VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

POČÍTAČOVÉ HRY S TECHNOLOGIÍ UNREAL ENGINE

COMPUTER GAMES USING UNREAL TECHNOLOGY

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — JAKUB HAMAL

AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN ROUPEC, Ph.D.

SUPERVISOR

BRNO 2013

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2012/13

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Jakub Hamal

který/která studuje v **bakalářském studijním programu**

obor: **Aplikovaná informatika a řízení (3902R001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Počítačové hry s technologií Unreal Engine

v anglickém jazyce:

Computer Games Using Unreal Engine Technology

Stručná charakteristika problematiky úkolu:

Cílem práce je zvládnout technologii tvorby počítačových her. Autor v rámci práce vytvoří ukázkové hry, které budou tématicky vhodné k propagaci studia na ÚAI.

Cíle bakalářské práce:

Autor se seznámí s technologií tvorby počítačových her se zaměřením na Unreal Engine. Následně vytvoří ukázkové aplikace, které budou vhodné pro propagaci studia na ÚAI.

Seznam odborné literatury:

Thorn, A.: UDk Game Development. Course Technology, 2012.

Vedoucí bakalářské práce: Ing. Jan Roupec, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2012/13.

V Brně, dne 13. 02. 2013

L.S.

Ing. Jan Roupec, Ph.D.
Ředitel ústavu

prof. RNDr. Miroslav Doupovec, CSc., dr. h. c.
Děkan

ABSTRAKT

Obsahem této práce je seznámení a popis technologie Unreal Engine 3. Jedná se o technologii, která je využívána k vytváření her a aplikací pro různé platformy. Tato práce se zabývá tvorbou her pro PC. V první části práce jsou vysvětleny techniky, použité v Unreal Engine 3, v druhé je popsáno vývojové prostředí Unreal Development Kit a jak s ním pracovat. Poslední část se zabývá vytvořením ukázkové aplikace, ve které jsou předvedeny techniky popsané v prvních dvou kapitolách.

ABSTRACT

The content of this thesis is to present and describe Unreal Engine 3 technology. It is technology, which is used to creating games and applications for different platforms. This work deals with creating PC games. Techniques, use dun Unreal Engine 3 are subscribed in the first charter and Integrated Development Enviroment, known as Unreal Development Kit is subscribed in the second charter. The last charter deals with creating sample application, in which are demonstrated techniques and methods, subscribed in the first and second chapters.

KLÍČOVÁ SLOVA

UE3, Unreal, engine, UDK, prostředí, editor

KEYWORDS

UE3, Unreal, engine, UDK, enviroment, editor

PROHLÁŠENÍ O ORIGINALITĚ

Prohlašuji, že jsem tuto práci vypracoval samostatně, pod vedením pana Ing. Jana Roupce, Ph.D.

……..
Jakub Hamal
22. května 2013

BIBLIOGRAFICKÁ CITACE

HAMAL, J. *Počítačové hry s technologií Unreal Engine.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. 47 s. Vedoucí bakalářské práce Ing. Jan Roupec, Ph.D.

PODĚKOVÁNÍ

Chtěl bych poděkovat panu Ing. Janu Roupcovi, Ph.D. za poskytnuté rady a pomoc při psaní této práce.

Obsah

1 Úvod

Počítačové hry jsou nedílnou součástí softwarového průmyslu a vznikají za účelem bavit uživatele, nejlépe po dlouhou dobu. Jejich historie sahá do počátku padesátých let, kdy začaly vznikat první zábavné programy. Od prvních, převážně textových, her se dostáváme k hrám současné doby: 2D/3D strategie, střílečky z pohledu první osoby, různé simulátory, nebo hry vytvořené pouze pro hru více hráčů. Hry všech těchto kategorií jsou postavené na určité technologii- herní engine. V něm je obsaženo grafické zpracování, fyzika modelů, hudební podklad hry, různé vstupy/výstupy aplikace a program pro různé úrovně ve hře, tzv. Level editor. V této práci se budu věnovat engine Unreal od firmy Epic Games. Jedná se o historicky druhý herní engine vytvořený speciálně pro 3D aplikace.

1.1 Unreal engine historie

Vydání první verze Unreal engine se datuje do roku 1998, kdy byl použit ve stejnojmenné hře Unreal. Za autora lze považovat Tima Sweeneyho, který vytvořil hru ZZT, což je první oficiální hra firmy Epic Games (v té době Epic Mega Games). Při psaní ZZT použil Sweeney objektově orientovaný jazyk Turbo Pascal a vytvořil tak základ pro herní Engine Unreal 1. Společně s první verzí byl zveřejněn také speciální programovací jazyk UnrealScript, vytvořený právě pro tuto technologii.

Následující verze Unreal 2 byla ovlivněna očekávaným přesunem uživatelů od PC k herním konzolím (PlayStation2) a měla poskytovat základ jak pro PC hry, tak pro hry na PS2. Největší přínos této verze tedy spočíval v multiplatformovosti. Poprvé se objevila ve hře America's Army.

Vydání Unreal 3 hodně souviselo se zaváděním HD rozlišení na konzole. Po představení rozdílu v grafickém zpracování mezi Unreal 2 a Unreal 3 se podařilo vedení Epic Games přesvědčit Microsoft ke zdvojnásobení základní paměti RAM v jejich zařízeních (256MB =>512MB). V roce 2009 byla uvolněna základní verze Unreal 3 volně ke stažení. Jako vývojové prostředí slouží Unreal Development Kit (dále UDK), které je distribuováno spolu s touto technologií [1].

Verze Unreal 4 byla zatím pouze představena, ale žádná hra na ní postavena není. Její vydání je svázáno s vydáním konzole Play Station 4, na které je očekáváno její největší využití.

Mezi jednotlivými verzemi nelze najít žádný vyloženě velký rozdíl z hlediska programování a funkčnosti. Při srovnání her stejného žánru používajících technologii Unreal 1 a Unreal 3, tak funkčně jsou hry v podstatě identické. Vyvíjelo se grafické zpracování a zobrazení, což se promítlo v podpoře nově vydávaných grafických ovladačů (DirectX) a postupným vylepšováním zobrazovací technologie. Na Obr. 1 je vidět, že grafický základ všech verzí je velmi podobný, ale rozdíl je v detailním zobrazení. Na postavě vlevo, vytvořené pomocí technologie Unreal 1, je poznat že k vymodelování bylo použito podstatně méně bodů (vrcholů) a vykreslení je více hranaté než u postavy vpravo, vymodelované technologií Unreal 3, u které je patrné zaoblení všech přechodů a hran.

Obr. 1 Grafická podobnost mezi UE1, UE2 a UE3

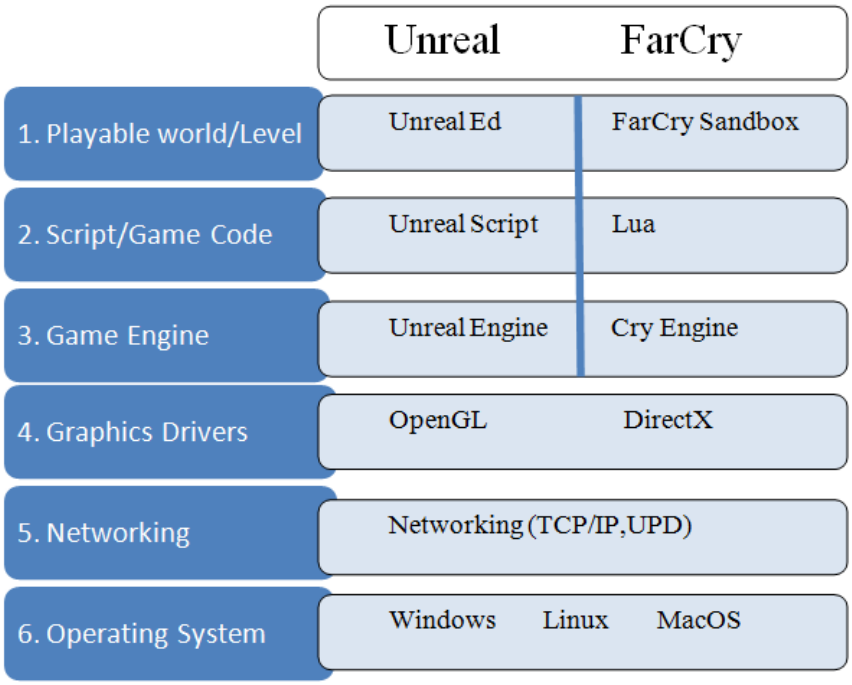
1.2 Obsah práce

Tato práce se zabývá tvorbou her a aplikací využívajících technologii Unreal Engine 3. Pomocí této technologie lze vytvořit mnoho druhů aplikací od ukázkových scenérií přes mapy a módy do hry Unreal Tournament až po samostatně fungující hry. Práce je pojata jako tutoriál a z toho důvodu je rozdělena do tří hlavních kapitol. V první kapitole jsou vysvětleny techniky, s nimiž se v Unreal Engine pracuje. V další kapitole je představeno vývojové prostředí UDK a vysvětleno jak v něm pracovat a závěrečná kapitola se věnuje aplikací prvních dvou při tvorbě aplikace. Pro tvorbu hry jsem si vybral verzi UDK-2012-07, a to z důvodu nedostatečné kompatibility nejnovější verze UDK-2013-02.

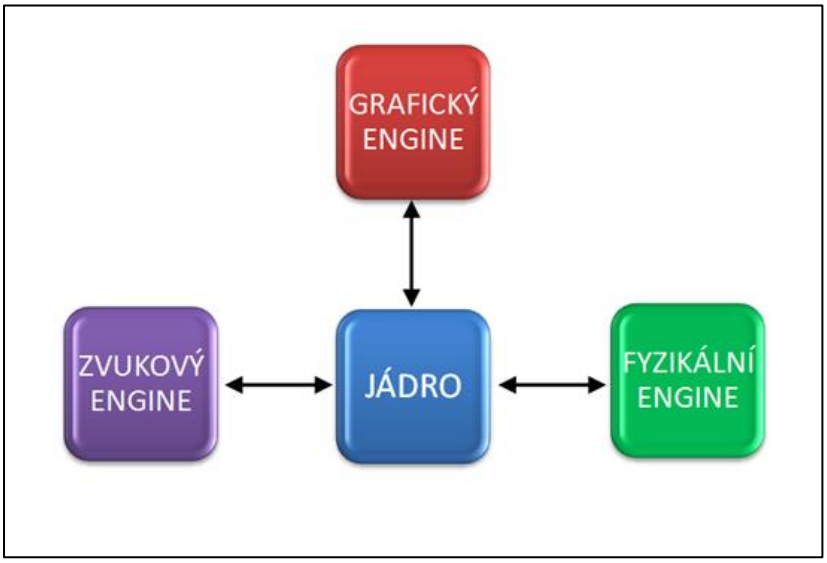
2 Unreal engine 3

Při vývoji počítačových her je využíváno mnoho herních engine. Každý z nich má svoji unikátní část, tzv. jádro, které vytváří základní odlišnost mezi nimi. Dále se mohou, ale nemusí, lišit ve způsobu práce s grafikou, se zpracováním zvuku, fyzikou, přístupu na internet a vývojovým prostředím. Mezi nejznámější neplacené herní engine patří právě Unreal Engine a Cry Engine od firmy Crytek vyvinutý pro hru FarCry a volně dostupný od roku 2011. Naopak mezi placené engine Source Engine od fimy Valve použitý např. ve hře Half-Life 2 a jejích modifikacích. Na Obr. 2 je vidět porovnání technologií použitých v UE a Cry Engine. Je zde vidět, že části 1 (Vývojové prostředí), 2 (skriptovací jazyk) jsou rozdílné, ale položky 4, 5 a 6 týkající se podporovaných grafických ovladačů, přístupu k síti a podporovaných Operačních Systémů už jsou společné pro oba.

Dále je nutné si uvědomit, že herní engine není pouze jádro napsané v nějakém programovacím jazyku, ale je to soubor několika částí, které spolu přes toto jádro komunikují. Tyto části jsou sice nazývány jako engine, ale bez jádra nemohou fungovat a každá z nich se zabývá pouze určitou oblastí, tudíž nelze vytvořit fungující hru s použitím jen jedné z nich. Na Obr. 3 jsou zobrazeny nejdůležitější části herního engine a znázorněna jejich komunikace s jádrem. Tyto části budou v této kapitole postupně probrány a vysvětleny.

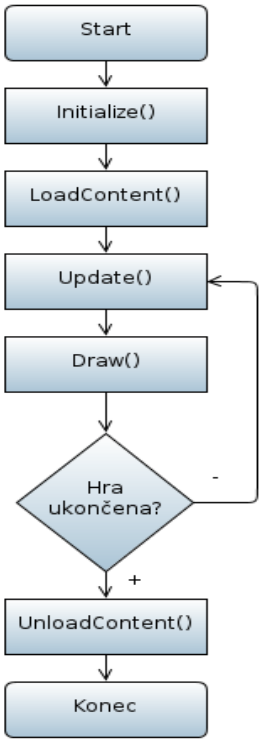


Obr. 2 Porovnání Unreal Engine a Cry Engine



Obr. 3 Architektura Engine

Řízení událostí UE3 je obstaráváno herní smyčkou. Mezi takto řízené události patří např. komunikace mezi jednotlivými částmi engine nebo vstupy a výstupy ovládání. Obecná herní smyčka je na Obr. 4. Znázorňuje spuštění aplikace, poté inicializaci a načtení obsahu, výpočet událostí, následné zobrazení výsledků a porovnání, zda byla aplikace ukončena. Pokud ne, provede se nový výpočet a vykreslení a opět se testuje ukončení hry. Tento cyklus se opakuje do okamžiku ukončení hry. Poté následuje odstranění dat z paměti a ukončení aplikace.



Obr. 4 Obecná herní smyčka

2.1 Grafický engine

V této kapitole se budeme zabývat grafickou částí UE3. Jedná se o nepostradatelnou část herního engine, bez které by hra byla v podstatě jen program zpracovávající určitá data a reagující na vstupy od uživatele, ale bez jakékoliv vizuální zpětné vazby. Má na starost hlavně objekty, jejich osvětlení a stínování a samozřejmě zobrazení samotné aplikace na obrazovce. Zajišťuje samozřejmě i různé animace při spuštění nebo vypnutí a různé In-Game movies, ale pro seznámení s UE3 nám postačí první 3 zmíněné. Pro vykreslování obrazu používá UE3 multi-vláknový renderovací systém Gemini 64- bit HDR. Zkratka HDR znamená High Dynamic Range a je to technologie umožňující větší dynamický rozsah expozice scény. Mezi metody pro výpočet a vykreslení obrazu, podporované tímto systémem, patří mimo jiné per-pixel lighting (výpočet vykreslení pro každý pixel zvlášť) a Photon lighting (technologie pro výpočet globálního osvětlení). Ze softwaru pro zpracování obrazu na PC je podporován Direct3D 9 popř. jeho novější verze Direct3D 10 nebo 11.

2.1.1 Gemini

Systém Gemini používá vykreslovací postup na bázi vrstev. V praxi to znamená, že výsledný obraz je složený z několika podobrazů, přičemž na každém z nich je zachycena část výsledného celku. Tyto podobrazy jsou zastoupeny vrstvami. Všechny vrstvy jsou oindexovány a tento index určuje pořadí na vykreslení. Obecně jsou vrstvy s vyšším indexem vykresleny nad vrstvami s indexem nižším. Kromě toho je možné použít pro každou vrstvu zvlášť tzv. blending function, což je funkce sloužící k míchání vrstev a s její pomocí se nastavuje průhlednost, prolínání a mísení jednotlivých vrstev. Samotné vykreslování je řízeno objektovým programováním a funkcemi takto napsanými.

Kromě indexovaných vrstev je možné pracovat s vrstvami, které navolí programátor. Toto se používá např., pokud je při vykreslování pozadí potřeba obejít nějaké omezení, dané systémem Gemini. Tato funkce dělá z Gemini velmi flexibilní vykreslovací systém.

Souřadnicový systém se shoduje se souřadnicovým systémem OpenGL, tzn. osa x je rostoucí zleva doprava a osa y roste zespodu nahoru. Počáteční bod (0,0) se na obrazovce nachází v levém spodním rohu. Při práci s objekty je každému objektu přidělen referenční bod, kolem kterého dochází k rotaci a změně měřítka. S výjimkou čar se referenční body nachází ve středu objektu, v případě čar je to první bod v řadě [2]. Na Obr. 5 je ukázka kódu, pomocí kterého Gemini vykreslí čtverec o straně 100 a středem posunutým ze souřadnic (0,0) na (100,100).

```
local rect = display.newRectangle(0,0,100,100)
rect.x = 100
rect.y = 100
```

Obr. 5 Gemini kód

2.1.2 Objekty v UE3

Velmi důležitou funkcí, kterou má grafický engine na starost je práce s objekty. Pod tím si můžeme představit funkce pro jejich vkládání do scény, vykreslování a práce s osvětlením a stínováním těchto objektů. Objekty lze rozdělit na statické a dynamické. U statických objektů se jejich vlastnosti nemění v závislosti na čase, tzn. po jejich vložení do scény je vypočítáno osvětlení, stínování a kolize a tyto hodnoty zůstávají stejné, dokud uživatel nezmění jejich umístění popř. další vlastnosti.

Druhým typem objektů jsou objekty dynamické. Ty se chovají přesně naopak. Po vložení do scény jim jsou vypočítány vlastnosti, ale ty se mohou měnit v závislosti na různých vnějších podnětech. V UE3 jsou používány 3 typy objektů. První z nich jsou Brushe. Ty spadají do kategorie statických objektů a jedná se o objekty vytvořené přímo v UDK. Dalším typem objektů jsou StaticMesh objekty. Ty lze zařadit jak do statických, tak do dynamických objektů. Záleží pouze na uživateli. Tyto objekty byly vytvořeny v modelovacím programu a následně vloženy do editoru. Poslední typ jsou SkeletalMesh objekty. Ty jsou podobné jako StaticMesh, ale oproti nim mají navíc kostru, která umožňuje animovat tyto objekty ve hře a jedná se o čistě dynamické objekty. Skeletal Mesh jsou také objekty vytvořené v externím modelovacím programu.

Při vytváření aplikace se používají všechny tyto typy, ale každý má svůj účel. Brush objekty jsou použity pro vytvoření základního tvaru, StaticMesh pro docílení detailnosti a pro pohyblivé objekty, jako např. auta, použijí se SkeletalMesh.

2.1.3 Osvětlení

Další, velmi důležitá, funkce grafického engine je vytvoření osvětlení a stínování scény. Jedná se o jednu z nejdůležitějších částí při tvorbě hry. Hráč jakožto jakýkoliv jiný člověk má určitou představu, jak by osvětlení mělo vypadat. Přesněji řečeno očekává, že objekt vrhá stín určité velikosti určitým směrem jako v reálném světě. Také je kladen důraz na větší či menší kontrast a nasvícení pouze části objektu. Toto všechno samozřejmě záleží na umístění zdroje světla a jeho vzdálenost od objektu. Pokud se nepovede správně rozmístit zdroje světla, tak je těchto požadavků dosaženo jen velmi obtížně a ve výsledku může špatné osvětlení pokazit celkový dojem ze hry.

V UE3 jsou podle využití ve scéně rozlišovány 4 druhy zdrojů světla:

- **Point lights –** bodové osvětlení, svítí všemi směry.
- **Spot lights –** bodové osvětlení, svítí pouze určitým směrem, př. reflektor.
- **Directional lights –** používá se pro simulaci slunečních paprsků.
- **Sky lights –** světlo okolí, nahrazováno LightMass systémem [10].

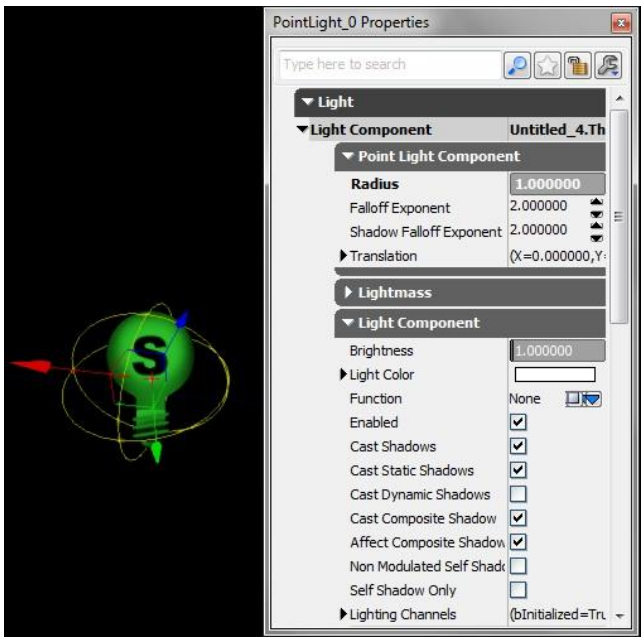
Také je možné nastavit jednotlivým objektům, aby vyzařovaly světlo. Takto nastavené objekty se poté chovají jako Point lights, ale na rozdíl od nich vyzařované světlo neovlivňuje ostatní objekty ve scéně. Příklad tohoto použití je na Obr. 6 jako fosforeskující cedule nouzového východu.

Obr. 6 Světlo vyzařované objektem

2.1.4 Light mapy a Shadow mapy

Pokud hovoříme o osvětlení scény a jeho výpočtu, je třeba si uvědomit, že se nepočítá každý zdroj zvlášť. Výpočet probíhá současně pro celou scénu a zahrnuje všechny světelné zdroje v ní umístěné. Takto vypočítané nasvětlení je uloženo do Light mapy, popř. Shadow mapy pokud hovoříme o stínech. Jedná se v podstatě o kopii scény, ve které je uloženo veškeré vypočítané osvětlení a vystínování, ale nic víc. Pokud by byla scéna spuštěna bez Light mapy, tak by nebyly vidět objekty. Byl by to pohyb poslepu. Naopak pokud by se zobrazila pouze Light mapa, tak by uživatel viděl pouze světlo a stíny v místech umístění objektu, ale neviděl by celkový vzhled objektu.

Light i Shadow mapy zahrnují jak statické tak i dynamické osvětlení. Rozdíl mezi nimi je stejný jako u statických a dynamických objektů. Statické osvětlení je při spuštění předpočítáno a dále se ve hře nemění a má velmi vysokou kvalitu. Oproti tomu dynamické osvětlení se s každým pohybem přepočítává a po skončení výpočtu se uloží nové. Pokud je prováděno příliš mnoho výpočtů v krátkém čase, kvalita zpracování může být horší.



Obr. 7 Zdroj světla s vlastnostmi

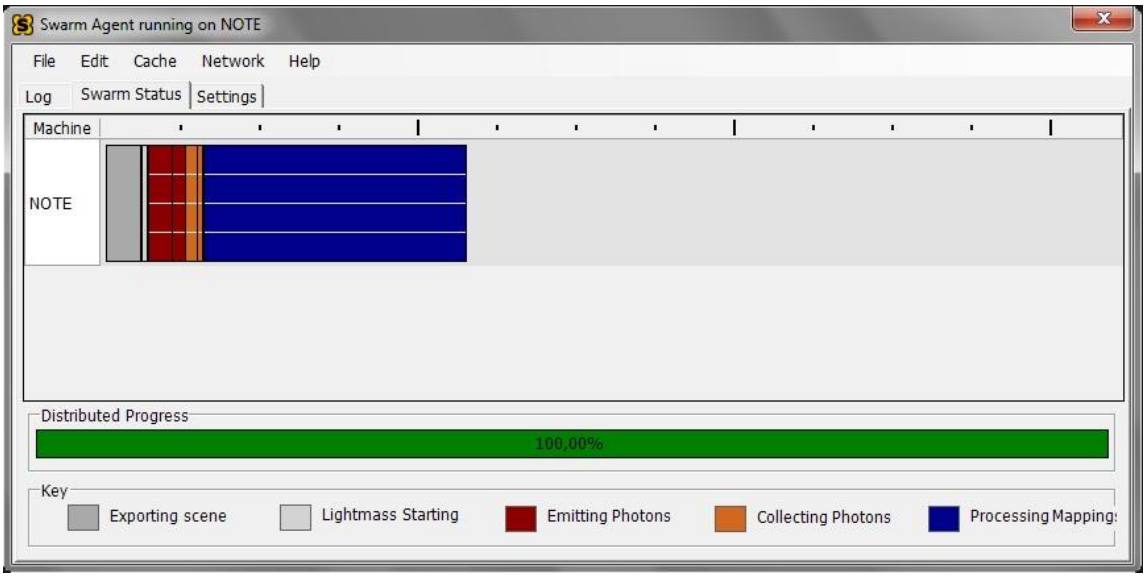
2.1.5 Lightmass systém

Pro vytváření co nejdetailnějšího osvětlení se UE3 používá kombinace Light a Shadow map se systémem Lightmass. Pomocí tohoto systému lze vytvářet detailnější nasvětlení, než pomocí Light map, ale za cenu vyšší výpočtové náročnosti. Slouží mimo jiné pro výpočet různých dynamických prvků, jako je třeba stínování oblastí a rozdílnost v odrazovosti (světlo při odrazu od objektu přebírá některé jeho vlastnosti, např. změní barvu v závislosti na barvě povrchu) ze kterých poté vytváří tzv. mapu světla s různými akcemi světla. Jeho umístění ve scéně je kolmé na standardní Light a Shadow mapy, takže Lightmass tyto komplexy v podstatě překryje a doplní o kvalitnější osvětlení. Vzhledem k náročnosti na výpočet je jen na vývojáři a jeho vybavení, zda tento systém použije nebo ne.

I bez použití Lightmass lze dosáhnout kvalitního nasvětlení a rozdíl mezi použitím a nepoužitím tohoto systému není nijak velký, jak je vidět na Obr. 8. Levá část je s použitím Lightmass a vidět hlavně přesnější vykreslení stínů než na pravé části, kde je Lightmass vypnut a stíny jsou nedokonalé [3].

Obr. 8 Rozdíl v nasvětlení s Lightmass a bez

Průběh výpočtu světla s použitím Lightmass je řízeno aplikací Unreal Swarm, která umožňuje také řízení výpočtu na principu vzdáleného přístupu. Pomocí Unreal Swarm lze monitorovat postup vytváření Lightmass a sledovat, jaká část výpočtu právě probíhá, a které přístroje jsou aktuálně používány a v jaké míře. Na Obr. 9 můžeme vidět, jak vypadá okno Swarm Agent po skončení vytváření Lightmass.



Obr. 9 Unreal Swarm Agent

2.2 Kvalita obrazu

UE3 umožňuje měnit kvalitu vytvářeného a zobrazovaného obrazu při samotné tvorbě hry. Výhoda této možnosti je znatelná pokud do hry vkládáme objekty a prozatím nepotřebujeme vidět všechny detaily, např. pokud kontrolujeme umístění a návaznost textur, tak je praktičtější použít nižší kvalitu. Naopak při kontrole správného odrazu světla a stínování se uplatní vyšší kvalita.

2.3 Zvukový engine

Kromě grafického engine je nedílnou součástí celého kompletu zvukový engine. Jak už název napovídá, má na starost výpočet a přehrávání zvuku. To je popsáno speciálními třídami v programovém kódu, který je obsažen v jádru samotného UE3. Základní příkazy pro vytváření a zpracovaní zvuku jsou uloženy ve třídě UAudioDevice. Její obsah a název závisí na tom, pro jaké zařízení je hra vytvářena. Pokud se jedná o hru pro PC, používá se třída UALAudioDevice, u her po PS3 je to třída UPS3AudioDevice a pro hry na Xbox360 se použije třída UXeAudioDevice. Další, velmi důležitá třída je UAudioComponent objects. Ta obsahuje informace o objektech, jejichž zvuková stopa může být přehrána pomocí UAudioDevice. Její podtřída USoundNode objects zase uchovává aktuální zvuková data, která jsou vložena do scény. Jelikož je UE3 technologie pro 3D aplikace tak je samozřejmá i podpora 3D zvuku [4].

2.4 Fyzikální engine

Poslední částí UE3, kterou si představíme, je fyzikální engine. Ten zajišťuje detekci a výpočet kolizí a interakci mezi objekty a okolním světem a následné promítnutí výsledků v aplikaci a v neposlední řadě i samotnou fyziku scény. V UE3 je pro fyziku používán systém PhysX od firmy Nvidia. Pro výpočet kolizí tento systém používá metodu rigid body, která bude popsána později. Pro práci s fyzikou a kolizemi je nutné objektům typu Brush, StaticMesh a SkeletalMesh, popsané výše, přiřadit kolizní box a vlastnosti. PhysX s nimi pak pracuje jako s fyzikálními objekty a ty se dělí na KActor a KAsset. Kromě kolizí lze objektu přiřadit silové zatížení popř. předem danou reakci na některý podnět. V UE3 jsou aplikovány tři typy silových modelů:

- **Forces**- působení konstantní síly (gravitace).
- **Impulse**- reakce na jednu nebo více předem daných akcí (výbuch).
- **Constraint**- cyklické silové zatížení, používá se pro neustále se opakující akce a reakce (houpačka).

Poté již záleží na uživateli, jak s těmito modely naloží dále a jak detailní fyziku chce vytvořit. Důležité je zvolit fyzikální systém, podle kterého se kolize a fyzika bude zpracovávat. Tyto systémy jsou v UE3 dva:

- **Kinematická fyzika**- provádí přesně to, co uživatel chce; ignoruje kolize.
- **Dynamická fyzika**- realistické reakce na kolize; pro ovládání je nutno užít Force nebo Constraint model.

2.4.1 PhysX

Jedná se o fyzikální engine sloužící k vytváření fyziky v reálném čase. Vyvinut byl firmou Aegie v roce 2004, která se ale v roce 2008 stala součástí společnosti Nvidia, zabývající se výrobou a distribucí grafických karet a softwaru pro ně určeného. Fyzikální engine programátorům umožňuje výpočet a zpracování fyziky bez nutnosti psaní vlastního (zpravidla složitého) kódu. Původně byl systém PhysX používán na PPU (physics processing unit) kartách, což byly grafické karty se speciální jednotkou pro výpočet fyziky. Po přechodu firmy Aegia pod firmu Nvidia se od tohoto modelu upustilo a byl nahrazen modelem GPU (graphics processing unit), používaným dodnes. Tento model je podstatně rychlejší a efektivnější díky vysoce paralelní struktuře zpracování. GPU jednotka může být usazena na vrcholu grafické karty, pokud je jí PC vybaveno, nebo je integrována přímo do základní desky. Integrované GPU ale nedosahují takových výkonů jako samostatné grafické karty [5].

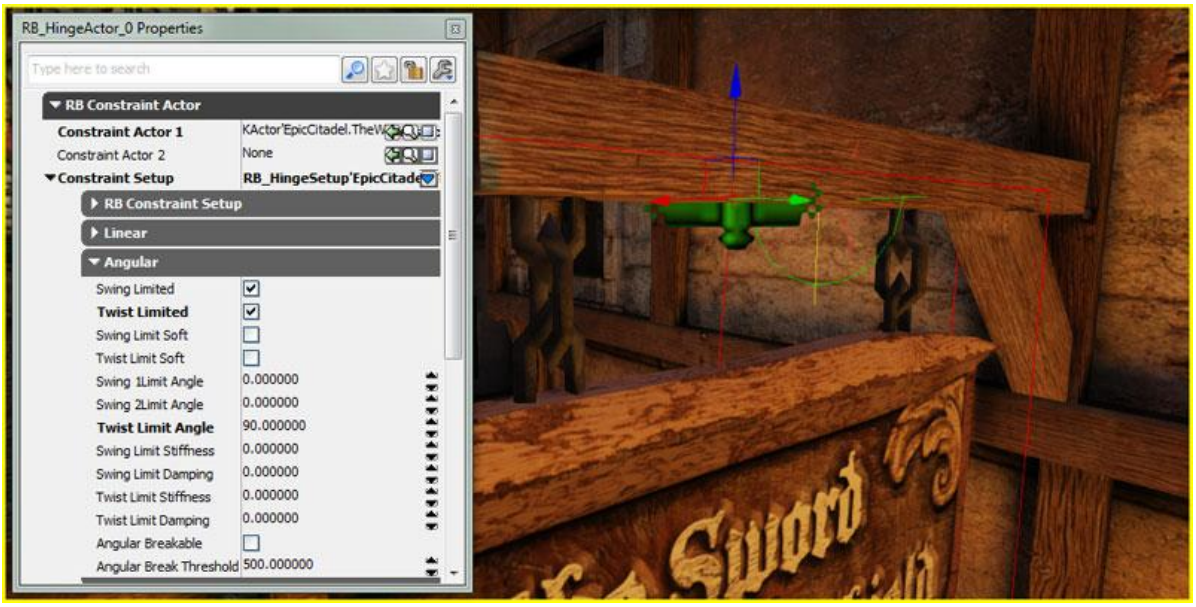
2.4.2 Rigid bodies

Jak bylo zmíněno, systém PhysX pro kolizní modely používá metodu rigid bodies. Tato metoda spočívá ve vytváření fyzikálně tuhých objektů, které se při kontaktu nedeformují, kolem objektů vložených do scény. Tento systém se používá i ve fyzikálních simulacích a to z důvodu menší výpočtové náročnosti. Charakteristická vlastnost tohoto fyzikálního výpočtu je, že takové objekty se nechovají stoprocentně realisticky při kolizích, ale velmi se tomu přibližují [6].

Přiřadit, např. objektu typu StaticMesh, model rigid body je možné několika způsoby. První z nich je automatické vložení pomocí nástroje Add RigidBody, který se nachází v panelu nástrojů, který se zobrazí po kliknutí pravým tlačítkem na daný objekt. Po užití tohoto nástroje bude automaticky vygenerován rigid body, kopírující geometrii základního objektu. Druhý způsob je náročnější a spočívá ve vytváření rigid body pomocí Brushe. Tímto způsobem se vytváří nehomogenní kolizní systém jednoho objektu, což znamená, že kolizní model jednoho objektu je vytvořen z více částí a není pro celý objekt stejný. Lze tak dosáhnout velmi unikátních kolizních vlastností jako např. díra ve zdi, skrz kterou lze nejen viděl, ale je možné skrz ni třeba střílet. Tohoto by automatickým generováním dosáhnout nešlo.

2.4.3 KActor

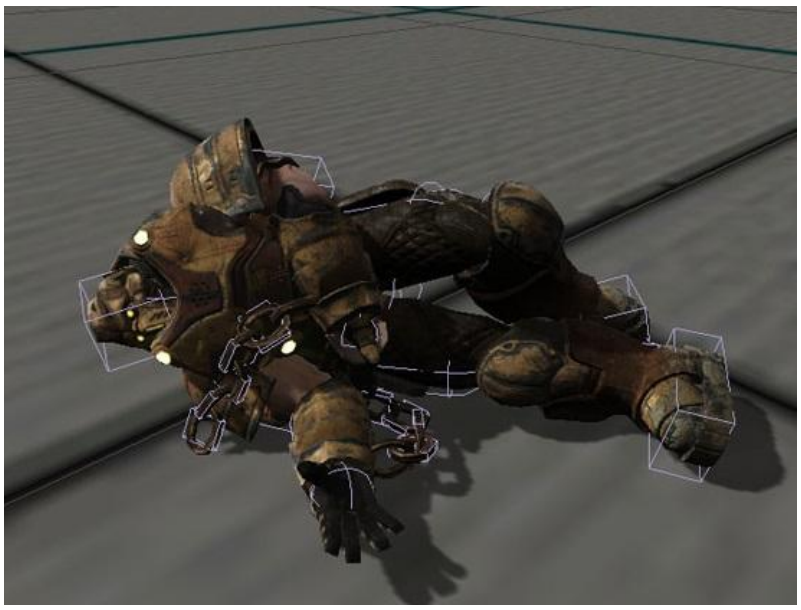
Prvním typem fyzikálních objektů jsou objekty KActor. Objekty tohoto typu se používají pro tzv. homogenní fyzikální model. To znamená, že fyzikální a kolizní model je stejný pro celé těleso a skládá se pouze z jedné části. Pro vytvoření tohoto typu objektů slouží jako základ objekt typu StaticMesh. Do scény je ale vložen jako KActor objekt. Takto vložený objekt je stále registrován jako StaticMesh bez kolizních vlastností. Poté se, jedním z výše popsaných způsobů, k tomuto objektu vytvoří kolizní model rigid body a nakonec jsou tyto dva modely spojeny do finálního KActor objektu s kolizními vlastnostmi [7]. KActor objekty se používají pro oživení jednoduchých modelů jako je např. krabice, která se po kolizi začne pohybovat, dveře otevírající se zatlačením, nebo prosté houpající se cedule, jejíž KActor model je zachycen na Obr. 10.



Obr. 10 Objekt typu KActor

2.4.4 KAsset

Druhým typem fyzikálních objektů jsou KAsset objekty. Tento typ se na rozdíl od KActor používá výhradně pro animované objekty nebo pro objekty, u nichž je nemožné dopředu předpovědět chování při animaci. Jako základ se používají objekty SkeletalMesh a druhá odlišnost od KActor je nehomogenní fyzikální model. Zjednodušeně řečeno se SkeletalMesh rozdělí na části, které se hýbou a části sloužící jako podpěrné body nebo klouby. Jelikož se SkeletalMesh skládá z tzv. kostí, tak kolem každé kosti, která je zahrnuta v pohyblivých částech objektu, se vytvoří samostatné rigid body a výsledná fyzika a kolizní model se neustále přepočítává. Takto rozdělený model můžeme vidět na postavě zobrazené na Obr. 11. Kvádry kolem různých částí zastupují jednotlivé rigid body, jejichž poloha a natočení se neustále mění a proto musí být s každou změnou fyzika a kolize přepočítany.



Obr. 11 KAsset model pro animaci postavy

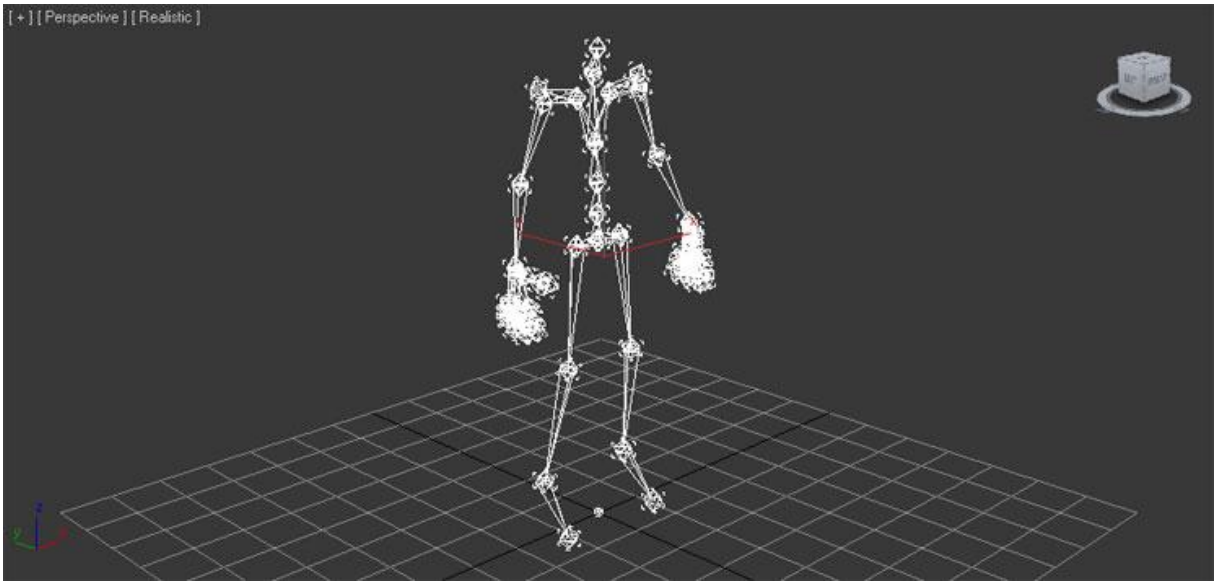
2.5 Animační systém

Jak již bylo zmíněno výše, animace se vytváří na objektech, které obsahují skelet neboli kostru. Aspekty, které ovlivní samotné animování, lze rozdělit do dvou kategorií. První z nich je animování samostatných prvků a druhá je vytváření kombinací těchto jednotlivých animací. Pokud tedy na jednu kost skeletu působí jedna animace, výsledný animační model je závislý pouze na parametrech této kosti. Jestliže ale na jednu kost působí více animací, tak výsledný model se rovná kombinaci všech působících animací a k tomu jsou započítány vlastnosti samotné kosti jako např. rychlost a směr pohybu nebo okamžité natočení. V UE3 jsou rozlišovány tři druhy animací: Skeletální animace, Morph animace a Face animace.

2.5.1 Skeletal animace

Jedná se o animace jednotlivých SkeletalMesh objektů. Vytvoření a import těchto objektů jsme si popsali výše, takže můžeme rovnou přejít na samotné animování. Při vytváření animace se každé kosti přidělí vrcholy s indexováním 0.0 – 1.0. Tento index určuje procentuální ovlivnění kosti při animaci.

Důležitá vlastnost skeletálních animací se týká výpočtu animace. Ten se neprovádí pro každý vrchol samostatně a následně spojení výsledků do jedné animace, ale počítá se pro celý objekt. Tento výpočtový model je použit z důvodu celkového zrychlení animací, protože výpočet pro jednotlivé vrcholy je jak výpočtově tak časově podstatně náročnější. Jakmile je animace na objektu vytvořena, je uložena jako samostatný model, nikoliv tedy jako součást daného objektu. To se provádí z důvodu pozdějšího užití stejné animace na jiném objektu. Ukázkový skelet postavy rozložený na kosti a vrcholy je vidět na Obr. 12.



Obr. 12 Skelet pro animaci v 3ds Max

2.5.2 Morph animace

Druhým typem animací jsou animace typu Morph. U tohoto typu, na rozdíl od Skeletal animací, je výpočet prováděn pro každý animační vrchol samostatně a výsledná animace je dána kombinací těchto výpočtů. Vzhledem k použité výpočtové metodě se mezi Morph animace řadí animace deformací a interakcí mezi objekty. Pro vytvoření Morph animace je třeba interakce mezi minimálně dvěma objekty. Jeden z nich je označen jako zdrojový (vyvolávající deformaci nebo interakci) a ostatní jsou cílené (podle nich se deformace řídí). Takto označené objekty se pro řízení animování spojí do jednoho modelu funkcí MorphTargetSet. Samotné animování probíhá jako výpočet a znázornění stavu vrcholů zdrojového objektu při deformaci. Tento výpočet je řízen parametry vrcholů cíleného objektu. Tyto vrcholy se musí nastavovat ručně a pro správný chod Morph animace musí mít cílený objekt stejný počet vrcholů jako objekt zdrojový.

2.5.3 FaceFX animace

Poslední druh animací zastupuje animování mimiky obličeje. Stejně jako u Morph animací, i zde se objekty spojují do modelu MorphTargetSet. U tohoto typu se ale nejedná o několik objektů se vzájemnou interakcí, ale o jednotlivé kosti a vrcholy skeletu (konkrétně obličeje), které jsou zapojeny do mimiky tváře. Animace je počítána jako animace celku jako u Skeletal animace a to umožňuje podstatné zrychlení jinak velmi složitých animací. Využití tohoto systému je zejména při různých video sekvencích, ve kterých je zobrazena mluvící postava. Mimika obličeje při mluvě se pak provádí pomocí FaceFX. Pro práci s těmito animacemi je ve vývojovém prostředí integrován nástroj FaceFX Studio [8].

2.6 UnrealScript

Poslední technologií, používanou v UE3, je programovací jazyk UnrealScript. Jedná se o programovací jazyk, ve kterém jsou naprogramovány funkční prvky hry, jako např. chování umělé inteligence nebo instrukce prováděné při startu a v průběhu chodu aplikace. Jeho autorem je Tim Sweeney a jako základ použil jeho dřívější programovací jazyk ZZT-oop. Jedná se o objektově orientovaný jazyk velmi podobný jazyku Java, ale na rozdíl od Javy, UnrealScript rozlišuje velká a malá písmena. Jelikož se jedná o objektově orientované programování, tak jako základní konstrukční prvek jsou použity třídy. V nich jsou uloženy metody a funkce použité k jednotlivým objektům. Jednotlivé třídy jsou uloženy v samostatných souborech, jejichž název se shoduje s názvem třídy. Pro tyto soubory je charakteristická přípona .uc nebo .uci.Velmi podstatnou vlastností UnrealScriptu je absence podpory mnohonásobné dědičnosti mezi třídami. V praxi to znamená, že každá třída může dědit vlastnosti pouze od jedné rodičovské třídy.

Syntaxí se UnrealScript podobá nejen Javě, ale obsahuje také společné znaky s jazyky C/C++. Tyto dva jazyky byly použity při programování jádra UE, ovšem při používání nekomerční verze UE je tento kód nepřístupný. Modifikovat lze pouze kód napsaný v Unrealscript, nebo vytvářet jednoduché skripty pomocí nástroje Kismet. Vytvářet a modifikovat třídy v jazyce Unrealscript lze např. v programu Microsoft Visual C# Express, ale existují i programy speciálně pro to určené, jako třeba Unreal X-Editor [9].

Jednou z nejdůležitějších tříd je třída GameInfo. Obsahuje totiž většinu z hlavních příkazů pro chod hry. Na Obr. 13 je ukázka kódu z třídy GameInfo, konkrétně funkce StartMatch, která je volaná při zahájení hry. Velmi užitečná věc pro programátora, která byla použita při programování UE3 pomocí UnrealScript, je okomentování jednotlivých funkcí ve třídách. Téměř každá funkce začíná komentářem, který vysvětluje, k čemu tato funkce slouží. Tento způsob komentování umožňuje podstatně lepší orientaci v kódu a zároveň pomáhá novým vývojářům snadněji porozumět způsobu, jakým je UnrealScript napsán. Komentáře mohou být jednořádkové, uvedené znaky // a ukončené koncem řádku, nebo víceřádkové, které jsou ohraničené symboly /* na začátku a symboly */ na konci komentáře.

```
/* StartMatch()
Start the game - inform all actors that the match is starting, and spawn player pawns
*/
function StartMatch()
{
    local Actor A;

    if ( MyAutoTestManager != None )
    {
        MyAutoTestManager.StartMatch();
    }

    // tell all actors the game is starting
    ForEach AllActors(class'Actor', A)
    {
        A.MatchStarting();
    }

    // start human players first
    StartHumans();

    // start AI players
    StartBots();

    bWaitingToStartMatch = false;

    StartOnlineGame();

    // fire off any level startup events
    WorldInfo.NotifyMatchStarted();
}
```

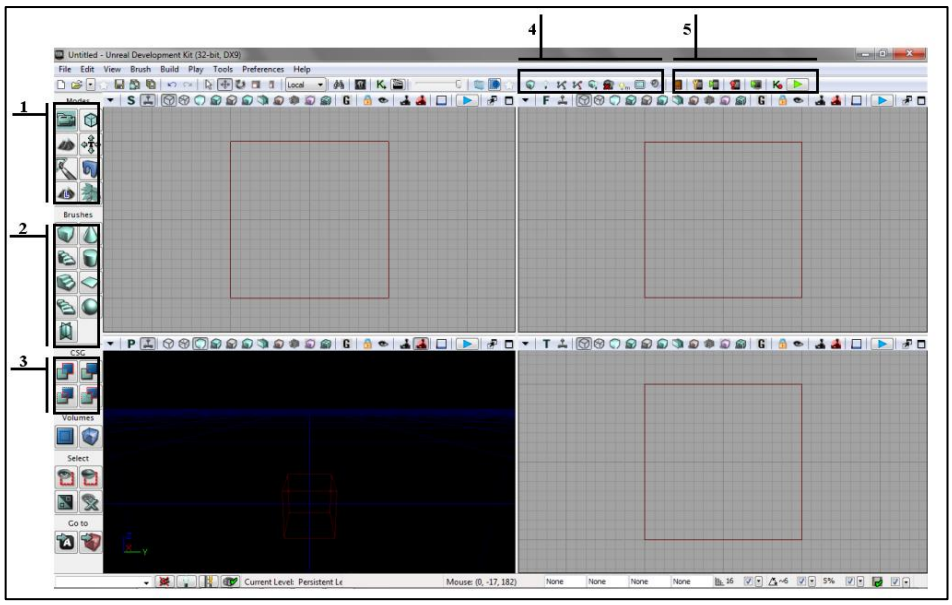
Obr. 13 UnrealScript – ukázka

3 Unreal Development Kit

V této kapitole se budeme zabývat softwarem Unreal Development Kit (dále jen UDK), což je oficiální vývojové prostředí pro UE3. Distributor nabízí dvě licence tohoto programu. První z nich slouží pro nekomerční účely a je zcela zdarma, ale je omezena z hlediska přístupu ke zdrojovému kódu jádra. Druhá verze je pro komerční účely a je k dispozici za prvotní poplatek 99 dolarů, ale určité procento z prvních výdělků je odvedeno firmě Epic Game, vlastnící práva na UE. Pomocí UDK je možné vytvářet hry a aplikace pro PC a pro iOS (operační systém produktů firmy Apple) a spolu s ním jsou dodávány programy SpeedTree Modeler, sloužící k vytváření animačních efektů, Unreal Frontend, který slouží k exportu aplikací z UDK do samostatného souboru a Unreal iOS Configuration, používaný pro konfiguraci aplikací pro iOS. V této kapitole si postupně popíšeme vývojové prostředí UDK, jak vkládat do scény objekty a pracovat s nimi, vytváření skriptů pomocí Kismet, nástroje pro práci s animacemi, vytvoření uživatelského rozhraní scény a export výsledné aplikace.

3.1 Vývojové prostředí

Vývojové prostředí je velmi podobné jako u programů pro modelování a kreslení. Hlavní část obrazovky je zastoupena různými, maximálně čtyřmi, pohledy na vytvářenou scénu a na stranách jsou umístěny nástroje a panely. Konfigurace pohledů záleží čistě na zvyku vývojáře, základní nastavení je 2*2 pohledy. V UDK jsou rozlišeny čtyři typy pohledu: čelní pohled, pohled na půdorys, pohled ze strany a pohled zobrazující aktuální vzhled vytvořené scény. Další možností konfigurace zobrazení je volba viditelnosti u daného pohledu. V praxi to znamená zobrazení reálného nasvětlení, odstranění stínů, zobrazení pouze hran objektů nebo prosté osvětlení scény vytvořené editorem. Detailnější rozbor vývojového prostředí je na Obr. 14 s odkazy na objasnění jednotlivých nástrojů v Tab. 1. V oblasti 1 se nachází různé editační módy, z nichž aktivní může být pouze jeden, v oblasti 2 jsou funkce na vytváření Brush objektů, oblast 3 zahrnuje vkládání Brush objektů do scény, v oblasti 4 jsou nástroje pro kompilaci různých částí aplikace a v oblasti 5 jsou funkce pro spuštění v testovacím módu na různých zařízeních.



Obr. 14 UDK uživatelské rozhraní

Oblast	Tlačítko	Funkce
1	Camera Mode	Volný pohyb kamerou
	Geometry Mode	Editace geometrie objektů
	Terrain Editing Mode	Nástroj pro editaci krajiny
	Texture Alignment Mode	Nástroj pro editaci textur
	Mesh Paint Mode	Kreslení na objekty typu Mesh
	Static Mesh Mode	Mód pro úpravu speciálních vlastností statických objektů
	Landscape Mode	Mód pro editaci krajiny, nahrazuje Terrain Editing Mode
	Foliage Mode	Nástroj pro vytváření a editaci rostlin
2	Brushes	Přednastavené geometrické tvary s možností editace vlastností
3	CSG Add	Vložení geometrického objektu do mapy
	CSG Substract	Vyhloubení prostoru do objektu
4	Build	Nástroje pro kompilaci geometrie, světla, zvuku a cest
	Graphic Quality	Nastavení kvality zobrazení
5	Test on	Spuštění aplikace v testovacím módu na PC, konzoli nebo iOS

Tab. 1 Popis uživatelského rozhraní UDK

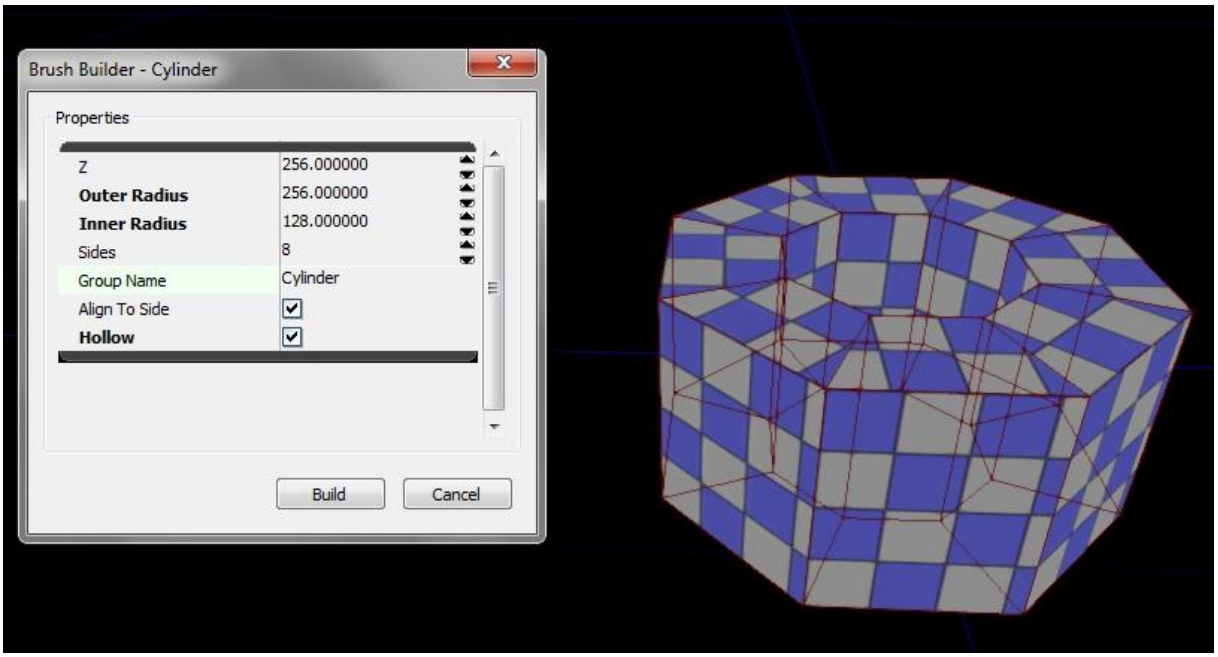
3.2 Content Browser

Tento nástroj patří mezi nejdůležitější věci, které se týkají UDK. Jedná se v podstatě o složku, ve které jsou uloženy všechny textury, materiály, StaticMesh objekty, SkeletalMesh objekty, různé zvukové efekty a animační modely vytvořené v UDK. V této složce se také provádí import textur a objektů z modelačních programů, nastavování jejich vlastností a vytváření nových materiálů. Všechny tyto funkce si popíšeme níže. Spolu s UE a UDK jsou ve volně dostupném balíčku k dispozici i některé originální textury a objekty od firmy EPIC.

3.3 Vytvoření objektu

V první kapitole jsme si popsali druhy objektů, s nimiž se v UE3 pracuje. Nyní si ukážeme, jak tyto objekty vytvořit a následně vložit do scény, abychom s nimi mohli dále pracovat. Jednoduché statické objekty typu Brush je možné vytvářet přímo v editoru UDK přes panel nástrojů Brushes. V UDK je přednastaveno devět základních tvarů od jednoduchého kvádru až po točité schodiště. Po zvolení požadovaného tvaru se otevře okno nastavitelných vlastností, kde se nastaví rozměry, počty vrcholů, sklon atd. Jakmile si nastavíme všechny vlastnosti, následuje umístění vytvořeného objektu ve scéně. Zatím je do scény vložen pouze geometrický obrys objektu. Aby byl Brush objekt vložen natrvalo do scény, je třeba použít funkci z panelu CSG, konkrétně CSG Add. Pomocí této funkce je vytvořený model vyplněn. V panelu CSG se ještě nachází funkce CSG Substract, která má přesně opačný účel tzn., že slouží k vyhlubování prostoru. Dalším krokem je poté nanesení textury na tyto objekty. To bude popsáno v další části. Objekty typu Brush, vytvořené přímo v editoru, mají po vložení do scény automaticky přiřazeny kolizní model, který kopíruje geometrický obrys objektu [10]. Ukázka vložení Brush objektu je zobrazena na Obr. 15.

Pro vytvoření objektů StaticMesh a SkeletalMesh se používají různé externí modelovací programy jako je např. Blender, nebo 3ds MAX. Jakmile je v takovém programu model vytvořen, tak jeho importování do scény se provede přes Content Browser, což je nástroj vývojového prostředí UDK. Poté už stačí model vložit do scény a dále s ním podle libosti pracovat a upravovat ho. Takto vytvořené modely jsou ve většině případů importovány do editoru spolu s texturou, takže odpadá povinnost texturování. Naproti tomu tyto objekty postrádají kolizní model, který je třeba vytvořit jedním ze způsobů, popsaných v první kapitole. Tyto objekty se používají pro oživení hry a vytvoření detailnosti scény.



Obr. 15 Vytvoření Brush objektu

3.3.1 Import a tvorba textur

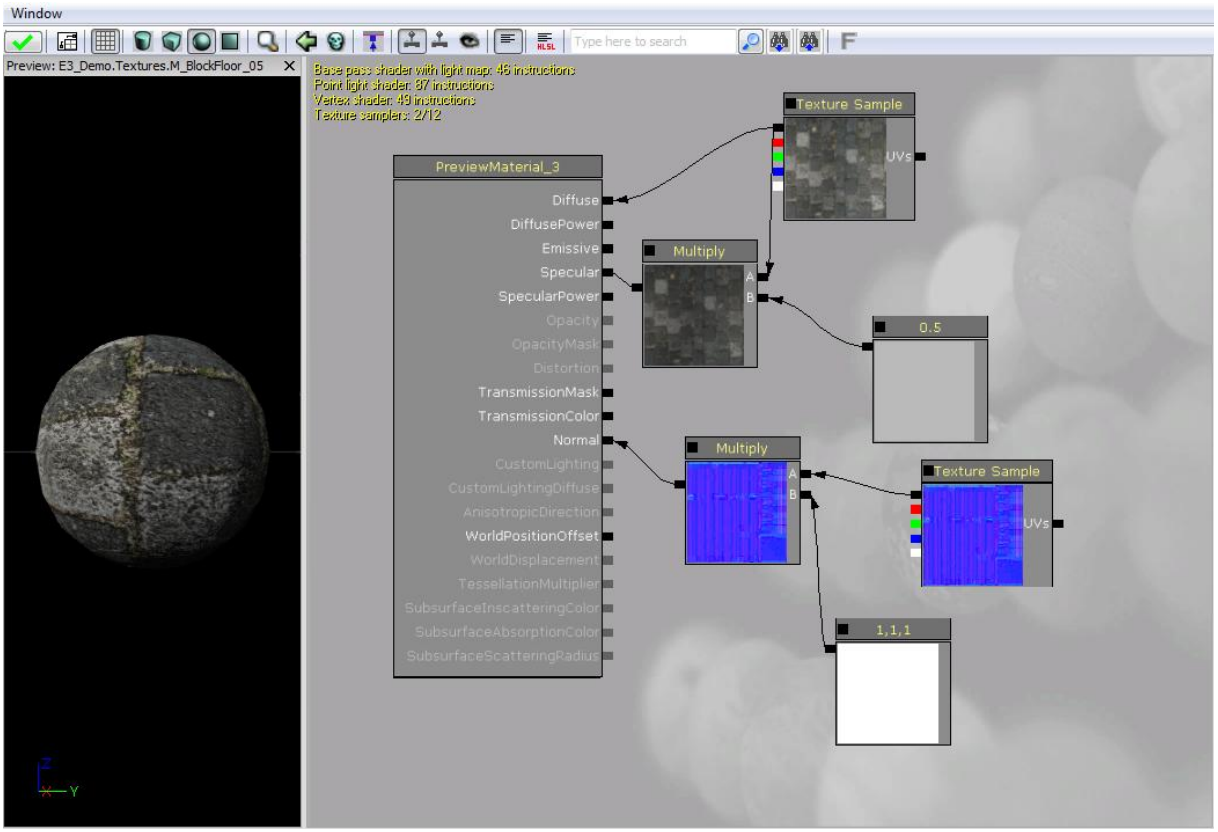
Po vytvoření objektu a vložení do scény je třeba k danému objektu vložit textury. Pojmem textura je myšlen grafický vzhled objektu, který uživatel vidí ve hře. Jako základ slouží libovolný obrázek, ze kterého se po importu do UDK vytvoří přiřazením vlastností materiál. Ten je následně použit jako textura na pokrytí objektu. Důležité je, aby importovaný obrázek byl ve formátu .bmp a jeho velikost byla mocnina 2 např. 1024x512 pixelů [10]. Pro lepší přehlednost je dobré vytvořit si v Content Browser vlastní balíček, do kterého se budou ukládat všechny importované textury a modely. Ušetří to čas při následném použití.

Po importu obrázku do UDK se v Content Browser obrázek jeví jako Texture2D. Aby mohl být aplikován na objekt, musí z něj být vytvořen materiál. Přes funkci CreateNewMaterial se otevře okno pro vytváření materiálu Material Editor a následně se přiřadí daný obrázek. Novému materiálu lze přiřadit řadu vlastností. Vytváření těchto vlastností se podobá blokovému programování (výběr a spojování funkčních bloků). Takto lze materiálu přiřadit nespočet vlastností. Mezi základní vlastnosti patří:

- **Diffuse color –** zastupuje množství světla, které se od povrchu odrazí rovnoměrně všemi směry, hodnota (1,1,1) znamená 100% odraz do všech směrů. Tato vlastnost nezávisí na úhlu pohledu, ale závisí na normálách jednotlivých pixelů.
- **Emissive color** – tato vlastnost udává, kolik světla daný materiál vyzařuje. Ačkoliv se poté materiál chová jako zdroj světla, tak vyzařované světlo neovlivňuje ostatní povrchy scény.
- **Specular color** – stejně jako Difuse color představuje odrazivost povrchu. V tomto případě se ale nejedná o rovnoměrný odraz všemi směry. Tato vlastnost je závislá na úhlu pohledu a sice největší odraz probíhá ve stejném směru, v jakém směřuje pohled.
- **Opacity** – vztahuje se k průsvitnosti povrchu a vyjadřuje množství světla, jež povrchem prostoupí. Při nastavení na 1 materiál nepropouští žádné světlo, naopak při nastavení na 0 je materiál zcela průsvitný.
- **Normal** – pomocí této funkce se nastavuje normála povrchu, podle které se určuje orientace povrchu. Tato vlastnost je ovlivňována jak odrazivostí, tak zrcadlením daného povrchu.

Obrázek se do editoru nahraje přes blok TextureSample. Tento blok má na pravé straně jeden vstup, do kterého se připojují editační funkce týkající se obrázku samotného jako je např. natočení nebo zvětšení. Na levé straně se pak nachází 5, barevně rozlišených, výstupů, které se připojují do hlavního panelu nastavení materiálu. Hlavní výstup je označen černou barvou a připojuje se k funkci Diffuse v panelu materiálu.

Material Editor také umožňuje vytvářet materiály pomocí sloučení, mísení nebo prolínaní několika základních vrstev, maximálně lze užít 13 obrázků pro vytvoření jednoho materiálu. Tato podpora je velmi důležitá, neboť velmi často nenajde vývojář obrázek, odpovídající jeho představě, a proto je lepší si vytvořit vlastní, než použít jiný a zároveň tak snížit grafickou kvalitu scény. Výsledný materiál se pak aplikuje do scény označením požadovaných povrchů, označením materiálu v Content Browser a následným použitím funkce add_selected_material. Na Obr. 16 je ukázka materiálu dostupného s UDK. V levém horním rohu je koule s výsledným vzhledem materiálu a na pravé straně jsou blokově nastaveny jeho vlastnosti. Jedná se o materiál použit v sérii Unreal Tournament.



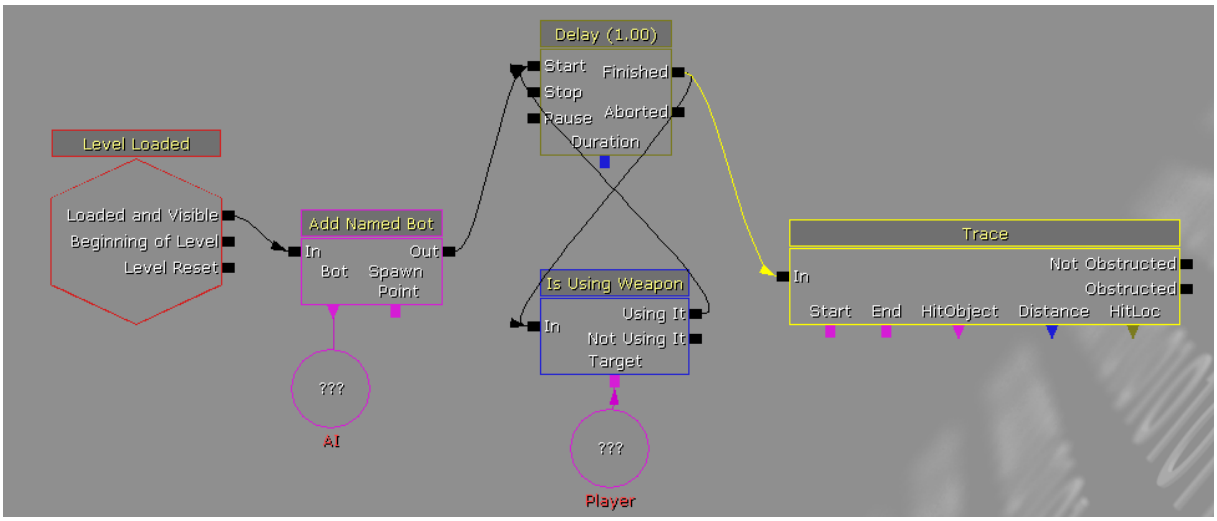
Obr. 16 Ukázka materiálu

3.4 Unreal Kismet

Kromě vytváření objektů a práci s nimi lze přímo v editoru vytvářet jednoduché skripty, ovlivňující aplikaci. K tomu slouží skriptovací prostředí Unreal Kismet. Skripty se vytváří pomocí funkčních bloků, které se navzájem propojují dráty. Bloky zastupují události a akce vytvořené uživatelem a díky podpoře hierarchie skriptů lze i velmi složité a obsáhlé cykly přehledně upravovat a řídit. Vytváření skriptů probíhá velmi logicky a řídí se zásadami matematiky, podmíněné logiky, hierarchií uspořádání a provádění akcí. Díky takto vytvořeným skriptům lze rozšířit již aplikované funkce, jako je třeba chování umělé inteligence, nebo vytvořit úplně nové např. vytvoření nového uživatelského rozhraní nebo použití nové animace. V Kismet systému jsou rozlišeny čtyři kategorie funkčních bloků:

- **Event** – bloky vytvářející vstup do následné sekvence, popř. reagují na událost zastoupenou ve scéně. Tyto bloky jsou zastoupeny diamantem s červeným obrysem.
- **Action** – tyto bloky zastupují akci, vyvolanou blokem Event nebo přímo ze scény. Zobrazeny jsou jako obdélník se vstupy na levé straně, výstupy na pravé straně a konektory pro podmínky na spodní straně.
- **Contition** – podmínkové bloky. Nejsou ovlivněny scénou samotnou, ale ovlivňují následný chod dané sekvence skriptu. Zastoupeny jsou obdélníkem s modrým rámem.
- **Variable** – bloky zastupující proměnné. Uchovávají v paměti určitá data používaná během celého cyklu. Zastoupeny jsou kolečkem, jehož rám je barevně zbarvený podle následného užití bloku.

Vytvořené sekvence je možné uložit pro použití v jiných scénách. Takto uložené skripty jsou v Content Browser zobrazeny jako KismetSequence a pro jejich použití stačí použít funkci ImportSequence. Obr. 17 zachycuje jednoduchou sekvenci s použitím všech blokových kategorií. Její funkce spočívá ve vytvoření zástupce umělé inteligence při načtení scény, kontroly zda je v dosahu jiný hráč nesoucí zbraň a pokud ano tak ho sledovat. Blok Delay slouží ke zpoždění smyčky, aby nedocházelo k zahlcení skriptu a z tohoto důvodu by měl být použit po a před každým blokem kategorie Action.



Obr. 17 Unreal Kismet ukázka

3.5 Práce s animacemi

Podstatnou součástí jakékoliv aplikace je animační systém. Špatně vytvořené nebo umístěné animace mohou zkazit celkový dojem ze scény. V UDK se práce s animacemi rozlišuje do tří hlavních částí, konkrétně animace jednoho objektu nebo jeho části, propojení animací několika a animace vedlejších efektů. Pro každou část je v UDK speciální nástroj.

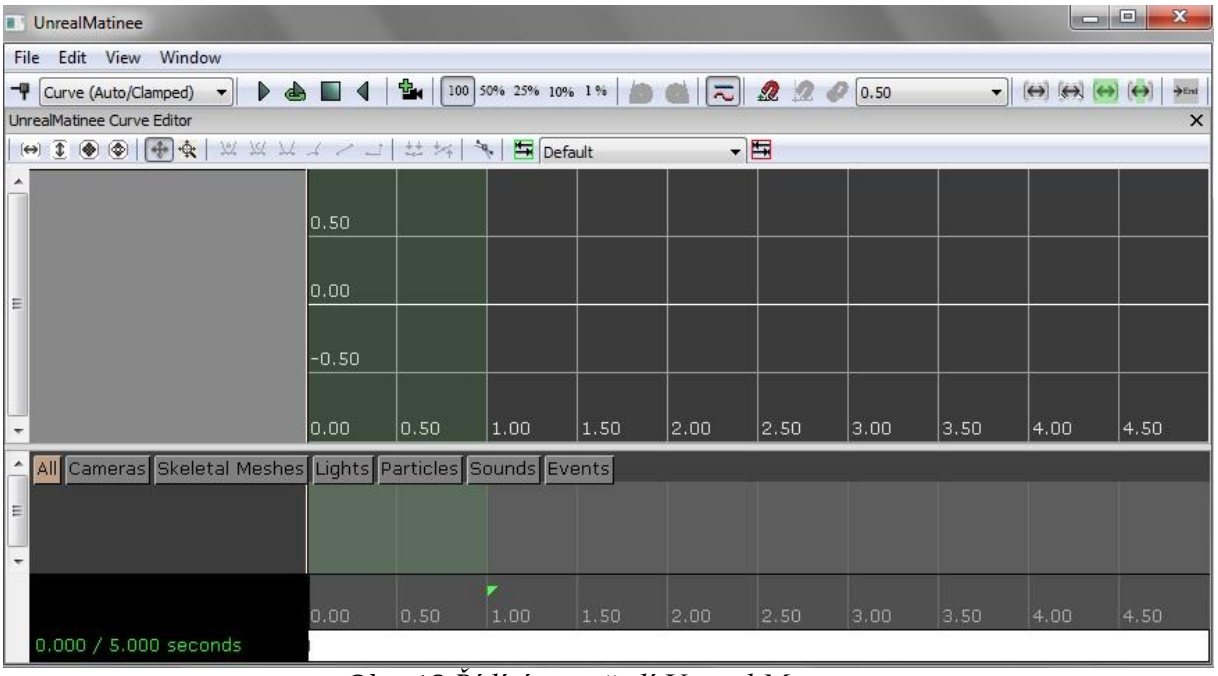
3.5.1 Unreal Matinee

První nástroj pro práci s animacemi, obsažený v UDK, je Unreal Matinee. Slouží k ovládání jednotlivých animací a video-sekvencí a jejich následné zapojení do scény. Velmi důležité je propojení se systémem Kismet. V něm je Matinee zastoupeno stejnojmenným blokem, který slouží pro zapojení výsledné animace do scény. Jelikož se jedná o řízení a časování animací, tak blok Matinee je obsažen vstupy. Do nich se připojují akce spouštějící popř. zastavující nebo jinak ovlivňující přehrávání animace, jako např. načtení scény nebo interakce určitých objektů.

Po vložení tohoto bloku do Ksimet skriptu a jeho následném otevření se zobrazí vývojové prostředí. To můžeme vidět na Obr. 18 a je velmi podobném různým programům pro editace videa. Poté následuje načtení sekvence z Content Browser a nastavení vlastností. Animaci je možno sledovat a editovat pomocí několika nástrojů:

- **Tracks** – stopy; sekvence jednotlivých animací.
- **Groups** – skupiny; do nich se řadí stopy, vztahující se k jednomu objektu.
- **Curves** – křivky; vykreslují časový průběh hodnot dané stopy.
- **Keyframes** – klíče; do nich se zapisují klíčové hodnoty jednotlivých křivek.
- **Tangents** – tangenty; určují tvar křivky v klíčových hodnotách.

Těmito nástroji lze kontrolovat danou animaci a sledovat její průběh. Výsledné hodnoty jednotlivých vrcholů, uložených v Keyframes, jsou na křivkách vyznačeny a jejich posloupnost lze sledovat na časové ose, zobrazené ve spodní části editoru.

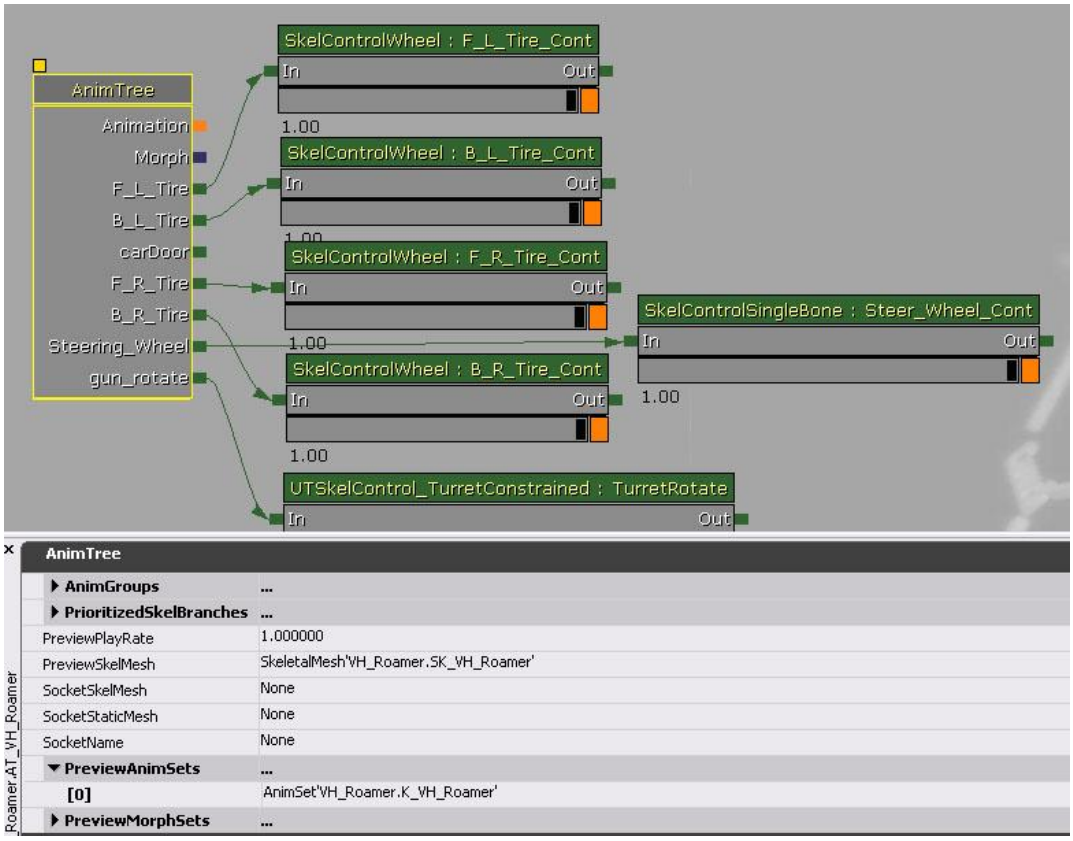


Obr. 18 Řídící prostředí Unreal Matinee

3.5.2 Unreal AnimTree

Tak jako Material Editor slouží k vytváření a práci s texturami a materiály, AnimTree umožňuje to stejné s animačními sítěmi. Jedná se o řízení chodu několika vzájemně propojených animací objektů typu SkeletalMesh. Také AnimTree je přímo propojen se systémem Kismet, ale není vybaven vlastním editorem. Pro vytvoření nové sítě animací slouží jako základní prvek je AnimTree Node, který se vloží do Unreal Kismet. Ten je opatřen dvěma hlavními konektory Animation a Morph. Ty umožňují vytvářet síť animací na přiřazeném objektu. Kromě těchto dvou konektorů může mít AnimTree Node libovolný počet dalších konektorů. V tomto případě se ale nejedná o vstupy/výstupy, nýbrž o ovládací prvky animací jednotlivých kostí skeletu. Okno editoru se podobné jako okno Material Editor. Pravá část se soustředí na vytváření animační sítě, vlevo lze vidět náhled přiřazeného skeletu a ve spodní části jsou umístěny jeho vlastnosti s možností úprav.

Jednoduchý AnimTree model je zobrazen na Obr. 19. Jedná se o řízení animace u bojového vozidla použitého v sérii Unreal Tournament, které je volně použitelné spolu s UDK. Na AnimTree Node je ke dvěma základním konektorům přidáno sedm nových, které se vztahují ke kostem skeletu. Mezi animované kosti u tohoto modelu patří kosti předního levého a pravého kola, zadního levého a pravého kola, dveří, kost pro natáčení volantu a kost otočného děla. Bloky ovládající animace jednotlivých kostí jsou zastoupeny funkcemi, které jsou naprogramovány spolu s herním jádrem. Po jejich připojení na konektory jednotlivých kostí se funkce s danou kostí spojí pomocí jejího názvu a vytvoří se tak odkaz na volání funkce pro připojenou kost.



Obr. 19 AnimTree bojového vozidla z UT

3.5.3 Unreal Cascade

Poslední nástroj pro práci s animacemi je Unreal Cascade. Ten slouží k řízení tzv. částicových animací jako třeba výstřely, exploze nebo mlha. Tyto animace jsou speciální v tom, že se nevztahují k objektům, ale jsou vytvářeny z částic, což jsou body vygenerované v prostoru a vystřelovány do okolí. Těmto bodům se říká emitory a jejich sloučení do skupiny se nazývá částicový systém. Prvním krokem je tedy vytvoření částicového systému a určení základní vlastnosti. Ta je volena podle následného využití systému:

- **Mesh** – speciální vlastnost, generující místo částic StaticMesh objekty (trosky po výbuchu).
- **Beam** – částice se generují mezi dvěma určenými body a jsou propojené (sníh).
- **Trail** – částice jsou spojené a při pohybu vytváří jednu stopu (výstřel).
- **Fluid** – vlastnost pro simulaci kapalin pomocí PhysX Fluid Solver.

Další krok je načtení tohoto systému do editoru. Ten je velice podobný jako u Matinee, tzn. obsahuje časovou osu, panel pro výběr částicového systému a jeho jednotlivých bodů, panel pro určení hodnot jednotlivých bodů a okno pro simulaci výsledné sekvence. Jakmile je sekvence, následuje její vložení do scény. V té je uložena jako Emitter a všechny tyto sekvence jsou sdružovány do modelu Sprite, což je plocha obdélníkového tvaru, natočená vždy kolmo ke kameře.

3.6 Tvorba uživatelského rozhraní aplikace

Uživatelské rozhraní je první věcí, která se zobrazí po spuštění aplikace a je tedy velice důležitá. Lze ho rozdělit na dvě hlavní části:

- **User Interface (UI)** – menu aplikace s různým nastavením.
- **Heads-up-Displays (HUD)** – informační panely.

V UE3 je možné vytvářet uživatelské rozhraní několika způsoby. Prvním způsobem je Canvas systém. Tento způsob se používá pro vytvoření uživatelského rozhraní nad samotnou aplikací, tzn. v pozadí je vidět scéna. Druhým typem je samostatné menu, zobrazené po startu aplikace, ale před načtením scény. Pro tento typ se používá systém ScaleformGFx, podporující technologií Adobe Flash. Jednoduché uživatelské rozhraní bylo možné vytvářet také přes Kismet, konkrétně funkci UIScene. Ta byla ale koncem roku 2010 z editoru odstraněna.

3.7 Export aplikace

Nedílnou součástí každého vývojového prostředí je nástroj pro vyexportování výsledné scény do samostatné aplikace, aby nemusela být neustále spouštěna přes editor. K tomuto účelu je spolu s UE3 a UDK dodáván nástroj Unreal FrontEnd. Ten umožňuje spojovat několik vytvořených scén, přidat k nim řídící soubory v jazyce UnrealScript atd. a to celé spojit do jedné aplikace a následně tento balíček vyexportovat jako instalační soubor pro vybrané cílové zařízení [10].

4 Ukázková aplikace

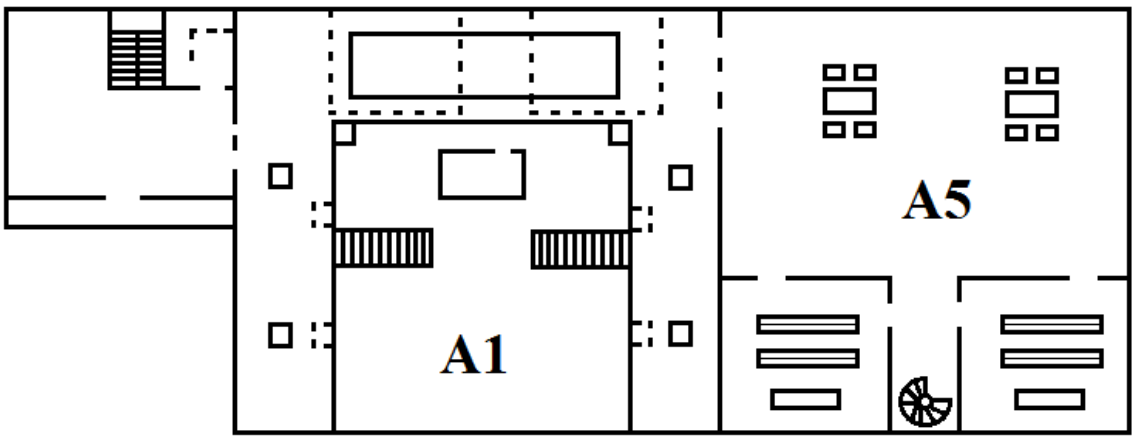
V závěrečné kapitole se budeme věnovat vytvoření jednoduché ukázkové aplikace, na které budou předvedeny některé techniky, představené dříve. V UDK je možné vytvářet dva základní druhy aplikací a sice hry nebo ukázkové scenérie. Pro tuto práci jsem zvolil hru, konkrétně 3D akci z pohledu první osoby s aplikací eliminačního módu. Aplikace je zaměřena spíše na předvedení skriptovacích možností UDK a z toho důvodu nejsou použity složité objekty vytvořené speciálně pro tuto hru, ale většina mapy je vytvořená z jednoduchých statických objektů. Hra bude mít vytvořené vlastní uživatelské rozhraní a nebude tedy použito rozhraní volně dostupné s UDK. Při vytváření jsem postupoval podle několika základních bodů:

- Návrh a vytvoření scény a grafického vzhledu.
- Vytvoření osvětlení.
- Naprogramování umělé inteligence.
- Vytvoření uživatelského rozhraní.

Při vytváření aplikace není nutné postupovat přesně podle těchto bodů, ale je třeba dodržet určitý chronologický postup. Není možné vytvářet osvětlení, pokud nevíme, jak scéna bude vypadat, nebo programovat umělou inteligenci, když není jasné, jak se má chovat. Naopak příliš nezáleží na tom, jestli uživatelské rozhraní vytvoříte před nebo po vytvoření scény.

4.1 Návrh scény

Jelikož by tato aplikace měla určitým způsobem propagovat studium na FSI VUT v Brně, tak jsem aplikaci umístil do areálu fakulty, konkrétně přízemí a prvního patra budov A1 a A5, Jelikož přízemí budovy A1 je hlavní vchod do fakulty, tak je tato místnost brána jako hlavní i v této aplikaci a hráč se bude objevovat právě zde. Pro přechody mezi patry bude sloužit několik schodišť. V místnosti A1 jsou umístěna 2 přímá, ve vedlejší místnosti bude umístěno schodiště s mezipatrem a v místnosti A5 jedno točité schodiště. Návrh scény je zobrazen na Obr. 20.



Obr. 20 Návrh scény

4.2 Vytvoření mapy

Po rozvržení scény následuje její vymodelování. Pro vytvoření základního tvaru byly použity jednoduché statické objektů typu Brush, vytvořené přímo v editoru. Také jsou použity objekty, dostupné přímo v UDK, konkrétně světla. Vytvoření místností a průchodů mezi nimi bylo provedeno pomocí vložení kvádrů funkcí CSG Add a jejich následným vyhloubením funkcí CSG Substract. Poté byly tyto místnosti stejným způsobem rozděleny na patra a v místnosti A1 byl vytvořen ochoz. Pomocí kvádrů tak byly vytvořeny židle i stoly a zábrany, zamezující pád z ochozu. Pro přechod mezi patry bylo vytvořeno schodiště pomocí nástroje pro tvorbu schodišť v panelu Brushes. Do vymodelované mapy je potřeba umístit body, sloužící pro navigaci a pohyb umělé inteligence. To se provede kliknutím pravého tlačítka do scény a použití funkce Add->Node->PathNode. Tímto příkazem se navigační bod do scény vloží a stačí ho jen umístit. S těmito body bude dále pracováno při programování postav umělé inteligence v Kismet. Poslední krok při tvorbě scény je umístění bodu, ve kterém bude začínat hru postava uživatele. Postup je stejní jako p navigačních bodů, jen místo PathNode je zvolen příkaz PlayerStart.

4.2.1 Textury

Jakmile je mapa vymodelována, je třeba vytvořit její grafický vzhled. K tomu slouží textury. I když byla mapa vytvořena podle interiéru budov A1 a A5, grafický vzhled byl použit vlastní a to ze dvou důvodů. Zaprvé z důvodu grafické náročnosti na výpočet při použití originálního vzhledu z fotografií a zadruhé kvůli špatnému kontrastu při zobrazení na PC. Z těchto důvodů byly pro základ textur použity obrázky, volně dostupné na webových stránkách, zabývajících se touto tématikou. Na Obr. 21 jsou tyto textury zobrazeny.

Obr. 21 Použité textury

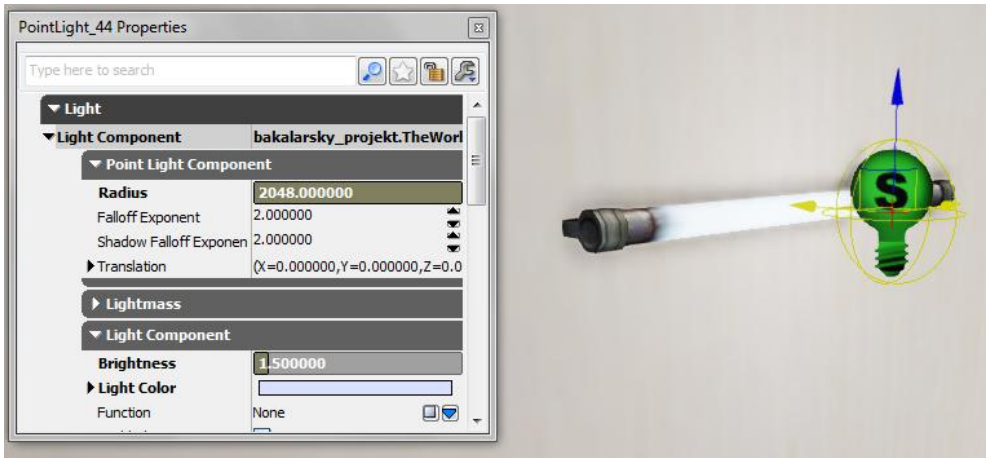
Použití textur	
1	Podlaha
2	Sloupy
3	Schody
4	Stěny
5	Lavice
6	Strop
7	Zábrany na ochozu
8	Tabule
9	Židle, stoly

Tab. 2 Použití textur

Tyto obrázky byly importovány do UDK a v Content Browser z nich byl vytvořen materiál pomocí nástroje Material Editor. Tento postup je popsán v předchozí kapitole. Nanesení materiálu na objekty se provede označením daného materiálu v Content Browser, poté označením ploch v aplikaci, na které má být daný materiál použit, a nakonec použití funkce Add Selected Material z panelu, zobrazeném po kliknutí pravým tlačítkem v aplikaci.

4.3 Osvětlení

Další fází, po vymodelování mapy a nanesením textur, je vytvoření osvětlení scény. To se provádí ve dvou krocích. První krok spočívá ve vkládání a umísťování objektů, které budou zastupovat zdroje světla, do scény. To je velmi důležité, protože světlo vyzařující odnikud, nepůsobí dobře. Druhým krokem je vytvoření tohoto zdroje. Pro tyto objekty byly použity objekty typu StaticMesh, volně dostupné s UDK. Po jejich umístění do scény byl před tyto objekty vytvořen zdroj světla. V této scéně jsou všechny zdroje světla zastoupeny typem Point Light. Jeho vložení do scény se provede kliknutím pravým tlačítkem a zvolením příkazu Add->Actor->PointLight. Po vložení byly tyto zdroje umístěny těsně před objekty, které je zastupují a byly jim nastaveny vlastnosti. Na Obr. 22 je zachycen zvolený objekt spolu s parametry světla, použitého ve scéně. Pro výsledný výpočet světla je použit systém Lightmass a doba tohoto výpočtu byla necelé 2 minuty.



Obr. 22 Světelný zdroj

4.4 Uživatelské rozhraní

Aby byla výsledná aplikace jedinečná, tak bylo vytvořeno vlastní uživatelské rozhraní. Jedná se o rozhraní, využívající systému ScaleformGfX a skládá se ze dvou hlavních částí: úvodní menu a závěrečné menu. Pro takovéto rozhraní slouží jako základ klip vytvořený v jakémkoliv Flash Editoru. Pro tuto aplikaci byly klipy vytvářeny v programu Adobe Flash Professional CS5. Klip pro úvodní menu obsahuje tlačítka pro spuštění hry, přechod na informační stránku a ukončení hry a je zobrazen na Obr. 23. Tato tlačítka jsou opatřena kódem v jazyku Java, konkrétně obsahují jednoduchý příkaz „fscommand (“ “)“, pomocí kterého je po stisknutí tlačítka vytvořena proměnná s hodnotou v závorce. S touto proměnou se bude dále pracovat při importu rozhraní do aplikace pomocí Kismet.

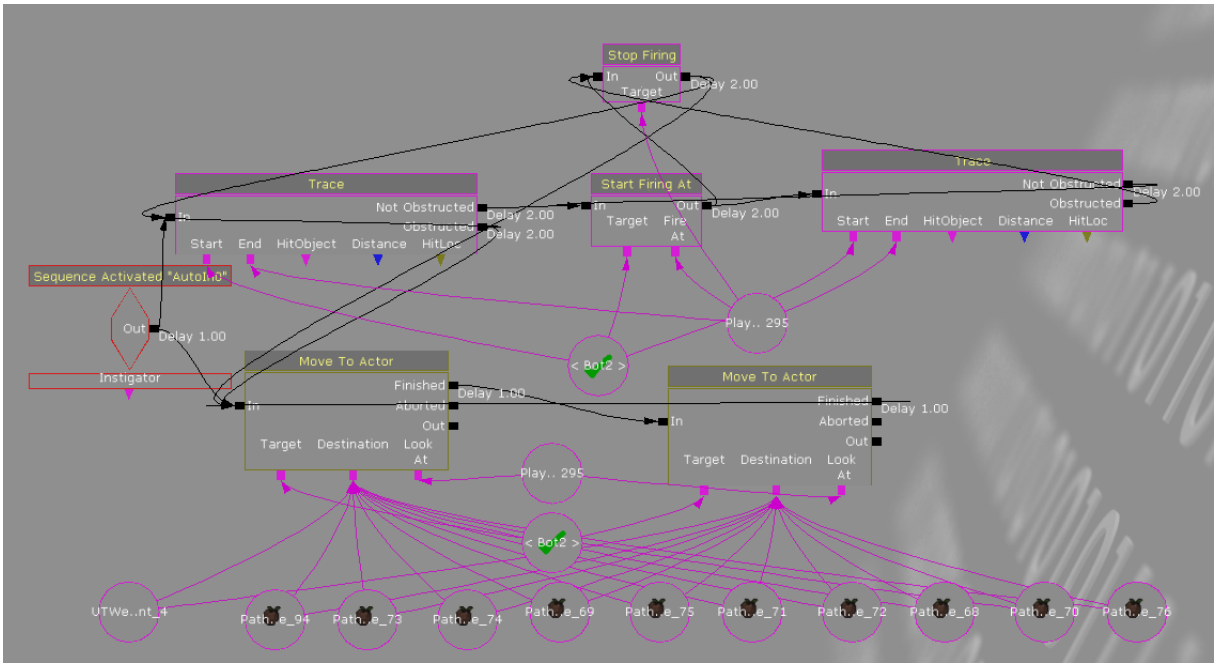
Obr. 23 Úvodní menu

Druhý klip byl vytvořen pro zobrazení informací o aplikaci. Na něm je použito pouze jedno tlačítko sloužící pro návrat do úvodního menu. Poslední klip slouží pro závěrečné menu a obsahuje tlačítko pro ukončení aplikace a tlačítko pro návrat do úvodního menu. Tyto klipy se importují do UDK, stejně jako obrázky pro textury, přes Content Browser.

4.5 Kismet

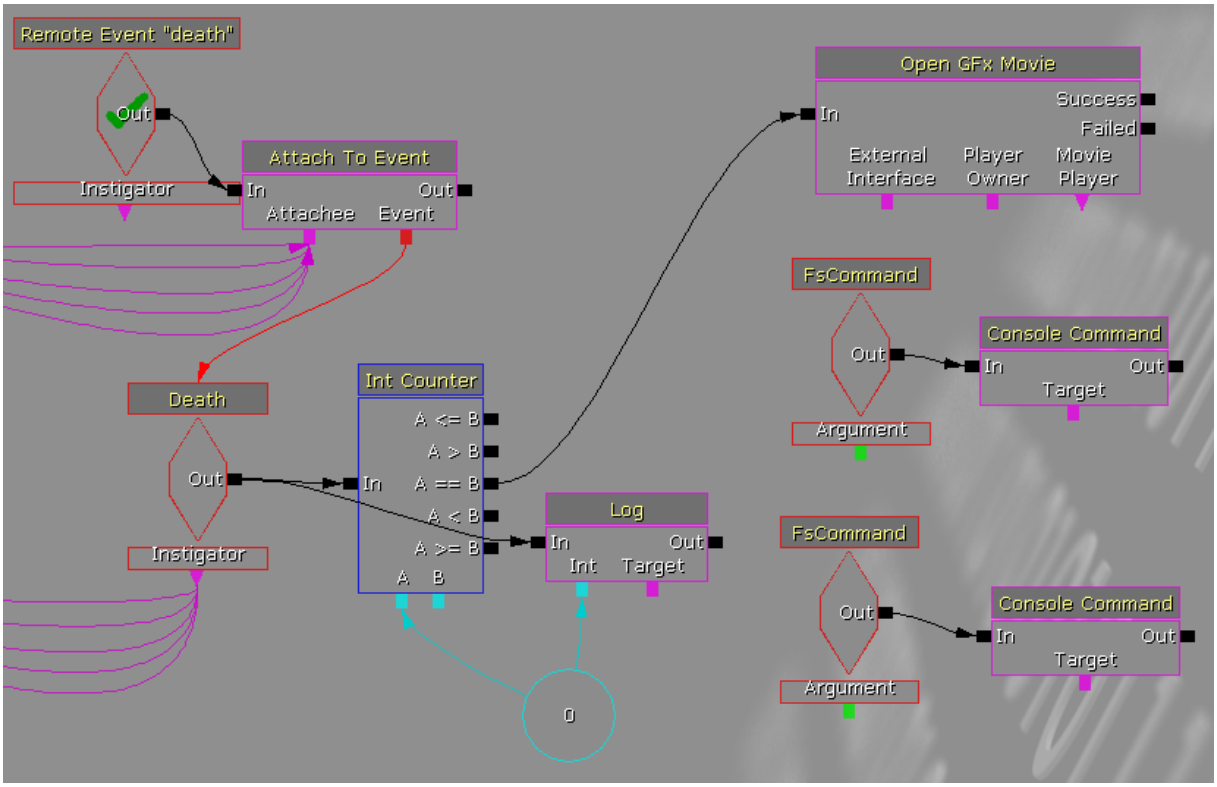
Posledním krokem k dokončení aplikace bylo vytvoření postav umělé inteligence a nahrání vytvořeného uživatelského rozhraní. Na těchto dvou věcech budou ukázány možnosti skriptování v prostředí Kismet. Základním prvkem celého skriptu je blok Level Loaded, který zastupuje načtení aplikace a na nějž jsou napojeny všechny ostatní bloky. Pro vytvoření postavy umělé inteligence slouží blok Spawn Actor, napojený na Level Loaded a určený proměnnou a místem vytvoření. V této aplikaci je vytvořeno celkem pět postav umělé inteligence na různých místech a u všech je nastaveno pouze jedno vytvoření do scény, aby se neobnovovaly po zničení. Jejich chování je naprogramováno v sekvencích Bot1-Bot5, které se liší pouze vytyčením oblasti pohybu přiřazené postavy.

Tyto sekvence se skládají ze dvou hlavních částí. První část určuje samotné chování postavy. Ta má za úkol hledat postavu uživatele (blok Trace), pokud ji vidí, tak na ni vystřelit (blok Start Firing), poté se znovu rozhlédnout a pokračovat popř. přerušit střelbu (Stop Firing), záleží na tom, zda uživatele stále vidí. Současně s hledáním a střelbou na hráče se musí postava pohybovat (blok Move To, s připojenými navigačními body). Celá tato sekvence je na Obr. 24.



Obr. 24 Kismet skript pro umělou inteligenci

Další sekvence se týká ukončení hry po eliminaci všech pěti postav. K tomu slouží funkce Remote Event. Ta je připojená na všech pět postav umělé inteligence a skládá se ze dvou částí: Activate Remote Event, který jen určuje, kterých věcí se to bude týkat a Remote Event, ke kterému je připojená podmínka a akce po jejím splnění. Tyto bloky jsou spolu spojeny pomocí unikátního názvu. Podmínka je tvořena zjištěním, jestli je postava naživu (blok Pawn->Death), na kterou je připojeno jednoduché počítadlo, vytvořené z bloku Log. Jakmile je v počítadle hodnota pět, což je porovnáváno blokem Compare, spustí se sekvence závěrečného menu. Tato sekvence je na Obr. 25.

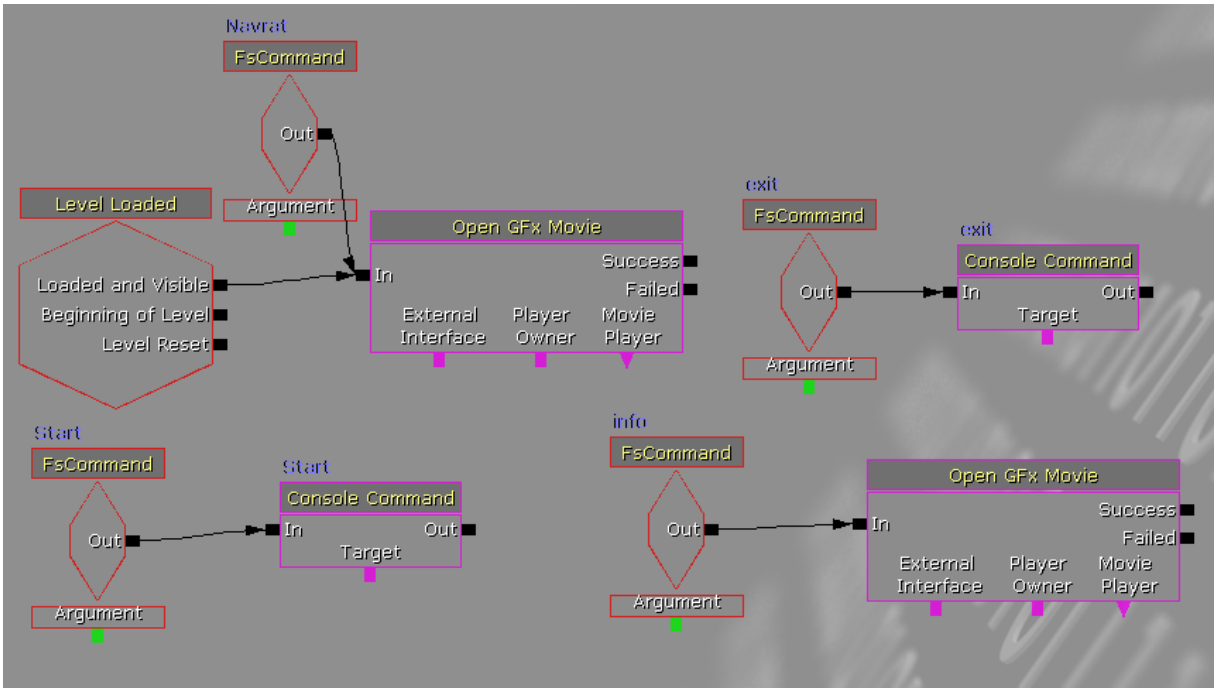


Obr. 25 Kismet skript pro ukončení hry

Poslední část skriptu je tvořena spouštěním a funkčností uživatelského rozhraní. Při spuštění hry je zobrazeno úvodní menu (blok OpenGfXMovie, připojený na Level Loaded). Funkce tlačítek v tomto klipu je řízena blokem Fscommand, který se načte ke klipu a v jeho názvu se určí tlačítko, ke kterému se vztahuje. To je provedeno vepsáním hodnoty proměnné, která byla určena při tvorbě klipu. Poté se k tomuto bloku připojí blok Console Command. Příkazy přiřazené tomuto bloku budou při jeho aktivaci načítány přímo do ovládací konzole aplikace. V úvodním menu jsou tři tlačítka s následující funkcí:

- **Spustit hru** – open DM-Bakalarsky_Projekt; spustí hru.
- **Info** – k tomuto tlačítku je připojen blok OpenGfXMovie; spustí klip s informacemi o hře.
- **Konec hry** – exit; ukončí aplikaci

V informačním klipu je pouze jedno tlačítko, po jehož stisknutí se znovu zobrazí úvodní menu. Závěrečné menu obsahuje dvě tlačítka: Návrat do menu, po jehož stisknutí se otevře klip s úvodním menu a Konec hry, které je řešeno stejně jako u úvodního menu. Část skriptu s uživatelským rozhraním je na Obr. 26.



Obr. 26 Kismet skript pro uživatelské rozhraní

4.6 Export aplikace

Jak bylo řečeno na konci minulé kapitoly, exportování aplikace probíhá pomocí Unreal FrontEnd. Po jeho spuštění se nastaví, aby byla aplikace vyexportována pro PC, zvolí se mapy, ze kterých se bude výsledná aplikace skládat a nakonec spustí export. Výsledkem je instalační soubor, po jehož spuštění a dokončení instalace je aplikace zcela nezávislá na UDK.

4.7 Ukázky ze scény

Obr. 27 Přízemí budovy A1

Obr. 28 Pohled z ochozu v A1

Obr. 29 Chodba budovy A5

Obr. 30 Přednášková místnost budovy A5

Obr. 31 Postranní místnost vedle A1

5 Závěr

Cílem této práce je seznámení se s technologií pro tvorbu počítačových her a aplikací Unreal Engine 3. Práce je zpracována jako tutoriál a je tu možnost jejího budoucího využití jako základního návodu, jak s touto technologií pracovat. V první kapitole jsou vysvětleny nejdůležitější techniky, kterých Unreal Engine 3 využívá, a které je třeba pochopit pro práci s touto technologií. V druhé kapitole je popsáno vývojové prostředí spolu s některými nástroji a jak s nimi pracovat. Poslední část práce se zabývá vytvořením ukázkové aplikace a předvedením některých technik, popsaných v prvních dvou kapitolách. Zejména jsou ukázány skriptovací možnosti nástroje Unreal Kismet.

Při vypracovávání této práce jsem se seznámil s technologii Unreal Engine 3 a jejím vývojovým prostředím UDK a vzhledem k funkčnosti výsledné aplikace jsou všechny cíle zadání považovány za splněné.

Seznam použité literatury

[1] THOMSEN, Mike. History of the Unreal Engine. In: Video Games, Wikis, Cheats, Walkthroughs, Reviews, News & Videos - IGN [online]. [23 February 2010] [cit. 2013-03-16]. Dostupné z: http://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine

[2] NORTON, James. Gemini Update. In: *Stoked Software Blog* [online]. [26 May 2012] [cit. 2013-03-16]. Dostupné z: http://blog.stokedsoftware.com/blog/2012/05/26/gemini-update

[3] WRIGHT, Daniel. Lightmass Static Global Illumination. In: *UDN - Unreal Development Network* [online]. © 2001-2012 [cit. 2013-04-01]. Dostupné z: http://udn.epicgames.com/Three/Lightmass.html

[4] AUTOR NEUVEDEN. Audio System. In: *UDN - Unreal Development Network* [online]. © 2001-2012 [cit. 2013-04-01]. Dostupné z: http://udn.epicgames.com/Three/AudioSystem.html#Overview

[5] *PhysX* [online], poslední aktualizace 12. května 2013 14:18 [cit. 14. 5. 2013], Wikipedie. Dostupné z: http://en.wikipedia.org/wiki/Physx

[6] NVIDIA. PhysX Features (2. X). In: *NVIDIA Developer Zone* [online]. © 2013 [cit. 2013-04-01]. Dostupné z: https://developer.nvidia.com/physx-features-2x

[7] AUTOR NEUVEDEN. Using KActors. In: *UDN - Unreal Development Network* [online]. © 2001-2012 [cit. 2013-04-05]. Dostupné z: http://udn.epicgames.com/Three/UsingKActors.html

[8] AUTOR NEUVEDEN. Introduction to FaceFX. In: *UDN - Unreal Development Network* [online]. © 2001-2012 [cit. 2013-04-12]. Dostupné z: http://udn.epicgames.com/Three/IntroductionToFaceFX.html

[9] *Unrealscript* [online], poslední aktualizace 22. dubna 2013 4:33 [cit. 14. 5. 2013], Wikipedie. Dostupné z: http://en.wikipedia.org/wiki/UnrealScript

[10] THORNE, Alan. *UDK Game Development*. Boston: Cengage Learning, 2012. ISBN 1-4354-6018-9.

Seznam příloh

Obsah DVD

- Elektronická verze této práce
- Instalační soubor editoru UDK-12-07
- Instalační soubor ukázkové aplikace
- Návod pro zprovoznění editoru a aplikace