ktorý v prípade potreby vytvára 2D vektorovú grafiku s animáciami pomocou nástroja Microsoft Blend. Tieto animácie budú napojené na viewmodel. Grafik môže testovať tieto animácie pomocou časti frameworku Bodylight nazvanej Animtester.

V závere vytvorí programátor užívateľské rozhranie aplikácie, v ktorom budú spomínaná 2D vektorová grafika, grafy, tlačidlá a podobne. Tieto prvky sú napojené na viewmodel pomocou tzv. Data Bindingu, ktorý umožňuje jednoducho definovať prepojenie kontroliek a dát z modelu v jazyku XAML.



Obrázok 4 - Diagram návrhu simulácií [9]

4.4 Rozšírenie o 3D a rozdiely oproti 2D

Posunutie konceptu simulátorov z 2D prostredia do 3D nie je len rozšírenie o jednu dimenziu. V prípade 3D aplikácií musíme klásť väčší dôraz na dej samotnej simulácie. Vo väčšine prípadov máme v dispozícii len jednu scénu, kde sa celý scenár odohráva (napríklad operačný sál). Hráčovi hry (študentovi) musí byť napovedané v ktorej časti hry sa nachádza. Zároveň mu musí byť jasné, aké úlohy môže v danom momente vykonať. V 2D aplikácii sme jednoducho vedeli načítať ďalšiu stránku aplikácie, v ktorej boli ďalšie grafy, ďalšie 2D vektorovo animované postavičky.

V 3D prostredí musíme postupovať trošku inak. Na objekt, ktorý nás momentálne zaujíma musíme upozorniť iným spôsobom. Týchto spôsobov je mnoho. Môžeme vziať kameru, zakázať jej interaktivitu, premiestniť ju na také miesto v scéne, kde chceme a v zábere bude náš objekt záujmu. Ďalším spôsobom je možnosť zvýraznenia daného objektu. Tu môžeme zaradiť zmenu farby, blikanie objektu, vysvietenie objektu atď.

Ďalším spôsobom ako viesť hráča sú textové oznámenia. Tie môžeme mať v 2D vrstve nad samotnou 3D aplikáciou, alebo to môžu byť popisy, ktoré sa zobrazujú nad objektami v 3D scéne.

To ako budú jednotlivé herné mechaniky v hre fungovať určí herný návrhár (game designer). Tento návrhár môže byť jedna a ta istá osoba ako autor scenáru, alebo s ním bude úzko spolupracovať. V ideálnom prípade autor scenáru vytvorí tento scenár a herný návrhár ho s ním konzultuje a následne sa snaží vytvoriť scénu v ktorej budú potrebné objekty a dejová línia korešpondujúca so scenárom. Dejová línia je reprezentovaná úlohami, ktoré musí hráč pri úspešnom prechode hry splniť.

Scenár hry bude samozrejme iný ako v prípade 2D aplikácie a bude počítať s možnosťou interakcie v 3D hre. Bude založený na herných mechanikách, ktoré máme k dispozícii a korešpondujú s našou simuláciou.

4.4.1 Herné mechaniky

Herné mechaniky sú podľa definície sadou pravidiel, ktorá nám umožňuje vytvárať tzv. gameplay hry a hru ako celok [10]. Herné mechaniky nám definujú interaktivitu hry a určujú, čo a akým spôsobom v hre môžeme vykonať. Počet a typ mechaník, ktoré môžeme nájsť v komerčných hrách je rôzny. Tieto mechaniky sa samozrejme líšia podľa typu hier. Návrhom týchto mechaník v hre sa zaoberajú herný návrhári (game designer).

Najprv si môžeme uviesť niekoľko bežne používaných mechaník v hrách:

- Definícia pohľadu hráča na scénu, napríklad pohľad prvej osoby
- Spôsob ovládania hry –napr. používanie myši verzus klávesnice
- Levely - hra je rozdelená na niekoľko úrovní. Tie na začiatku majú ľahšiu náročnosť a zväčša slúžia na to, aby si hráč osvojil herné mechaniky.
- Úlohy a misie – hráč ich splní a posúvajú ho ďalej v deji.
- Systém odmeňovania hráča za splnené úlohy

V prípade našich simulátorov sme sa rozhodli pre pár jednoduchých mechaník:

- Interaktivita s objektmi pomocou popiskov – po kliknutí na interaktívny objekt sa zobrazí kontextové menu možností daného objektu.
- Rozhodovací systém – hráč sa môže rozhodnúť, čo v danej chvíli spraviť
- Hráč by mal byť schopný spraviť zlý krok – človek sa najlepšie učí na vlastných chybách, preto v každom kroku bude mať hráč pomerne veľa možností, čo v hre vykonať.
- Jednoduchý pohľad s možnosťou otáčania – pre tento pohľad sme sa zatiaľ rozhodli kvôli technickým obmedzeniam v Silverligte (nemožnosť schovať, alebo zablokovať kurzor myši)
- Zmena pohľadu v hre - v prípade potreby vieme obmedziť hru tak aby sa hráč díval na objekt, na ktorý chceme upútať pozornosť.
- Scenár je založený na modely z Modelicy – hra ovplyvňuje vstupy a je ovplyvňovaná výstupmi z modelikového modelu

Čo teda potrebujeme pre nášho dizajnéra, aby vedel efektívne pracovať? V prvom rade návrhár potrebuje obsah scény. Preto samozrejme potrebujeme 3D grafika, ktorý vytvorí potrebné 3D modely objektov v scéne. V prípade postáv vrátane kosterných animácií. K jeho práci bude potrebný 3D modelovací nástroj, z ktorého následne vyexportuje modely s animáciami v potrebnom formáte.

Ďalej musíme podotknúť, že úloha nášho dizajnéra bude odlišná od jeho klasického významu. Náš game dizajnér bude zároveň zastávať aj úlohu level dizajnéra a bude sa snažiť vytvoriť dej v danej scéne. To znamená, že podľa zadania zo scenára vytvorí statickú 3D scénu a ju potom rozšíri o interakciu.

Predpokladajme, že náš dizajnér nie je programátor a chceme mu čo najviac uľahčiť jeho prácu. Preto budeme potrebovať nástroj, ktorý mu v tomto všetkom pomôže. Budeme potrebovať nástroj ktorý bude spĺňať tieto požiadavky:

- Jednoduché rozmiestňovanie objektov po 3D scéne
- Jednoduché nastavovanie parametrov objektov v scéne
- Vytváranie deju scenára pomocou vizuálneho vyjadrenia
- Možnosť používania všetkých herných mechaník
- Jednoduché napojenie deju na modely v jazyku Modelica
- Nástroj umožní tvorbu nelineárneho deja

Samozrejme, niektoré z požiadaviek nie sú dosiahnuteľné bez spolupráce programátora. Ten vytvorí sadu malých znovu použiteľných herných komponent, ktoré bude dizajnér používať pri vytvorení scény a jej deja. Týmto sa chceme vyhnúť skriptovaniu deja, ako to je vo väčšine hier zvykom a udržať tak scenár jednoducho modifikovateľným.

Kapitola 5

Návrh 3D časti simulátorov

V tejto časti popíšeme základné koncepty použité pri implementácii. Pred tým ako budeme schopní rozprávať o samotnom nástroji pre tvorbu simulácií, tak sme museli vytvoriť a navrhnúť takzvaný herný engine. Ten by nám mál ponúkať základnú funkcionalitu, ktorú od neho budeme požadovať.

Keďže XNA je framework, ktorý ponúka len správu obsahu, jeho vykresľovanie, a potrebné matematické knižnice pre prácu s 3D, tak budeme musieť postaviť nejaké robustnejšie riešenie nad týmto frameworkom.

5.1 Herný Engine

Čo by mal takýto engine splňovať? V hernom svete existuje momentálne pomerne veľa herných enginov. Všetky určitým spôsobom zjednodušujú prácu vývojárovi a snažia sa vývojára ušetriť od komplikovaných matematických výpočtov, problémom s importom obsahu hry, implementácie fyzikálnych zákonov, umelej inteligencie, hľadania ciest (path finding) a podobne.

My sme si pre náš engine stanovili tieto požiadavky:

- Bude obsahovať graf scény – hierarchické usporiadanie objektov podľa priestorového a logického hľadiska
- Jednoduché vyjadrenie kamery – bude obsahovať objekt kamera, ktorého nastavenia budú určovať vlastnosti perspektívy
- Kreslenie priehľadných objektov – priehľadné objekty v scéne budú vykresľované podľa vzdialenosti od kamery
- API pre klikanie na objekty – engine nám povie na aký objekt v scéne sme klikli
- Bude obsahovať voľne dostupný fyzikálny engine – v prípade požitia kamery z pohľadu prvej osoby sme sa rozhodli integrovať základnú fyziku

- Stav scény pôjde jednoducho uložiť – budeme potrebovať serializáciu najlepšie do binárneho formátu
- Načítanie 3D animovaných postáv a prehrávanie týchto animácií – modelovacie nástroje exportujú sadu animácií ako jednu veľkú animáciu. Preto je nutné túto animáciu rozseknúť na menšie časti.

5.1.1 Graf scény

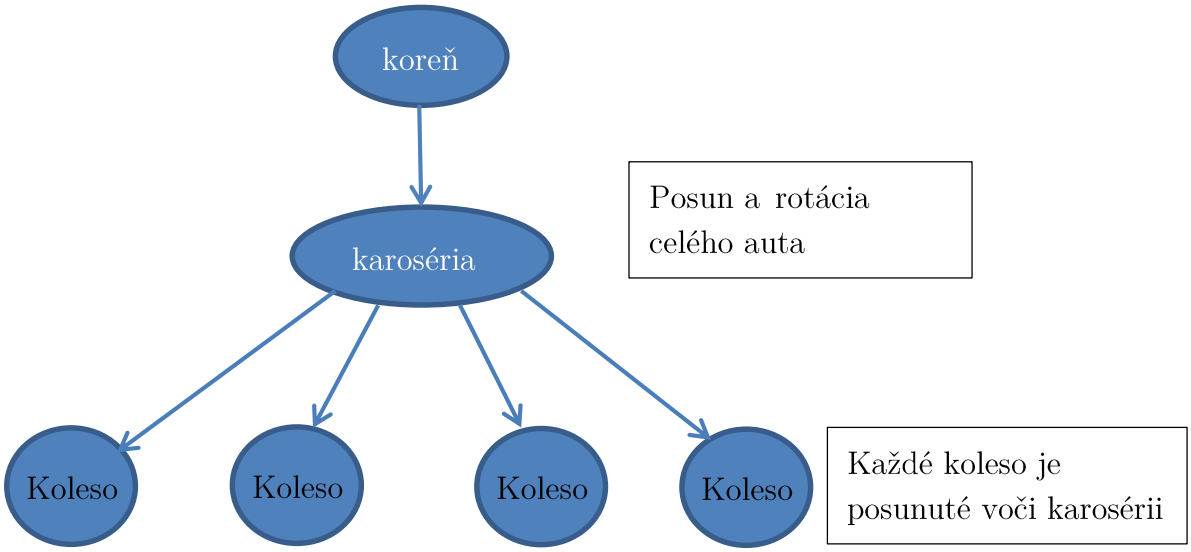
Graf scény je štruktúra, v ktorej sú usporiadané všetky objekty scény tak, že logicky a priestorovo súvisiace objekty sú zoskupené. Toto usporiadanie nám efektívne dovoľuje meniť transformácie objektov a aj jednotlivých skupín.

Graf scény je n-árny strom. Každý objekt v scéne môže mať v scéne ľubovoľný počet detí ale práve jedného rodiča. Existuje viacero variant tohto grafu. Najčastejšia variant udržiava vo vnútorných uzloch iba transformácie a v listoch stromu sa nachádzajú jednotlivé geometrické objekty.

Namiesto nastavenia transformácie každého telesa absolútne je vhodnejšie mať v uzloch transformácie hierarchicky a relatívne k predkovi daného uzlu. To nám umožňuje následnú zjednodušenú editáciu 3D scény, keď upravením transformácie jedného objektu ovplyvníme absolútne transformácie celého podstromu tohto uzlu [11].

V našom engine sme sa rozhodli pre mierne odlišnú variantu, a to takú že každý uzol bude niesť v sebe relatívnu transformáciu voči predkovi a zároveň každý uzol (aj vnútorný) môže a nemusí mať na sebe nejaký 3D geometrický objekt (Mesh).

Môžeme si to ozrejmiť na príklade jednoduchého modelu auta. Model sa bude skladať z karosérie a štyroch kolies. Kolesá majú relatívnu polohu voči pozícii karosérie. Týmto môžeme napríklad jednoducho dosiahnuť animáciu otáčania kolies. Pohyb auta po scéne dosiahneme zmenou relatívnej polohy karosérie ku koreňu scény.



Ďalšia výhoda tejto reprezentácie je možnosť vytvorenia hierarchie obálok (BVH[4]), ktorá nám môže urýchliť vykresľovanie scény. Každý objekt v scéne má svoju obálku. V našom prípade to bude osovo zarovnaný bounding box (kváder, ktorého hrany sú zarovnané s osami *x,y,z*). Uzol v scéne má taký bounding box, ktorý zahrňuje **všetkých jeho potomkov**. To nám urýchli vykresľovanie v prípade, že sa nejaká celá časť stromu nachádza mimo pohľadu kamery. To znamená, že ak sa naše auto dostane mimo pohľadu kamery, tak sa engine nebude snažiť vykresľovať karosériu vrátane kolies, čo značne ušetrí výpočtový čas na grafickej karte.

5.2 Editor

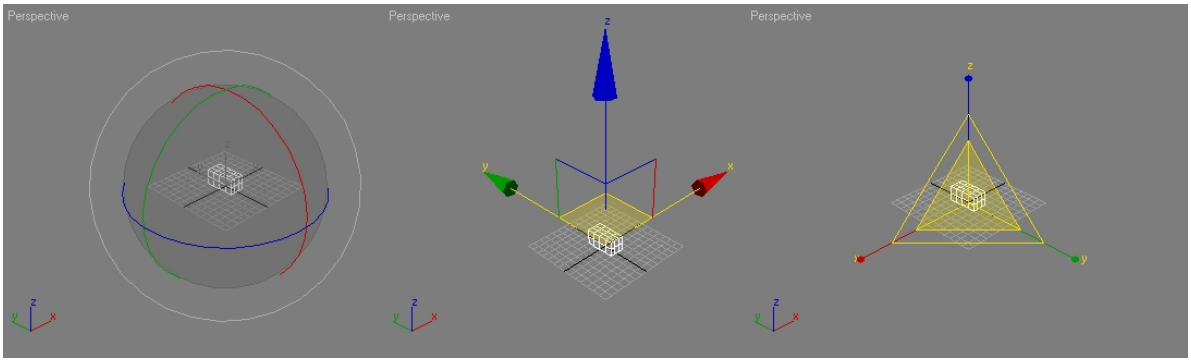
Nad spomínaným herným enginom postavíme editor, ktorý nám uľahčí množstvo práce. V editore budeme môcť pridávať a mazať objekty zo scény. Ďalej budeme môcť týmito objektmi ľubovoľne manipulovať. Keďže bude mať scéna hierarchickú štruktúru

[4] Bounding volume hierarchy – hierarchia obálok

budeme potrebovať textové vyjadrenie stromu. Každý objekt v scéne bude mať svoje vlastnosti ako farba, nastavenia materiálu a podobne. Preto budeme potrebovať časť editoru, v ktorej budeme editovať tieto vlastnosti. Poslednou časťou bude editor scenáru, ktorý bude zakreslený pomocou vhodnej reprezentácie.

5.2.1 3D časť editora

Bude existovať pohľad na scénu ktorý budeme môcť jednoducho rotovať a posúvať. Objekty si budeme môcť vyberať a jednoducho meniť ich transformácie. V tomto prípade nemá zmysel znovu vynaliezať koleso a rozhodli sme sa inšpirovať v úspešných 3D nástrojoch. Objekty budeme môcť posúvať, rotovať, a zväčšovať/zmenšovať pomocou manipulátorov (gizmo).



Obrázok 5 - Gizmá v programe 3ds Max

Na obrázku môžeme vidieť jednoduché vyjadrenie giziem v programe 3Ds Max. Prvé gizmo slúži k rotácii, druhé k posunu a tretie k zväčšovaniu/zmenšovaniu. Každé z týchto giziem pracuje v nejakej ose. Objekty na scéne posúvame ťahaním za jednu z ôs.

5.2.2 Editácia parametrov objektov

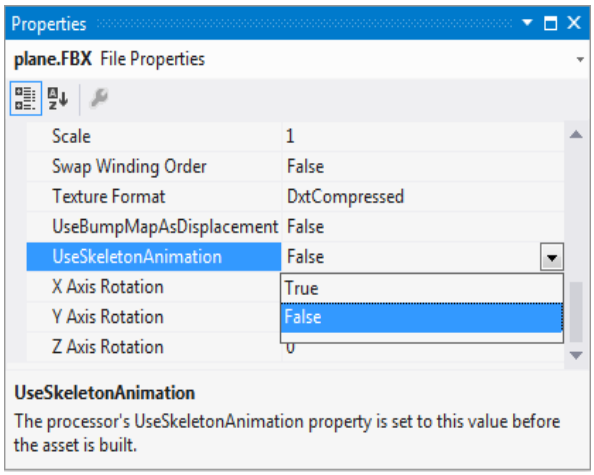
Každý z objektov v scéne má svoje parametre. Každý objekt zároveň bude môcť niesť v sebe objekty ako sú mesh, collider, kameru. Týchto kombinácii je mnoho.

Všetky objekty v scéne budú inštanciou nejakej triedy. V ediore budeme chcieť editovať práve parametre týchto objektov. Preto môžeme vytvoriť generický editor, ktorému podsunieme objekt, ktorý je inštanciou triedy a v editore zobraziť korešpondujúce vyjadrenie tohto objektu.

```
public class EngineModelProcessor : ModelProcessor
{

    [DefaultValue(false)]
    [Browsable(true)]
    public bool UseBumpMapAsDisplacement { get; set; }

    [DefaultValue(false)]
    [Browsable(true)]
    public bool UseSkeletonAnimation { get; set; }
    .
    .
    .
}
```

Kód 1

Opäť sa môžeme inšpirovať vo fungujúcich aplikáciách. Jednou z nich je Visual Studio, ktoré pre nastavenia projektu a importérov používa práve dekompiláciu tried. Toto získavanie informácií sa nazýva reflexia. Pomocou reflexie sme schopní zistiť všetky informácie o danom type. Pre naše účely bude úplne stačiť ak dokážeme získať verejné premenné a vlastnosti (property) z tried a ich typ. K týmto si následne vytvoríme potrebnú prezentáciu v editore. Môžeme si všimnúť, že Visual Studio používa pre property typu `bool` dropdown list, ktorom si užívateľ môže zvoliť True, alebo False. Pre naše účely budeme musieť vytvoriť podobný editor ako používa Visual Studio.



Obrázok 6 - Nastavenia vo Visual Studio

5.2.3 Reprezentácia scenára

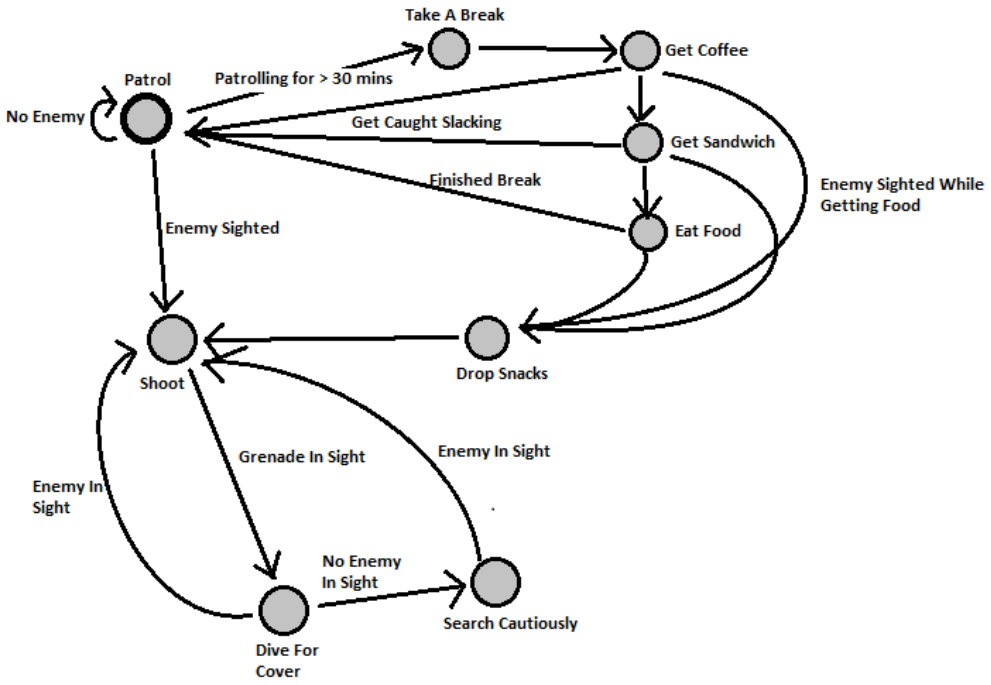
Scenár sme sa rozhodli reprezentovať konečným stavovým automatom (FSM[5]). Použitie stavových automatov v hrách nie je zriedkavé. Požívajú sa pomerne často pri kontrolovaní stavu umelej inteligencie, kde práve rozhodovanie nejakej entity s umelou inteligenciou je riadené stavovým automatom.

FSM sa skladá zo štyroch hlavných prvkov:

- Stavy,
- Prechody
- Pravidlá, ktoré musia byť dodržané, aby došlo k prechodu medzi stavmi
- Vstupné udalosti

Tento koncept môžeme použiť aj na náš scenár. V našom prípade vstupmi pre tento FSM budú akcie ktoré spraví hráč, výstupy z modelu a čas. Jednotlivé stavy budú definovať miesto v scenári, kam sa hráč dostal. Prechody budú ovplyvňované práve pomocou vstupov z tohto FSM.

[5] Finite State Machine



Obrázok 7 - Vyjadrenie AI pomocou FSM [12]

Na obrázku vidíme vyjadrenie AI nejakej postavy v hre pomocou FSM. Vidíme, že postav sa rozhoduje podľa nejakých vstupov z okolia. Toto vyjadrenie je zaujímavé aj z hľadiska nelinearity deja v hre. V určitom momente sa hráč môže rozhodnúť pre jednu z n alternatív a dej podľa toho bude pokračovať ďalej v príslušnej časti FSM.

Veľkou výhodou tejto reprezentácie je okamžitá vizualizácia, kde momente návrhu scenára vidíme možné prechody týmto scenárom.

Kapitola 6

Implementácia

V tejto kapitole popíšeme základne triedy vytvoreného 3D enginu a editora postaveného nad týmto enginom. Ako sme spomínali, použitá platforma bude Microsoft Silverlight.

6.1 Základ 3D enginu

Základným prvkom pre vykresľovanie 3D obsahu v Silverlighte je komponenta `DrawingSurface`. Obmedzením je, že táto komponenta môže byť len jedna vo vizuálnom strome aplikácie. `DrawingSurface` nám ponúka všetko, čo potrebujeme pre vytvorenie tzv. hernej slučky. Kontrétne sú to tieto 2 prvky:

- event `Draw` - v event handleri pre túto eventu môžeme vykresľovať primítíva do plochy `DrawingSurface.`
- metóda `Invalidate` – vynúti ďalšie zavolanie eventy `Draw.`

Takto vieme vytvoriť metódu, ktorej volanie sa bude pravidelne opakovať. `DrawingSurface` nám ponúka samozrejme eventy pre spracovanie vstupu z klávesnice a myši. Samotné vykresľovanie primitív prebieha pomocou metód triedy `GraphicsDevice`.

6.1.1 Trieda `EngineSurface`

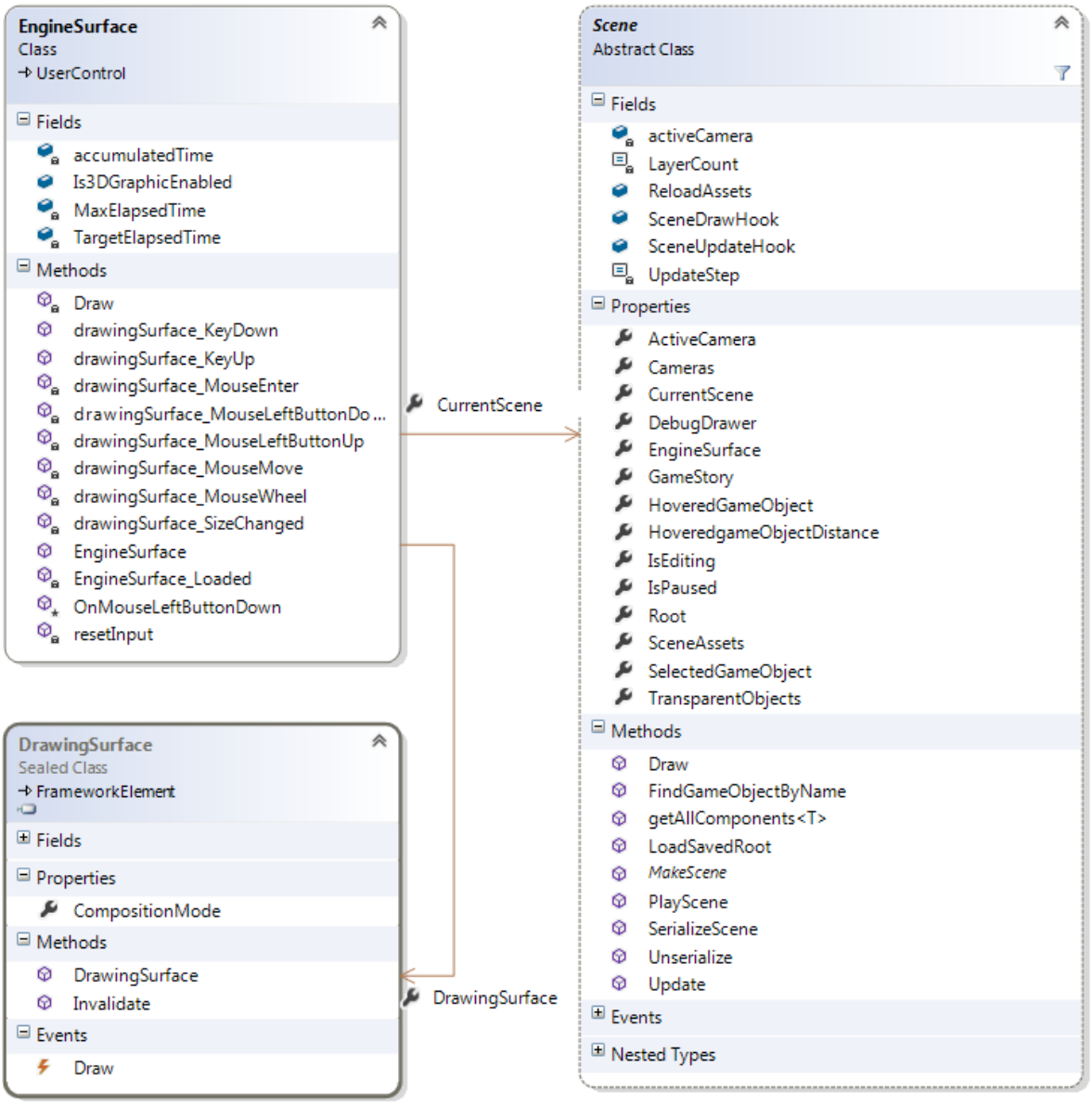
Táto trieda je obalovou triedou pre `DrawingSurface.` Obsahuje event handlery pre všetky potrebné vstupy z klávesnice a myši. Informácie o vstupe zapisuje do statickej triedy `Input.` Metóda `Draw` je implementáciou event handleru z eventy `Draw`, ktorá je obsiahnutá v `DrawingSurface`. V tejto metóde sa deje niekoľko vecí. Na začiatku tejto metódy sa nastavia správne stavy pre grafickú kartu. Ďalej náš engine umožňuje tzv. fixed Update Time. To umožňuje volanie metódy `Update` zo scény tak, aby sa vykonala napr. presne 60 krát za sekundu. V tejto metóde budú implementované veci ako animácie, fyzika a Fixed Update Time nám zaručí rovnakú rýchlosť týchto animácií pri hocijakom FPS.

Táto metóda sa zavolá toľko krát, aby platilo spomínane pravidlo. Potom sa zavolá samotné vykresľovanie primitív zo scény. Nakoniec zavoláme metódu `Invalidate`.

6.1.2 Trieda `Scene`

Táto trieda je abstraktná a vyžaduje jej rozšírenie. Obsahuje základnú funkcionalitu pre zobrazenie, ukladanie a manipuláciu so scénou. Základné a podstatné časti tejto triedy:

- `ActiveCamera` – obsahuje odkaz na kameru, ktorou momentálne renderujeme scénu.
- `GameStory` - vyjadrenie scenáru pomocou FSM
- `GameAssets` – má v sebe uložené jednotlivé modely, textúry, ktoré používame v scéne
- `Root` – odkaz na koreň stromovej štruktúry objektov v scéne
- `SelectedObjectChanged` – event, ktorý sa zavolá ak sme v scéne klikli na iný objekt
- `SerializeScene, Deserialize` – metódy pre uloženie/načítanie scény
- `TransparentObjects` – list objektov v scéne, ktoré majú priehľadný materiál



Obrázok 8 - Diagram zobrazujúci triedy DrawingSurface, EngineSurface a Scene

Scéna obsahuje metódy `Draw` a `Update`. V metóde `Draw` sa najprv vykreslia všetky nepriehľadné objekty zo stromovej štruktúry. Následne prepneme stav na grafickej karte tak, aby sme vypli zápis do Z-Bufferu, ktorý obsahuje hĺbku jednotlivých pixelov v priestore. Potom začneme kresliť priehľadné objekty a to tak, že ich kreslíme v poradí od najvzdialenejších po najbližšie pomocou alfa blendingu.

Tie objekty (resp. ich časti), ktoré sa nachádzajú za solídnymi sa nevykreslia vôbec. Zoradenie objektov nám zabezpečí správny výstup, v prípade, že máme niekoľko transparentných objektov v scéne za sebou.

6.1.3 Trieda GameObject

`GameObject` je základným stavebným prvkom enginu. Inštanciu tejto triedy budeme nazývať game objekt. Hierarchia týchto game objektov tvorí graf scény. Každý `GameObject` má niekoľko detí a práve jedného rodiča. `GameObject` je zároveň kontajner pre komponenty, ktoré určujú výzor a správanie sa tohto game objektu. Základne časti tejto triedy sú:

- `Children` – list detí tohto GameObjectu
- `Parent` – rodič tohto GameObjectu
- `Components` – list komponent
- `Transform` – určuje relatívnu transformáciu k predkovi
- `Bounds` – osovo zarovnaný bounding box, určuje obálku celého podstromu tohto objektu
- `Draw` – metóda, ktorá vykreslí tento objekt. V skutočnosti len volá metódu `Draw` z jednotlivých komponent. Potom sa volá táto metóda z potomkov tohto objektu.
- `Update` – volá metódu `Update` z jednotlivých komponent a potom zavolá `Update` svojich potomkov
- `OnSelect`, `OnHover` – eventy, ktoré sa zavolajú, v prípade, ak označíme, resp. položíme myš nad daný objekt v scéne.
- `Enabled` – ak je tato premenná `false`, tak sa celý podstrom daného game objektu nebude vykresľovať ani aktualizovať
- `Active` – značí, čie je daný objekt aktívny, hodnota je `false`, ak sa game objekt nachádza v podstrome game objektu, ktorý ma `Enabled` nastavené `false`

Ak máme v dispozícii koreň stromu scény, tak stačí zavolať jeho `Update`. To spôsobí postupné volanie tejto metódy na všetkých potomkoch tohto koreňu. Zavolaním metódy `Draw` dosiahneme vykreslenie celého podstromu.

Táto trieda obsahuje logiku pre klikanie na objekt. V prípade kliknutia na objekt to dá vedieť všetkým komponentám a zavolá príslušné eventy. Game objekt si zároveň počíta veľkosť bounding boxu.

6.1.4 Trieda Transform

Táto trieda reprezentuje pozíciu game objektu v priestore. V každej metóde `Update` z game objektu sa prepočíta absolútna transformácia objektu pomocou absolútnej transformácie jeho predka a svojich lokálnych transformácií.

```
public void ComputeAbsoluteTransform()
        {
            if (GameObject.Parent != null)
                AbsoluteTransform = Matrix.CreateScale(Scale) * Rotation *
Matrix.CreateTranslation(Position) *
GameObject.Parent.Transform.AbsoluteTransform;
            else
                AbsoluteTransform = Matrix.CreateScale(Scale) * Rotation *
Matrix.CreateTranslation(Position);
         }
```

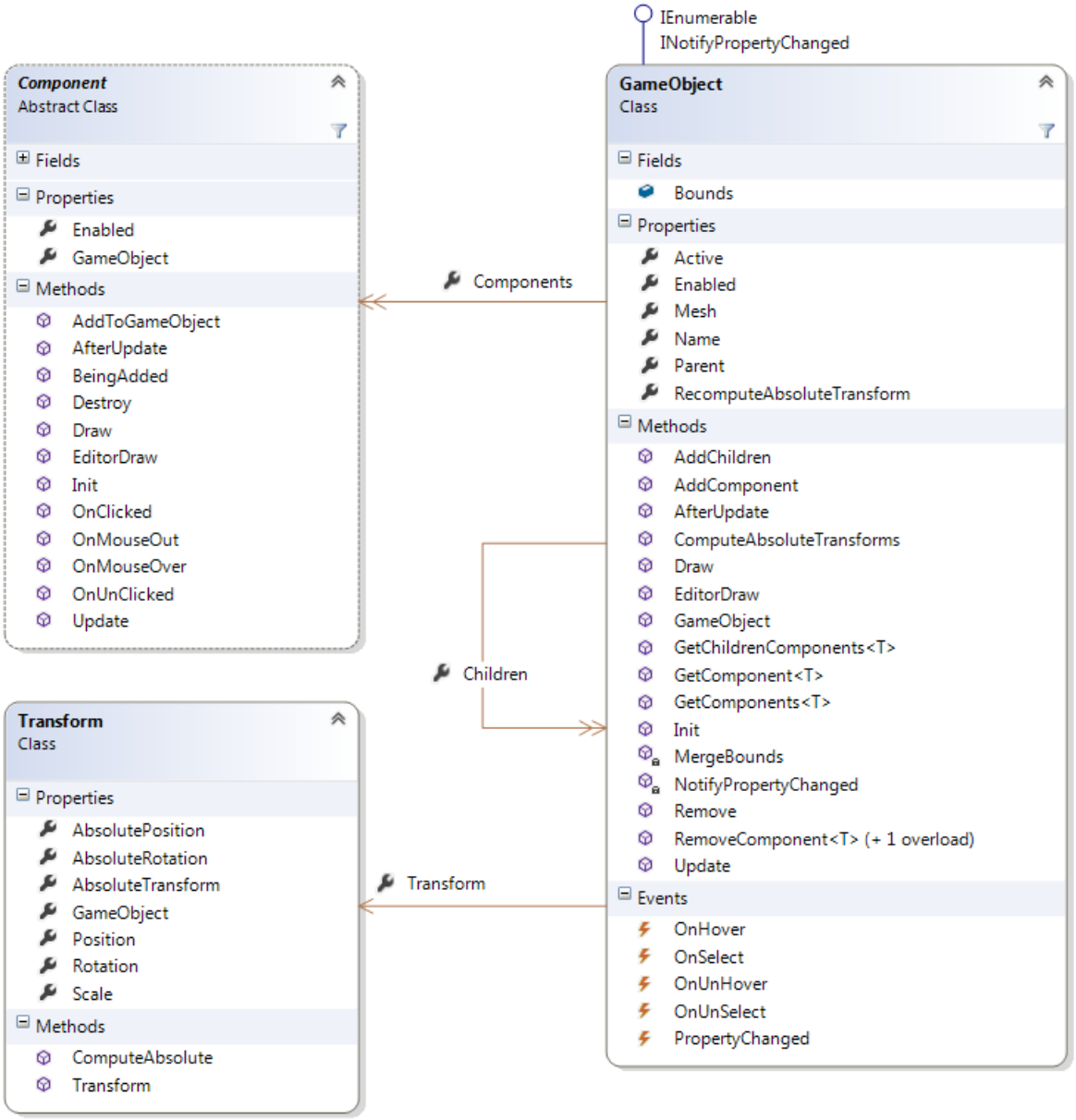
Kód 2 - Počitanie absolutných transformácií

`AbsoluteTramsform` je matica veľkosti 4x4. Ako vidieť z kódu, jej hodnota sa vyráta postupným násobením matíc, ktoré určujú zväčšenie, rotáciu, pozíciu a absolútnu transformáciu predka. Táto trieda nám zároveň dovoľuje zistiť a nastaviť absolútnu pozíciu a absolútnu rotáciu objektu.

6.1.5 Trieda Component

Táto trieda je v engine jednou z najpodstatnejších. Inštancie potomkov tejto triedy budeme nazývať komponenty. Samotná trieda `Component` je abstraktná a má

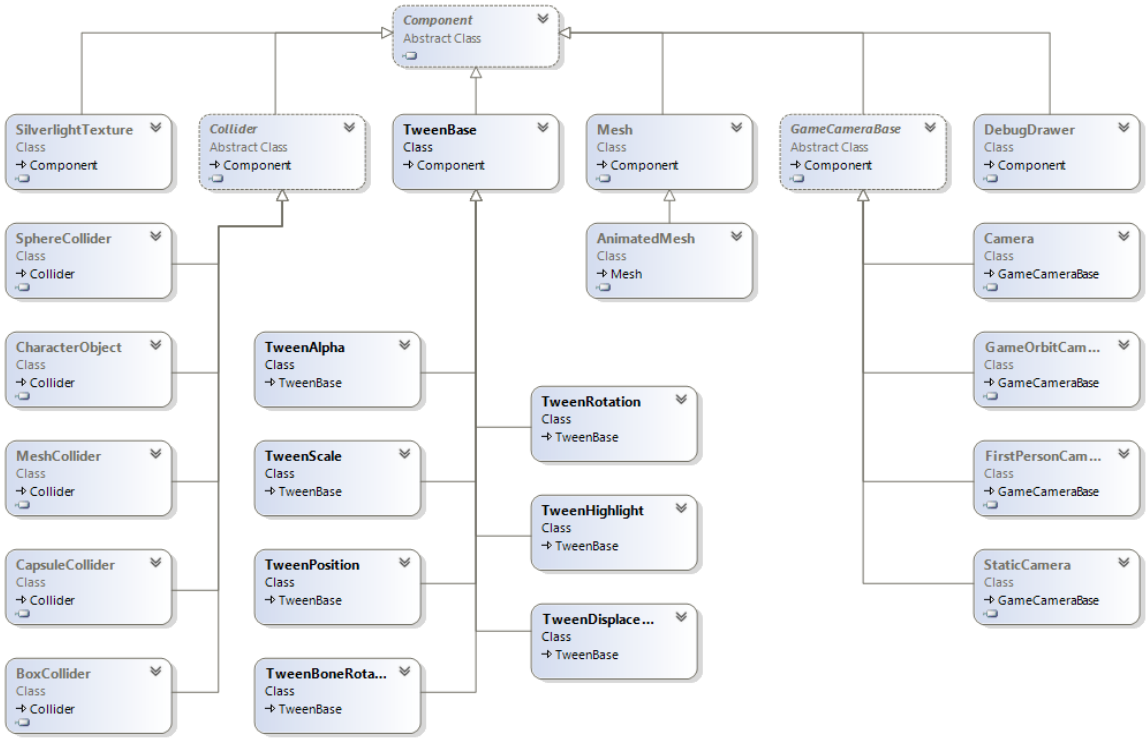
úzke prepojenie s game objektom. Každý game objekt môže obsahovať niekoľko komponent. Tie určujú výzor a správanie sa daného objektu v scéne. Tieto komponenty sa budú na objekty pridávať v editore. Ako príklad si môžeme uviesť mesh, fyzikálny collider, alebo animáciu objektu.



Obrázok 9 - Triedy tvoriace graf scény

Konkrétna komponenta vzniká rozšírením triedy `Component` a implementáciou niekoľkých základných metód. Je nutné podotknúť, že komponenta musí obsahovať bezparametrický konštruktor, aby bolo možné komponentu vytvoriť v editore. To nám zároveň umožní aj jednoduchšiu serializáciu. Základné metódy sú:

- `Update` – implementáciou tejto metódy dostávame reálnu funkcionalitu komponenty. V tejto metóde implementujeme väčšinu logiky komponenty.
- `Draw` – v tejto metóde by malo byť implementované vykresľovanie primitív na obrazovku.
- `AfterUpdate` – je volaná po tom ako sa vykonal update na všetkých objektoch a komponentách v scéne. Táto metóda je napríklad vhodná na zisťovanie informácie, či sme v `Update` objektom pohli, a podobne.
- `BeingAdded` – metóda sa zavolá po tom, ako pridáme inštanciu komponenty na nejaký game objekt. Slúži na nastavenie dodatočných verejných parametrov komponenty, závislých na konkrétnom game objekte (v konštruktore ešte nepoznáme game objekt).
- `Init` – zavolá sa v momente ak načítame scénu a začneme ju prehrávať. Metóda bude slúžiť k nastaveniu dodatočných privátnych parametrov ktoré komponenta vyžaduje pre svoju funkcionalitu.



Obrázok 10 - Diagram základných komponent

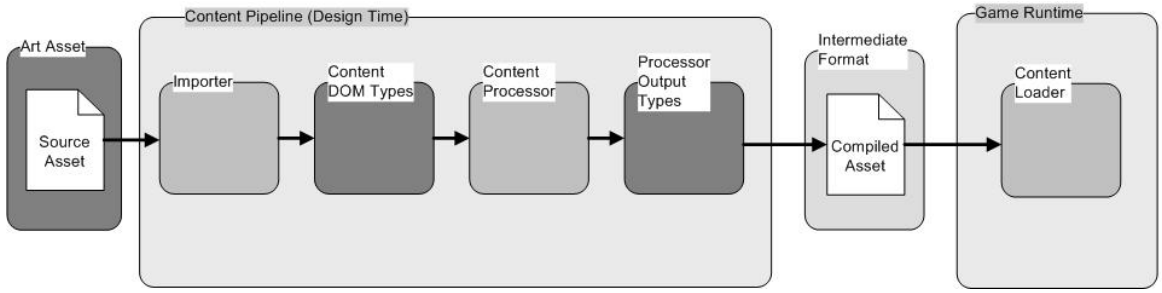
Na obrázku Obrázok 10 môžeme vidieť diagram tried niektorých komponent z enginu. Spomenieme štyri základné:

- `Mesh` - reprezentuje tvar 3D objektu
- `GameCameraBase` – reprezentuje kameru
- `TweenBase` – základ animácií
- `Collider` – tvar objektu použitý pre fyziku

6.1.6 Importovanie obsahu hry

XNA je schopné importovať množstvo súborov (assetov). Môžeme importovať 3D modely vo formáte *.fbx a *.x, ďalej obrázky vo formáte *.bmp,*.png, zvuky vo formáte *.wav, *.mp3, *.wmv. Autori frameworku XNA sa rozhodli, že importovanie týchto formátov bude prebiehať počas návrhu. Dôvodom pre toto rozhodnutie bolo to, aby samotná hra (game runtime) nemusela referencovať knižnice pre načítanie spomenutých súborov [13].

Takto vznikla takzvaná content pipeline. Každá XNA hra je vo visual studiu prepojená s content pipeline projektom. Tento projekt obsahuje súbory (assety) ktoré chceme v hre používať. Kompiláciou tohto projektu nám vznikne serializovaná podoba runtime objektov, ktoré budeme v hre používať.

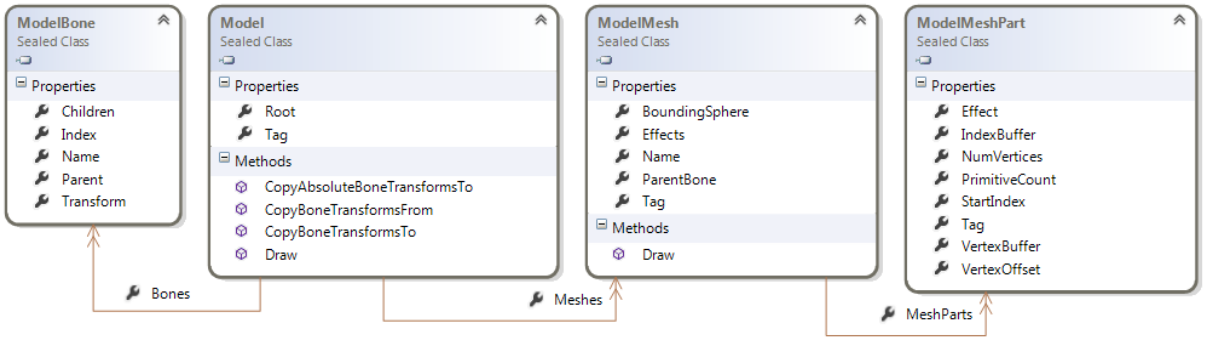


Obrázok 11 - Schématické znázornenie importovania assetu [14]

XNA nám dovoľuje pre každý súbor napísať vlastný importér, alebo použiť nejaký externý aj v natívnom jazyku ako je napr. C++. Tento importér vytvorí nejakú hierarchickú štruktúru. Nad ňou potom pracuje content processor, ktorý z tejto štruktúry vyrobí objekty kompatibilné s runtime typmi. Tieto objekty sú následne serializované do formátu XNB. Content processor môžeme takisto rozširovať o vlastnú funkcionalitu. Runtime obsahuje len knižnice na deserializáciu tohto formátu.

6.1.7 Importovanie 3D modelov

Naše modely máme vo formáte FBX, ktoré dokáže XNA importovať. Problém nastáva v prípade obmedzení v Silverlighte. Ako bolo spomenuté v kapitole s možnými technológiami, tak Silverlight nedokáže získať dáta zo štruktúr, ktoré sú určené pre grafickú kartu. To znamená, že sa nevieme dostať k dátam z Vertex a Index bufferu, ktoré udržujú pozície vertexov v priestore a ich prepojenie, ktoré tvorí trojuholníky.



Obrázok 12 - Štruktúra 3D modelu v XNA

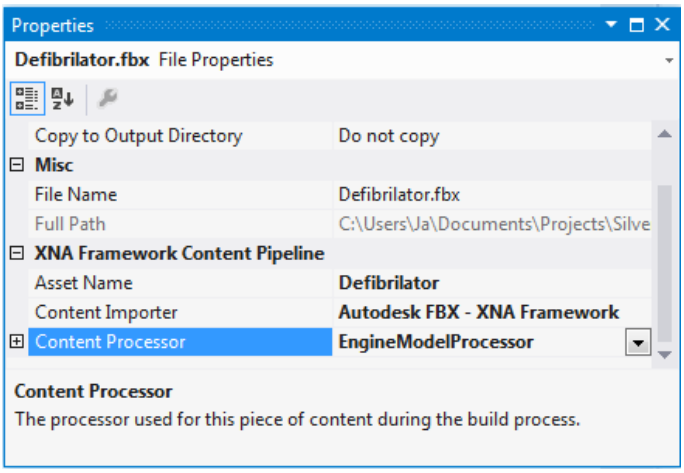
Základnou entitou pre zobrazenie 3D modelu v XNA je trieda `Model`. `Model` obsahuje hierarchickú štruktúru kostí. Na každej z týchto kostí môže a nemusí byť pripevnený `ModelMesh`. Tento `ModelMesh` obsahuje zoznam jeho častí – `ModelMeshPart`. Tie už obsahujú Index a Vertex Buffer, ktoré sa posielajú na grafickú kartu.

Spomínané triedy obsahujú atribút `Tag`, ktorý je typu `Object`. Rozšírením Content processoru môžeme do tohto atribútu uložiť ďalšie informácie, ktoré o 3D modely potrebujeme.

V našom engine vzniklo rozšírenie triedy `ModelProcessor`, ktoré pre každý `ModelMesh` uloží do jeho atribútu `Tag` slovník, ktorý obsahuje:

- `BoundingBox` – obálka danej časti modelu
- `Vertices` – list vertexov v priestore
- `Indices` – indexy vertexov v predchádzajúcom liste, kde každá trojica určuje trojuholník

Takto sme schopní sa v hre dostať k potrebným dátam. Tie budeme potrebovať v prípade zisťovania udalostí ako je kliknutie na 3D objekt, alebo v prípade vytvárania fyzikálnych colliderov daného objektu.



Obrázok 13 - Nastavenie importeru a procesoru pre model Defibrilator.fbx

6.1.8 Trieda Mesh

`Mesh` je rozšírením triedy `Component` a jeho úlohou je vykresliť danú časť modelu na správnom mieste a poskytovať API pre získanie informácií o kliknutí myši.

Keďže `Model` má podobnú hierarchiu ako naša hierarchia game objektov, tak môžeme pomerne jednoducho z modelu vytvoriť strom game objektov s korešpondujúcimi transformáciami, ako majú kosti v triede `Model`. Každá kosť má svojho predka a zároveň relatívnu transformáciu k nemu. Túto transformáciu jednoducho rozložíme na pozíciu, rotáciu, a zväčšenie pomocou matematických knižníc z XNA.

Trieda `Mesh` teda obsahuje referenciu na inštanciu triedy `ModelMesh`. Ak použijeme opäť príklad s autom, tak v prípade správneho vymodelovania auta 3D grafikom budeme schopní zvlášť manipulovať s karosériou a zvlášť kolesami. `Mesh` obsahuje zároveň list materiálov, ktoré korešpondujú materiálmi, ktoré boli v 3D modelovacom nástroji nastavené pre každú časť modelu.

Materiál obsahuje základne parametre ako sú: priehľadnosť, základná farba, farba zrkadlového odrazu, textúra. Tieto parametre budeme môcť následne jednoducho meniť v editore, alebo animovať počas hry.

6.1.9 Ukladanie scény

Kvôli veľkosti uložených scén sme sa rozhodli pre serializér, ktorý bude vytvárať binárne súbory. Bola použitá externá open sourcová implementácia serializéru [15]. Tá je založená na reflexii a ukladá do súboru všetky verejné vlastnosti a premenné serializovaných inštancií. Výhodou tohto serializéru je možnosť preťažiť serializáciu niektorých typov a jeho rýchlosť.

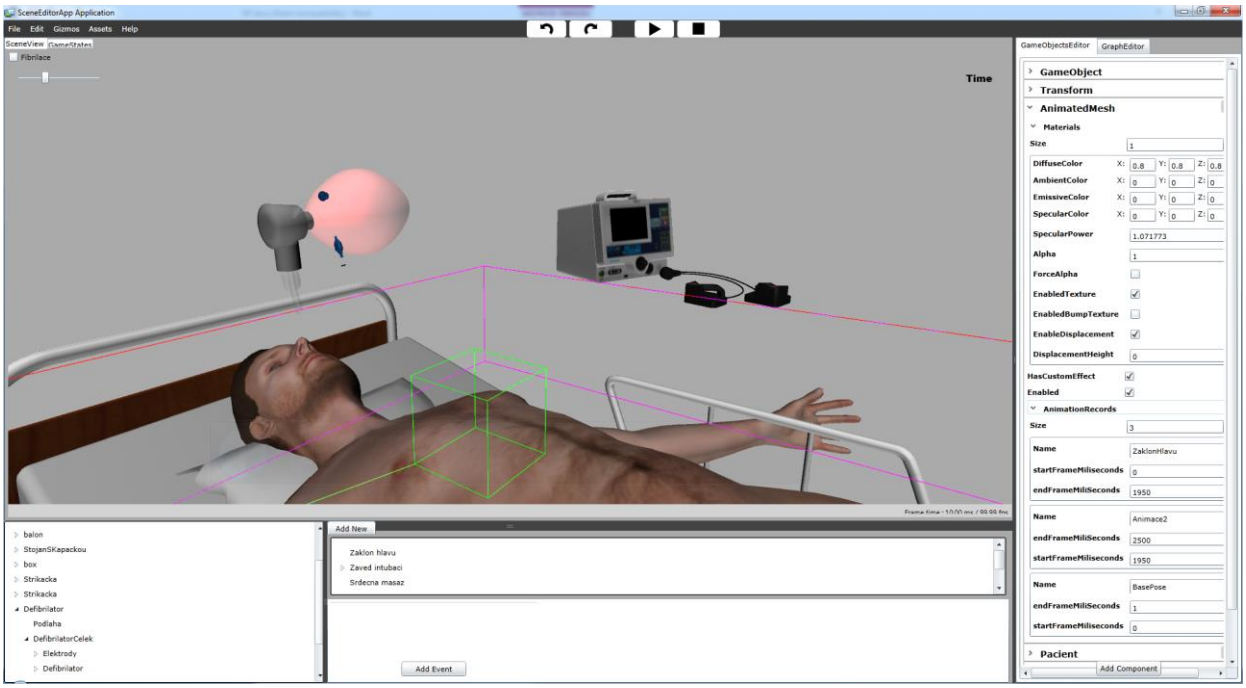
Náš engine obsahuje kód pre serializáciu typov z XNA ako sú `Mesh` a `ModelMesh`. Ako bolo spomenuté v predchádzajúcej kapitole, kópie týchto inštancií máme uložené v contnent projekte vo formáte xnb. Preto pri serializácii týchto tried ich nepotrebujeme serializovať znovu, ale len načítať ich inštanciu z xnb súboru.

Z tohto dôvodu vznikla trieda `AssetDatabase`, v ktorej máme uložené mená a inštancie všetkých potrebných assetov, ktoré sa v scéne používajú. A pri deserializácii scény sa najprv načítajú tieto assety. To znamená, že serializovaná podoba triedy `ModelMesh` obsahuje len jej kľúč v slovníku `AssetDatabase`.

6.1.10 Implementácia fyziky

Pre implementáciu fyziky bol vybraný open source projekt JigLibX [16]. Obsahuje všetku potrebnú funkcionalitu, ktorá by sa nám mohla zísť. Obsahuje väčšinu potrebných kolíznych plôch ako sú guľa, kváder, kapsula a mesh. Dokáže optimálne riešiť kolízie medzi spomínanými entitami. Tento fyzikálny engine bol mierne prispôsobený a vznikli komponenty, ktoré reprezentujú spomínané kolízne plochy.

6.2 Editor



Obrázok 14 – Vytvorený editor

Nad vytvoreným enginom vznikol editor, v ktorom je implementovaná základná funkcionalita pre editovanie scény a scenára. Základnou funkcionalitou editoru je vytvorenie takejto scény a následne testovanie scenára. Preto editor umožňuje prehrať navrhnutú scénu so scenárom. Editor môžeme rozdeliť na niekoľko častí.

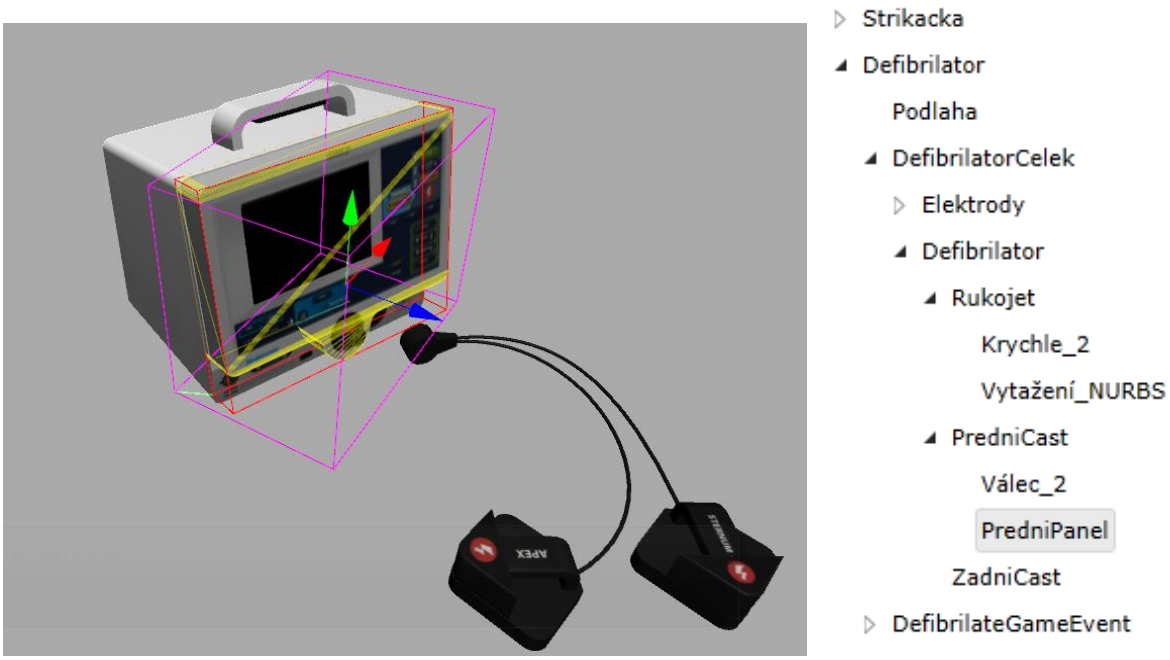
6.2.1 3D pohľad na scénu

Tento pohľad umožňuje rozmiestňovanie objektov po scéne a ich rôznu transformáciu pomocou giziem. Zároveň v ňom môžeme označovať objekty. Označením objektu sa zvýrazní jeho obrys a ružovou farbou sa zobrazí osovo zarovnaný bounding box. Editor nám umožňuje pohľad rotovať a posúvať pomocou myši:

- **Stlačením stredného tlačidla myši + ťahanie myšou** – rotovanie pohľadu
- **Stlačením stredného tlačidla myši + ľavý Alt + ťahanie myšou** – posúvanie kamery v rovine kolmej na pohľad

Výber giziem je možný v hlavnom menu, alebo pomocou skratiek:

- **Q** – gizmá sú vypnuté
- **W** – gizmo pre transláciu
- **E** – gizmo pre rotovanie
- **R** – gizmo pre zväčšenie/zmenšenie

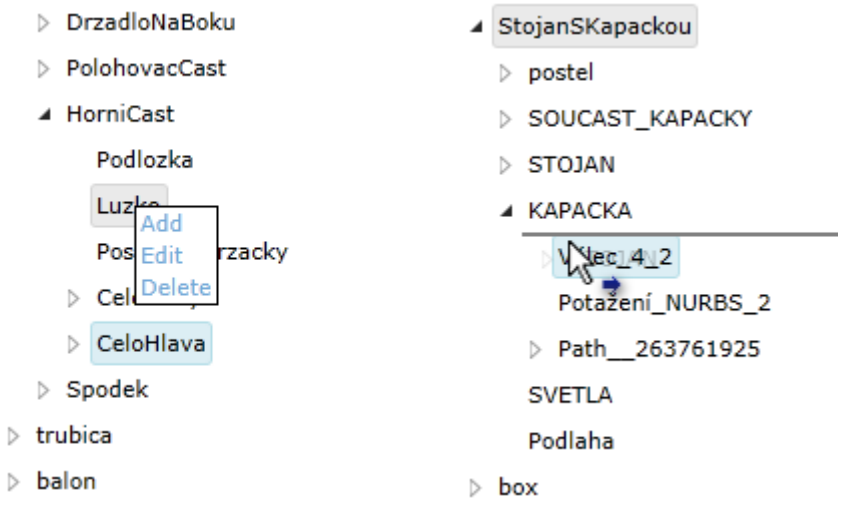


Obrázok 15 - Zobrazenie objektu v 3D pohľade a v grafe scény

6.2.2 Zobrazenie grafu scény

Dôležitou časťou editora je zobrazenie grafu scény v textovej stromovej podobe pomocou tzv. Treeview. V tomto pohľade môžeme tak isto zvoliť objekt, ktorý chceme editovať. Tento pohľad korešponduje s 3D pohľadom na scénu. V prípade, že klikneme na objekt v 3D pohľade, tak sa označí aj v Treeview a naopak. Na predchádzajúcom obrázku si môžeme všimnúť štruktúru defibrilátoru, tak ako ju vymodeloval 3D grafik.

Treeview nám umožňuje pomocou drag&drop meniť hierarchickú štruktúru game objektov a pomocou kontextového menu pridávať a mazať game objekty zo scény. Za túto funkcionaliu môžeme vďačiť knižniciam z Silverlight toolkit.



Obrázok 16 - Kntextové menu a drag&drop funkcia

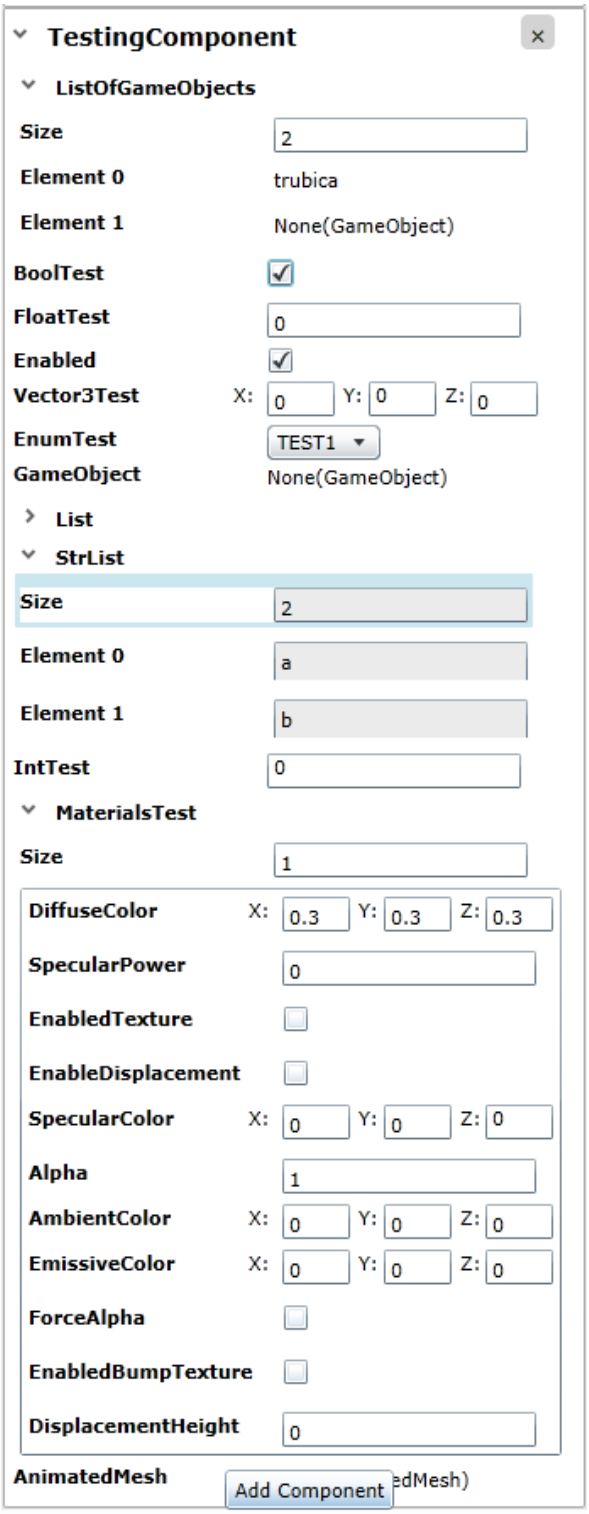
6.2.3 Editovanie game objektov

Hlavnou časťou editora je editor game objektov, kde môžeme nastavovať jednotlivým game objektom jednak vlastnosti, ale aj prepojenia medzi nimi. Zároveň nám umožňuje mazať a pridávať týmto game objektom komponenty.

Editor nám umožňuje editovať verejné parametre a vlastnosti (property) inštancie nejakej triedy. Podporované sú základné dátové typy `bool`, `int`, `float`, `string`. Ďalej typy z knižnice XNA: `Vector3`, `Vector2`. Ďalej môžeme editovať hodnotu výčtového typu `enum`. Z typov patriacich do enginu môžeme editovať referenciu na nejakú komponentu, alebo game objekt. Špeciálnym prípadom je materiál, u ktorého editujeme jeho parametre. Editor podporuje editovanie listov spomínaných dátových typov.

V neposlednom rade nám táto časť editoru umožňuje pridávať a mazať jednotlivé komponenty z game objektov.

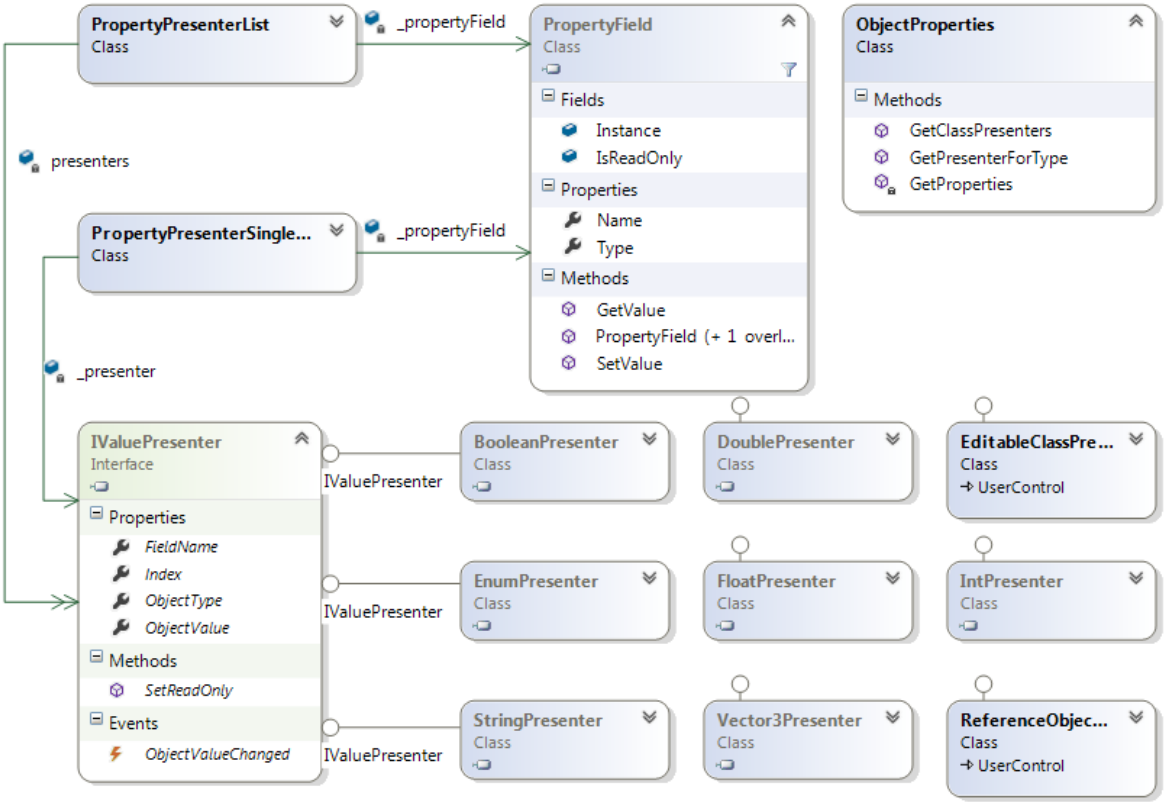
```
public class TestingComponent :
Component
{
    private bool _boolTest = true;
    public enum MyEnum
    {
        TEST1,
        TEST2,
        TEST3,
        TEST4,
    }

    public bool BoolTest
    {
        get { return _boolTest; }
        set
        {
            if (value != _boolTest)
            {
                _boolTest = value;

Debug.WriteLine("Changing
                    value to:" + value);
            }
        }
    }

    public float FloatTest
    {
        get;
        set;
    }
    public Vector3 Vector3Test;
    public MyEnum EnumTest;
    public GameObject GameObject;

    public List<GameObject>
ListOfGameObjects
    {
        get;
        set;
    }
    public List<float> List;
    [ReadOnly]
    public List<string> StrList;
    public int IntTest;
    public List<Material>
MaterialsTest;
    public AnimatedMesh AnimatedMesh;
```



Obrázok 17 - Vyjadrenie triedy pomocou kódu a jej zobrazenie v editore

Na predchádzajúcom obrázku je znázornený editor vlastností pre jednu konkrétnu komponentu `TestingComponent`. Základné dátové typy sú vyjadrené pomocou textového poľa, ktoré akceptuje daný formát. Typ `bool` je vyjadrený pomocou zaškrtávatka. Enum zobrazujeme pomocou otváracieho sa listu.

Typy `GameObject` a `Component` sú vyjadrené textovou podobou a v prípade, že sú nenulové, tak editor zobrazí ich meno. Referenciu na iný game objekt v hre vytvoríme tak, že vezmeme game objekt v Treeview a pomocou drag&drop ho pretiahneme na príslušné miesto v editore. Podobné je to s komponentami. Do komponenty môžeme pretiahnuť taký game objekt ktorý obsahuje komponentu potrebného typu.



Obrázok 18 - Diagram tried editora pre editáciu vlastností

Základným stavebným kameňom pre editovanie parametrov je trieda `PropertyField`. Jedna inštancia tejto triedy korešponduje s jednou premennou, resp. vlastnosťou konkrétnej inštancie objektu. Obsahuje potrebné informácie ako je jej meno a typ a referenciu na inštanciu objektu. Pomocou metód `SetValue` a `GetValue` sme schopní nastaviť resp. získať hodnotu danej premennej danej inštancie.

Pomocou statickej triedy `ObjectProperties` získame použitím reflexie zoznam týchto `PropertyField` pre editovanú inštanciu objektu. Z tohto zoznamu následne vytvorí zoznam 2D Silverlight kontroliek, ktoré sa zobrazia v editore.

Pre každý podporovaný typ existuje vlastná kontrolka, ktorá zobrazí a umožní editáciu jeho hodnoty. Tieto kontrolky implementujú rozhranie `IValuePresenter`. Toto rozhranie obsahuje všetko, čo potrebujeme pre editáciu hodnoty nejakej premennej a to jej typ, hodnotu, meno a v prípade, že sa nachádza v liste, tak aj jej index. Rozhranie definuje eventu, ktorá sa zavolá z kontrolky v prípade, že užívateľ zmenil hodnotu.

Ďalšou časťou sú `PropertyPresenterSingleValue` a `Propertypresenter List`. Tieto triedy obsahujú referenciu na `PropertyField` a potrebné/ú 2D kontrolku. Zabezpečuje zisťovanie zmeny hodnoty uloženej v editovanom objekte a následnú zmenu hodnoty v 2D kontrolke/ách a naopak. Tento prístup umožňuje zobrazovanie aktuálnych hodnôt v editore aj keď sa s 3D objektom niečo deje (napríklad posúvanie, alebo animácia).

Takto môžeme editovať parametre ľubovoľných inštancií tried, ktoré budeme v hre používať. Tento prístup použijeme aj v prípade editovania parametrov modelikových modelov a editovania scenára.

Jazyk C# nám dovoľuje použiť pre jednotlivé elementy aplikácie takzvané atribúty. Jedná sa o dodatočné informácie, ktoré môžu slúžiť napríklad kompilátoru, serializéru a podobne. Pre funkcionalitu editoru bola vytvorená sada týchto atribútov:

- `[HideInEditor]` – Použitie pred vlastnosťou, alebo premennou triedy. Daný člen nebude viditeľný v editore aj napriek tomu, že je verejný.

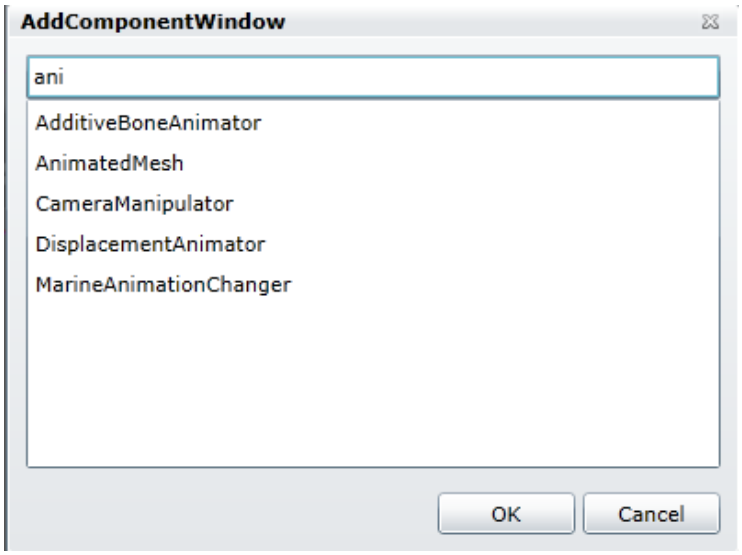
- `[ReadOnly]` – Použitie pred vlastnosťou, alebo premennou triedy. Daný člen bude v editore viditeľný, ale nebudeme ho môcť editovať. Príklad na obrázku č.Obrázok 17, kde property `StrList` nemôžeme editovať.
- `[IsEditable]` – Použitie pred triedou. Tento atribút určuje spôsob zaobchádzania s triedou v editore. Ak je pred triedou tento atribút, tak editor ďalej rozoberie inštanciu tejto triedy a zobrazí ju. Príkladom je trieda `Material`, pre ktorú nemáme definovaný zvlášť presenter, ale obsahuje tento atribút. Takto môžeme editovať parametre materiálu ako to je zobrazené na obrázku č.Obrázok 17.

Takto môžeme čiastočne na úrovni kódu definovať ako bude vyzerať editor danej triedy, najčastejšie komponenty.

Výhodou je možnosť editovania vlastností (property). Tá sa navonok chová ako členská premenná, ale v implementácii nám umožňuje pridanie logiky. Príkladom je `property BoolTest` z obrázku č.Obrázok 17, ktorá v momente keď zmeníme jej hodnotu, tak vypíše text do konzoly. Tento prístup umožňuje programovanie funkcionality v editore. Booleovská vlastnosť sa môže napríklad takto chovať ako tlačidlo, kde v jej logike **nezmeníme** jej hodnotu, ale zavoláme nejakú funkciu.

Nevyhnutnou časťou editora je možnosť pridávania a mazania komponent z game objektov. Mazanie prebieha jednoducho pomocou tlačidla v editore pri každej komponente. Pridávanie komponent funguje pomocou jednoduchého dialógu po kliknutí na tlačidlo Add Component. V tomto dialógu sa zobrazí zoznam všetkých tried, ktoré dedia od triedy Component. Dialóg obsahuje jednoduchý filter, v ktorom môžeme filtrovať komponenty podľa názvu triedy.

Editor následne vytvorí inštanciu danej komponenty a pridá ju do listu komponent označeného game objektu. Z tohto princípu vyplýva podmienka bezparametrického konštruktora danej komponenty. Inak by editor nevedel vytvoriť potrebnú inštanciu. Počet a typ komponent na jednom game objekte **nie je** obmedzený.



Obrázok 19 - Pridávanie komponenty

6.2.4 Ďalšia funkcionalita editora

Podstatnou funkcionalitou, ktorú editor podporuje je možnosť spustiť si danú scénu. Aby bolo scénu možne editovať rozumným spôsobom, tak v editačnom móde sú všetky komponenty v scéne pozastavené. To umožňuje nastaviť počiatočné rozmiestnenie scény a počiatočné parametre objektov, ktoré sa v priebehu hry potom môžu zmeniť na základe užívateľových vstupov a akcií, ktoré vykoná.

Pozastavenie komponent je jednoduché. V editore sa **nezavolá** funkcia Update, v ktorej je sústredená funkcionalita komponenty. Takto môžeme editovať scénu, ktorá sa tvári staticky a po jej spustení budú všetky komponenty aktívne, t.j všetky animácie a objekty používajúce fyziku budú aktívne až pri prehrávaní scény.

Táto funkcionalita je nevyhnutná, ale dala by sa považovať aj za nevýhodu tohto editora. Preto by bolo možné v ďalších verziách editoru povoliť vykonávanie metódy `Update` pre niektoré komponenty. Užitočné by to bolo hlavne u animácií, kde by sme nemuseli zakaždým spustiť danú scénu.

V momente spustenia scény si editor uloží jej serializovanú podobu. Následne môžeme scénu testovať. Aktívne sú všetky komponenty a vykonáva sa na nich metóda `Update`. V momente, keď je scéna spustená, tak môžeme editovať parametre jednotlivých game objektov a komponent. 3D pohľad editora na scénu sa zmení na

herný pohľad a funkcionalita označovania objektov v 3D pohľade je pozastavená. Objektami potom **nemôžeme** hýbať pomocou giziem.

Po skončení testovania klikneme na tlačidlo Stop. Do editora sa nahrá scéna, ktorú sme uložili pred jej spustením. Takto sa všetky zmeny, ktoré boli vykonané v stave spustenia zrušia.

Počas testovania editora sa ukázalo ako nevyhnutná podpora undo a redo. Ak sa užívateľ preklikne, alebo omylom posunie nejaký objekt, tak bolo preňho komplikované vrátiť tieto zmeny naspäť. Momentálne editor podporuje vrátenia zmien vykonaných pomocou giziem a v 2D editore game objektov, v ktorom je podporovaná zmena hodnôt parametrov. Akcie, ako pridanie, mazanie game objektov, zmena počtu objektov v listoch nie sú podporované. Predísť takejto nutnosti sa dá častým ukladaním scény.

Obrázok 20 - Hlavná lišta editora podpora undo/redo a spúšťania scény

Kapitola 7

Tvorba scenárov

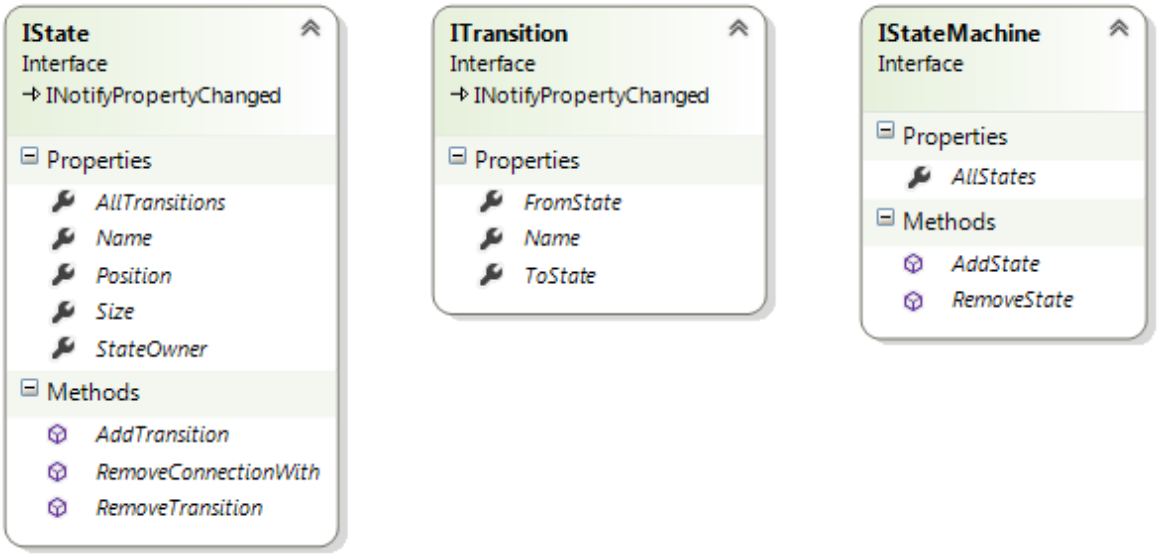
Ako sme spomenuli v predchádzajúcich kapitolách, rozhodli sme sa dej, teda jednotlivé úlohy, ktoré musí hráč resp. študent vykonať, reprezentovať pomocou stavových diagramov. V definícii scény si budeme ukladať vyjadrenie jedného takéhoto stavového diagramu, ktorý bude definovať správny postup hráča scénou respektíve scenárom.

Výhodou tejto reprezentácie je vizuálne vyjadrenie scenára a pomerne veľký prehľad o úlohách, ktoré má hráč v scenári vykonať. Preto bolo nutné naimplementovať časť editora, ktorý by vedel vizuálne vyjadriť inštanciu nejakého stavového diagramu a vedel by ho jednoducho editovať. Rozhodli sme sa pre klasické vyjadrenie stavového diagramu, kde jednotlivé stavy sú zakreslené pomocou grafického tvaru a prechody sú orientované krivky. Požiadavky na editor stavového diagramu sú jasné:

- Jednoduché pridávanie a mazanie stavov
- Jednoduché mazanie a pridávanie prechodov
- Možnosť meniť vizuálnu podobu, teda rozmiestnenie a veľkosť uzlov
- Možnosť zmeny názvu prechodov a stavov

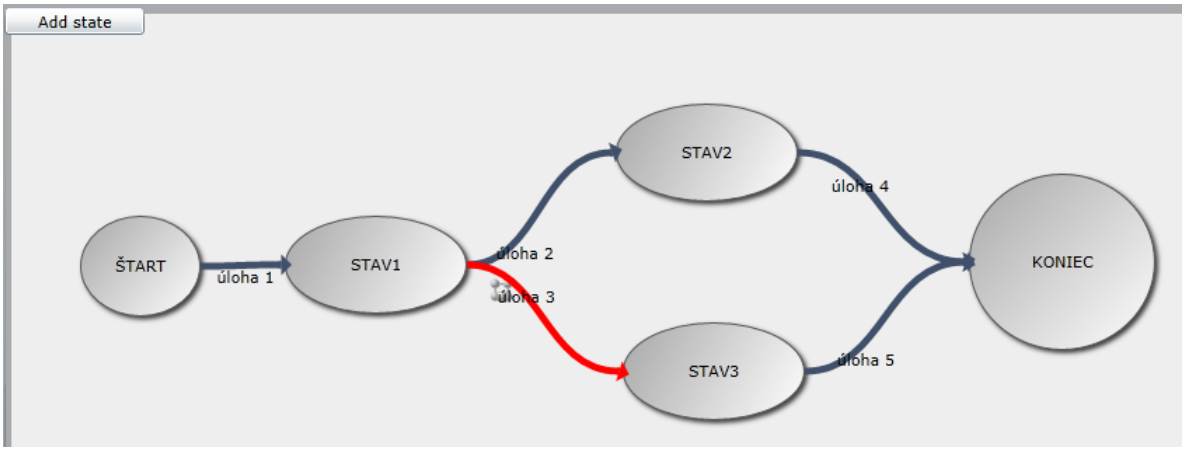
Bohužiaľ nebol v dispozícii žiaden open source diagram editor pre Silverlight a preto vznikala táto časť editoru od nuly.

Pre požiadavky editovania a rozšíriteľnosti editora stavových diagramov vzniklo niekoľko rozhraní, ktoré definujú ako majú vyzerať triedy vyjadrujúce prechody, stavy a celý stavový automat. Sú to `IState`, `ITransition` a `IStateMachine`. Editor pracuje s týmito rozhraniami kvôli možnosti rozšírenia editora.



Obrázok 21 - Rozhrania definujúce stavový automat

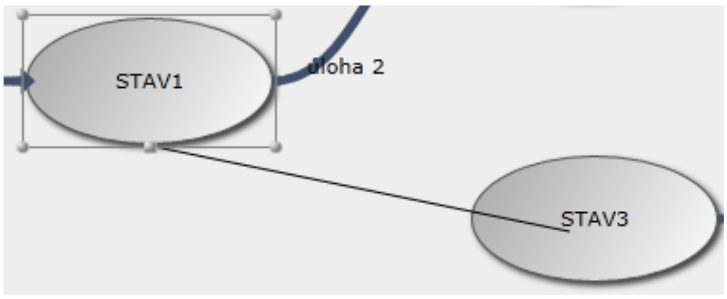
Stavy obsahujú definície metód s ktorými pracuje editor, ako sú pridávanie, mazanie prechodov a podobne. Každý stav a prechod má svoje meno. U stavov je definovaná ich poloha v rámci diagramu a veľkosť 2D objektu, ktorý je vizuálnou podobou stavu.



Obrázok 22 - Editovanie stavového automatu

Na obrázku č.Obrázok 22 vidíme výslednú podobu editora stavového automatu. Pomocou tlačidla *Add state* jednoducho pridáme prázdny stav. Stavy

a prechody v editore môžeme označiť. Pre každý stav môžeme zmeniť jeho pozíciu a veľkosť pomocou manipulátorov, ktoré sa zobrazia na označenom stave. Nové prechody vytvárame pomocou myši, keď prepojíme jeden stav s cieľovým stavom.



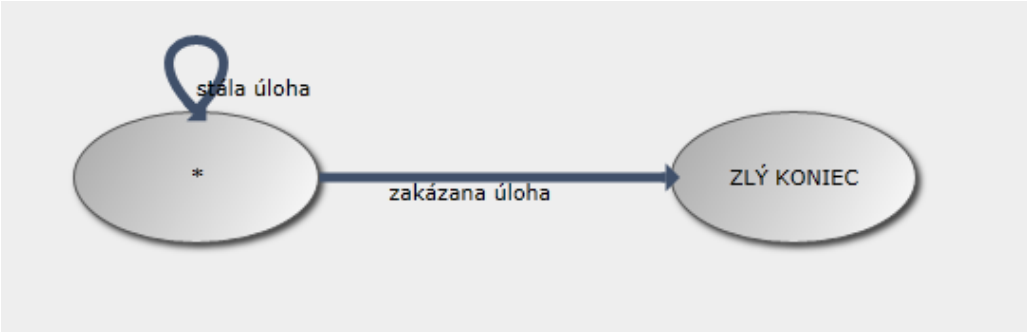
Obrázok 23 - Vytvorenie prechodu zo stavu 1 do stavu 3

Mazanie stavov a prechodov funguje stlačením tlačidla *Delete* na klávesnici v momente keď sú označené. Editor stavových automatov má aj druhú časť, v ktorej sa zobrazuje detail označeného stavu alebo prechodu. Tam sa už zobrazujú vlastnosti implementácií spomínaných troch rozhraní.

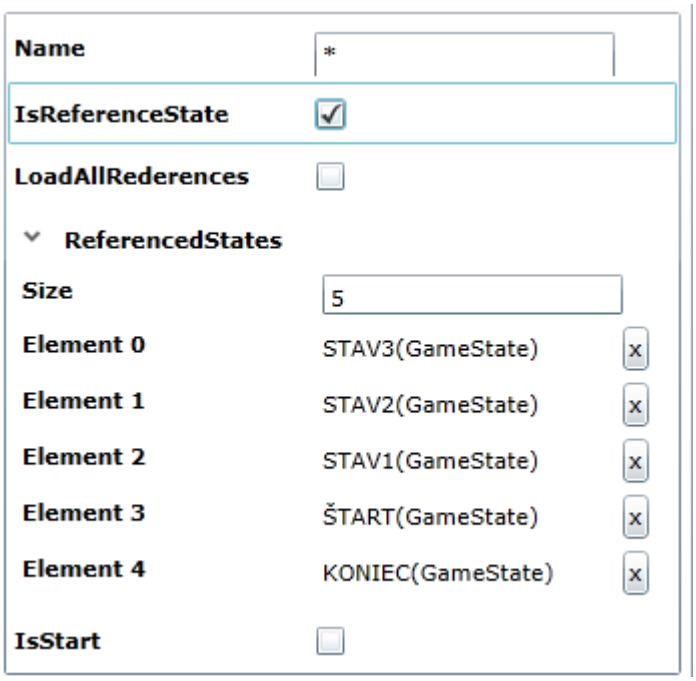
Táto časť editora funguje na rovnakom princípe ako editor game objektov. V detaile daného stavu alebo prechodu zobrazíme vlastnosti týchto objektov, ktoré získame pomocou reflexie.

Tieto triedy sú `GameState`, `GameStateMachine` a `GameStateTransition`. U každého stavu môžeme určiť niekoľko parametrov. Môžeme meniť jeho meno. Ďalej je možné nastaviť stavu, či je stavom počiatočným ak sa hra zapne. Nakoniec môžeme u stavov nastaviť aby sa chovali ako referencia na iný stav alebo viacero stavov. To nám značne môže uľahčiť návrh daného stavového automatu. A mali by sme sa vyhnúť rôznym problémom ako je prekríženie kriviek, ktoré definujú prechody.

Príklad je zobrazený na nasledujúcom obrázku. Stav s názvom * je referenciou na všetky ostatné stavy v hre. Hráč môže v každom momente hry tým pádom vykonať úlohu s názvom *stála úloha* a po jej vykonaní sa vrátiť späť do predchádzajúceho stavu. Obdobne hráč môže vykonať stále úlohu s názvom *zakázaná úloha*, tak dôjde k neúspešnému ukončeniu hry. Bez týchto referenčných stavov by sme museli vytvoriť prechod medzi každým stavom a stavom s názvom *ZLÝ KONIEC*.

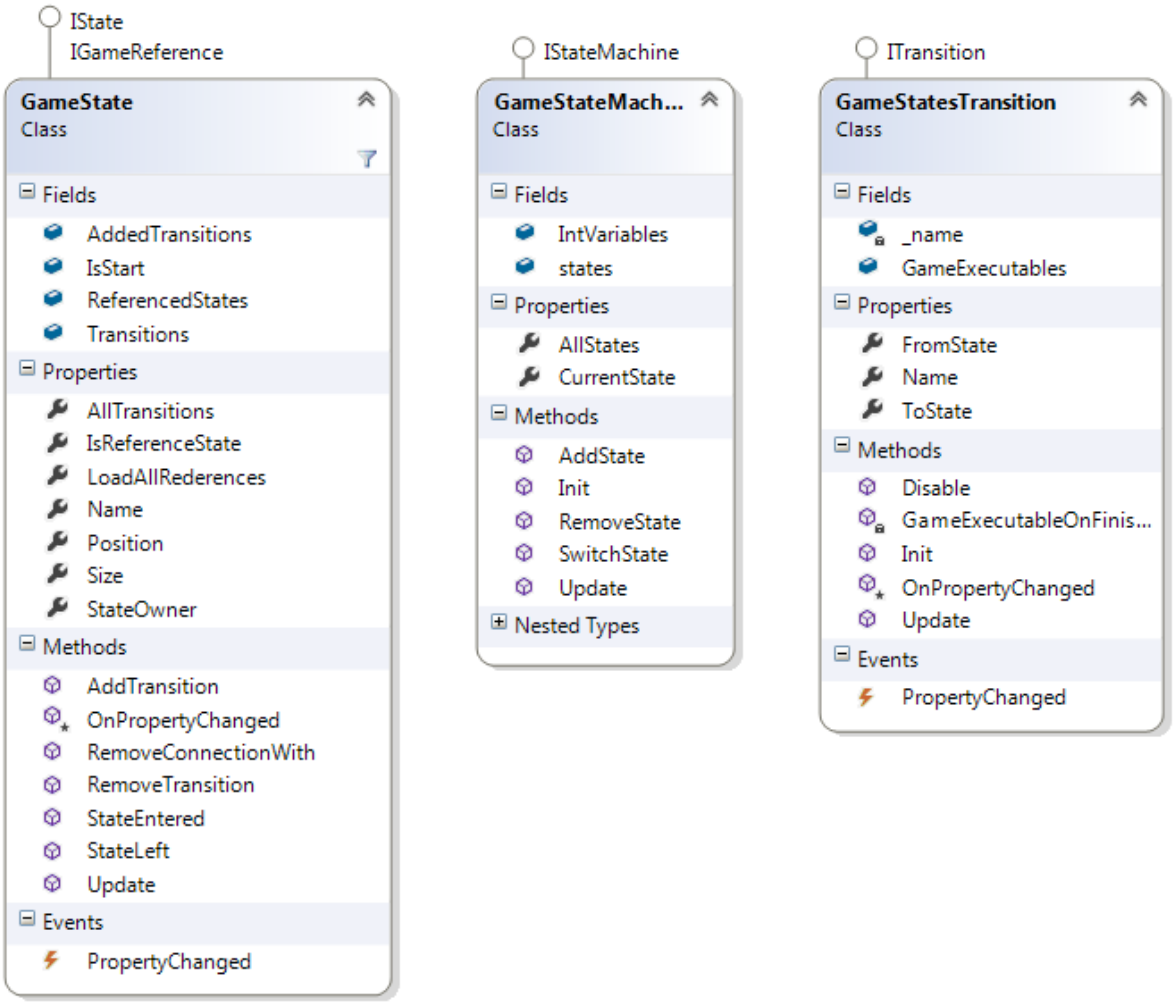


Obrázok 24 - Ukážka referenčných stavov



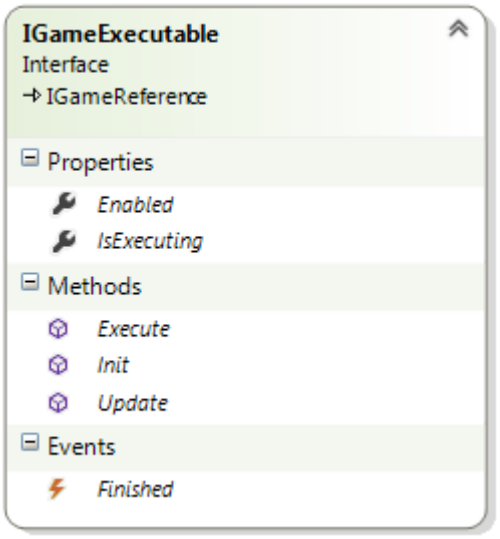
Obrázok 25 - Detail stavu s názvom „*“

Tieto triedy obsahujú logiku stavového automatu. Stavový automat obsahuje referenciu na súčasný stav. Každá z týchto tried obsahuje metódu `Update`, ktorá funguje podobne ako u game objektov. Táto metóda sa volá len na aktívnom stave a na prechodoch, ktoré z tohto stavu vychádzajú. Takto môže daný stav zistiť, či došlo k splneniu nejakého prechodu a zmení súčasný stav na cieľový stav definovaný týmto prechodom.



Obrázok 26 - Diagram tried definujúcich stavový automat

Na obrázku č.Obrázok 22 máme zároveň znázornený jednoduchý scenár. Vidíme, že existujú dva rôzne spôsoby ako sa dostať do stavu s názvom *KONIEC*. Jednotlivá úloha je určená triedou, ktorá implementuje rozhranie `IgameFinishable`. Každý prechod v stavovom automate obsahuje zoznam referencií na takéto úlohy. Ak sa takáto úloha vykoná, dochádza k zmene stavu v automate. To môže nastať nejakou akciou užívateľa, zmenou hodnoty v modelikovom modele, alebo uplynutím času. Toto rozhranie by sme mohli nazvať ako abstrakciou úlohy, ktorú vykoná užívateľ (hráč), alebo sa vykoná sama, na základe nejakého iného podnetu.



Obrázok 27 - Rozhranie IGameExecutable

Rozhranie `IGameExecutable` sa skladá z:

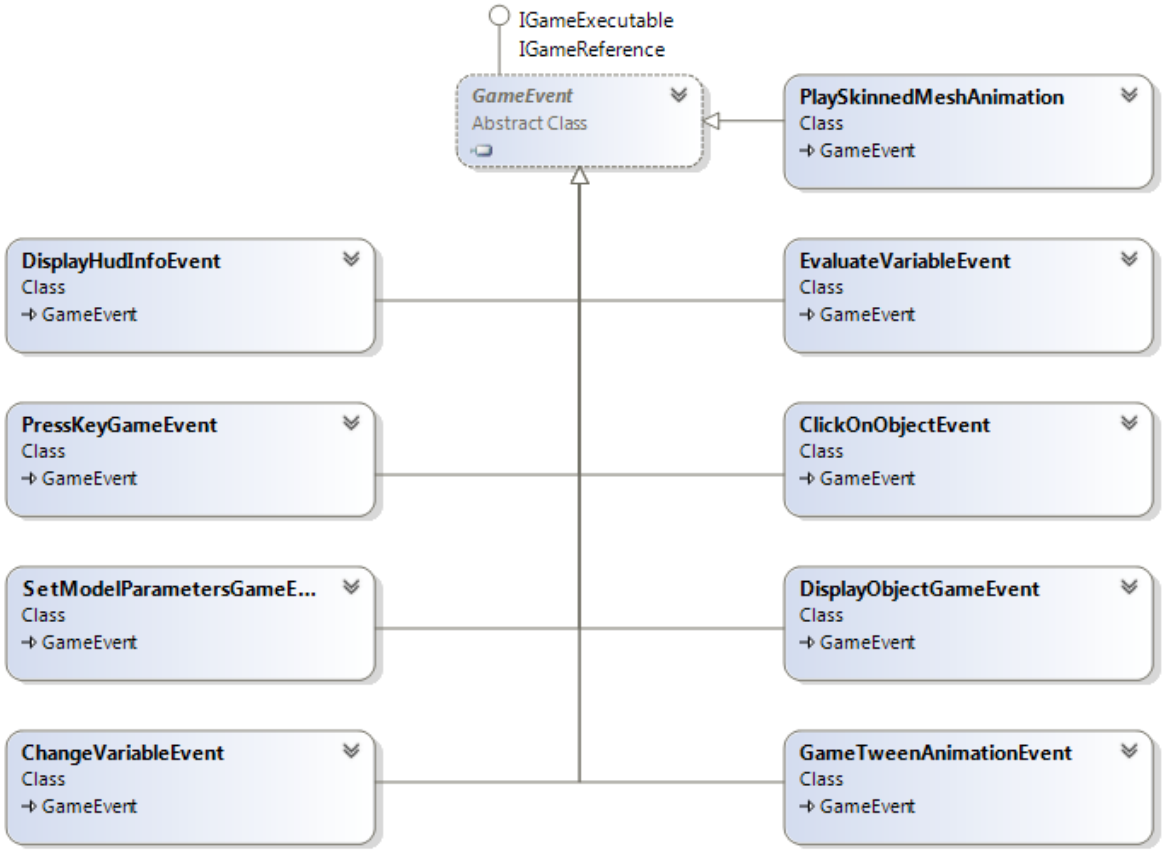
- `Enabled` – vlastnosť ktorá určuje, či je daná úloha aktívna. Aktívna je práve vtedy keď sa na nej vykonáva metóda `Update`
- `IsExecuting` – určuje, či je daná úloha vykonávaná užívateľom
- `Execute` – implementácia vykonania danej úlohy
- `Init` – inicializačná metóda
- `Update` – metóda sa volá z metódy `Update` príslušného prechodu
- `Finished` – eventa, ktorá sa zavolá keď sa dokončí daná úloha do konca. Na túto eventu je napojený prechod a ak sa zavolá dôjde k zmene stavu.

Toto rozhranie implementujú dve triedy. Prvá z nich je trieda `GameEvent`. Samotná trieda je abstraktná a obsahuje málo funkcionality. Avšak jej potomkovia sú základným prvkom pre tvorbu scenárov. Každá táto trieda je časťou skladačky, z ktorej bude následne game designer vytvárať scenár podľa zadania.

Na nasledujúcom obrázku môžeme vidieť potomkov tejto triedy. Z názvov týchto tried je pomerné jasné akú majú funkcionalitu. Ako príklad uvedieme spustenie animácie, zobrazenie textovej informácie na obrazovke, nastavenie parametru modelu

alebo počkanie na vstup užívateľa (kliknutie na objekt, alebo stlačenie klávesy). Inštanciu tejto triedy budeme nazývať herná udalosť.

Tieto herné udalosti nám budú stačiť na vytvorenie ukážkového scenára, ktorý si popíšeme v ďalšej kapitole.

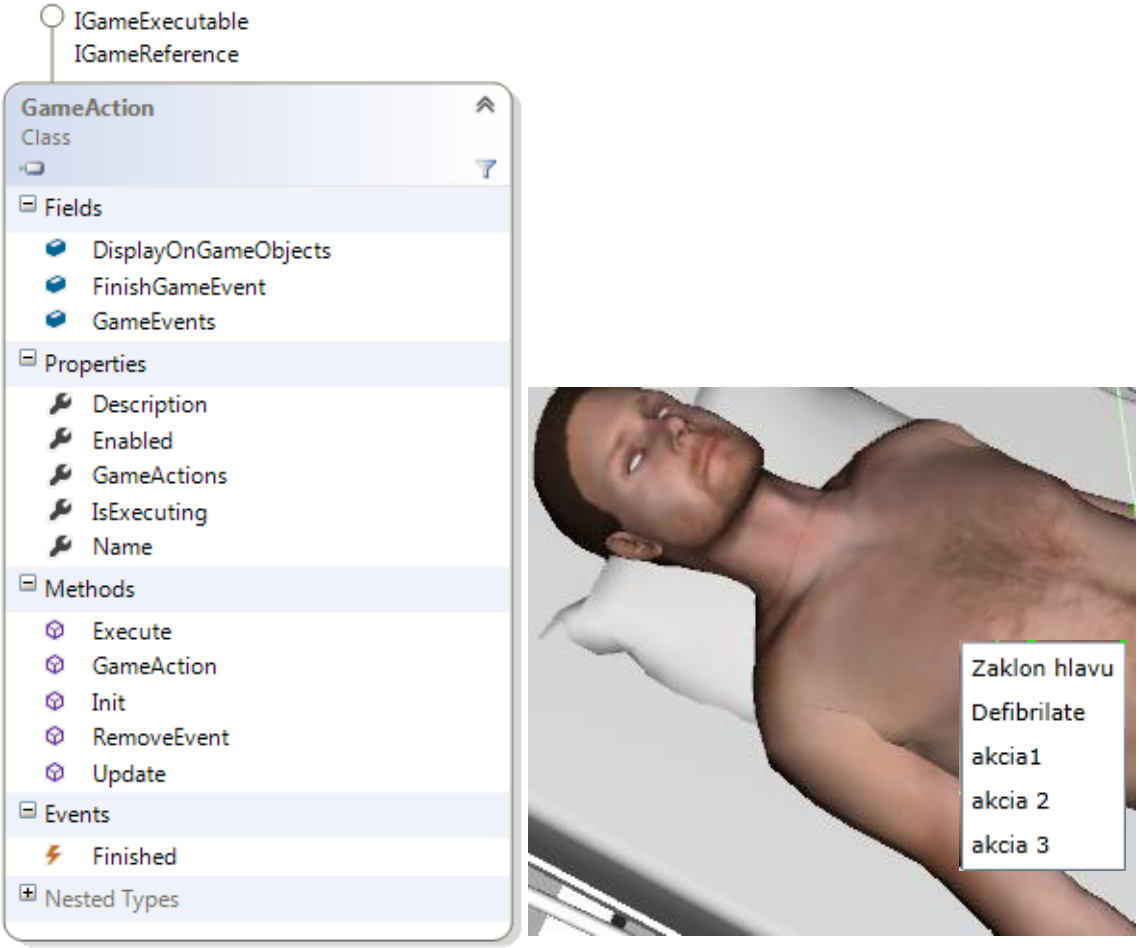


Obrázok 28 - Diagram tried dediacich od GameEvent

Druhou triedou, ktorá dedí od `IGameExecutable` je trieda GameAction. Tá obaľuje funkcionalitu ovládania scény pomocou popiskov. Každá GameAction má definované svoje meno a 3D objekty zo scény, na ktorých sa má zobraziť vo forme popisku.

Každý prechod má definovaný zoznam inštancií objektov, ktoré implementujú rozhranie `IgameExecutable`. Editor podporuje do tohoto zoznamu pridať inštancie práve triedy `GameAction` (budeme ich nazývať herné akcie) a tieto inštancie vytvoriť a editovať.

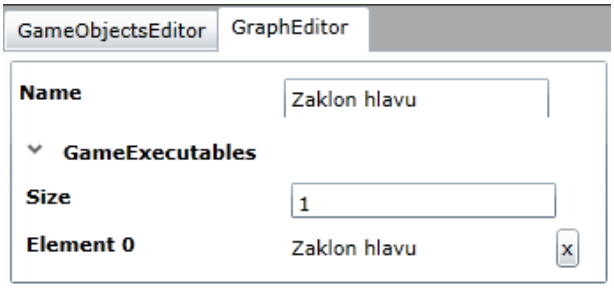
Každá herná akcia v sebe zahrňuje niekoľko herných udalostí, ktoré na seba nadväzujú. Takto môžeme spojiť jednoduchšie herné udalosti do komplexnejšej sady príkazov. Po tom ako užívateľ klikne na popisok danej akcie môžeme napríklad zobraziť varovný text v ktorom užívateľa vyzveme k stlačeniu nejakého tlačidla. Následne prehrajeme nejakú animáciu.



Obrázok 29 – Trieda GameAction a popisky v scéne

Editor herných akcií obsahuje ich zoznam, v ktorom môžeme vybrať akciu, ktorú chceme editovať. Tento zoznam podporuje drag&drop, a tak si môžeme poradie akcií v zozname jednoducho meniť. Toto poradie nijak neovplyvňuje poradie akcií v hre, slúži len ako pomoc pre lepšie usporiadanie. Z tohto zoznamu môžeme pomocou

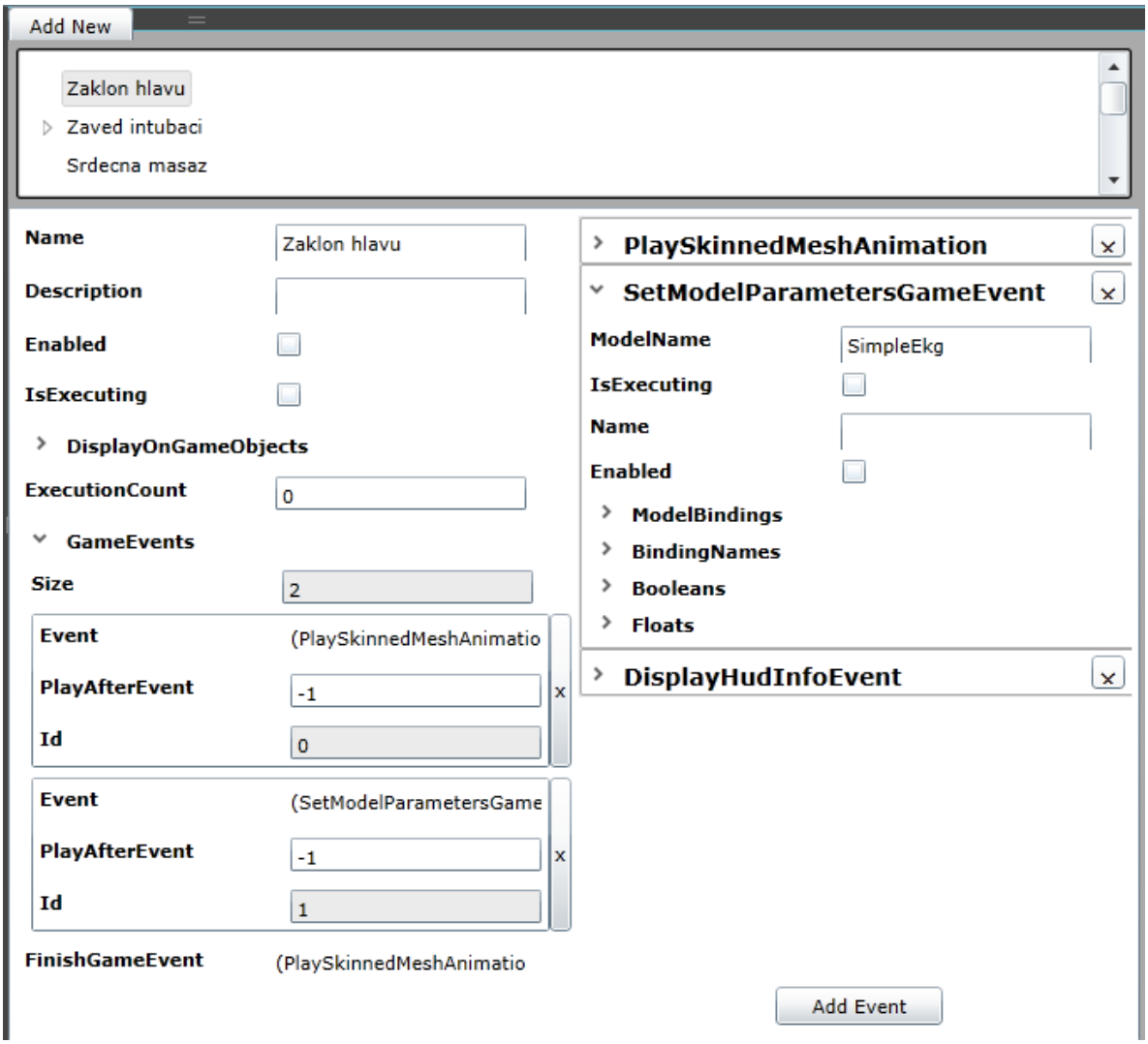
drag&drop pridať akciu do prechodu v hre a to tak, že pridáme hernú akciu do zoznamu `GameExecutables` v detaile prechodu zobrazeného na nasledujúcom obrázku.



Obrázok 30 – Detail prechodu

V editore herných akcií môžeme tieto akcie pridávať a mazať. Každá z týchto akcií obsahuje zoznam herných udalostí, ktoré môžeme tak isto mazať, pridávať a editovať.

Herný dizajnér si zároveň určuje poradie v akom sa budú tieto herné udalosti vykonávať. V detaile udalostí môže editovať ich parametre. Parametre jednotlivých udalostí si popíšeme v nasledujúcej kapitole.



Obrázok 31 – Detail hernej akcie

Kapitola 8

Ukážkový scenár

Ako ukážkový scenár bola vybraná kardiopulmonálná resuscitácia. Jedná sa o jednoduchý scenár, v ktorom užívateľ musí zachrániť pacienta pomocou umelého dýchania, srdečnej masáže a defibrilácie.

Aplikácia sa snaží simulovať kroky, ktoré lekár musí vykonať v praxi. V textovej podobe by sme mohli scenár vyjadriť nasledovne.

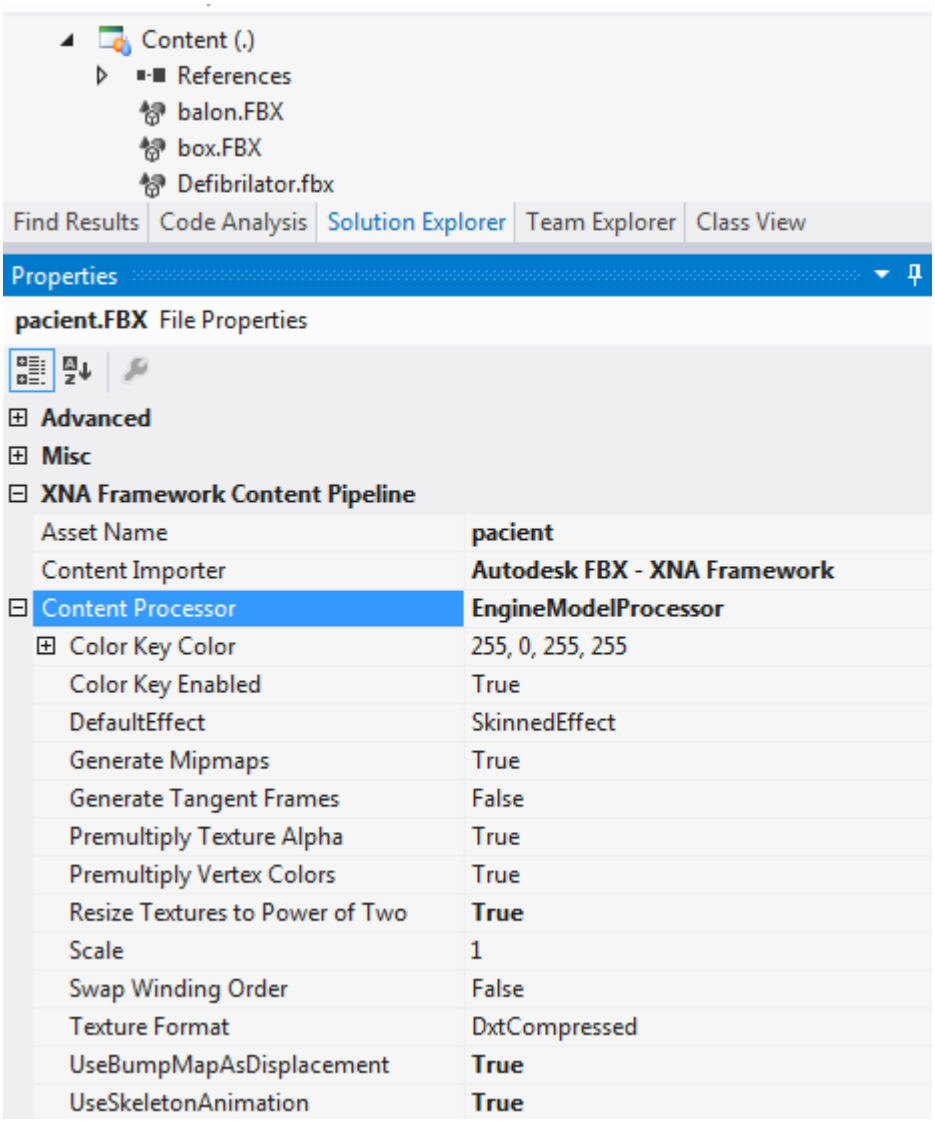
Máme miestnosť, v ktorej leží pacient. Pacient prestane dýchať a dochádza k zástave srdca. Lekár musí následne vykonať potrebné kroky. Pred začatím resuscitácie musí pacientovi zakloniť hlavu a zaviesť intubáciu.

Následne dochádza k masáži srdca a umelému dýchaniu. Takto užívateľ pokračuje nejakú dobu. Pacient sa nepreberá. Lekár musí použiť defibrilátor. Umiestní elektródy na správne miesto a zapne elektrický výboj. Pacient sa stále nepreberá. Lekár opakuje tieto dva úkony pokiaľ sa pacient nepreberie.

Takýto scenár môžeme jednoducho popísať pomocou stavového automatu. Prvým krokom, ktorý musí herný dizajnér vykonať je pripravenie 3D scény.

8.1 Príprava scény

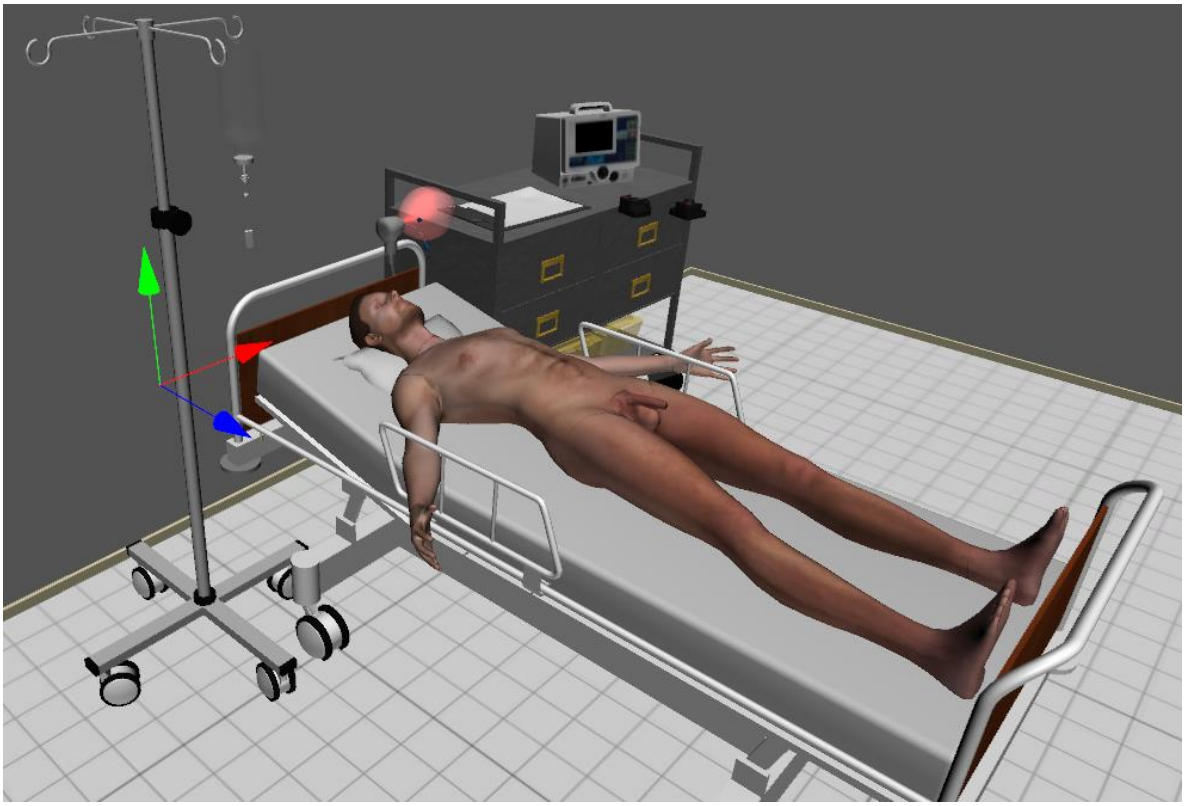
Od 3D grafika sme mali pripravené potrebné 3D modely a v prípade pacienta aj s kosternými animáciami. Tieto všetko modely pridáme do Content projektu, ktorý bude prepojený s našou hrou. V nástroji Visal studio jednoducho nastavíme vlastnosti importovania jednotlivých modelov. Hlavným nastavením pre každý model je použitie nášho content processoru. Ten pridá pre každý model dôležité informácie. U tohto procesoru môžeme nastaviť ďalšie parametre, ako je načítanie kosterných animácií, zmenšenie veľkosti modelu a ďalšie.



Obrázok 32 - Nastavenie importéru pre model pacienta

Takto pripravené modely môžeme následne použiť v našej scéne. Za použití editora im nastavíme správnu veľkosť, pozíciu a rotáciu. To sa deje jednoducho. Najprv vložíme modely do scény pomocou hlavného menu (Assets -> Add 3D model) a v následnom dialógu vyplníme meno modelu. Potom označím model v 3d pohľade, alebo v stromovej štruktúre a použitím giziem meníme jeho pozíciu, rotáciu a veľkosť. Takto si vytvoríme statickú podobu scény.

Na predchádzajúcom obrázku sú znázornené vlastnosti importovania nášho hlavného modelu pacienta v scéne. Ten používa kosterné animácie a zároveň má nastavený parameter `UseBumpMapAsDisplacement`. Tento parameter zaručí importovanie modelu tak, aby sme mohli použiť displacement, to jest vyťahovanie vertexov pozdĺž normály. Táto vlastnosť zaručí možnosť animovania dýchania. Oblasť modelu, v ktorej pacient dýcha je vymedzená textúrou v odtieňoch šedej, nakreslenou v modelovacom nástroji od 3D grafika.

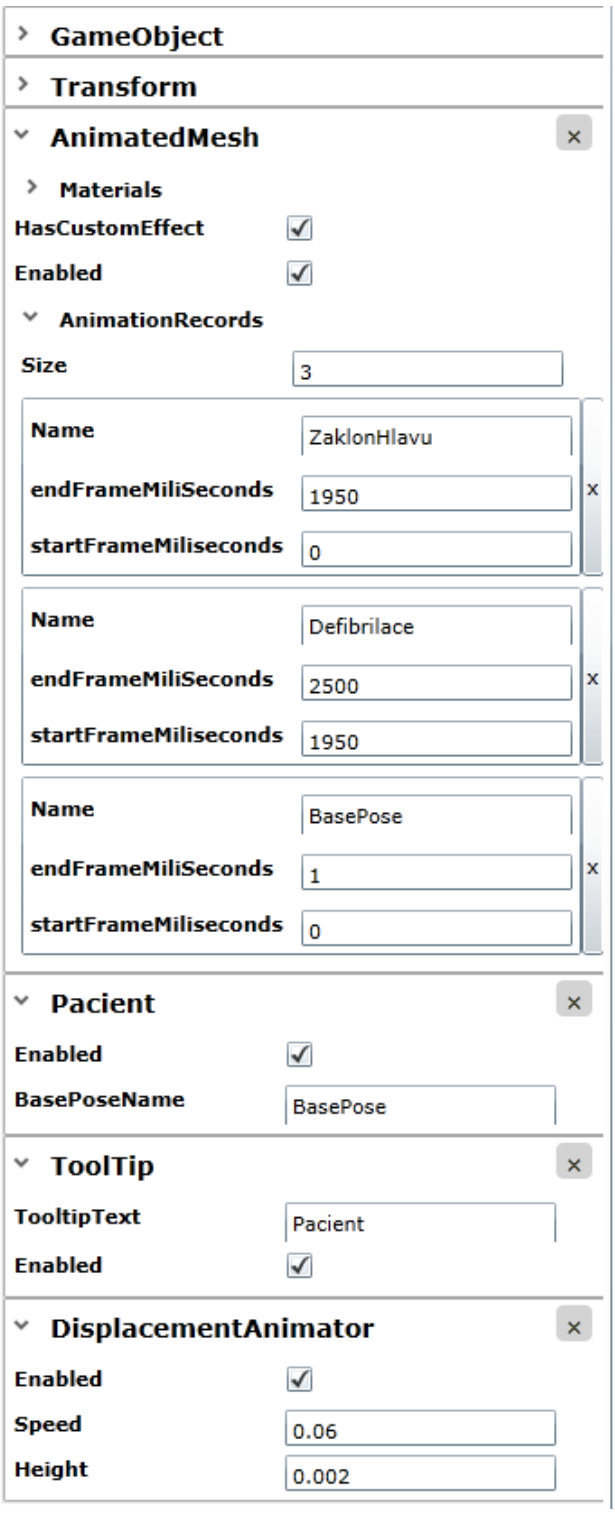


Obrázok 33 - Podoba scény po rozmiestnení 3D modelov

Ďalej môžeme upraviť parametre jednotlivých objektov v scéne. Tu spomeniemedve hlavné časti 3D scény. Model pacienta a model defibrilátora.

8.1.1 Model pacienta

Model pacienta obsahuje kosterné animácie a zároveň bude na ňom prebiehať animácia dýchania. Túto funkcionalitu zabezpečíme pridaním a nastavením správnych komponent na game objekt. Po pridaní modelu do scény sa vytvorí game objekt s komponentou `AnimatedMesh`. V nej si môžeme nastaviť detaily animácií, ktoré máme v dispozícii od 3D grafika. Modelovacie nástroje, ktoré pracujú s formátom fbx väčšinou exportujú všetky animácie ako jednu veľkú animáciu, preto trieda `AnimatedMesh` obsahuje funkcionalitu na rozdelenie tejto animácie na viacero menších. V našom prípade pacient obsahuje dve animácie. Prvou je zaklonenie hlavy a druhou je nadskočenie pacienta po šoku z defibrilátoru. Od grafika vieme kedy dané animácie končia a začínajú, a tak v editore môžeme nastaviť `AnimationRecords`. O animáciu dýchania sa stará komponenta `DisplacementAnimator`, ktorá mení jeden parameter v materiále pacienta a grafický shader následne reaguje zmenou pozície vertexov na obrazovke.



Na model pacienta pridáme komponentu `Tooltip`, ktorá zobrazí popisok objektu po najetí myšou nad objekt. Komponenta `Pacient` prehrá animáciu po načítaní scény s definovaným názvom.

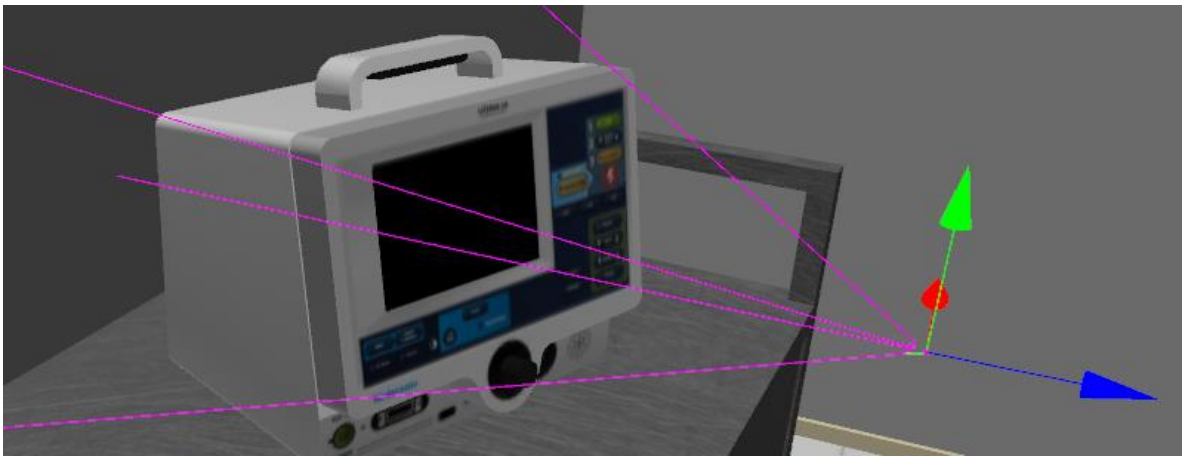
8.1.2 Defibrilátor

Model defibrilátoru neobsahuje žiadne animácie. V scéne však zastáva podstatnú úlohu. Defibrilátor bude obsahovať krivku EKG pacienta a bude možné naň kliknúť a tento graf takto priblížiť. Pre oba prípady obsahuje engine potrebné komponenty. Prvou je komponenta `SilverlightTexture`. Tú pridáme na pripravený objekt v scéne, ktorý je obdĺžnikového tvaru. V tejto komponente stačí nastaviť meno textúry, pod ktorou sa nachádza v pripravenom slovníku takýchto textúr. Kreslenie grafu do tejto textúry musí zabezpečiť programátor v kóde za pomoci funkcií z engine a zo Silverlightu. Takto môžeme na 3D objektoch zobrazovať vektorovú grafiku a grafy, ktorými disponuje Silverlight.

Obrázok 34 - Zobrazenie detailu defibrilátoru

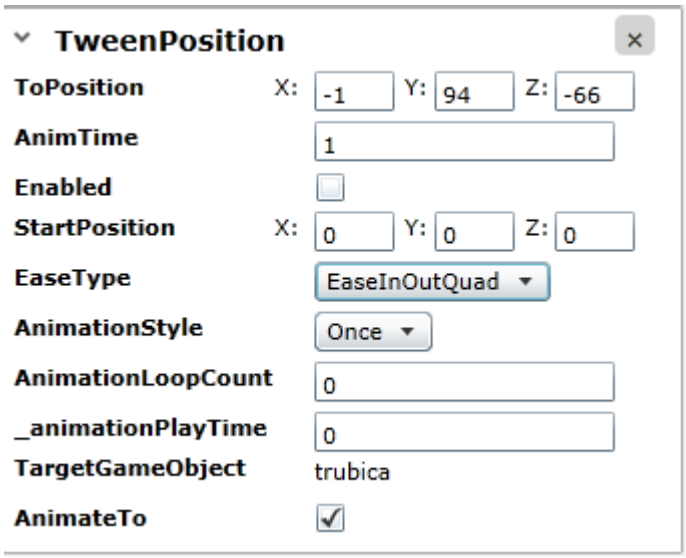
Graf na predchádzajúcom obrázku je vykreslený pomocou knižníc z frameworku Bodylight. Pomocou tohto frameworku je graf zároveň napojený na jednoduchý model simulujúci EKG. Podobne ako graf, tak aj model pridá do scény programátor.

Ďalej pridáme na defibrilátor komponentu `ActiveObject`, ktorá má dve úlohy. Táto komponenta zvýrazní farbu objektu, ak sa nad ním nachádza myš. A po kliknutí na objekt dôjde k jeho priblíženiu tak, ako to je vidieť na predchádzajúcom obrázku. Táto komponenta obsahuje referenciu na kameru, ktorá určuje ako bude vyzerať pohľad po priblížení na objekt. Po kliknutí na defibrilátor dôjde k interpolácii a teda animácii súčasného pohľadu a pohľadu určeného statickou kamerou. Stlačením tlačidla escape sa pohľad opäť oddiali.



Obrázok 35 - Nastavenie pohľadu statickej kamery

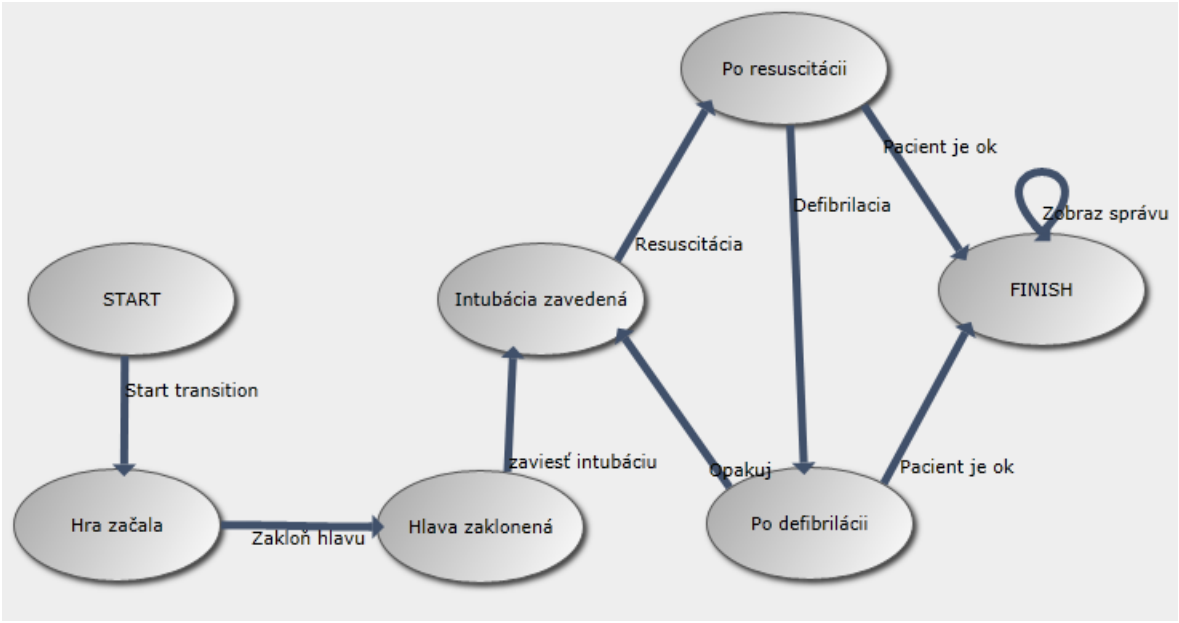
Ako ďalšie nastavenia v scéne môžeme nazvať prípravy animácií jednotlivých objektov, ktoré sa budú spúšťať ako reakcie na užívateľove kroky. Prehrávať sa budú z herných udalostí, ktoré sme popísali v predchádzajúcej kapitole. Tieto animácie majú na začiatku názvu triedy slovo `Tween`. Sú to jednoduché komponenty, ktoré menia jeden z parametrov objektu. Základne sú: zmena pozície, priehľadnosti, farby. V parametroch môžeme meniť hodnoty, ako čas animácie, počet opakovaní, krivku animácie (takzvaný easing).



Obrázok 36 - Animácia pohybu intubačnej tubice

8.2 Pridanie scenára

Pre takto pripravenú scénu môžeme začať navrhovať stavový automat.



Obrázok 37 - Vyjadrenie deja pomocou stavového automatu

Na predchádzajúcom obrázku vidíme vyjadrenie deja pomocou stavového automatu. Tento automat korešponduje s textovou podobou, ktorou sme popísali scenár na

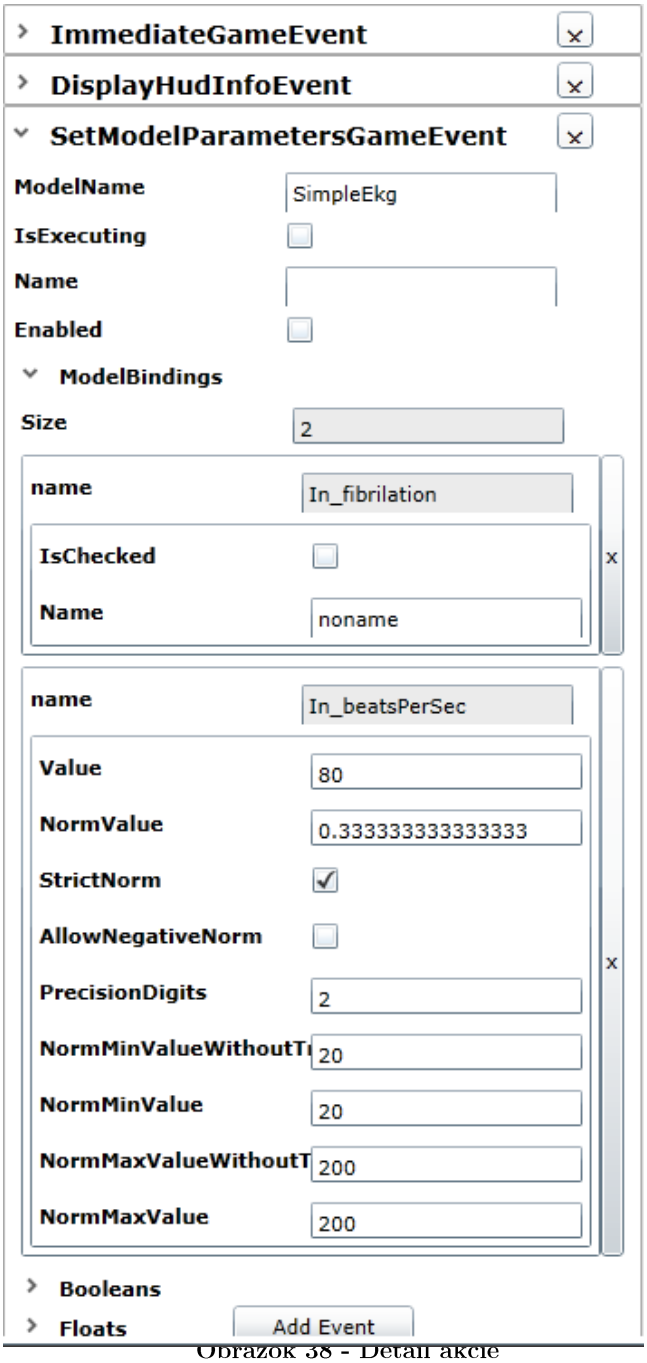
začiatku kapitoly. V nasledujúcej časti textu popíšeme jednotlivé prechody z tohto stavového automatu.

8.2.1 Start transition

Prechod, ktorý sa vykoná okamžite po štarte hry. Hráčovi vysvetlíme, aká je situácia pomocou textu na obrazovke a zároveň nastavíme správne parametre modelu, ktorý reprezentuje EKG pacienta.

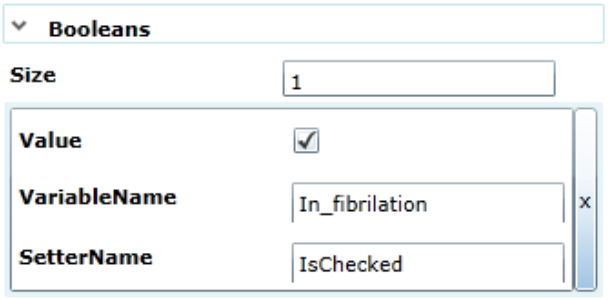
Prechod start transition je napojený na hernú akciu s názvom štart hry. Na obrázku vpravo vidíme detail herných udalostí, z ktorých sa táto akcia skladá. Ako prvá je `ImmediateGameEvent`, ktorá značí, že akcia sa začne hneď sama vykonávať bez užívateľovho vstupu.

Ďalej máme hernú udalosť `DisplayHudEvent`, ktorá vypíše na obrazovku ľubovoľný text definovaný v tejto udalosti. Ako posledná sa zavolá `SetModelParametersGameEvent`. Tá nám dovolí meniť vstupy modelu. Všetky vstupy modelu sú vypísané v poli `ModelBindings`. Z obrázku vidíme, že náš model má dva vstupy: `In_fibrilation` a `In_beatsPerSec`.



Obrázok 38 - Detail akcie

Konkrétne nastavíme parameter `Is_checked` z vstupu `In_fibrilation` na false. To dosiahneme tak, že v poli `Booleans` pridáme záznam s týmito informáciami. Takto bude model vracať EKG krivku pri fibrilácii pacienta.

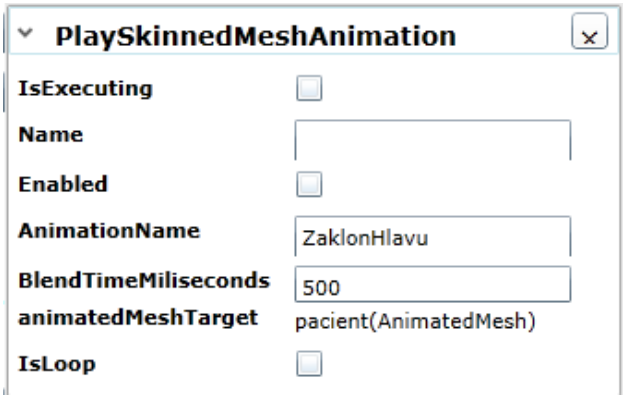


Obrázok 39 - Nastavenie premenných modelu

Obrázok 40 - Textová informácia

8.2.2 Zakloň hlavu

Tento prechod je napojený na akciu s menom Zakloň hlavu. Akcia je pomerne jednoduchá a užívateľovi po kliknutí na pacienta dovolí zakloniť hlavu. Obsahuje jedinú hernú udalosť `PlaySkinnedMeshAnimation`. V tejto udalosti nastavíme meno animácie, ktorá sa má prehrať a do `AnimationMeshTaregt` natiahneme referenciu na model pacienta v scéne. (Pomocou použitia drag&drop zo stromovej štruktúry scény).



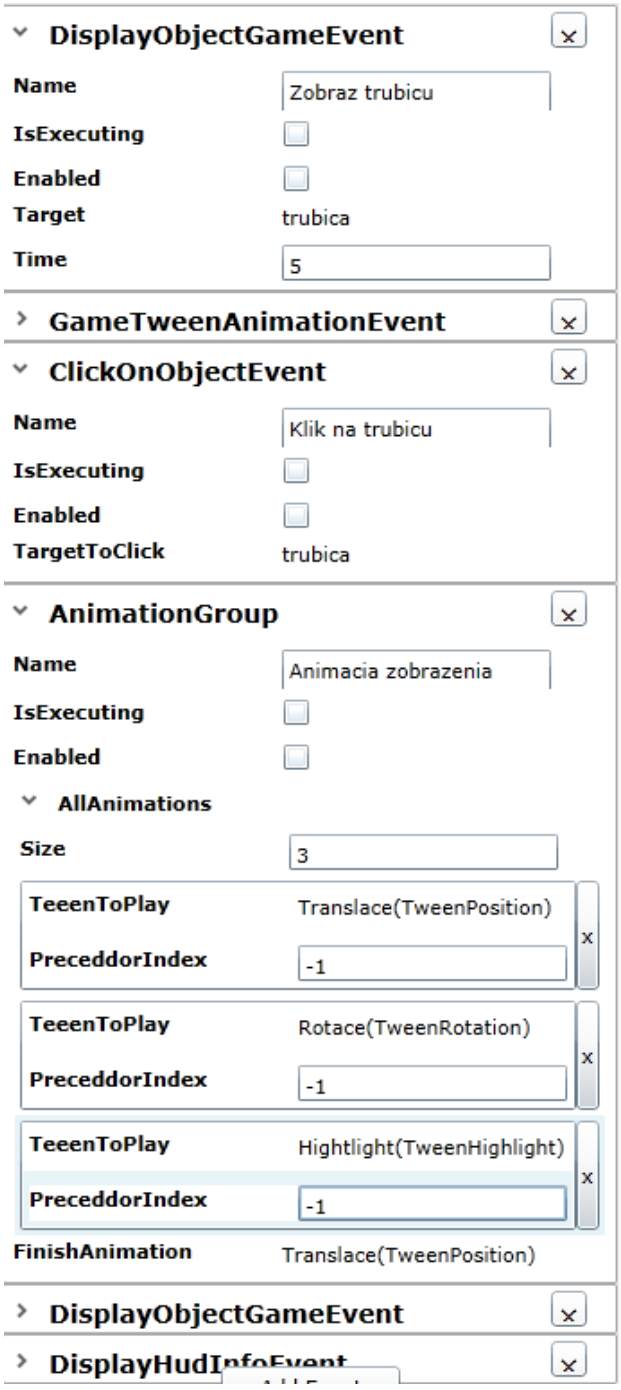
Obrázok 41 - Prehrávanie animácií

8.2.3 Zaviesť intubáciu

Tento prechod je prepojený tiež s jednou hernou akciou. Táto akcia zobrazí intubačnú trubicu a následne, po kliknutí na ňu, sa spustí animácia jej aplikácie. Následne sa zobrazí dýchací vak.

Na obrázku môžeme, vidieť jednotlivé herné udalosti. `DisplayGameObject` zobrazí skrytý objekt pomocou animácie priehľadnosti. `GameTweenAnimationEvent` spustí konkrétnu animáciu, na ktorú má referenciu. `ClickOnObjectEvent` čaká, kým užívateľ neklikne na konkrétny objekt. `AnimationGroup` dokáže spustiť niekoľko animácií naraz.

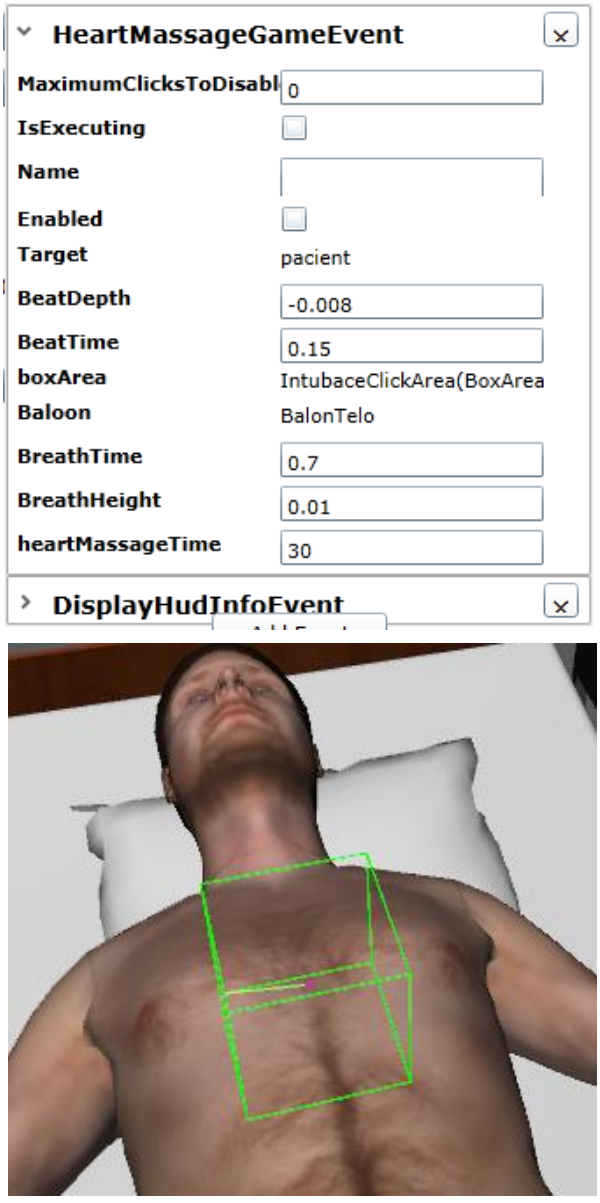
Kombináciou týchto udalostí dosiahneme požadovaný efekt.



Obrázok 42 - Detail akcie

8.2.4 Resuscitácia

Herná akcia, ktorá odpovedá tomuto prechodu vyzerá v editore pomerne jednoducho. Obsahuje dve herné udalosti `HeartMassageGameEvent` a `DisplayHudInfoEvent`. Podstatnou je prvá z nich. Funguje tak, že umožní klikať na trup pacienta a na dýchací vak za sprievodnej animácie nádychu, alebo výdychu hrudníku. Parameter `Target` obsahuje referenciu na model pacienta v scéne. Ak klikneme na tento objekt tak sa odchytí kliknutie a herná udalosť sa pozrie, či kliknutie bolo v oblasti definovanej komponentou `BoxArea`, na ktorú obsahuje ďalšiu referenciu. Ak áno, prehrá sa animácia stlačenia hrudníku. Podobne je to s kliknutím na dýchací vak. Tam sa nemusí kontrolovať pozícia kliknutia. Parameter `HeartMassageTime` určuje, ako dlho bude táto udalosť aktívna. Ostatné parametre určujú hĺbku a čas animácie hrudníku pacienta.

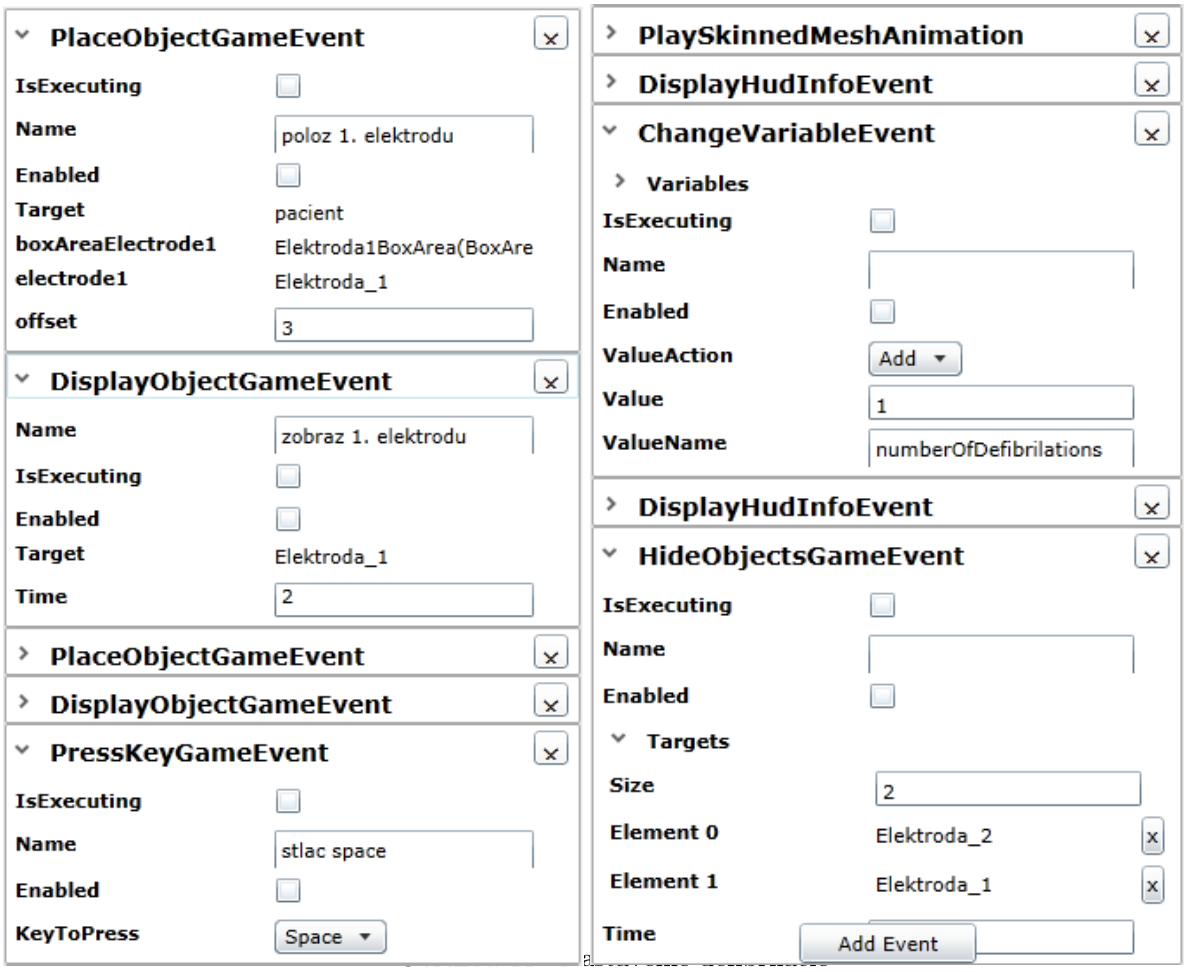


Obrázok 43 - Resuscitácia

8.2.5 Defibrilácia

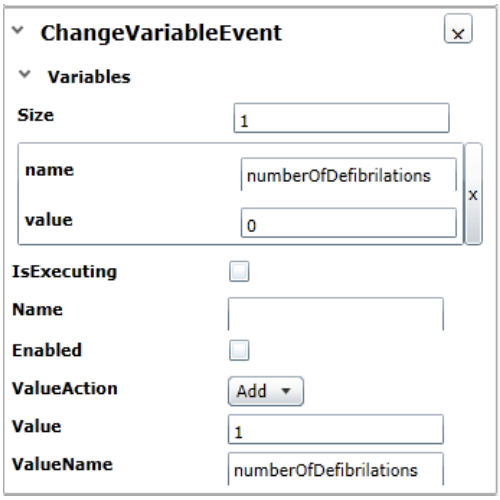
Tento prechod je prepojený s hernou akciou, ktorá umožní užívateľovi priložiť elektródy a následne zapnúť defibrilátor. Prvou použitou hernou udalosťou je `PlaceGameObjectEvent`. Tá obsahuje referencie na dva objekty: objekt, na ktorý musíme kliknúť a objekt, ktorý sa na toto miesto kliknutia presunie. V našom prípade klikneme na pacienta a elektróda sa presunie na miesto kliknutia. Podobne ako pri srdečnej masáži, herná udalosť obsahuje referenciu na komponentu typu `BoxArea`. Tá určuje miesto, do ktorého musíme kliknúť.

Následne sa elektróda ukáže na mieste, na ktoré sme klikli. Tento postup zopakujeme aj pre druhú elektródu.



Ak máme priložené obe elektródy, zobrazí sa textová informácia, ktorá užívateľa vyzve k stlačeniu tlačidla na klávesnici, čo je detekované vďaka `PressKeyGameEvent`, ktorá čaká na stlačenie tlačidla, ktoré si môžeme v editore vybrať. V našom prípade sme vybrali medzerník. Ďalej sa prehrá kosterná animácia elektrického šoku na pacientovi. Potom pomocou `HideObjectsGameEvent` schováme elektródy.

Na záver využijeme ďalšiu hernú udalosť `ChangeVariableGameEvent`. Stavový automat obsahuje zoznam celočíselných premenných, ktoré môžu slúžiť ako počítadla. Vo vlastnosti triedy s názvom `Variables` môžeme jednoduchým trikom editovať zoznam týchto premenných, ktorý je v skutočnosti uložený na inštancii stavového diagramu. Preto, ak budeme mať viacero týchto udalostí, tak každá z nich ukáže rovnaký zoznam. Ďalej už len nastavíme, čo sa má stať s premennou nastavenou vo `ValueAction`. Môžeme od danej premennej odčítať, pričítať, alebo ju nastaviť na konkrétnu hodnotu. V našom prípade zväčšíme hodnotu premennej s názvom `numberOfDefibrilations` o jedna.

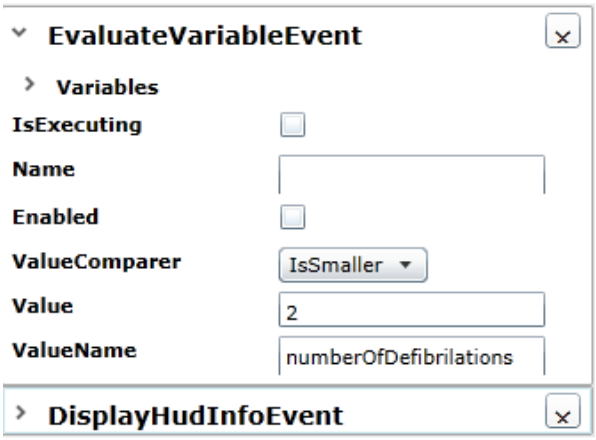



```
[DoNotSerialize]
public List<GameStateMachine.IntVariable> Variables
{
get
        { return
Scene.CurrentScene.GameStory.StateDiagram.IntVariab
les;    }

set
        {
Scene.CurrentScene.GameStory.StateDiagram.IntVariab
les = value;
        }
}
```

Obrázok 45 - Zmena premenných v stavovom diagrame

Po defibrilácii sa stavový automat dostane do stavu s názvom Po defibrilácii. V tomto stave máme dva prechody, ktoré budú kontrolovať hodnotu uloženú v premennej numberOfDefibrilations.

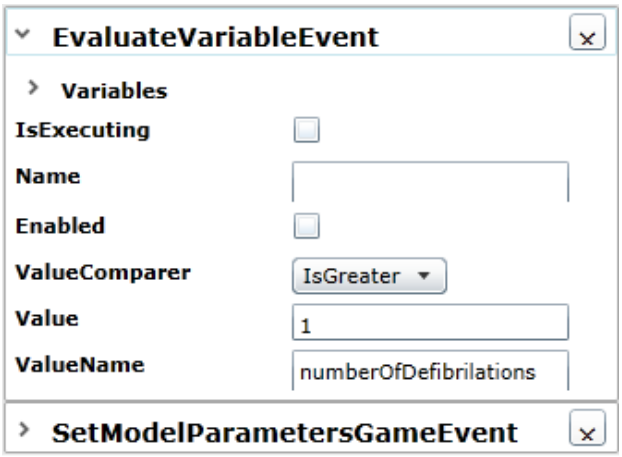
8.2.6 Opakuj



Obrázok 46 - Akcia opakuj

V tomto prípade máme v akcii pridanú jednu hernú udalosť, ktorá bude kontrolovať hodnotu premennej zo stavového diagramu. Takto môžeme jednoducho povedať koľko krát musí užívateľ resuscitovať a defibrilovať pacienta. Vo vlastnosti `Variables` máme vypísané všetky premenné zo stavového automatu. V našom prípade sme sa rozhodli, že stačí ak užívateľ bude defibrilovať pacienta dva krát. Týmto prechodom sa vrátime opäť do stavu Intubácia zavedená.

8.2.7 Pacient je ok



Obrázok 47 - Akcia pacient je ok

Tento prechod je prepojený s akciou, v ktorej sa nachádza tiež udalosť `EvaluateVariableEvent`. Tá zistí, či je hodnota uložená v parametri väčšia ako jedna, ak áno akcia sa vykoná a potom sa nastavia parametre v modely. Konkrétne nastavíme premennú `In_fibrilation` na `false`, teda presne naopak, ako tomu bolo na začiatku hry. To zaručí, že krivka EKG začne ukazovať tep pacienta.

8.2.8 Zobraz správu

Obrázok 48 - Akcia zobraz správu

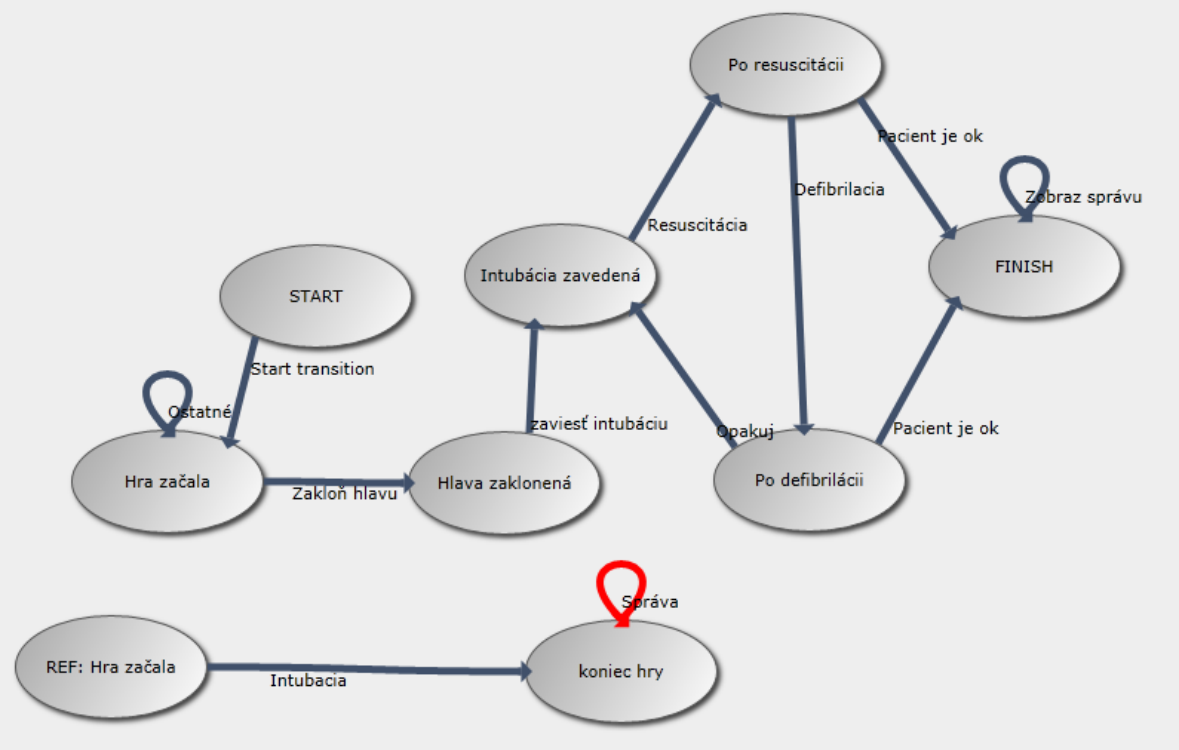
Akcia tohto prechodu zobrazí informáciu o úspešnej záchrane pacienta.

8.3 Ovládanie

Ovládanie samotnej hry je iné ako ovládanie editoru. V scéne máme objekt s kamerou `GameOrbitCamera`. Pohľad tejto kamery sa ovláda jednoducho. Kliknutím a ťahaním myši kameru rotujeme a keď navyše podržíme medzerník, kamerou môžeme hýbať v rovine kolmej na jej pohľad.

Ovládanie objektov sa deje klikaním na nich a následným výberom z možností, ktoré môžeme vykonať.

8.4 Ďalšie možnosti



Obrázok 49 - Stavový diagram s ďalšími prechodmi

Na predchádzajúcom obrázku môžeme vidieť diagram rozšírený o ďalšie prechody. Tieto prechody reprezentujú kroky, ktoré užívateľ môže vykonať, ale nie sú správne. Ako prvý je prechod zo stavu *Hra začala* s názvom *Ostatné.* V tomto prechode je

pridaná akcia defibrilácie. To znamená, že užívateľ môže defibrilovať pacienta hneď na začiatku hry. Táto defibrilácia však nebude mať žiaden efekt na dej a po jej vykonaní sa stav zas vráti do stavu *Hra začala*.

Ďalšou možnosťou, ako sa vysporiadať so zlými krokmi užívateľa je vytvoriť chybový stav a nesprávne kroky presmerovať do tohto stavu. Na obrázku máme stav *REF: Hra začala*, ktorý je referenciou na stav *Hra začala* a z neho prechod *Intubácia*, ktorý nás zavedie do stavu *koniec hry*. V tom nás hra upozorní, že sme zaviedli intubáciu v dobe, keď sme nemali a musíme hrať celú hru od znova.

Kapitola 9

Zhodnotenie a záver

V tejto diplomovej práci sme sa zaoberali tvorbou simulačných hier. Ak sa takéto hry pokúša vytvoriť team o malom počte ľudí, tak je nutné vytvárať nástroje, ktoré proces tvorby takejto hry uľahčia. 3D hry sú všeobecne známe veľkým rozpočtom a veľkou náročnosťou na ľudské zdroje. Práve pre malý team je výhoda používať a recyklovať už hotové veci a preto sme sa rozhodli pre použitie platformy Silverlight, v ktorej bol implementovaný aj framework Bodylight, ktorého časti boli v tejto práci použité [8].

Nad základným enginom bol pomocou Siverlightu vytvorený komplexný editor scény a scenárov. Práve implementáciu editora značne uľahčilo použitie platformy Silverlight.

V práci bol vytvorený systém na definovanie a úpravu scenárov pomocou stavových diagramov. Tento postup tvorby scenára by sa dal použiť pre rôzne enginy a ukázal sa ako vhodný a rozšíriteľný. Stavový diagram vizuálnou formou popisuje dej v scéne a nevyžaduje znalosť programovanie. Jednotlivé prechody stavového automatu sú rozdelené na menšie nadväzujúce znovupoužiteľné časti, s ktorými pracuje herný dizajnér. Tento dizajnér sa spolieha na funkčnosť jednotlivých častí, z ktorých postupne skladá celkový scenár. Programátor má na starosti tvorbu jednotlivých častí, z ktorých skladá dizajnér scenár.

Bolo zachované prepojenie s frameworkom Bodylight a došlo k rozšíreniam, kde modely z Modelicy môžu zasahovať do deja a ich výstupy môžu byť zobrazované v 3D scéne. Takto dokážeme vytvoriť 3D simulácie založené na fyziologických modeloch.

Možnosti editora sme ukázali na vytvorení jednoduchej hry odpovedajúcej scenáru kardiopulmunálnej resuscitácie. Demonštrovali sme ako sa dá pomocou základných herných udalostí poskladať komplexnejší scenár. Výsledná aplikácia je použiteľná a jej rozšírením a doplnením o spomínané fyziologické modely by mohla vzniknúť aplikácia použiteľná vo výuke.

Je nesporné, že simulačné hry vo výuke majú veľký potenciál a preto by som videl aj potenciál v tejto práci a v jej rozšírení, poprípade použitia niektorých jej častí. Avšak porozmýšľal by som nad všetkými plusmi a mínusmi použitej platformy. V prípade pokračovania by bolo nutné zapracovať na samotnom engine a jeho performance, poprípade zlepšenia celkového vizuálu 3D prostredia za použitia techník ako sú realtime tiene, realistickejšie osvetlenie scény a podobne. Bolo by nutné zlepšiť použiteľnosť častí editoru a zlepšiť tak efektivitu práce s ním.

Silverlight sa ukázal ako nie príliš vhodnou technológiou pre túto problematiku. Problémom bola implementácia vlastného 3D enginu. Vytvorený engine nakoniec splňoval všetku potrebnú základnú funkcionalitu, avšak je nutné podotknúť, že je to práca jedného človeka a engine by potreboval optimalizácie a zdokonaliť niektoré jeho časti. Bolo potrebné množstvo snahy, kým boli viditeľné nejaké výsledky a pri použití spomínanej Unity 3D by sa čiastočné výsledky dostavili ihneď. Nemuseli by sme implementovať časť editoru, serilizáciu scény, implementáciu fyziky a optimalizácie všemožného druhu a mohli by sme sa zamerať na prepojenie s modelmi v jazyku Modelica a navrhovaním scenárov.

V súčasnej dobe sa opúšťa od tvorby hier na zelenej lúke a využívajú sa herné enginy. Pri svojich riešeniach ostávajú hry a herné štúdia, ktoré majú hotové vlastné herné enginy a bolo by pre nich náročnejšie a často aj zbytočné prechádzať na iné technológie. Pri vývoji od piky sa samozrejme oplatí použiť niečo hotové. Z môjho pohľadu by bolo súčasne jednoduchšie upraviť framework Bodylight aby fungoval mimo Silverlight, ako sa pokúšať o vymyslenie kolesa.

Momentálne je nutné myslieť aj na multiplatformnosť aplikácii a Silverlight s podporou 3D bohužiaľ podporuje len Windows. Musíme myslieť aj na mobilné zariadenia a tablety a preto vidím veľkú budúcnosť technológií založených na používaní open sourcovej implementácií prostredia .NET : Mono. V súčasnosti existujú implementácie tohto prostredia pre väčšinu mobilných a desktopových zariadení. Jednou z technológií používajúcich Mono je Unity 3D.

Kapitola 10

Literatúra

[1] A. Sliney a D. Murphy, „JDoc: A Serious Game for Medical Learning,“ *IMCLab, Dept of Computer Science.*

[2] G. P. Tolentino, „Usability of Serious Games for Health,“ Universidade Estadual de Montes Claros, 2011.

[3] Clinispace, „About product,“ [Online]. Available: http://www.clinispace.com/product.html.

[4] Microsoft, „What Is a XNA Profile?,“ [Online]. Available: http://msdn.microsoft.com/en-us/library/ff604995.aspx.

[5] U. 3D, „Unity 3D about,“ [Online]. Available: http://unity3d.com/company/public-relations.

[6] „What are the pro/cons of Unity3D,“ [Online]. Available: http://gamedev.stackexchange.com/questions/8118/what-are-the-pro-cons-of-unity3d-as-a-choice-to-make-games.

[7] J. Kofránek, „Modelica,“ rev. *MEDSOFT*, 2013, ISSN 1803-8115.

[8] F. Ježek a P. Privitzer, „SADA VÝUKOVÝCH SIMULÁTORŮ - VÝSLEDKY VÝVOJE FRAMEWORKU,“ rev. *MEDSOFT*, 2013, ISSN 1803-8115.

[9] M. Tribula, M. Vavrek a O. Michal, „TVORBA 3D VÝUKOVÝCH APLIKACÍ POMOCÍ TECHNOLOGIE MICROSOFT SILVERLIGHT,“ *MEDSOFT,* 2012, ISBN 978-80-904326-5-9 .

[10] Wikipedia, „Game mechanics,“ [Online]. Available: http://en.wikipedia.org/wiki/Game_mechanics.

[11] B. Beneš, J. Sochor, P. Felkel a J. Žára, Moderní počítačová grafika, 2005, ISBN 80-251-0454-0.

[12] M. Evans, „Finite State Machines (are Boring),“ [Online]. Available: http://martindevans.me/heist-game/2013/04/16/Finite-State-Machines-(Are-Boring)/.

[13] „Game Creation with XNA/3D Development/Importing Models,“ [Online]. Available: http://en.wikibooks.org/wiki/Game_Creation_with_XNA/3D_Development/Importing_Models.

[14] Microsoft, „What is the Content Pipeline?,“ [Online]. Available: http://msdn.microsoft.com/en-us/library/bb447745.aspx.

[15] M. Talbot, „Silverlight Serializer,“ [Online]. Available: http://whydoidoit.com/silverlight-serializer/.

[16] JigLibX. [Online]. Available: http://jiglibx.codeplex.com/.

[17] T. Susi, M. Johannesson a P. Backlund, „Serious Games – An Overview,“ rev. *School of Humanities and Informatics*, 2007.

[18] J. Kofránek, „INTEGROVANÉ MODELY FYZIOLOGICKÝCH SYSTÉMŮ,“ Peaha, 2011.

Príloha A

Zoznam použitých skratiek

3D – Three dimensional (troj-rozmerný priestor)

3D – Two dimensional (dvoj-rozmerný priestor)

API – Application programming interface

IDE – Integrated development enviroment (vývojové prostredie)

UK – Univerzita Karlova

VR – Virtuálna realita

PC – Presonal computer (osobný počítač)

XAML – Extensible Application Markup Language, rozšírenie XML popisujúce grafické rozhranie aplikácií

EKG –Elektrokardiogram

OSX – operačný systém pre PC Macintosh

WISWYG – What you see is what you get

.NET – prostredie pre beh aplikácií pod operačným systémom Windows

Príloha B

Obsah priloženého CD a návod na ovládanie

V hlavnej zložke CD sa nachádza súbor README.txt. V tomto súbore sú popísané základné inštrukcie pre spustenie a ovládanie hry a editora, ktoré sa nachádzajú v zložke **Game a Editor**. V zložke **Documents** je umiestnená elektronická kópia hlavného dokumentu tejto diplomovej práce. V zložke **Project** je dôležitá podzložka **Pacient3D**, v ktorej je umiestnená projektová štruktúra otvoriteľná vo Visual Studiu.

Ovládanie hry

Ovládanie hry sa deje pomocou myši klikaním na jednotlivé objekty. Pohľad kamery sa ovláda jednoducho. Kliknutím a ťahaním myši kameru rotujeme a keď navyše podržíme medzerník, kamerou môžeme hýbať v rovine kolmej na jej pohľad.

Ovládanie editora

Ovládanie 3D pohľadu na scénu sa deje pomocou myši:

- **Stlačením stredného tlačidla myši + ťahanie myšou** – rotovanie pohľadu
- **Stlačením stredného tlačidla myši + ľavý Alt + ťahanie myšou** – posúvanie kamery v rovine kolmej na pohľad
- Objekty vyberáme kliknutím na ne

Výber giziem je možný v hlavnom menu, alebo pomocou skratiek:

- **Q** – gizmá sú vypnuté
- **W** – gizmo pre transláciu
- **E** – gizmo pre rotovanie
- **R** – gizmo pre zväčšenie/zmenšenie

Úprava stavových diagramov

- Pridanie pomocou tlačidla add state
- Označenie stavu a prechodu kliknutie myšou
- Mazanie pomocou tlačidla delete na klávesnici

- Pridanie prechodu pomocou myši – pomocou drag&drop zo spodného stredného manipulátoru každého stavu na stav cieľový
- Zmena tvaru stavu pomocou štyroch manipulátorov

Úprava komponet a game objektov a herných akcií

- Pridanie komponenty pomocou tlačidla Add Component
- Zmena parametrov vyplnením daných input polí
- Zmena referencií na iné objekty sa deje pomocou drag&drop zo stromovej štruktúry objektov, zoznamu herných akcií, atď

Ovládanie ďalších častí editoru sa deje pomocou príslušných tlačidiel