Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Prezentace Centra počítačové grafiky a vizualizace ve virtuální realitě

Plzeň 2019 Jiří Noháč

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů. Veškerý použitý software, který byl za účelem této práce použit byl získán a použit legálním způsobem.

V Plzni dne 2. května 2019

Jiří Noháč

Abstract

The aim of this thesis is to explore a field of gaming development for available development environments. Uncover their advantages, disadvantages and finally choose only one of them for development of application in virtual reality, which will include 3D model of laboratory and results of work in the field of the graphic science at the department of informatics. In these days development of graphical environments is really expensive and every tool that can simplify work on a project is extremely valuable. Many talented developers created countless graphical development environments over decades, which they used themselves or distributed to different studios or to the general public. Based upon this knowledge, several game development environments will be chosen, known and less known, paid and freely accessible and evaluate their strengths and weaknesses not just in casae of virtual reality, but also in field of extensibility, usability, functionality and price polices towards the studio but also to the final customer. After carefully selected development environment is the goal to convert narrow spectrum of projects developed at the department of informatics into virtual reality. The main purpose of the thesis is to demonstrate the essence of a problem and it's possible solution in the most understandable way to the general public, future practitioners or scientists. The main aquisition of the thesis is in demonstration of projects and researches, which are currently developed in our faculty. Thanks to a virtual reality, participants will be able to actually try and engage themselves to these projects. Next aspect of this thesis is to extend insight and understanding of virtual reality in constantly growing world, where finding the right development tool for virtual reality is very desirable, because trend tends to go this way and the amount of aplications for virtual reality is constantly increasing.

Abstrakt

Cílem této práce je prozkoumat dostupná herní vývojová prostředí v oblasti herního vývoje. Odhalit jejich přednosti, nedostatky a nakonec vybrat pouze jedno prostředí pro vývoj aplikace na virtuální realitu, která bude zahrnovat vymodelování prostoru laboratoře a výsledků prací v oboru grafiky na katedře informatiky. V dnešní době je vývoj grafických prostředí velmi nákladný a každý nástroj, který může zjednodušit práci na projektu, se nesmírně cenní. Mnoho talentovaných vývojářů během desítek let vypracovalo nespočet grafických vývojových prostředí, které sami využívají nebo je distribuují jiným studiím či široké veřejnosti. Na základě tohoto poznatku bylo cílem bakalářské práce vybrat vhodné herní vývojové prostředí, známé i méně známé, placené i volně přístupné a vyhodnotit výhody i nevýhody nejen okolo virtuální reality, ale i co se týče rozšířitelnosti, využití, funkcionality a cenové politiky vůči studiu, ale i finálnímu zákazníkovi. Po pečlivém výběru vývojového prostředí je cílem práce převést úzké spektrum projektů vyvíjené na katedře informatiky do virtuální reality. Hlavním účelem je představit veřejnosti nebo jinému okruhu aplikantů či vědců podstatu problému a jeho možné řešení v co nejsrozumitelnější formě. Hlavní přínos práce spočívá v prezentování projektů a výzkumů, které se u nás na fakultě momentálně vyvíjejí. Díky virtuální realitě si tyto projekty budou moci účastnící sami vyzkoušet a osahat na vlastní kůži. Dalším aspektem práce je rozšíření náhledu a porozumění ve stále se rozrůstajícím světě virtuální reality, ve kterém by bylo žádoucí najít ten správný vývojový nástroj, jelikož trend se ubírá tímto směrem a aplikací na virtuální realitu neustále přibývá.

Obsah

1 Nalezení vhodného vývojového prostředí

Zadáním bakalářské práce je se seznámit se dostupnými vývojářskými prostředky pro vývoj softwaru, zaměřeného na počítačové hry či jiné projekty vyžadující práci s grafickými prvky. Toto vývojářské prostředí se v anglickém jazyce nazývá „engine“, jinými slovy motor, který slouží jako základní kámen pro ostatní komponenty k dosažení požadovaného výsledku. Tyto enginy jsou výsledkem mnoha desítek let pokroku v oblasti grafiky, fyziky a jiných podstatných oblastí na poli vědy, které vývojáři postupně začali vytvářet již v minulém století. Na základě zjednodušení vývoje jak pro programátory, tak pro herní designery, se začala vytvářet sofistikovaná vývojová prostředí (enginy), která měla za hlavní cíl zrychlit vývoj herního softwaru a také zlepšit porozumění v rámci celého projektu mezi různými vývojovými týmy pro lepší komunikaci a práci na produktu. Herní enginy se s časem stávaly dokonalejšími a mnohem důležitějšími než samotný produkt, který na nich byl vytvářen. Nicméně v posledních dvou desetiletích se vývoj takovéhoto softwaru stal natolik náročným, že se mnoho vývojářů rozhodlo nadále engine nevyvíjet a raději zakoupit licenci od jiného studia. Současně každé velké studio jako např. Electronic Arts, Ubisoft, Blizzard-Activision, Konami atd. mají jeden nebo více herních enginů pro různé typy projektů, na kterých pracují. Každý z těchto enginů byl při jeho tvorbě zaměřen na určité vlastnosti a schopnosti potřebné k vývoji konkrétního produktu. Během let tak vznikly různé druhy enginů, které se neustále rozvíjejí a vznikají novější a lepší verze.

Pro účely mojí práce se zaměřím pouze na úzké spektrum enginů, které mají tu vlastnost, že lze na jejich jádru napsat program pro virtuální realitu. Virtuální realita (dále jen VR) se v posledních letech velmi rozšířila mezi studia po celém světě. V minulosti se již objevilo několik iterací VR, nicméně žádná z nich se mezi uživateli nerozšířila natolik, aby se z VR stala nová platforma. Hlavním důvodem byl nedostatečný výkon hardwaru, který byl v produktech použit, ale i s nimi souběžně nabízené produkty. Díky tomu o VR lidé ztratily zájem a tato technologie se na pár let vytratila z obzoru pozornosti. Vše se ale změnilo, když v roce 2011 předvedl Palmer Lucky svůj prototyp VR. Jeho vizí bylo vytvořit co možná nejkvalitnější displej společně s gyroskopickými čidly a zároveň se snažil, aby tato jednotka byla dostupná širokému spektru uživatelů. V roce 2012 požádal v rámci kick-starteru o

přispění na výrobu tohoto projektu, a tak vznikl Oculus Rift. Během let se vydalo mnoho postupných vývojových verzí, ale až teprve v roce 2016 se objevila první oficiální verze pro uživatele. Tento rok by se dal označit jako velmi významný, jelikož v témže roce byl také vydán od firmy HTC společně se společností Valve virtuální kit HTC Vive. Z počátku vývoje Oculus Rift byly verze „developmentu kitu“ určeny pro používání na stacionárním místě společně s ovladačem, ale HTC Vive již od začátku disponoval možností pohybovat se se soupravou v prostoru, díky přibaleným speciálním ovladačům, ale hlavně pomocí systému snímání pozice brýlí. To mělo za následek rozdělení v prvopočátcích vývoje softwaru na tyto konkretní platformy. Jinými slovy na Oculus Rift byly orientovány produkty spíše pro pocit „vsedě“ nebo „vestoje“, oproti tomu HTC Vive produkty vyžadovaly mírný pohyb v prostoru. Během dvou dalších let přibyli do VR další menší hráči, kteří přidávali různá další vylepšení, jako je například vyšší rozlišení displeje, lepší obnovovací frekvence, větší zobrazený úhel pohledu, nižší váhu soupravy a další vylepšení. V dalším postupu práce se nyní budu pouze soustřeďovat na ty herní enginy, které podporují virtuální soupravu HTC Vive nebo Oculus Rift. Detailnější popis analýzy lze nalézt v kapitole 4.1.

K dnešnímu datu lze na trhu nalézt mnoho enginů. Každý z těchto enginů má své klady a zápory, různé druhy podpory externích zařízení na konkrétním enginu a v neposlední řadě jaké druhy licencí jsou nutné pro jejich používání nebo pro vydání konkrétního produktu. Některé enginy mají také rozdílné vydavatelské politiky vzhledem k vydávání softwaru na konkrétním enginu. S ohledem k zadání, musím nadále připravit takové enginy, které podporují virtuální soupravu HTC Vive, Oculus Rift nebo lze nějakým způsobem přidat do enginu podporu vývoje pro tyto produkty. Toto omezení mi velmi zjednoduší mou práci, jelikož k minulému roku stále není mnoho enginů, které podporují VR. V přiložené tabulce uvádím vybranou skupinu enginů, které podporují VR a vyberu tři z nich, které budou nejvhodnější pro další analýzu k finálnímu výběru enginu pro další postup v bakalářské práci.

Název enginu	Platformy
CryEngine	Windows, OS X, Linux, PlayStation 3 a 4, Wii U, Xbox 360 a One, iOS, Android
Unreal Engine 4	Windows, OS X, Linux, PlayStation 3 a 4, Wii U, Xbox 360 a One, iOS, Android
Unity	Windows, OS X, Linux, PlayStation 3,4 a Vita , Wii U, 3DS, Xbox 360 a One, iOS, Android, Windows Phone, BlackBerry 10, Tizen, Unity Web Player, WebGL, Android TV, Samsung Smart TV
Lumberyard	Windows, PlayStation 4, Xbox One
Skyline	Windows, iOS, Android, Windows Phone
Cobra Engine	Windows, OS X,PlayStation 4, Xbox One
Creation Engine	Windows, OS X, PlayStation 3 a 4, Xbox 360 a One, Nintendo Switch
RE Engine	Windows, PlayStation 4, Xbox One

Tabulka 1.1: Tabulka enginů a jejich podporovaných platforem

Název enginu	Prog. jazyk	Skriptování	Licence
CryEngine	C++	Lua, C#	Proprietární software
Unreal Engine 4	C++	GLSL, HLSL, Cg, UnrealScript, C++, Blueprints	Proprietární software
Unity	C++, C#	C#, Cg, HLSL	Proprietární software
Lumberyard	C++	Lua	Proprietární software
Skyline	C++	C#	Proprietární software
Cobra Engine	...	...	Privátní software
Creation Engine	C++	...	Privátní software
RE Engine	...	...	Privátní software

Tabulka 1.2: Tabulka enginů a jejich vlastností

Po podrobném prostudování dostupných herních enginů k roku 2018 jsem dospěl k závěru, že se momentálně nachází na trhu osm herních enginů, které splňují požadavek na vývoj softwaru pro VR. V tabulce 1.1 a 1.2 lze vidět seznam těchto enginů. Tabulka dále obsahuje v jakém jazyce byl engine napsán, jaký jazyk je podporovaný pro psaní skriptů, v neposlední řadě je dále uvedeno pro jaké další platformy je engine podporovaný. Poslední data symbolizují jaký druh licence je nutný k jeho legitimního použití. Vzhledem k tomu, že v rámci bakalářské práce je nutné použít legální software, musím z výběru odstranit enginy: RE Engine, Creation Engine a Cobra Engine. Tyto enginy jsou pod ochrannou známkou firmy, která je momentálně vlastní a nemůže ji použít nikdo jiný, než firma samotná, nebo podpůrná studia. V konkrétním případě RE Engine vlastní Capcom, Creation Engine vlastní Bethesda Softworks a Cobra Engine vlastní Frontier Developments. Mezi další důležitá kritéria patří dostupná podpora ze strany vydavatele včetně fáze vývoje v konkrétním stavu enginu a kompatibilita mezi různými druhy platforem, na které lze konkrétní projekt z enginu přenést. Ná základě těchto požadavků jsem vyřadil engine Skyline, který se v momentální době stále nachází ve stavu průběžného vývoje, kde se na dalších platformách pracuje a pak engine Lumberyard od Amazonu, jenž obsahuje velmi malou podporu platforem k exportu projektu. Z výběru enginů mi tedy už jen zbyly: Unity engine od Unity Technologies, Unreal Engine 4 od Epic Games a CryEngine od Cryteku.V následujících kapitolách provedu rozbor historie každého z těchto tří enginů a následně zanalyzuji samotné prostředí, uživatelskou náročnost na ovládání a typy licencí nutných pro jeho používání, případně pro samotné uvedení projektu na trh. V kapitole 4 se dále zaměřím na analýzu výběru soupravy mezi Oculus Rift a HTC Vive a následující kapitola 4.2 bude věnována dané soupravě. [1] [2] [4] [9] [11] [12]

2 Vývojářská prostředí a jejich historie

V kapitole 1 jsem na základě kritérií vybral tři vhodné enginy k další analýze. V této sekci se budu zaobírat historií každého z enginů a přehledem verzí, které se objevily během let na trhu.

2.1 Unreal Engine

V květnu roku 1998 byl od společnosti Epic Games uveden jejich první herní engine, který byl pojmenován po stejnojmenné hře „Unreal“. Jádro Unreal enginu bylo vytvořeno na architektuře jazyka C++ a mezi první podporované platformy patřily Windows, Linux a Mac OS. Prvotními vlastnostmi enginu bylo vykreslování objektů (scén), detekce kolizí, umělá inteligence, viditelnost, správa síťového protokolu (známo také pod názvem networking) i kontrola nad souborovým systémem. V průběhu jednoho roku pak studio stále vylepšovalo stávající verzi enginu. Z důvodu rapidního vývoje renderovacích technologií konce 20. století byla první verze enginu navržena pro šest různých verzí. Stavěn byl původně pro technologii 3Dfx, S3 Metal, PowerVR SGL a později i Direct3D 5 až 7 společně s OpenGL, bohužel OpenGL nebyl dále rozvíjen kvůli nedostatečné úrovni potřebného softwaru mezi hardwarem a operačním systémem pro komunikaci s grafickou kartou (v anglickém jazyce „driver“). Poslední nouzová renderovací technologie byla schopná fungovat s jakýmkoliv hardwarem (to znamená i v případě, že počítač neměl nainstalovaný žádný grafický akcelerátor).
Verze 2.0 byla poprvé představena pro hru „America’s Army“. Hlavním cílem nové verze bylo předělat a upravit engine tak, aby byl použitelný i pro jiné typy her než FPS (First Person Shooter) žánry. Mezi další novinku patří podpora DirectX 8, 9 a OpenGL. Mezi zásadní nové technologie patří: Static meshes, Animated skeletal meshes, Vertex meshes, Height-Mapped Terrain atd. Posledním velkým krokem je implementace Karma-physics SDK (Software Development Kit) na propočítání fyziky pohybujících se objektů (postav, strojů), přesnosti kolizí a odrazů drobných částeček.
Třetí verze enginu byla vyvinuta v roce 2006 a byla natolik kvalitní, že ji hojně začala využívat studia po celém světě. Unreal engine byl odděleně upraven každým studiem a na základě jejich interních požadavků pro

daný produkt byly připraveny takové hry, jako: Medal of honor: Airborne, Batman: Arkham Asylum nebo kupříkladu trilogii Mass Effect. Díky podpoře DirectX 9 až 11 a nové verze OpenGL byla rozšířena podpora nových platforem. Mezi nové polatformy patří Xbox 360, PlayStation 3, IOS, OS X, Android, PlayStation Vita a Wii. Později roku 2011 byl engine upraven pro Adobe Flash Player. Zásluhou většího rozšíření mezi platformami byl kromě her využit i v jiných oblastech ať už v umění nebo v simulátorech.
Poslední a nynější verze 4.0 byla představena již v roce 2012. Mezi hlavní novinky patří implementace nového debuggeru „Kismet“ a grafického zápisu kódu „Blueprint“. V posledních verzích byla nově přidána podpora virtuální reality a všech základních virtuálních headsetů na trhu. [2] [3]

Unreal Development Kit (UDK)

UDK je volně šiřitelné vývojové prostředí pro tvorbu map, modů a her v Unreal Enginu 3, který byl pro veřejnost poprvé uvolněn v listopadu 2009 s regulérními měsíčními aktualizacemi, opravami a novými funkcemi. Pro nekomerční využití je zdarma, avšak dle licence se pro komerční využití musel uhradit Epic Games prvotní poplatek 99 USD a pak následně 25 % z celkového zisku nad výdělek 50 000 USD. [2] [3]

2.2 CryEngine

Prvni verze CryEnginu byla vytvořena v roce 1999 německým studiem Crytek. První představení onoho enginu byla na výstavě ECTS roku 2000, kde ohromila publikum svou technologickou demoverzí. Vývojáři nadále pokračovali s vydáváním nových ukázkových projektů, tehdy nazývané X-Isle (později známé jako Far Cry). 2. května 2002 Crytek představil veřejnosti svoji verzi herního enginu pod jménem „CryEngine“.
První verze pro komerční využití se datuje k roku 2004, kdy Crytek vydal jejich první hru na tomto enginu, Far Cry. Původně byl CryEngine jen visuální prezentací pro grafické čipy firmy Nvidia, ale nakonec se z dema vyklubala plnohodnotná hra. Díky úspěchu Far Cry si engine licencovala mnohá studia jako například NCsoft. Předností enginu byla podpora pixel a vertex shaderů verze 3.0. Později však práva na hru Far Cry a CryEngine (Ubisoft svojí verzi CryEnginu dále nazývá DUNIA engine) získala firma Ubisoft, ale vývoj CryEnginu mohl stále pokračovat pod vedením Cryteku.
Další verze CryEnginu byla prvně použita u hry Crysis a následně Crysis: Warhead. Nová verze nabízela podporu nových platforem, jako jsou PlayStation 3 a Xbox 360. Licence enginu již získala další studia jako na-

příklad Avatar Reality, Entropia Universe, XLGames, Reloaded Studios, a Games Academy GmbH.
Důležitým milníkem Cryteku byla však verze 3.0, kterou předvedli na Game Developers konferenci v roce 2009. Představená verze již plně podporovala DirectX 11 a přidala mezi podporované platformy Nintendo Wii. Hlavním úspěchem bylo vydání verze volně k použití pro nekomerční vývoj her. Tato verze byla vydána 17. srpna 2011 pod jménem CryEngine 3 SDK. Účelem bylo přiblížit vývoj počítačových her širšímu publiku a použití ve školách jako naučný software pro studenty. Posledním velkým titulem vydaným studiem byl třetí díl Crysis, který na upraveném CryEnginu 3 vydali v roce 2013.
Od konce roku 2013 se Crytek rozhodl, že jejich další verze CryEnginu nebude mít žádné číslo. Důvodem pro toto rozhodnutí bylo, že tento nový engine nemá téměř nic společného s předchozími verzemi CryEnginu. Také to znamenalo změnu dosavadní ochranné známky „CryENGINE“ na „CRYENGINE“. Tato verze CryEnginu nově podporuje Linux a konzole nové generace, jako jsou PlayStation 4 s Xbox One.
Stávající verze, číslo pět, byla oznámena na konferenci GDC 2016. Přináší podporu DirectX 12, vývoje pro VR (HTC Vive, Oculus Rift, Playstation VR a OSVR) a skriptování v jazyce C#. [4] [5] [6] [7]

2.3 Unity engine

Unity je multiplatformní herní systém vyvinutý společností Unity Technologies, který se používá především pro vývoj trojrozměrných(dvourozměrných) videoher a simulací pro počítače, konzole či mobilní zařízení. Engine byl poprvé oznámen pouze pro OS X na konferenci Apple Developers Worldwide v roce 2005 a od té doby byl rozšířen na cílových 27 platforem. Do dnešního dne již bylo vypuštěno šest verzí.
Pro Apple Design Awards na veletrhu Worldwide Developer Conference 2006, Apple, Inc. jmenovala Unity na druhé místo v kategorii grafika pro Mac OS X. Unity Technologies říká, že je to poprvé, co byl někdy navržen nástroj pro tvorbu hry. Průzkum časopisu Game Developer v květnu 2012 označil Unity za špičkový herní engine pro mobilní platformy. V červenci 2014 získala společnost Unity ocenění „Nejlepší engine“ na výročním britském ročníku Develop Industry Excellence Awards.
Po vydání Unity 5 společnost Unity Technologies kritizovala velký objem rychle vygenerovaných her, zveřejněných na distribuční platformě Steam nezkušenými vývojáři. Generální ředitel John Riccitiello v rozhovoru prohlásil,

že se domnívá, že to je vedlejší účinek úspěchu Unity ve vývoji her. V prosinci 2016 společnost Unity Technologies oznámila, že změní systém číslování verzí pro Unity od identifikátorů založených na sekvenci po rok vydání, aby přizpůsobil verzi s jejich častějším uvolňováním přídavků a oprav.
Unity podporuje vytváření projektů na 27 různých platformách, kupříkladu iOS, Android, Windows, Mac, PlayStation 4, Xbox One atd. Unity dříve podporovala 7 dalších platforem včetně svého vlastního nástroje Unity Web Player. Unity Web Player bylo rozšíření prohlížeče, které bylo podporováno pouze v systémech Windows a OS X. Jelikož byl zastaralý, došlo k jeho nahrazení ve prospěch novějšího WebGL. Unity také společně s Nintendem zavedli speciálni verzi Unity SDK pro herní platformu Nintendo Wii, kterou Nintendo dodává s každou vývojářskou licencí Wii U. [8] [9] [10]

3 Finální srovnání všech vybraných enginů

Ze všech postupně zmíněných kritérií pro výběr vhodného enginu se zaměřím na vyhodnocení všech kladů a záporů. Dále zmíním přístupnost a efektivitu vývoje her a aplikací, kde jiné herní enginy mohou mít výhodu nad ostatními. Společně s předešlými požadavky se zaměřím na uživatelskou náročnost použití aplikace, obtížností importovat různé druhy aktiv (modely, textury, data atd.) a také na to, jaké licence jsou nutné pro používání a produkování projektů na trh. Na závěr postupně odeberu některé z nich a vyberu jeden vhodný engine pro další postup při tvorbě bakalářské práce.

3.1 Unreal Engine 4

Unreal Engine 4 jsem testoval na verzi 4.17.2 s licencí pro osobní použití. Jedna z hlavních předností tohoto enginu se ukrývá v jednoduchosti používání jeho nativního editoru při vkládání prvků do scény a následné úpravy jejich vlastností, změny pozice, ale i nastavení nasvícení scény atd. Další z přínosů je rozhodně velmi kvalitní editor textur a jejich vlastností, který se nazývá „Material editor“. Tato vnitřní aplikace slouží k nastavení vlastností textur, kupříkladu jak se má daná textura chovat, když na ní svítí světlo, nebo jak se mají různé spekulární mapy chovat na konkrétním objektu. Ale asi nejzajímavější je, že zde existují nastavení pro změnu modularity, která mají za následek možnost vytvoření náhodně poskládaných vrstev textury společně s jejich nastavením a tím vytvoření unikátního setu vlastností pro daný model. Tím se dá zaručit unikátnost vzhledu daného modelu, pokud vytváříme modely, které složí pro strukturu stavby nebo jiných méně komplexních systémů.
Kromě samotného editoru pro umístění objektů, tak další zajímavou vlastností unreal enginu jsou takzvané „Blueprints“, tyto tzv. šablony si lze představit jako grafický programovatelný systém aplikace, který vytváří konkrétní schopnosti a vlastnosti příslušné scény, objektu, modelu, textury atd. Tyto „Blueprints“ nejsou nic jiného než kód aplikace, který se vytváří pomocý grafických segmentů, které skládáme do sebe jako stavebnice z LEGA. Tyto segmenty mezi sebou mohou komunikovat, přistupovat navzájem do svých vnitřních systému a měnit je v reálném čase. Unreal Engine tento

systém ještě zjednodušil tím, že psaní kódu nahradil obrazovým spojování uzlů, které symbolizují určitý objekt, vlastnost, funkci a ty následně skládáme dohromady. Podle toho, jak jsou entity propojeny, nakonec symbolizují sekvenci akcí, které se mají provést. Mnoho Blueprintů lze získat přímo z „Starter Packu“, což je přibalený startovní balíček od Epic Games přímo v Unreal Enginu. V tomto balíčku můžeme nalézt kupříkladu systém pro jednoduchou tvorbu hry z pohledu první osoby, pohledu třetí osoby atd. Další Blueprinty lze zakoupit na „Marketplace“, kterou připravilo Epic Games jako platformu, kde lidé přidávají svoje práce k dalšímu použití za poplatek pro třetí stranu. Ty si lze pak snadno importovat do Unreal Enginu pro použití v projektu nebo nadále modifikovat. S těmito všemi komponentami ale Epic Games ještě nekončí a nabízí také mnoho tutoriálů a materiálů o vývoji jak pro začátečníky, tak i pokročilé programátory společně s fórem, na kterém se řeší jak triviální, tak i složité otázky na různá témata. Díky této zákaznické podpoře a mnoha materiálů k prostudování se řadí i v mnoha komunitách jako velmi vhodný herní engine této doby pro malé a střední firmy.
Co se týče základních vlastností uvnitř samotného editoru, každý objekt má nástroje tam, kde by je člověk očekával. Rozšířené nastavení a pokročilé ovládání jsou jsou povětšinou skryta. Pravděpodobným důvodem může být opatrnost ze strany vývojářů, aby nezkušení programátoři nezasahovali do nastavení, kterému nemusí rozumět. S tím souvisí i jejich popisky, kde u všech nastavení je přibližný popis co konkrétní nastavení provede s komponentou, ale uvítal bych v některých případech detailnější popis či obrázek, dokonce si myslím, že pro nalezení konkrétního nastavení musí uživatel dlouze hledat mezi všemi nabízenými možnostmi to co chce změnit. Z celkového hlediska lze usoudit, že Unreal Engine obsahuje mnoho prvků a technologií, jak přistupovat ke scéně, ale aby je změnil, musí si v některých případech přečíst značnou část manuálu k ovládnutí konkrétních komponent.
K dalším jistým problémům také dochází při optimalizací veškerého kódu, modelů, textur atd. Jelikož lze importovat do vybraného projektu mnoho druhů formátů pro modely i textury, kde jejich rozdíly mohou zapříčinit nekompatibilitu mezi některými z nich. Jednou z chyb, které jsem si všiml je, že Unreal vcelku toleruje větší množství chyb uvnitř importovaných aktiv. Vzhledem k této situaci často dochází, že programátor přehlédne varování nebo neprovede u konkrétního druhu aktiva operaci, aby engine správně načetl konkrétní věc. Tato benevolenci vzhledem k načítání aktiv může mít dosti fatální důsledky, proto jsem se při každém importu raději podíval do „Message Log“, kde se zobrazují všechna dosavadní doporučení a varování enginu. [2]

3.2 Unity engine

Unity Engine jsem testoval ve verzi Unity 2017.2.0 s licencí pro osobní použití. Tento engine patří již k velmi populárnímu vývojářskému softwaru mezi skupinou profesionálů tak i nadšenců, či začínajících vývojářů. Jednoznačnou výhodou tohoto enginu je jeho jednoduchost pracovat se základními objekty ve hře. Základní grafické prostředí je přehledné a snadno se do něj uživatel dostane po pár hodinách, ale rozhodně nevetší plus se týká cenové politiky a kvalitní optimalizace.
Momentálně se Unity engine nachází ve verzi 2017.2.2. Unity podporuje DirectX, Vulkan a OpenGL ES pro mobilní zařízení. Unity nyní nabízí ze svého sortimentu čtyři možné cenové politiky. Personal, Plus, Pro a Enterprise. Každý ze stupňů představuje jistou verzi zákaznické podpory a možnosti zasahovat do základního kódu programu. Personal představuje jen základní možnosti bez žádné bonusové podpory. Pro verze má oproti Plus verzi tu výhodu, že u Pro verze lze zasahovat do samotného jádra enginu. Poslední licence, Enterprise, která nad Pro verzí získává nadstandardní „cloudové“ služby a větší možnosti při managementu projektu. Na základě těchto finančních rozložení může využívat unity engine takřka každý za účelem výuky, či s cílem vytvoření profesionální aplikace.
Jednou z hlavních předností, kterou tento engine disponuje jsou nízké požadavky na hardware, jak k práci nad samotným projektem, tak i po samotném exportu na vybranou platformu (vzhledem k implementovaným technologiím). Tímto důkazem může být mnoho aplikací vytvořených, jak pro stolní počítače (7 Days to Die, Rust, ...), tak i na mobilní zařízení (Bad Piggies, Deus Ex: The Fall, ...), které ve většině případů lze spustit i na starších strojích. Díky nízkým nárokům enginu se lze při vývoji více soustředit na samotný vývoj aplikace aniž bychom museli zkoumat, jaký druh algoritmu použít na konkrétní řešení problém k jeho urychlení. Unity engine se proto hojně využívá jako výukový nástroj v mnoha školách od základních až po univerzitní. Z mé zkušenosti mohu poznamenat, že s importy aktiv jsem neměl žádně problémy, engine umí různé druhy modelů, textur atd. Velmi cenným přídavkem by byl systém skriptování s uživatelským rozhraním, který disponuje výše zmíněný Unreal Engine. [9]

3.3 CryEngine

CryEngine jsem testoval ve verzi 5.1.1, která byla s licencí pro osobní užití a testování. Pro vydání produktu na trh je nutné Cryteku předvést demo projektu a následně od společnosti získat licenci na konkrétní aplikaci vy-

cházející z dema.
Pokud bychom se podívali do historie použití a vývoje CryEnginu, zjistili bychom, že hlavní devízou celého vývojového prostředí je s velkou pravděpodobností jeho důraz na využití nejmodernějších technologií směrem ke grafické stránce věci, kdy už první Far Cry měl v té době ohromující vizuální stránku a o několik let později v roce 2007 Crysis, kde bylo poprvé ve velké míře použito DirectX 10, což bylo v daném roce nevídané. Bohužel to sebou neslo i jisté potíže. V tomto případě je to náročnost jak při vývoji, tak i ve finálním produktu. CryEngine momentálně patří mezi hardwarově a vývojově nejnáročnější herní enginy vůbec. V době, kdy se Crysis dostalo do obchodů, neexistoval žádný samostatný grafický čip, na kterém by bylo možné dosáhnout požadované snímkové frekvence. Jedinou možností bylo propojení čipů pro aktivní zapojení více karet najednou - SLI (Nvidia) nebo CrossFire (AMD, dříve Radeon). Přibližně po roce byly na trh uvedeny řady grafických karet, které tímto výkonem disponovaly. Problém náročnosti na hardware přetrvává dodnes, kdy hry, které vycházely na tento engine, byly s velkou pravděpodobností velmi náročné a zákazníci museli snižovat detaily, aby hra pracovala plynule. S tím také souvisí optimalizace. Přestože se studia snažila hru optimalizovat, tak se jim to v mnoha případech nepovedlo a dnes můžeme vidět spoustu her, které jsou na CryEnginu postavené, u kterých optimalizace nebyla doladěná do takové míry, jak bychom si přáli (viz. kupříkladu hra Lichdom: Battlemage). Mezi další problém, co tuto situaci podpořil, je problém optimalizace na konzolích. Od verze enginu 3.0 byla přidána podpora na Xbox a PlayStation. Vývojáři tak už mohli připravovat multiplatformní tituly pro své hry, ale mělo to jeden háček, CryEngine byl primárně optimalizován pro stolní počítače, a tak přidaná podpora pro konzole výše popsaný problém ještě zhoršila. Díla vytvořená na tomto enginu trpěla propady snímkové frekvence a to až pod hranici hratelnosti. Problém s optimalizací vyřešila až verze 5.0 vydaná v roce 2016, která přidala i podporu VR. Vzhledem k předchozím informacím, by se dalo usuzovat, že CryEngine v mém případě nesplňuje kritéria pro použití v rámci bakalářské práce, ale jsou tu jistá kritéria, která by mohla vznést enginu do mého výběru.
Během předchozího desetiletí(13 let) se spousta lidí naučila s tímto prostředím pracovat. Prostředí a vývojářské mechanizmy se nezměnily, jinými slovy, to co platilo před 10 lety platí i dnes s jistými obměnami. Proto lze na internetu získat mnoho návodů, jak s tímto enginem pracovat. Další výhodou je široká možnost pomoci na oficiálním fóru Cryteku, kde lze dohledat řešení na konkrétní problém. Pro nekomerční použití je licence zcela zdarma, kde každý, kdo se registruje na oficiální stránce Cryteku, si může engine stáhnout a vyzkoušet. Při obchodním využití enginu lze za poplatek při škálovaných

cenách Cryteku získat modely, textury, zvuky, částicové efekty atd., které vytvořili sami autoři Cryteku nebo přispěvatelé komunit. Pro velké firmy je tu konečně Enterprise edice, která nabízí všechna dosavadní zvýhodnění plus získává přistup ke kódu CryEnginu včetně samotné podpory zaměstnanců Cryteku. [7]

3.4 Závěrečné shrnutí a výběr enginu

Po dosavadním vyzkoušení každého z enginů, se velmi těžko definuje finální kandidát pro další postup v práci. Každý z enginů disponuje řadou výhod nad ostatními, ale také i nevýhod, které mohou způsobit při tvorbě projektu značné potíže. Kupříkladu CryEngine je ze všech enginů nejnáročnější, jelikož všechna nastavení, modely, textury, nasvícení atd. zpracovává v reálném čase, což se může zdát jako velké plus, jelikož zobrazuje scénu přesně tak, jak bude ve výsledku vypadat, ale to zapříčinilo na mé sestavě, že snímková frekvence klesala pod 25 snímků za vteřinu při úpravách scény. To mělo za následek špatné ovládání programu a některé operace trvaly neúměrně dlouho dobu. Unreal a Unity pracují na principu omezení nastavení při editaci scény, díky tomu lze s aplikací lépe manipulovat společně s ostatními nastaveními. Je nutné ale také zmínit, že u všech tří enginů lze nastavit detaily, s kterými se pracuje v režimu návrhu. To může ovlivnit výkon aplikace, ale ztrácíme opět kvalitu zobrazení a tím pádem i věrnost finálního výsledku práce. Tímto způsobem jsem se pokoušel u CryEngine nastavit nižší detaily, abych docílil více snímků za sekundu. Ke zlepšení došlo, ale výkon se stále nepřibližoval Unreal či Unity enginu. Pro vývoj aplikace ve virtuální realitě je potřeba dosáhnout snímkové frekvence alespoň 90. Pokud by hodnota klesla pod 90 FPS, kompenzuje virtuální headset pokles snímků tak, že vytvoří poloviční vertikální synchronizaci (V-Sync), která uzamkne snímky na 45 FPS.
Další podstatnou vlastností, s kterou jsem byl velmi spokojen, jsou Blueprinty v Unreal enginu. Jejich princip je velmi jednoduchý a kód se v nich píše velmi snadno. Zpočátku jsem musel prostudovat mnoho materiálů a tutoriálů, jak s tímto systémem správně pracovat, abych dokázal využít jeho předností, ale nakonec svou komplexností a jednoduchostí překonal ruční psaní kódu jak v CryEnginu tak v Unity. Společně s tím bych chtěl zmínit také velmi kvalitní debugger pro Blueprinty a Material Editor. Při spuštění konkrétního projektu lze v Bluperintu snadno vidět, jak se konkrétní procedury aktivují pro lepší pochopení možné chyby v algoritmu nebo opomenutí možností, které mohou nastat při používání a Material Editor zjedno-

dušuje a urychluje vytvoření textur pro modely projektu. Tímto předběžně zařazuji Unreal Engine do konečného porovnání.
U Unity enginu jsem byl ze začátku zahlcen rozmístěním a možnostmi nástrojů, ale po přečtení tutoriálu jsem vždy nalezl, co jsem potřeboval vyzkoušet a ve výsledku jsem si na ovládání zvykl. Bohužel Unity byl poslední, který jsem zkoušel a celkově mi přišel velmi podobný Unreal Enginu jak rozsáhlostí nabízených funkcí, tak jejich uspořádáním. Hlavním trumfem, který engine nabízí, byla jeho enormní podpora zařízení, na které bylo možné výsledný produkt exportovat. V tabulce 1.1 lze získat představu o podporovaných zařízeních. Počet platforem, které tento engine podporuje, mě utvrdil v mém výběru jako druhého finalisty v posledním porovnání. Rozhodnutí bylo primárně určeno z důvodu, kdy CryEngine je stále velmi náročný na hardware a má menší podporu zařízení. Důležité je také zmínit narůstající problémy studia Crytek, kdy se v posledních letech firma dostala do finančních potíží, které mohou vést v blízké době k menší průbojnosti podpory samotného enginu. Proto si vybírám tyto favority: Unity a Unreal enginem.
Vítěze finálního duelu bude velmi těžké zvolit, jelikož oba enginy disponují silným počtem podporovaných platforem, kvalitním uživatelským rozhraním, širokou možností importovat různé druhy aktiv atd. Oba jsou hojně využívány jak komunitou, tak samotnými vývojáři po celém světě. Jediné, v čem se od sebe liší je, jakým způsobem se vytváří samotný kód programu. Unreal disponuje silným nástrojem Blueprint, kdežto Unity využívá tradiční systém psaní kódu. Blueprinty jsou velmi intuitivní a snadno se s nimi pracuje, ale nedokáží provádět určité operace, kdy jsem se setkal s případem, kde bylo výhodnější kód napsat ručně pro jisté typy příkladů (načítání/ukládání dat do souboru). Závěr je tedy takový, že k datu zpracovávání bakalářské práce je pro tvorbu VR ideální Unity či Unreal engine a je jen na konkrétním programátorovi, který ze zvolených si sám vybere. Z osobního hlediska mi více vyhovoval Unreal engine a jeho systém Blueprintu, který usnadňuje zápis kódu, pro urychlení procesu implementace a dále velmi příjemného editoru. S tímto poznatkem jsem nakonec vybral jako vítěze Unreal Engine 4 namísto Unity.

4 Rozbor zadání druhé sekce práce

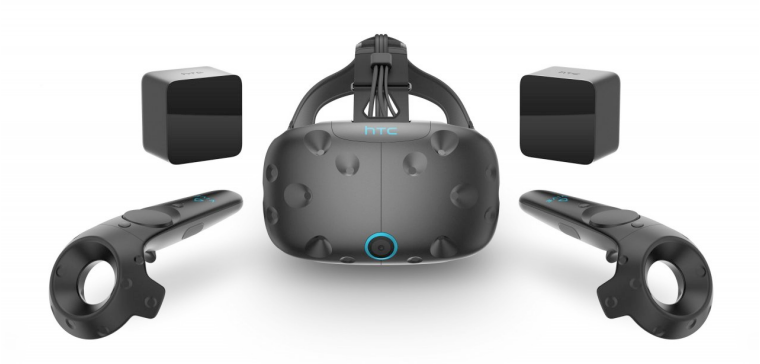
Z minulých kapitol, které se zabývaly herními enginy, jsem si postupně podle stanovených kritérií, jak měřitelných, tak subjektivních preferencí, postupně analyzoval herní enginy, našel jejich výhody, ale také jejich slabiny. Na základě těchto informací jsem postupně zužoval seznam herních enginů, až nakonec zbyl jen jeden jediný engine. Tímto enginem se stal Unreal Engine 4. Po ustanovení již finálního enginu mohu začít s tvorbou a náplní druhé části bakalářské práce.
Zadáním praktické části práce je na základě vybraného enginu vypracovat vizualizaci projektů, kterými se na katedře informatiky, speciálně v Centru počítačové grafiky zaobírají. Výsledky prací oboru grafiky v konkrétní divizi musím následně přenést tak, aby tyto informace byly vhodně reprezentovatelné pomocí VR. Po selekci projektů je nutné sestavit scény, kde si bude možné projekty detailněji prohlédnout, popřípadě mezi nimi procházet. S kompletním návrhem aplikace a všech jejich komponent je součástí bakalářské práce také implementace návrhu do vybraného herního enginu a jeho rozšíření tak, aby bylo možné v budoucnu aplikaci dále rozvíjet.

4.1 Test vybraných souprav pro VR

V rámci seznámení se s dostupnými prostředky pro VR na Katedře Informatiky a Výpočetní techniky jsem otestoval technologie Oculus Rift DK2 a HTC Vive. Jak již bylo řečeno v kapitole 1, Oculus Rift DK2 i HTC Vive jsou speciálně navržené soupravy pro VR. Oculus Rift DK2 pro jeho fungování požaduje video výstup kompatibilní se specifikací HDMI 1.3 popřípadě DisplayPort 1.2 a nebo vyšší. Dále potřebuje jeden USB 3.0 a dva USB 2.0 porty. Minimální nutný hardware pro dostačující funkčnost je 8 GB paměti, procesor Intel i3 nebo AMD Ryzen 3 popřípadě jedny z vyšších řad. U grafické karty je nutné vlastnit alespoň NVIDIA GTX 960/1050 Ti nebo AMD Radeon R9 290/RX 470 popřípadě vyšší modely. V neposlední řadě pak Windows 8.1 nebo Windows 10. U HTC Vive je nutné mít video výstup kompatibilní s HDMI 1.4 nebo DisplayPort 1.2 popřípadě vyšší. Pro přenos informací je nutné mít jeden USB 2.0 port. Minimální povinná specifikace je 4 GB paměti, procesor Intel i5 4590 nebo AMD FX 8350 popřípadě vyšší,

grafická karta NVIDIA GTX 1060 nebo AMD Radeon RX 480 nebo vyšší. Podporované operační systémy jsou pak Windows 7, 8.1 a 10. Z výše uvedených minimálních požadavků lze vydedukovat, že tyto soupravy si žádají nadstandardní hardware k jejich chodu. Pokud vezmeme v úvahu, že je třeba 90 snímků za sekundu k plynulému chodu a uvědomíme si, že rozlišení, které je u obou 2160x1200 pixelů je o 25 % více pixelů, než má dnešní standard full HD (1920x1080 pixelů). Představené hardwarové požadavky jsou tedy opodstatněné. Produkty jsou představeny na obrázcích 4.1 a 4.2. [11] [12] [13] [14]

Obrázek 4.1: Obrázek produktu Oculus Rift DK2.



Obrázek 4.2: Obrázek produktu HTC Vive a příslušenství.

Analýzu obou produktů jsem započal s Oculus Rift DK2. Test proběhl na

dvou získaných programech speciálně napsaných pro tuto platformu. První z nich zahrnoval dům na italské riviéře, kde se uživatel pomocí klávesnice a myši nebo ovladače mohl procházet po domě a jeho zahradě. Druhé demo zahrnovalo simulaci jízdy na horské dráze. Obě dema byla koncipována pro ovládání v sedě i ve stoje, ale vzhledem k tomu, že jsem musel používat klávesnici a myš, rozhodl jsem se pro test v sedě. Obě dema využívala všech senzorů pro pohyb ve VR. Díky tomu jsem si mohl udělat dobrou představu o chování souprav při jejich používání. Pro HTC Vive jsem zkoušel další dema, zcela jiného konceptu. První demo z těchto dvou zahrnovalo různé typy scén, kde bylo představeno mnoho typů ukázek, jak lze s VR pracovat a jaké techniky pro utváření atmosféry lze připravit. Druhé přídavné demo byla hra, kde bylo cílem vzdorovat vlnám útoků nepřátel za pomocí zbraní uvnitř domu, ve kterém se hráč nacházel. Hráč se mohl pohybovat jak pomocí vlastního těla, tak teleportačním mechanismem. Kromě všech těchto způsobů, hráč sbíral kolem sebe zbraně a náboje, které následně použil ke své obraně. Obě tato dema mi posloužila jako základní měřítka toho, co HTC Vive dokáže a jak přesné pohyby a manévry lze s ovladači a tělem konat do té doby, než bude systém zahlcen. Důležité je také zmínit, že HTC Vive vyžaduje pro pohyb v prostoru alespoň 2 x 1,5 m volného místa ke správnému chodu celé soustavy. Pro testování obou sestav jsem zkoušel ovládání jak s ovladačem klasickým, tak s ovladači se senzory pohybu a také jsem testoval různé pozice, v sedě a ve stoje. Vzhledem k tomu, že se na hardwarové úrovni Oculus Rift DK2 a HTC Vive se od sebe neliší, jinými slovy rozlišení displeje, typ displeje ani úhel pohledu vykazují malé rozdíly, zaměřil jsem se na kvalitu zachycení pohybu osoby, ovládání konkrétního testovaného dema za pomocí periferií a v neposlední řadě pohodlí uživatele včetně kvality zpracování obou sestav. [13] [14] [15] [16]

U zařízení Oculus Rift DK2 je při ovládání v sedě záznam pohybu velmi dobrý. Souprava detekuje otáčení hlavy kolem vlastní osy, ale také náklon hlavou do stran, která dříve nebyla podporována. Při ovládání ve stoje zůstává vjem stále stejný, ale pohyby těla už Oculus Rift DK2 začíná zaznamenávat hůře, hlavně když se uživatel se soupravou předkloní, nedokáže se „headset“ přizpůsobit změně směrové orientace. Pro testování ovládání jsem použil Xbox 360 bezdrátový ovladač. Dema byla na ovladač připravena a vše fungovalo perfektně, ale schopnost vtáhnutí do VR nebyla úplně ideální za situace kdy ovladač nepohyboval směrování, ale určoval směr těla. To mělo v některých případech za následek neadekvátní pohyby v prostoru. Oculus Rift DK2 je kvalitně zpracován. Jedná se o kombinaci plastu a kovu, kde kov drží základní konstrukci a pro snížení váhy i ceny je obal z tvrdého plastu. Co se týká pohodlí nošení soustavy se Oculus Rift DK2 řadí mezi nadprů-

měrný produkt, kdy celá sestava na hlavě není nad míru těžká, díky tomu lze se soustavou snadno pohybovat bez žádné komplikace. Jediné k čemu jsem měl trochu výhrady, je značný tlak směrem na nosní přepážku, který Oculus Rift DK2 vyvíjí. Při dlouhodobém nošení způsobuje tedy otlačení nosu. [13] [14]
Při testování HTC Vive jsem byl příjemně překvapen. Souprava váží oproti Oculus Rift DK2 o několik gramů více (přibližně o 75 gramů), takže jsem očekával, že se bude s HTC Vive „headsetem“ pohybovat hůře, ale nebylo tomu tak. Rozdíl jsem pocítil, ale nebyl tak velký, jak jsem původně předpokládal. Na základě připravených zkušebních programů jsem otestoval jen pohyb ve stoje společně s HTC Vive ovladači. Snímání pohybů uživatele u HTC Vive je velmi dobře provedené, všechny pohyby co zaznamenával Oculus Rift DK2, umí HTC Vive také, navíc dokáže zaznamenat i pohyb předklonu hráče bez výše popsaných problémů. Použití ovladačů je velmi intuitivní a snímané pohyby, které dokáží zpracovat, jsou ve všech třech osách. Společně s „motion track padem“ (senzorem snímaní pohybu prstů na ovladači) a dalšími tlačítky ovladač hodnotím velmi pozitivně. HTC Vive tedy vítězí na poli kvality záznamu pohybu a ovládání. Testovanaá souprava je co do kvality zpracování velmi podobná Oculus Rift DK2, kdy hlavní šasi zařízení je z kovu a obal chrání plastový kryt, na kterém je umístěn senzor pro snímaní pohybu „headsetu“. Stejně jako Oculus Rift DK2 trápí HTC Vive stejný problém s otlačením nosní přepážky uživatele, kde zvýšená hmotnost ještě umocňuje vyvíjený tlak. [13] [14]
Sečteno podtrženo celkový výsledek je dosti vyrovnaný, kde oba produkty mají velmi totožné hardwarové požadavky, ale také jejich specifikace a dema se mohou hrát ve stoje či v sedě s klasickým ovladačem (vyjma některých testovacích scén). Pro účely mojí bakalářské práce jsem ale nakonec zvolil HTC Vive. Oculus Rift DK2 je původně navržen pro pocit v sedě. HTC Vive zvládá obojí, jak konfiguraci vnímání pocitu „v sedě“, tak „ve stoje“. Společně s mírně lepším záznamem pohybu v prostoru a vybaveností pohybovými ovladači (Oculus Rift se již dnes také prodává s jejich pohybovými ovladači, ale v době testování jsem ovladače neměl k dispozici) proto finálním vítězem a také produktem, který budu dále používat je HTC Vive.

4.2 Virtuální souprava HTC Vive

HTC Vive je virtuální souprava vyvíjená firmou HTC společně s firmou Valve Corporation. Souprava je dimenzována pro pohyb v malé místnosti, kde pro určení přesné polohy slouží bázový systém, který by se měl v nej-

lepším případě nacházet v rozích místnosti. Společně s pohybovými ovladači speciálně navrženými pro tuto platformu lze dosáhnout zajímavých realizací aplikací, využívajících pohyb v prostoru a interakci s prostředím.
HTC Vive byl poprvé představen Philem Chenem během HTC Mobile Congress v březnu roku 2015. První vývojové soupravy byly zaslány v srpnu a září roku 2015 a první verze pro spotřebitele byly vydány až 5. dubna 2016. V červnu 2017 Valve odhalil detaily druhé generace Vive ovladačů, které využívají systému sledování pohybu prstů pro další možné variace ovládání.[11]

4.2.1 Příslušenství

1. **HTC Vive souprava** – Zařízení disponuje dvěma displeji na technologii OLED, přičemž každý displej generuje obnovovací frekvenci 90 Hz s celkovým pozorovacím úhlem obou displejů 110 stupňů. Každý displej zobrazuje 1080x1200 pixelů, tedy dohromady získáme rozlišení 2160x1200 pixelů. Pro zajištění bezpečné orientace v prostoru obsahuje souprava přední kameru, která umožňuje uživateli pozorovat své okolí, aniž by si musel sundat brýle. Pro stanovení aktivního prostoru není navržen žádný automatický systém, proto je nutné předem pomocí ovladače nastavit aktivní prostor, který bude následně označovat vnější bariéru. Pokud se uživatel přiblíží k hranici vnější bariéry, bude o této situaci informován skrze vykreslení virtuální bariéry uvnitř soupravy. Souprava nadále obsahuje mnoho druhů senzorů, mezi které kupříkladu patří vnější systém infračervených senzorů, které přijímají paprsky od bázového systém k určení polohy. Mezi další senzory patří akcelerometr, gyroskop a proximity senzor. V lednu 2018 vydalo HTC novou verzi soupravy „Vive Pro“. Vive Pro vylepšilo rozlišení displeje pro každé oko na 1400x1600 pixelů, tedy v součtu 2800x1600. Přibyla další vnější kamera na brýlích, připojitelná sluchátka a došlo k celkové úpravě ergonomie soupravy včetně snížení váhy a zmenšení velikosti. [11]

2. **HTC Vive ovladač** – Bezdrátový ovladač si lze představit jako virtuální ruce ve VR. Ovladač obsahuje dotykový senzor, několik tlačítek a páčkových spouštěčů. Na jedno nabytí lze použít ovladač po dobu 6 hodin. Okolo kruhové části ovladače lze nalézt dvacet čtyři infračervené senzory, které mají za účel detekování pozice v prostoru.[11]

3. **HTC Vive bázový systém** – Také znám jako „Lighthouse tracking system“ jsou dvě černé skříňky, které emitují infračervené pulsy s frekvencí 60 pulsů za sekundu (maximální rozměr místnosti činí 4,5 x 4,5

metru). Infračervené paprsky jsou následně zachyceny brýlemi a ovladači s přesností méně než 1 cm. Bezdrátová synchronizace snižuje počet kabelů.[11]

4. **HTC Vive snímač** – Externí snímač slouží jako dodatečná proprieta k určení dalšího bodu v prostoru. Snímač si lze dokoupit separátně nebo je přibalen u dražší edice soupravy, v některých případech se též dodává s příslušnou aplikací. Snímač lze namontovat téměř kamkoliv, kupříkladu na židli, na zem, na vlastní tělo atd.[11]

5. **HTC Vive bezdrátový adaptér** - Společně s vydáním HTC Vive Pro byl také představen bezdrátový modul. Modul má za účel nahradit standardní připojení soupravy s PC pomocí speciálního bezdrátového zařízení, které bude posílat data místo po HDMI kabelu bezdrátově. Celý systém vyžaduje instalaci speciální karty (která je součástí balení) do počítače a připojení senzoru s dosahem 6 metrů od umístění a úhlem záběru 150 stupňů. Samotné bezdrátové zařízení pracuje v nezarušeném pásmu 60 GHz a pro přenos využívá kodeku XR od společnosti DisplayLink. Jelikož již není potřeba mít k počítači a soupravě připojen žádný kabel, je nutné soupravu napájet externě. HTC proto společně s bezdrátovým modulem přibalil i externí baterii.[18]

6. **HTC Vive „Delux Audio Strap“** - V červnu 2017 HTC uvolnila speciální „Delux Audio Strap“, což nebylo nic jiného, než jejich vlastní sluchátka přímo navržená pro sestavu HTC Vive. Společně s tímto příslušenstvím vylepšili i ergonomii samotných brýlí, aby byly více komfortní a lehčí pro snadnější použití včetně lepšího rozložení váhy.[11]

5 Témata pro realizaci VR

Na základě zadání bakalářské práce mám za úkol seznámit se s různými odděleními na katedře grafiky, kde mi vysvětlí, co na jednotlivých odděleních konkrétně zpracovávají, předvedou mi několik možných scén a jejich vizualizovatelných výstupů a výsledků. Posledním krokem je předběžný návrh o realizaci projektu společně s požadavky na konkrétní téma. Následně bych si měl ze všech těchto informací vybrat ta témata, která by zapadala do konceptu VR. V rámci katedry grafiky jsem se seznámil se třemi projekty, na kterých toto univerzitní pracoviště momentálně pracuje.
Jedná se o zpracování, analýzu a simulaci enzymů, kde data, získaná externě či interně v rámci Univerzity zpracují, následně analyzují a poté pro některé enzymy navrhnou i průběh či simulaci daného enzymu. Každý takovýto enzym se skládá z mnoha atomů prvků, převážně uhlíku, dusíku, kyslíku, ale také i síry. Tyto atomy následně s vazbami mezi sebou tvoří molekulu zkoumaného enzymu. Tento projekt mi představil a lehce nastínil pan Mgr. Martin Maňák. V rámci projektu mi poskytl zkušební data a dostatečné informace k navrhnutí dema pro VR. Dále mě pan Maňák požádal, aby se v demu enzym a jeho atomy zobrazovaly věrně a aby šlo s daným enzymem interagovat. Odkaz na článek zaobírající se konkrétním problém enzymů lze nalézt v referencích. [20]
Druhý projekt, který mi byl představen uvažuje problém týkající se komprese a minimalizace modelů. Vedoucím tohoto projektu je pan Doc. Ing. Libor Váša, Ph.D. V rámci projektu mi byl vysvětlen princip celé jeho práce včetně jeho týmu. Jak již bylo naznačeno výše, tématem projektu je nalézt vhodný algoritmus pro kompresi modelů a najít kompromis mezi kompresními poměry tak, aby kvalita původních dat zůstala téměř nezměněna. Následně si pan Váša představoval pro vizualizaci projektu ve VR různé typy modelů v různých kompresních poměrech. Tyto výsledky by pak byly zobrazeny ve VR, kde by následně bylo možné mezi projekty přepínat. [21]
Posledním tématem je dekompozice modelů a následně vhodná kategorizace do předem nadefinovaných šablon. Toto téma mi bylo představeno mým zadavatelem panem Ing. Petrem Vaněčkem, Ph.D. Téma se zaobírá problémem definice modelů, kde je v průmyslu nutné se rozhodnout, jakého tvaru je model vytvořený kupříkladu v AutoCADu (modelovací software speciálně navržen pro výrobní průmysl), tedy zda-li má tvar trubky, krychle, válce atd. Tímto problémem se zabývá tým řešitelů včetně pana Vaněčka. Pan Vaněček mi předvedl různé modely, které na základě algoritmu rozdělil do

příslušných kategorií a na závěr byl původnímu modelu připsán vektor stavových hodnot, který ho symbolizoval. Všechna nashromážděná data, která algoritmus načetl následně používal i při porovnávání dalších modelů. Na základě všech těchto informací mi bylo doporučeno, že by bylo vhodné pro mé demo vzít několik zkušebních modelů, předpřipravit si stavové vektory a následně při výběru jednoho z nich zvýraznit modely, které se nejvíce blíží konkrétnímu vybranému modelu.
Celý koncept si nyní rozdělím do čtyř kategorií. Základní místnost, projekt enzym, projekt komprese modelů a projekt kategorizace modelů.[22]

5.1 Základní místnost

Jako v každé hře nebo demu je nutné mít nějaký druh úvodního prostoru, kde je uživatel (hráč) seznámen s ovládáním. Tento stav jsem nazval „Základní místnost“. Základní místnost musí obsahovat nějaký druh tutoriálu a navíc nějaký druh popisu, co lze od dema očekávat, včetně způsobu, jak lze mezi projekty přecházet. Po dohodě s konzultantem zvolil jsem jako Základní místnost model učebna UC335, kde se momentálně nachází souprava HTC Vive, kterou má škola k dispozici. V počátečním návrhu konceptu místnosti jsem uvažoval, že po nasazení brýlí a spuštění mé aplikace se přenese uživatel virtuálně do středu místnosti. Celou tuto místnost jsem si změřil a v programu trial SketchUp Pro (software pro vytváření modelů a staveb) vymodeloval. Společně s místností jsem si dále také změřil vstupní dveře, školní stůl a také tzv. „Pískoviště“. Pískoviště je projekt vytvořený katedrou grafiky. Jedná se o dřevěnou vanu vyplněnou pískem, nad kterou ze shora svítí projektor napojený na kinect od Xboxu, který snímá vzdálenost písku od kinectu. Následně podle naměřené vzdálenosti vizualizuje krajinu. Celý tento systém jsem tedy také ve zjednodušené verzi přenesl do mého projektu.
Během zpracování jsem řešil problém přepínání mezi projekty v aplikace. Prvním řešením, nad kterým jsem uvažoval, bylo přepínání úrovní pomocí jednoduchého systému výtahu. Výtah by se kupříkladu po stisknutí tlačítka v počáteční místnosti vynořil ze země a následně by se hráči otevřely dveře. Hráč by vstoupil do výtahu a pro výběr projektu, který by si chtěl vyzkoušet by si zvolil konkrétní patro. Následně by se výtah zavřel a hráč by se fyzicky přesunul do nové pozice, kde by byl příslušný projekt nebo by se načetla nová úroveň. Druhým řešením by mohl být systém ovládacího panelu, kdy by si uživatel na panelu vybral jaký projekt by si přál vidět a pak by se opět fyzicky „teleportoval“ na místo, které si vybral, nebo by se opět načetla nová

úroveň. Po dohodě s konzultantem se nabízela možnost návrhu, kde by se v základní místnosti umístil stůl, na jehož povrchu by se nacházely helmy, které by symbolizovaly příslušné projekty. Hráč by si pro navštívení úrovně musel nasadit helmu, která by ho následně do konkrétního místa přenesla.
Po výběru bude tedy místnost obsahovat toto: Hráč je po zapnutí aplikace umístěn před stůl přibližně uprostřed místnosti, tak aby uživatel byl mezi snímači pohybu dobře umístěn. Dále se uživateli zobrazí 2D nabídka, která vysvětlí uživateli základní ovládání včetně nabízených projektů a jako poslední se uživateli zobrazí stůl s helmami, které se budou nacházet v rohu místnosti.

5.2 Prezentace projektu enzym

Po nasazení helmy v základní místnosti (helma pro tento level je modrá) se hráč přenese do nové „místnosti“ navržené pro demonstraci enzymu. Na základě požadavků konzultanta jsem vytvořil model místnosti, která se skládá z kontrolního panelu, podstavce pro enzym a následně po výběru konkrétního enzymu pak i model tohoto enzymu. Jelikož hlavním cílem je představit konkrétní enzymy, rozhodl jsem se umístit velký podstavec doprostřed místnosti, nad kterým se následně po výběru enzymu zhmotní model. Uživatel, který se po výběru projektu enzym přenese do tohoto levelu, se objeví přímo u kontrolního panelu, kde si bude moci vybrat na základě předpřipravených dat několik enzymů a jejich variant. Po výběru enzymu se ze země „vynoří“ velký podstavec a zobrazí se enzym nad ním. Enzym se následně začne otáčet pomalu kolem své osy.
Jak již bylo řečeno v předchozí kapitole, enzym se skládá z několika set atomů. Má data obsahovala přes několik tisíc atomů. Při pokusu o načtení kompletního enzymu a zobrazení všech jeho atomů, frekvence snímků za vteřinu klesla na hodnoty v řádu jednotek. Důvod toho je velmi prostý. Jelikož jsem chtěl vykreslovat přes pět tisíc atomů najednou, CPU nedokázal pojmout tolik požadavků na vykreslení. Proto jsem se rozhodl tato data zmenšit na takovou hodnotu, aby výsledný enzym vypadal dostatečně věrohodně a získal jsem redukovanou množinu atomů pro vykreslení enzymu. Po testování různých metod, jak zredukovat počet atomů pro konkrétní enzym, jsem použil vztah:

$$\begin{gathered}[B] = \{A_{k+n*i}\}, \quad k = N, \quad n = N \\ i = 0, 1, ..., \frac{size(A) - k}{n} - 1\end{gathered} \tag{5.1}$$

Vektor **A** definuje seznam všech atomů, **k** definuje posun v seznamu atomů, tedy od jakého atomu se začne, **n** definuje jak velké kroky jsou mezi hodnotami a **i** definuje index posloupnosti. Kupříkladu pro konkrétní vektor $A = [1, 2, 3, ..., 15], k = 3, n = 2$ získáme výsledný vektor $B = [3, 5, 7, 9, 11, 13]$ Následně pak vektor **B** je seznam vybraných atomů, které tvořily původní enzym. Jinými slovy, z původních dat daného enzymu si vyberu jen část, ale zachovám tím dostatečnou informaci pro zachování realistické vizualizace. Kontrolní panel nabízí výběr mezi redukcemi každého desátého prvku, každého patnáctého a pak každého dvacátého. Pro každý tento typ modelu jsem dále připravil dvě další varianty. Ve výsledky má uživatel na výběr z šesti možných typů enzymů.
Projekt enzym se začne po zobrazení na velkém podstavci uprostřed místnosti pomalu otáčet kolem své osy, nebo pomocí kontrolního panelu lze navolit možnost otáčet enzymem ručně. Různé druhy atomů jsou u konkrétního enzymu dále reprezentovány jinými barvami, aby bylo možné identifikovat, jaké podíly atomů konkrétní enzym obsahuje. Dále se nabízí na panelu možnost zredukovat zobrazení atomů jen na konkrétní prvky. Kupříkladu je možné si nechat zobrazit jen atomy uhlíku, přičemž ostatní atomy zmizí. Možnosti filtrace lze kombinovat mezi všemi atomy. Pokud by uživatel chtěl zobrazit jiný druh enzymu z původního seznamu, stačí opět u konzole nastavit konkrétní enzym a spustit nahrávací sekvenci, čímž se zobrazovaný enzym přepne na uživatelem zvolený.

5.3 Projekt kategorizace modelů

V dalším projektu se budu zabývat tématem pro dekompozici a kategorizaci různých typů modelů. Rozhodl jsem se na doporučení vedoucího práce, že pro demonstraci projektu využiji úzkou skupinu modelů, které jsem získal od zadávajícího projektu a k tomu také výsledky algoritmu kategorizace každého modelu. Tato data budou následně načtena a zobrazena na předem připravených platformách k prohlédnutí.
Uživatel, který se „ocitne“ v tomto projektu po nasazení helmy a následném přenesení se do konkrétní místnosti, bude mít již připravené modely kolem něj a také kontrolní panel. Uživatel si bude moci modely prohlédnout a pokud si žádá detailnější popis, vybere konkrétní model z nabídky. Vybraný model se následně přenese do středu místnosti, kde bude připravený velký stojan, nad kterým se model zobrazí a přizpůsobí svojí velikost k lepšímu zobrazení konkrétního modelu. Kolem stojanu a modelu budou další informace, například do jaké kategorie nejvíce zapadá, jak se model jmenuje atd.

Společně s vybraným modelem se u všech ostatních objektů, které sdílejí stejnou dominantní kategorii a přesahují určitou mez pravděpodobnosti zařazení do této kategorie, jejich barva se přizpůsobí tak, aby uživatel lépe rozeznal podobně vybrané modely. Tato změna má za účel simulovat, jak algoritmus definuje podobnost mezi různými typy modelů a přesnost tohoto určení. Každý model se bude pomalu otáčet v prostoru, pro lepší celkovou představu. Pokud by si uživatel chtěl porovnat dva modely mezi sebou pro přesnější analýzu mezi oběma modely, lze vybrat pomocí ovladače první a druhý model pro porovnání a ty se následně oba přenesou do středu místnosti vedle sebe. U každého z nich bude popsáno, do jaké kategorie spadají a následně bude vypsán i celkový stavový graf pro každý z modelů. Pro výběr porovnání jiného modelu lze zrušit výběr konkrétního modelu a vybrat si jiný k porovnání.

5.4 Projekt komprese modelů

Posledním projektem, který jsem zpracovával je komprese modelů. Pro toto konkrétní téma jsem se rozhodl připravit návrh, který se koncentroval na představení modelů modifikovaných kompresním algoritmem. Pro návrh jsem se rozhodl použít čtyři modely. Vybrané modely byly následně upraveny a uloženy za pomoci algoritmu, který je vyvíjen na katedře grafiky pro účely minimalizace velikosti modelů. Pro demonstraci této procedury jsem se rozhodl použít několik kompresních poměrů. Vytvořené modely budou následně načteny enginem a uživateli zobrazeny. Na základě těchto datových podkladů si lze v tomto projektu nastavit 16 kombinací pro konkrétní model k přímému porovnání.
„Místnost“, kterou jsem navrhl pro tento projekt, se skládá ze dvou podstavců a dvou kontrolních panelů k ovládání každého podstavce pro zobrazení konkrétního modelu a příslušných kompresních poměrů. Uživatel si na každém z těchto panelů vybere určitou instanci k prohlédnutí, která se po načtení zobrazí nad konkrétním podstavcem. Nad vyobrazeným modelem lze sledovat informace, jako je konkrétní kompresní poměr, velikost souboru na disku, počet polygonů atd. Každý z modelů se bude otáčet kolem své osy, ale bude také možné nastavit stacionární pozici aktivací ručního otáčení, které se bude nacházet na příslušném kontrolním panelu svázaného s modelem.

5.5 Rozšířitelnost aplikace

Jedním z bodů bakalářské práce je navrhnout všechny funkce popsané výše navrhnout tak, aby bylo možné v budoucnu přidat nové typy projektů, které by využívaly mnou vytvořené procedury jako stavební kameny. Rozšířitelnost jsem se rozhodl rozdělit do dvou rovin.
První rovina se bude týkat maximálního zjednodušení práce nad vytvořeným projektem v Unreal Engine 4, kde všechna data, modely, materiály i skripty budou přístupná mezi všemi možnými levely. Dále u všech skriptů a podprocedur bude detailně popsán jejich význam a využití. Posledním bodem je scéna, kterou bude moci další člověk využít jako základní kostru, kde si na základě vlastní potřeby upraví, co bude potřebovat či přidávat nové vlastnosti.
Druhá rovina rozšířitelnosti se zaobírá nastavením aplikace, která na základě konfiguračního souboru dokáže předpřipravit několik potřebných parametrů. Načtení a umístění levelů a s tím spjaté vygenerovaní helmy v základní místnosti. Dále také teleportace mezi příslušnými levely a zobrazení informací o nové úrovni v základní místnosti. Oba tyto systémy navzájem spolupracují a podílejí se na efektivnější tvorbě prostředí všech projektů, ale hlavně na budování nových témat.

6 Analýza komponent uvnitř projektů

U kreativních výstupů práce jsem se rozhodl vzít v úvahu všechna kritéria, která jsou nutná pro splnění návrhu (uvedených v předchozích kapitolách) a na jejich základě rozpracovat finální řešení problému k daným komponentám, ze kterých se bude aplikace skládat. Rozdělení komponent budu brát od těch nejméně složitých, až po komplexní spojení subsystémů do konkrétního celku představujícího projektu. Všechna představená řešení jsem navrhl na základě vlastního úsudku, názorů vyučujících a také se nechal inspirovat v knize *Unreal Engine VR Cookbook* od autora Mitche McCaffreyho.[19]

6.1 Statické modely

Pro vytvoření všech místností včetně učebny UC335 a ostatních dalších zařízení, jako například stoly, židle, podstavce a tak dále lze přistupovat několika způsoby. První způsob může být vyhotovení finálního modelu přeměřením příslušného předmětu a následně podle naměřených hodnot a dopočtení zbylých neznámých stran zkonstruovat konkrétní model. Druhá možnost je využít engine, kde lze pomocí předdefinovaných základních bloků, poskládat komplexnější strukturu. Tato metoda se velmi hojně používá pro vymodelování místností, či jednoduchých objektů. Bohužel pro modely, které vyžadují více drobných detailů, nelze tuto metodu doporučit. Poslední z možností, je převzít modely od jiných tvůrců a ty následně použít či upravit podle osobních požadavků. Modely získané tímto způsobem velmi šetří čas, jelikož vytvořit model konkrétního předmětu tak, aby dosahoval požadovaných kvalit, může zabrat velmi dlouho. Tato možnost má ale i své nevýhody. Licence použití konkrétního modelu může být velmi striktní v tom, jak lze daný objekt použít, ale také cena daného modelu může být neúměrně vysoká.
V rámci mého projektu jsem se rozhodl vyzkoušet a použít všechny tři možnosti, které jsem uvedl v předchozím odstavci. První metoda pro můj případ zahrnovala přeměření místnosti UC335, stolu na kterém je umístěn počítač, zařízení pro simulaci terénu nazvané „pískoviště“ a oken. Metodu číslo dvě jsem použil pro procedurálně generované mapy k vytvoření okolního prostředí každé úrovně, kde na vzhledu mimo centrum dění moc nezáleží. Generování stěn, podlah i stropů je ve výsledku mnohem jednodušší. Pro přípravu

rozšířitelnosti aplikace jsem navrhl soubor, na jehož základě se vytvoří konkrétní počet místností včetně vytvořených helem k přesunu mezi projekty. Je třeba zmínit, že například všechny atomy pro model enzymu jsou též procedurálně generované s využitím primitivního modelu koule. Díky tomu lze libovolně měnit vlastnosti každého z atomů podle potřeby. K poslední možnosti jsem se uchýlil jen u dvou případů, jedná se konkrétněji o model kancelářské židle, která slouží jako doplněk k základní místnosti a pak o model helmy, která slouží jako teleportační zařízení mezi projekty (levely aplikace). K tomuto rozhodnutí jsem dospěl na základě zjištění náročnosti vytvoření kvalitního modelu, který by v mém projektu vypadal dostatečně dobře, jelikož s vytvářením modelů jsem se seznámil až teprve v průběhu zpracování bakalářské práce a nemám mnoho zkušeností s vytvářením podobných modelů. Na závěr doplňuji, že v dalších kapitolách budou tyto modely představeny jak vypadají v editoru modelů, tak i následně v aplikaci po zakomponování textur.

6.2 Projekt enzym

Pro tento projekt rozeberu načtení a zobrazení atomů v prostoru, otáčení enzymu kolem své vlastní osy, kontrolní panel a všechny jeho funkcionality.

6.2.1 Způsob načtení atomů a jejich zobrazení

K vykreslení kompletního enzymu je nutné znát všechny jeho atomy, kde se v prostoru nalézají, z jakého prvku jsou a jaká je velikost konkrétního atomu. Od konzultanta jsem vyzískal data dvou enzymů, která používají ve svých výpočtech. Struktura tohoto souboru je rozdělena do dvou vrstev. První vrstva zahrnuje data o všech atomech, které se v konkrétním enzymu nalézají a druhá vrstva definuje vnitřní prostory mezi atomy, tzv. komůrky. Pro mé účely práce jsem se rozhodl využit pouze data pro vykreslení samotného enzymu. Na obrázku 6.1 představuji jeden z možných případů, jak konkrétní datový soubor vypadá. V přiloženém DVD lze nalézt tento soubor pro detailnější nahlédnutí. Z úryvku lze vyčíst, že pro každý atom je definována pozice ve třech rozměrech, jakého je typu prvku a pak rádius atomu (poloměr koule, která ho reprezentuje).
Načtení atomů je tedy velmi jednoduché. Provede se odstranění komentářů na počátku souboru, které slouží jako vysvětlení, co které hodnoty dále znamenají a také kolik atomů daný enzym obsahuje (nutné pro další použití). Dalším krokem algoritmu bude načtení řádků souboru s daty jednoho enzymu a jejich správné rozdělení. Po rozdělení všech hodnot se data umístí

```
# ATOMS(5579), FORMAT(CenterX CenterY CenterZ Radius Name)
18.16 95.732 23.522 1.55 Nitrogen
19.466 96.185 24.079 1.7 Carbon
20.643 95.597 23.318 1.7 Carbon
20.995 94.443 23.53 1.52 Oxygen
19.595 95.777 25.548 1.7 Carbon
20.961 96.116 26.15 1.7 Carbon
21.184 95.577 27.866 1.8 Sulfur
21.762 93.893 27.638 1.7 Carbon
21.251 96.372 22.426 1.55 Nitrogen
22.406 95.873 21.698 1.7 Carbon
23.628 96.081 22.579 1.7 Carbon
23.786 97.121 23.219 1.52 Oxygen
22.609 96.62 20.385 1.7 Carbon
21.491 96.45 19.396 1.7 Carbon
21.832 97.039 18.016 1.7 Carbon
20.655 97.058 17.148 1.55 Nitrogen
19.902 95.99 16.89 1.7 Carbon
20.207 94.813 17.425 1.55 Nitrogen

# CAVITIES(15)

.
.
.
# ID(1), SPHERES(8), FORMAT(CenterX CenterY CenterZ Radius)
28.281928419098712 77.72487264178568 32.84422124263192 1.8700000047683716
28.149172026598798 77.38977662453152 32.13023225656233 1.8700000047683716
27.70792395250555 78.02271311077587 32.17749623258916 1.8700000047683716
27.998477889577597 78.08230039113413 32.35114833185453 1.8700000047683716
28.008254217079305 77.77485879280174 32.106298980229255 1.8700000047683716
27.2587377312267 76.99867176349582 32.688071606955425 1.8700000047683716
27.083635593455725 76.92773287942639 32.67544011742601 1.8700000047683716
27.08747979611202 76.81358707704042 32.67975261388994 1.8700000047683716
```

Obrázek 6.1: Obrázek s ukázkou souboru data_atom01.txt

do příslušných seznamů. Tento proces se bude opakovat stále dokola do té doby, dokud počet načtených řádek se nebude rovnat počtu všech atomů. Dalším nutným úkolem, který je nutné provést pro správné fungování dalších komponent, je vypočítat geometrický střed konkrétního enzymu. Geometrický střed bude dále sloužit jako střed enzymu pro posun celého enzymu najednou a také pro otáčení enzymu kolem své vlastní osy. Výpočet jsem provedl pomocí průměru všech načtených pozic atomů. Předpis pro výpočet středu uvádím:

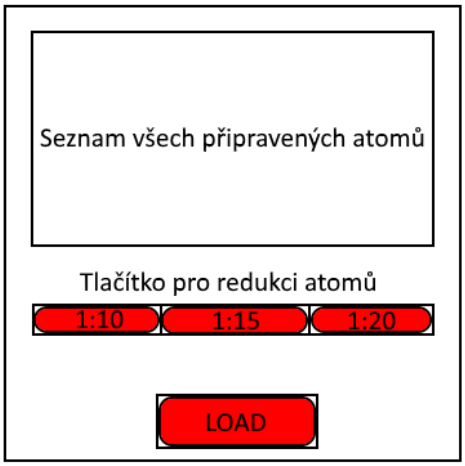
$$X_j = [\frac{\sum_{i=0}^{k} \text{atom}_i(x_j)}{n}] \qquad k = 0, 1, ..., n-1 \quad n = \text{počet atomů} \qquad (6.1)$$

Hodnoty $X = [x_1, x_2, x_3]$ jsou nyní geometrickým středem celého enzymu.

6.2.2 Kontrolní panel a další funkcionality

Kontrolní panel zprostředkovává komunikaci mezi schopnostmi systému projektu enzym a samotným uživatelem VR. Vzhledem k maximálnímu zjednodušení ovládání a omezení prostoru pro umístění dalších ovládacích prvků jsem se rozhodl, přenést většinu ovládání do tohoto panelu. Kontrolní panel umožňuje uživateli vybrat si enzym a následně ho přenést do VR. Předkládá

další možnosti, jako například rotovat s objektem podle libosti a v neposlední řadě také filtrovat enzym pro zobrazení jen atomů prvků, které chce uživatel vidět. Společně se všemi těmito prvky také kontrolní panel okolo enzymu zobrazuje data.
První část panelu (zleva) je věnována načítání enzymu. Pro tento systém výběru jsem navrhl jednoduchý seznam a tzv. „check box“. Seznam obsahuje všechny přístupné enzymy, ze kterých je možné si vybírat. Výběr konkrétního enzymu uživatel provede dotknutím se názvu enzymu a stisknutím tlačítka na ovladači. Po provedení této akce musí dále uživatel provést výběr redukce dat. To provede na základě stisknutí příslušného tlačítka, které bude v tento moment zvýrazněno. Na výběr jsou tři možnosti: 1:10, 1:15, 1:20. Po výběru obou těchto kritérií se zvýrazní velké tlačítko, které bude mít nápis „LOAD“ to následně odstartuje proces, který zobrazí daný enzym na podstavci. Pokud by si uživatel chtěl nechat zobrazit jiný enzym, musí provést stejné operace znovu, tak jak bylo popsáno. Nově vybraný enzym překryje ten původně načtený. Na obrázku 6.2 níže lze vidět schéma panelu.
Druhá část panelu (prostřední) zprostředkovává zobrazení informací o en-

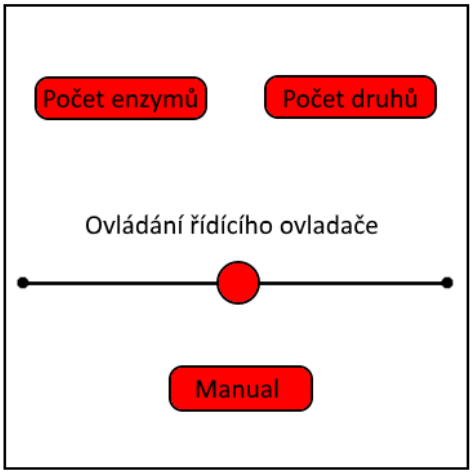


Obrázek 6.2: Schéma panelu pro načtení konkrétního enzymu.

zymu a také ovládání rotace na podstavci. Volba zobrazení různých dat je uživateli nabídnuta formou tlačítek. Tato data dále budou „levitovat“ v prostoru okolo enzymu. Každá z těchto informací bude moci být vybrána naráz či v různých variantách, takže je nutné navrhnout systém, který bude informace správně řadit pod sebe, či vedle sebe. Informace, které chci aby informační panel enzymu byl schopen zobrazit jsou: Počet atomů enzymu a počet atomů jednotlivých druhů. Data o počtu atomů enzymu umístím nad vrchol celého enzymu, ale data pro počet atomů jednotlivých druhů

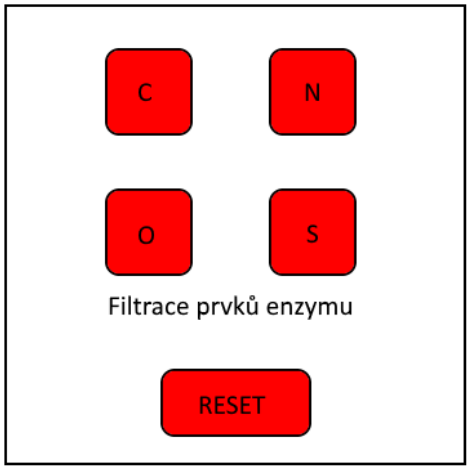
budou na boku enzymu, přesněji řečeno pokud bude uživatel stát čelem k enzymu, budou vidět na jeho pravé straně. Při výběru více typů atomů se zobrazí data pod sebou. K ovládání rotace enzymu bude mít na výběr uživatel dvě možnosti: Buď nechá enzym rotovat automaticky neustále, nebo zvolí možnost ručního ovládání pomocí tlačítka, které bude muset stisknout. Při zvolení této možnosti bude pomocí řídícího ovladače rotovat s enzymem. Uživatel provede úchop a následně pohybem své ruky bude provádět posun po kontrolním panelu, který poté ovlivní rotaci enzymu. Pokud by uživatel chtěl opět přejít do automatického otáčení enzymu, stačí znovu stisknout tlačítko pro manuální otáčení a systém zapne automatické otáčení enzymu. Na obrázku 6.3 níže lze vidět schéma panelu.

Poslední část (pravá) se soustřeďuje na filtraci dat z enzymu k zobrazení jen



Obrázek 6.3: Schéma panelu pro zobrazení dat a otáčení enzymu.

některých druhů atomů. K filtrování těchto dat je zde opět použit systém „check boxů“, které budou reprezentovat tlačítka na panelu. Každé tlačítko aktivuje konkrétní typy atomů, kupříkladu atom dusíku, uhlíku atd. Po stisknutí příslušného tlačítka se všechny atomy, které mají příslušně zvolený typ odstraní z enzymu a zůstanou všechny zbývající. Lze takto pokračovat i s ostatními typy, až by nakonec žádný nezbyl. K opětovnému zapnutí typu atomu stačí znovu stisknout tlačítko konkrétního typu. Při výběru nového enzymu se všechna předchozí nastavení na kontrolním panelu nastaví do původního stavu, aby nedošlo k žádné chybě při filtraci, zobrazení či rotaci enzymu. Schéma panelu je na obrázku 6.4.



Obrázek 6.4: Schéma panelu pro filtraci dat z enzymu.

6.3 Projekt kategorizace modelů

Projekt dekompozice a kategorizace modelů nyní rozdělím do sekcí. Proberu formát nezpracovaných dat a jejich následnou úpravu do nově vytvořeného souboru, kde popíši jeho novou strukturu. Za druhé upřesním kontrolní panel, který opět slouží jako zjednodušení operací mezi systémem a uživatelem.

6.3.1 Příprava dat a jejich formát

Ze získaných modelů od vedoucího projektu a vygenerovaných dat určujících kategorizaci těchto modelů jsem musel vymyslet, jak původní data přetransformovat do jednotné formy. První součástí bylo analyzovat obsah souboru **TestData4.similarity.xml**. Konkrétní soubor obsahuje název modelu a následně seznam všech modelů, které algoritmus vyhodnotil jako blízké originálu. Dalším typem souboru je pak kromě samotného modelu i jeho výsledná zpráva, kterou vytvořil algoritmus při zjištění do jaké skupiny patří (pláty, trubky, tyče atd.). Tento soubor je vygenerován pro každý model zvlášť a má příponu **.fts**. Konkrétní výsledná zpráva pro model je ukázána na obrázku 6.5. Pro zjednodušení práce se všemi představenými prostředky bylo nutné zakomponovat data ze souboru **TestData4.similarity.xml** a všech výsledků zpráv do jednoho kompaktního souboru.
Situaci jsem vyřešil vytvořením programu v Jave, kde jsem provedl načtení konkrétního xml dokumentu do paměti. Následně podle načteného názvu modelu byla přiřazena konkrétní data příslušného **fts** souboru a takto upravené informace se uložily do nového souboru. Pro zjednodušení nyní před-

```
PartsCount: 33
RecPartsCount: 29
PlaneCount: 12
CylinderCount: 17
RodCount: 6
CylHoleCount: 11
CompleteRodCount: 0
CompleteNonRodCount: 6
PlaneArea: 0.583614218694931
CylArea: 0.378829754039409
RodArea: 0.095686786488875
CylHoleArea: 0.283142967550534
VolumeToAreaRadius: 0.29182769247784
EigenValueRatio: 4.84897763403115
CocentricCylinders.N-tuples: 3
CocentricCylinders.Tubes: 0
SymmetryRSigma: 0.00404237350801481
```

Obrázek 6.5: Úryvek ze souboru **121$1.3D.STX.fts**

vedu na příkladu, jak může takový konkrétní soubor vypadat.
Každý model popsaný tímto souborem bude rozdělen na dvě části. První z nich definuje všechny podstatné informace o konkrétnímu modelu. Druhá část upřesní všechny modely, které jsou původnímu modelu podobné a jejich hodnotu vzdálenosti. První část popisu modelu:

„!název-modelu|počet-podobných-modelů|data-z-fts-souboru|"

Data z fts souboru jsou uspořádána následovně (názvy jsou přesně podle původního souboru fts):

„PartsCount,RecPartsCount,PlaneCount,CylinderCount,RodCount,
CylHoleCount,CompleteRodCount,CompleteNonRodCount,PlaneArea,
CylArea,RodArea,CylHoleArea,VolumeToAreaRadius"

Stanovení dat pro podobné modely jsou rozděleny do řádků, které vypadají následovně:

„název-modelu,hodnota-vzdálenosti-od-modelu;"
(poslední řádek je bez **středníku**)

Kompletní sestava dat pro konkrétní model a jeho podobných modelů může vypadat následovně, viz obrázek 6.6.

6.3.2 Kontrolní panel a další komponenty

V tomto bodě jsem se rozhodl pro kontrolní panel navrhnout systém porovnání dvou modelů. V minulých kapitolách jsem definoval, k jakým funkcím

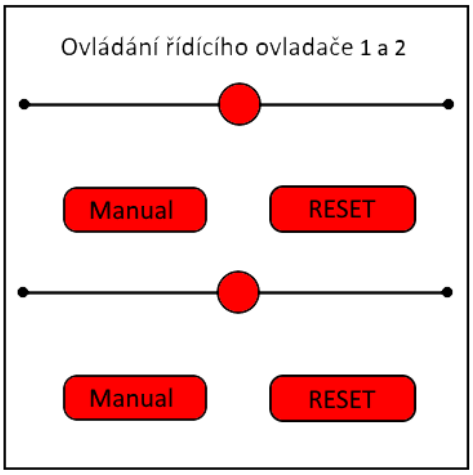
```
!cubical//121$1.3D.STX|14|33,29,12,17,6,11,0,6,0.583614218694931,
0.378829754039409,0.095686786488875,0.283142967550534,0.29182769247784|
cubical//121$1.3D.STX,0;
cubical//199988$6.3D.STX,0.482111636181243;
cubical//147346$11.3D.STX,0.50293585272444;
cubical//144$1.3D.STX,0.515726712833247;
rotational//161897$9.3D.STX,0.760009705131507;
rotational//140897$7.3D.STX,0.812971070385578;
rotational//402879$2.3D.STX,0.8983885045521;
plates//102$1.3D.STX,0.905896947404674;
rotational//402879$4.3D.STX,0.919335759738506;
plates//101737$1_item_3d-mod.stx,0.921753434932483;
plates//101754$1_item_3d-mod.stx,0.944215795612002;
rotational//402867$2.3D.STX,0.958999859971599;
rotational//402867$3.3D.STX,0.967391306798886;
plates//101705$1_item_3d-mod.stx,0.994968356204808
```

Obrázek 6.6: Úryvek ze souboru **SimilaritiesUE4.txt**

by uživatel měl mít přístup. Nyní tyto vlastnosti musím adekvátně upravit tak, aby mohl uživatel prostřednictvím kontrolního panelu pracovat s oběma modely současně, ale také aby každý z nich mohl dle potřeby odstranit z výběru a vybrat si k porovnání jiný. Dále by měl mít uživatel možnost si zobrazit u obou modelů jejich stavové vektory a další data. Panel bude opět rozdělen do segmentů. Mezi segmenty patří ovládání modelů a informace o modelech.

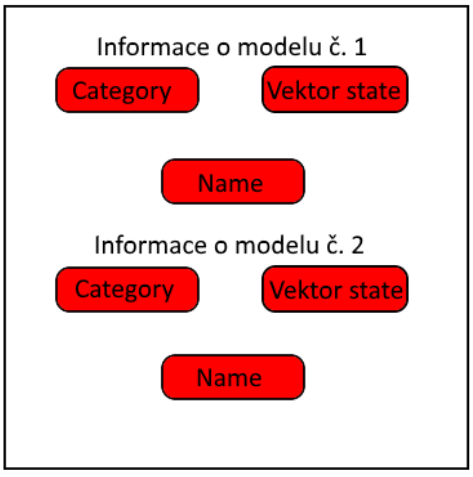
Část panelu zodpovídající za ovládání vybraných modelů a jejich odebrání bude na kontrolním panelu umístěna vlevo. Panel bude obsahovat možnost ručně či automaticky otáčet s každým vybraným objektem na podstavcích uprostřed místnosti. Pro přepínání mezi automatickým a ručním otáčením jednoho ze dvou objektů jsou určeny speciální dvě tlačítka, kde implicitní nastavení pro otáčení objektů bude automatické. Po jejich stisknutí se model zastaví a aktivuje na panelu řídící ovladač, se kterým následně uživatel bude moci pohybovat do stran a tím rotovat s objektem okolo své osy. Pokud uživatel bude chtít zvolit jiný objekt k porovnání, je k tomu připraveno speciální tlačítko, které bude pro každý vybraný model zvlášť. Po jeho stisknutí se model vrátí na své původní místo, kde si ho uživatel prvotně vybral. Nyní si uživatel opět může vybrat příslušným ovladačem nový model k porovnání. Celé schéma a rozložení všech ovládacích prvků lze najít na obrázku 6.7.

Druhá část kontrolního panelu obsahuje schopnost zobrazení konkrétních dat o vybraných modelech. Tato část se nachází na kontrolním panelu vpravo. Pro každý vybraný model lze kromě jejich vzájemného porovnání zobrazit pomocí kontrolního panelu: fts data modelu, název modelu a kategorii, do které byl podle algoritmu zařazen. Pro zobrazení fts dat jsem se rozhodl celý výpis hodnot zobrazit vedle konkrétního modelu. Tyto hodnoty jsou podle



Obrázek 6.7: Schéma panelu pro rotaci modelu a jeho zrušení výběru.

kategorie zobrazeny odshora dolů. Název modelu bude umístěn přímo nad samotným modelem zatímco informace o dominantní kategorii bude umístěna pod modelem. Všechna tato data budou směřovat k uživateli tak, aby mu byla stále k dispozici. Jinými slovy se data budou neustále natáčet za uživatelem podle pohybu po místnosti. Celé schéma a rozložení všech ovládacích prvků lze najít na obrázku 6.8 níže.



Obrázek 6.8: Schéma panelu pro zobrazení informací o modelu.

6.4 Projekt komprese modelů

U posledního projektu si analýzu rozdělím do dvou podsekcí. V první sekci stanovím, jak byly zpracovány přiložené modely a animaci(celkem dva modely a jedna animace) včetně načtení a zobrazení v místnosti. Druhá sekce rozebere vlastnosti, které dokáže uživatel pomocí kontrolního panelu upravit pro zobrazení všech modelů a animace v jednotlivých kompresních poměrech.

6.4.1 Komprimované modely

Pro předvedení konkrétního tématu modelů komprese jsem vyzískal od konzultanta projektu pro moji bakalářskou práci dva modely (plus dva osobní), včetně kompresního algoritmu pro přípravu komprimovaných verzí.
Pro představení algoritmu nyní musím vytvořit redukované verze všech modelů. Po testování schopností dodaného programu jsem se rozhodl stanovit 3 varianty plus základní. Každá z variant se bude lišit podle použití kompresního koeficientu uvnitř programu při aplikaci komprese na konkrétní vybraný model. Těmito koeficienty budou hodnoty: **0.001**, **0.01** a **0.1**. Po aplikaci komprese a následné dekomprese vznikne model ve formátu **.obj**, který následně použiji pro zobrazení v mé aplikaci.
Soubor OBJ je standardní textový formát pro ukládání 3D modelů, kde soubor je rozdělen do dvou částí. První část definuje body v prostoru, kde každý takový bod obsahuje x,y a z souřadnice v prostoru a přiřazení identifikačního čísla. Druhá část mapuje vytváření polygonů podle sekvencí identifikačních čísel bodů a z nich skládá kompletní síť finálního modelu. Níže přikládám ukázku pro obě tyto části.

6.4.2 Informace o komprimovaném modelu

Pro zobrazení dodatečných informací ke každému vygenerovanému modelu je nyní nutné navrhnout formát souboru, který bude obsahovat základní informace o modelu a typu komprese.
Pro každý model budu mít zahrnuto v konkrétním souboru jeho název a následně ke každé variantě komprese počet polygonů a vertexů, jaký koeficient byl použit v algoritmu při jeho kompresi, celkové snížení velikosti od původního modelu v procentech a v neposlední řadě také finální velikosti konkrétní verze modelu. Na tomto základě jsem navrhl tento systém ukládání informací do souboru, který je na obrázku 6.10. Data o každém modelu jsou rozdělena symbolem '|'. Symbol '!' rozděluje řetězec na dva díly, kde první díl definuje název modelu a druhý díl směs všech kompresních informací o modelu. Pokud rozdělíme druhou část podle znaku ';', získáme všechna data

```
# WaveFront format OBJ - Toto je znak komentare
# Seznam všech geometrických vektoru, kde (x,y,z[,w]) jsou souřadnice,
# w je volitelné a základná hodnota je nastavena na 1.0.
v 0.123 0.234 0.345 1.0
v ...
...
# Seznam všech texturových souřadnic, v (u, v [,w]) souřadnicích, tyto hodnoty
# se pohybují mezi 0 a 1, w je volitelné a základná hodnota je nastavena na 1.0.
vt 0.500 1 [0]
vt ...
...
# Seznam všech normálových vektorů v (x,y,z) formě
vn 0.707 0.000 0.707
vn ...
...
# Polygonální síť
f 1 2 3
f 3/1 4/2 5/3
f 6/4/1 3/5/3 7/6/5
f 7//1 8//2 9//3
f ...
...
```

Obrázek 6.9: Schéma souboru OBJ a jeho vlastností.

```
název modelu!
počet polygonů,počet vertexů,velikost souboru na disku,
kompresní hodnota algoritmu,procentuální snížení velikosti;
...;
...;
...;
název modelu2!
počet polygonů,počet vertexů,velikost souboru na disku,
kompresní hodnota algoritmu,procentuální snížení velikosti;
...;
...;
...;
```

Obrázek 6.10: Formát souboru pro uchování informací o konkrétním komprimovaném souboru.

pro konkrétní kompresní verzi modelu.
Na základě stanoveného formátu jsem sestavil soubor **CompressionData.txt**, ve kterém jsou umístěny všechna zmíněná dat. Pro konkrétní představu finální verze souboru je uveden obrázek 6.11

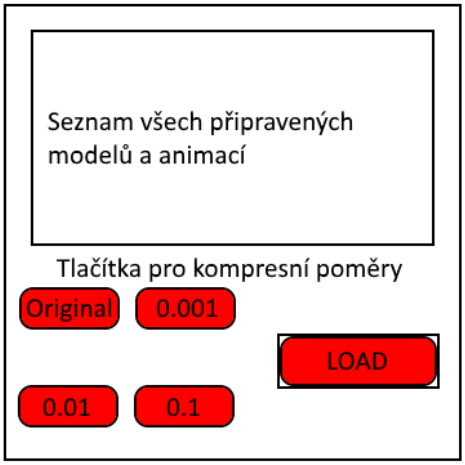
6.4.3 Kontrolní panel a další komponenty

Pro toto téma jsem se rozhodl upravit kontrolní panel tak, aby všechny funkce byly pouze pro jeden ze dvou reprezentovaných modelů. Panel bude obsluhovat přepínání instancí modelů na konkrétní platformě, kde se budou modely prezentovat. Mezi další schopnosti bude patřit kontrola rotace

```
WineBottle!
26496,72905,911 480 B,0,0;
26496,73166,17 684 B,0.001,98.06;
26494,68087,9 858 B,0.01,98.92;
25476,61764,7 916 B,0.1,99.14;|
Bimba!
17710,53120,595 007 B,0,0;
17710,53104,18 460 B,0.001,96.898;
17708,50246,9 009 B,0.01,98.49;
17708,47014,7 153 B,0.1,98.98;|
Skull!
80016,239922,2 910 436 B,0,0;
80016,239980,11 4716 B,0.001,96.06;
80016,236824,63 261 B,0.01,97.83;
79414,209981,31 757 B,0.1,98.81;|
Ogre!
24732,73423,839 549 B,0,0;
24732,73677,26 356 B,0.001,96.86;
24730,71278,14 217 B,0.01,98.31;
24596,66672,9 848 B,0.1,98.83;|
```

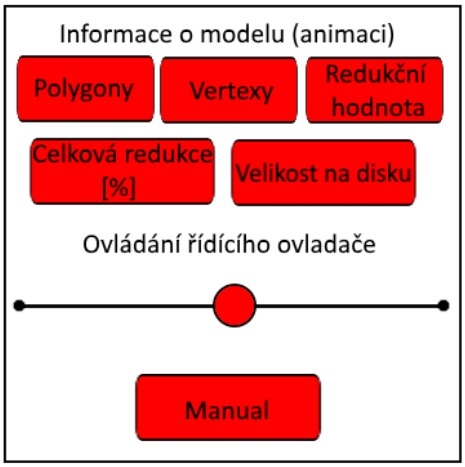
Obrázek 6.11: Soubor **CompressionData.txt**.

modelu ruční, či automatická a v neposlední řadě též zobrazení informací o konkrétním modelu (velikost modelu na disku, počet polygonů, konkrétní kompresní poměr atd.).
Pro načtení a zobrazení konkrétního modelu opět poslouží na levé části kontrolního panelu seznam modelů, mezi kterými si uživatel bude moci vybrat stisknutím konkrétního názvu modelu. Po vybrání se uživateli zvýrazní tlačítka, která symbolizují „zatrhávací položky výběru“. Uživatel si musí vybrat konkrétní kompresní poměr, který je napsán nad příslušným tlačítkem. Pokud jsou obě kritéria vybrána, uživateli se zvýrazní tlačítko „LOAD“, které po jeho stisknutí zobrazí konkrétně vybraný model. Pokud by si uživatel chtěl vybrat jiný model, nebo instanci konkrétního modelu, musí provést celou sadu operací znovu. Po novém výběru a načtení modelu se všechna nastavení kontrolního panelu týkající se rotace a zobrazení dat vrátí do základního nastavení. Schéma tohoto uspořádání kontrolního panelu lze nalézt na obrázku 6.12.
Pravá část kontrolního panelu je věnována zobrazení informací o konkrétní instanci modelu a ovládání rotace. Na panelu se nachází pět tlačítek, která po zapnutí, popřípadě vypnutí, zobrazí data konkrétního objektu. Mezi zobrazované informace patří: počet polygonů, počet vertexů, kompresní hodnota algoritmu, procentuální hodnota snížení velikosti modelu vůči původní verzi a velikost souboru na disku. Rotace modelu je v základu nastavena na automatické otáčení kolem svislé osy. Pokud by uživatel chtěl zapnout manuální otáčení, je na panelu připraveno tlačítko pro ruční ovládání pomocí připra-



Obrázek 6.12: Schéma seznamu všech modelů(animace) a kompresních poměrů.

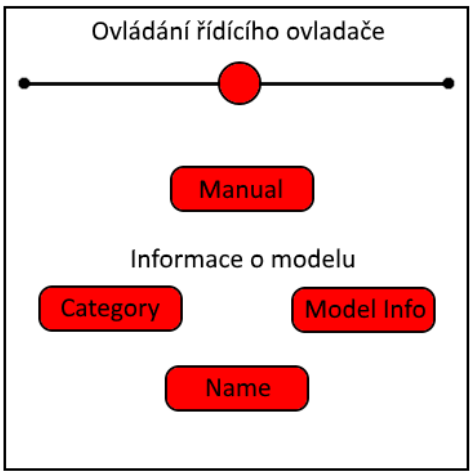
veného řídícího ovladače. Pokud by uživatel chtěl opět zapnout automatické otáčení modelu, stačí opět stisknout tlačítko pro ruční ovládání a tím se aktivuje zpětně automatická rotace jako v původním stavu. Schéma na obrázku 6.13 zobrazuje konkrétní část panelu.



Obrázek 6.13: Schéma panelu pro zobrazení informací o modelu a ovládání rotace.

6.5 Testování rozložení kontrolních panelů

Během vývoje aplikace jsem se rozhodl otestovat celou aplikaci za cílem vyzkoušení všech uživatelských ovládání, tedy přecházení mezi místnostmi a používání všech druhů panelů. Aplikaci jsem po jejím naprogramování nechal vyzkoušet svým kolegou a po důkladném ozkoušení všech možností uživatelského rozhraní měl pouze jednu výhradu k rozložení kontrolního panelu kategorizace modelů.
V kapitole 6.3.2 jsem nadefinoval rozloženi kontrolního panelu do dvou sekcí. První sekce sloužila pro rotaci vybraného objektu a jeho vrácení na původní pozici. Sekce druhá se pak zaměřila na zobrazení informací konkrétního modelu, nicméně každá ze sekcí byla určena pro oba modely naráz. Kolegovi toto řešení přišlo matoucí, a tak navrhl sjednotit všechny funkce pro jeden model do jedné sekce, kde budou funkce jak pro ovládání rotace a odstranění modelu z podstavce, ale také zobrazení příslušných informací. Finální rozložení lze nalézt na obrázku 6.14.



Obrázek 6.14: Schéma celého panelu kategorizace pro ovládání jednoho modelu.

7 Implementace projektů do Unreal Enginu

Nyní bych se vyjádřil k samotné implementaci všech komponent a projektů, které jsem v předchozích kapitolách podrobněji popsal a následně navrhl a připravil funkce, které bude každé demo projektu obsahovat. V dalším postupu si nyní rozdělím implementaci do šesti kategorií: „Základní místnost“, projekt enzym, projekt dekompozice modelů, projekt komprese modelů a jako poslední rozšířitelnost aplikace.
Vzhledem k tomu, že u každé kategorie je spousta podprogramů, které řídí různé části objektů, jsem se rozhodl detailněji popsat jen jednu místnost k předvedení, jak byla konkrétní místnost a její propriety vytvořeny. U ostatních předvedu jen významné procesy a záležitosti k pochopení konkrétního provedení. Ostatní podprogramy a nastavení lze následně nalézt přímo v projektu Unreal enginu, který bude přiložen na DVD. V této kapitole vynechám obrázky zdrojových kódů, které by zbytečně zabíraly mnoho místa a lze je najít opět v projektu Unreal enginu na DVD.

7.1 „Základní místnost“

Implementační část pro základní místnost si nyní rozdělím na dva postupné kroky. První část rozebírá složení všech modelů a vytvoření statického modelu místnosti. Ve druhé fázi uvedu procedury a jejich použití v základní místnosti.

7.1.1 Statický model a nastavení

„Základní místnost“ se skládá ze dvou vrstev. První vrstvu místnosti jsem vytvořil s pomocí Unreal Enginu, kdy bylo nutné sestavit okolní prostředí mimo konkrétní místnost, pro zachování věrného zasazení. Aby mohla základní místnost vůbec existovat, je nutno vytvořit podlahu, která se bude chovat jako zemský povrch, po kterém se bude moci uživatel pohybovat. Ohraničení, kam až uživatel dohlédne, je omezeno modelem místnosti UC335. Nad všemi úrovněmi včetně základní místnosti je nastaveno osvětlení, které má simulovat denní světlo. Všechny zmíněné součásti obaluje tzv. „sky box“. Sky box je druh simulace nebeské oblohy většinou reprezentované koulí nebo krychlí, která je z vnitřní strany namapována texturou.

Modely, které jsou zde použity: Místnost UC335, dva stoly, zařízení pro simulaci terénu nazvané „pískoviště“, model vstupních dveří, kancelářská židle a helmy. Všechny modely lze nalézt ve složce **CustomModels**.
Všechny místnosti, včetně té základní, jsou umisťovány vedle sebe do řady, aby bylo možné pomocí konfiguračního souboru **projects.txt** přidávat další místnosti s novými tématy. Textury pro modely jsem použil některé vlastní, ale také u některých modelů (kupříkladu stěn) i textury, které nabízel Unreal Engine v základním balíčku zadarmo.
„Základní místnost“ je počátečním umístěním ve VR celé aplikace, kdy po spuštění mé aplikace se do ní uživatel přemístí. Procedury, které jsou zde použity se dále využívají i v jiných projektech. Hned z počátku bych chtěl zmínit, že se budu pouze zabývat velmi obecně jak je naprogramováno snímání ovladačů a záznam pohybu uživatele. Unreal Engine při vytváření projektu pro VR vytvořil základní schéma řízení pro ovladače a také pro snímání pohybu v prostoru. Díky získanému základu kontroly ovládání pak bylo mnohem snazší upravit či přidat nové prvky tak, aby vyhovovaly konkrétním požadavkům. Všechny ostatní Blueprinty a podprocedury týkající se základní místnosti a vytváření dalších místností budou probrány detailněji.

7.1.2 Reprezentace charakteru a ovladačů

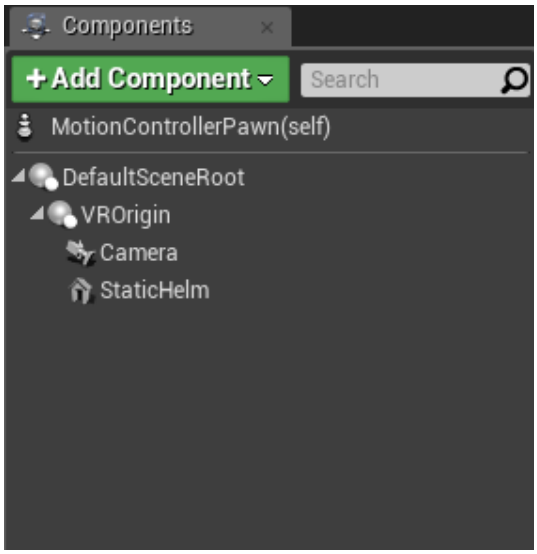
Mezi základní Blueprinty, které zpracovávají zdroje ze snímačů ovladačů a „headsetu“ patří MotionControllerPawn, BP-Motion-Controller a BP-PickupCube. Všechny tyto Blueprinty byly vytvořeny při zakládání projektu a výběrem nastavení projektu „Virtual Reality“ z možných typů zaměření.

Blueprint MotionControllerPawn

Tento Blueprint je potomkem od třídy „Pawn“, která symbolizuje ovladatelnou kameru, se kterou lze manipulovat a udávat jí pohybové příkazy. Tato „třída“ slouží pro vytvoření objektu reprezentujícího uživatele a obsluhu vstupů od ovladačů, které následně zpracuje a provede korespondující operaci. Blueprint provádí tři hlavní procesy:

- Nastavení výšky uživatele ve VR a registrace ovladačů,
- uchopení předmětů,
- teleportace po místnosti.

Na obrázku 7.1 je zobrazena hierarchie všech aktuálních komponent, ze kterých se Blueprint skládá.



Obrázek 7.1: Schéma všech komponent Blueprintu MotionControllerPawn.

Pro pochopení, co každá komponenta představuje, je teď stručně vysvětlím. Během práce přibudou další, které vysvětlím až konkrétním místě užití. Komponenty **DefaultSceneRoot a VROrigin** jsou tzv. **Scene Components**. Jedná se o typ komponenty, která kategorizuje ostatní přidružené do jednoho balíku, aby se s nimi lépe pracovalo. Nutno dodat, že pro každý Blueprint je vždy definovaný kořenový element, v tomto případě je to DefaultSceneRoot. Další komponentou je **Camera Component**, v našem případě s názvem Camera. Tato komponenta slouží k vytvoření kamery, kterou následně používáme pro simulaci očí uživatele ve virtuálním prostředí. Na obrázku 7.1 je vidět, že tato komponenta je umístěna hierarchicky pod scénou VROrigin, jinými slovy pokud budeme provádět se scénou VROrigin nějaké transformace, všechny členy, které do této skupiny patří provedou stejnou transformaci. Poslední objekt je **Static Mesh Component**(StaticHelm). Komponenta zprostředkovává zobrazení instance statického meshe, což je geometrický model, který se skládá ze statické množiny polygonů.
Bližší popis procesů:

1. **Nastavení výšky uživatele a registrace uživatelů** - podprogram, který na základě připojeného HMD (Head Mouted Display, tedy například PSVR, Oculus Rift a HTC Vive) přidruží typ snímání pozice brýlí s herním enginem a následně provede umístění kamery do prostoru. Druhá část algoritmu provede vytvoření instancí Blueprintu BP-Motion-Controller, která zobrazí pravou (pravý ovladač) a levou (levý ovladač) ruku. Left a Right Controller jsou proměnné do kterých jsou uloženy instance ovladačů a ty jsou následně přidruženy scéně

VROrigin.

2. **Uchopení/Upuštění předmětů** - tento proces provádí uchopení objektů, které musí být potomku typu „Actor“, tedy mají implementovány rozhraní „Pickup Actor Interface“. Pokud je rozhraní implementováno, může se uživatel přiblížit s ovladačem (virtuální rukou) ke konkrétnímu objektu a stisknutím s podržením „Trigger button“ dojde k připojení objektu k ovladači (ruce). Po uvolnění tlačítka dojde k upuštění předmětu. Co se bude dít konkrétně po upuštění předmětu si už definuje předmět sám. Tento proces tedy určuje, zdali bude předmět uchopen či upuštěn. Příkladem takového předmětu je kupříkladu BP-Pickup-Cube nebo HelmetVisor, kde oba tyto Blueprinty budou vysvětleny později.

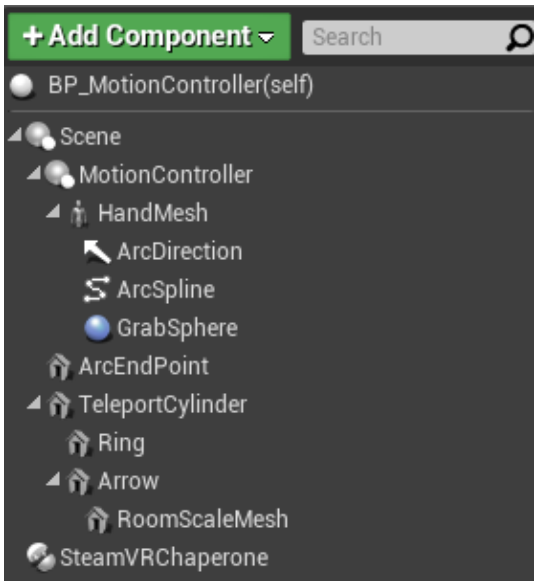
3. **Teleportace po herní ploše** - poslední z podprogramů je možnost teleportovat se za pomocí stisknutí dotykového tlačítka na ovladači a určení konkrétního cílového místa v aplikaci. Pro účely mé aplikace jsem ale tuto možnost kompletně celou deaktivoval, jelikož místnost a pohyb v ní jsem nastavil tak, že teleportace není zapotřebí.

Blueprint MotionControllerPawn jsem dále rozšířil o nové procedury, které budou postupně probrány ve zbývajících kapitolách implementace.

Blueprint BP-Motion-Controller

Blueprint BP-Motion-Controller je potomkem třídy „Actor“, která slouží pro reprezentaci ovladače (virtuální ruky) v aplikaci. Tento Blueprint má za úkol získávat přímé vstupy od ovladače přidruženému této „třídě“ a tyto změny zobrazit ve virtuálním prostředí. Dále zprostředkovává přidružení a „oddružení“ předmětu k ovladači (virtuální ruce). Poslední velkou částí je metoda obsluhující stisknutí dotykového tlačítka pro umožnění teleportace po místnosti. Tato „třída“ umožňuje ještě mnoho dalších funkcí, ale jak jsem již zmiňoval o pár odstavců výše, nebudu se jimi dále zabývat. V dalším kroku nastíním rozložení komponent tohoto Blueprintu.

Na obrázku 7.2 máme konkrétní schéma komponent. Blueprint je rozdělen do tří podkategorií. První kategorie **MotionController** slouží k reprezentaci virtuální ruky. Jako součásti skupiny přibyly nové komponenty. **Skeletal Mesh Component**, v našem případě HandMesh, slouží pro reprezentaci ruky, kde „skeletal“ znamená, že se jedná o druh animovaného statického meshe, který má definovanou vnitřní kostru se kterou lze aktivně manipulovat. Komponenta **Arrow Component**, u nás ArcDirection, funguje jako jednoduchá čára, která primárně slouží k určení toho, jakým směrem je objekt



Obrázek 7.2: Komponenty Blueprintu BP-Motion-Controller.

natočen. **Spline Component** v Blueprintu ArcSpline je druh zápisu cesty pro definování a použití pozičních dat. Komponenta **Sphere Collision**, u nás GrabSphere, slouží pro identifikaci kolize s jinými objekty. Druhá skupina **TeleportCylinder** se skládá z prvků, které „třída“ používá k teleportaci uživatele po herní ploše. Komponenta **SteamVRChaperone** slouží k získání vstupů ze SteamVR. SteamVR je služba od společnosti Valve nutná k chodu „headsetu“ HTC Vive.

Blueprint BP-Pickup-Cube

Blueprint BP-Pickup-Cube určuje, jak má vypadat předmět, u kterého když vytvoříme instanci z BP=Pickup-Cube, bude ho moci uživatel zvednout či položit. Třída není použitá v žádné scéně, ale je zde zmíněna hlavně proto, že na jejím základě jsou odvozené jiné Blueprinty, které se již v aplikaci nacházejí. V této kapitole rozepíši co je nutné nastavit a jaké skripty je nutné vytvořit a nastavit k úspěšné realizaci uchopitelného předmětu v rámci popisované „třídy“.

1. **Aktivace rozhraní Pickup Actor** - u Blueprintu je nutné v sekci **„Class Settings“** přidat rozhraní **Pickup Actor Interface**.

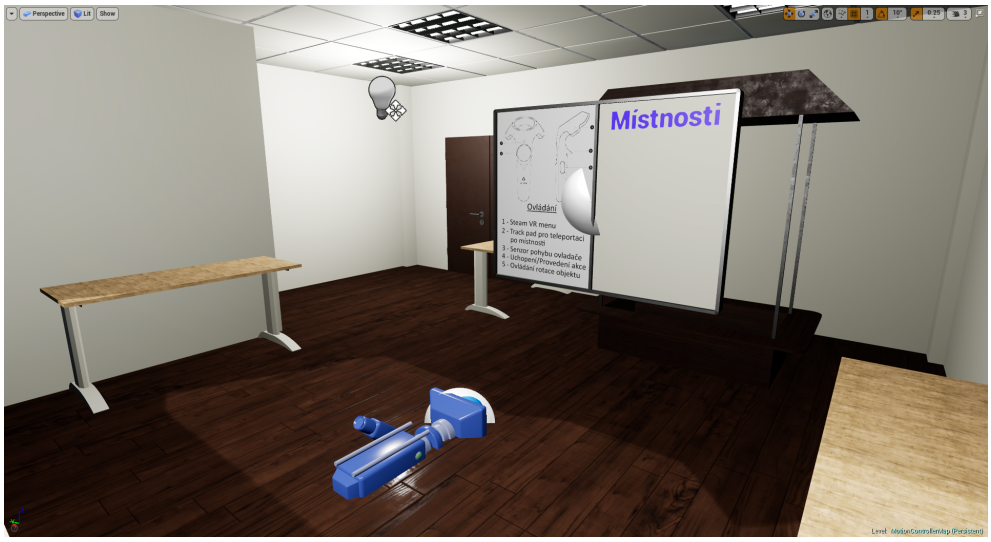
2. **Nastavení vlastností statického meshe** - dalším krokem je po přidání statického meshe, symbolizujícího konkrétní model předmětu, nastavit několik vlastností. V panelu detailů statického meshe musíme nastavit **Enable Gravity** na *true*, v kategorii Collision/ **Simulation**

Generates Hit Events na *true* a **Generate Overlap Events** na *true*.

3. **Vytvoření skriptů** - posledním krokem je napsat skript, který na základě události Pickup nebo Drop provede přidružení nebo odloučení předmětu.

7.1.3 Úvodní výuková lekce

Po spuštění aplikace se uživatel ocitne uprostřed základní místnosti. Po jeho levici (popřípadě pravici vzhledem k natočení uživatele při spuštění) bude levitovat nad zemí tabulka, která je rozdělena do dvou částí.
Levá půlka tabulky nabízí popis všech ovládacích prvků, které může uživatel používat k přesouvání se po místnosti. Pravá část tabulky informuje o všech místnostech, které momentálně moje práce obsahuje. Data, která tabulka zobrazí jsou název projektu a korespondující barva helmy.
Pro zobrazení informací jsem vytvořil Blueprint, který obsahuje všechny



Obrázek 7.3: Obrázek místnosti a umístění tabulky s informacemi o místnostech.

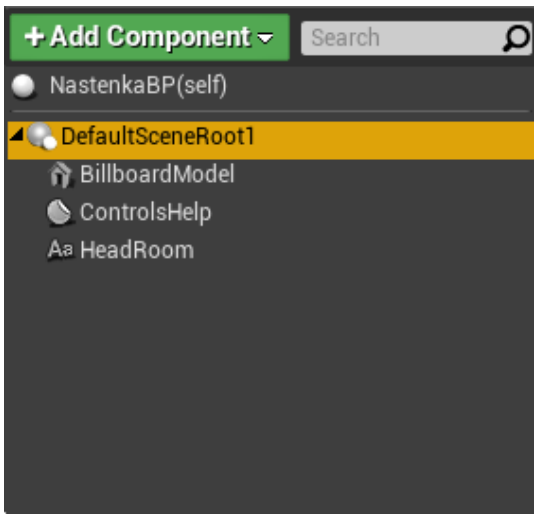
modely nutné ke sestavení tabulky, ale také skripty, které budou zprostředkovávat zobrazení informací o aplikaci a rozložení místností. Prvním krokem je umístit model nástěnky, reprezentující tuto tabulku, do prostoru. Tabulka je vytvořená externě a následně importována enginem. Jelikož levá část tabulky se nebude měnit, rozhodl jsem se do ní umístit obrázek, který ukazuje nápovědu a vysvětlení, co která tlačítka na ovladači znamenají. Pro druhou část tabulky již bylo nutné napsat proceduru, která z přiloženého souboru

projects.txt přečte seznam všech projektů, jejich názvy a ty zobrazí v pravé části. Následuje výpis konkrétního souboru:

Enzym!
Category!
Compression!

Po načtení všech těchto informací pak dále algoritmus vytvoří helmy podle počtu projektů, které budou umístěny na jeden ze stolů v základní místnosti. Finální částí funkce je tato data přenést na pravou část tabulky. To se provede za použití Unreal komponenty **Text Renderer**, tato komponenta vykresluje 2D texty v prostoru, kterému lze přiřadit velikost, font, barvu a další vlastnosti písma. Všechny názvy místností budou vypsány do seznamu odshora dolů, kde barva textu symbolizuje příslušnou helmu místnosti. Na obrázku 7.4 je složení všech komponent a jejich hierarchie Blueprintu NastenkaBP: [17]

Na obrázku přibyla nová komponenta **Decal Component**, u nás **Cont-**



Obrázek 7.4: Komponenty Blueprintu NastenkaBP.

rolsHelp. Tento druh komponenty pracuje jako typ materiálu na povrchu statického meshe (stručně řečeno, jedná se o nálepku na model). Nyní rozeberu každou část algoritmu nutnou k načtení místností a helem odděleně:

1. **Načtení všech místností a vytvoření helem v základní místnosti** - Algoritmus nejdříve načte soubor a rozdělí data do pole podle oddělovacího znaku '!'. Následně si uloží do proměnné **Count Rooms** podle velikosti načteného pole, kolik místností musí vytvořit. Dalším

krokem je provést programovou smyčku podle počtu místností a přidat do základní místnosti na předem připravený stůl helmy, které budou sloužit k přesunu mezi místnostmi.

2. **Načtení názvu místností pro vytvoření seznamu v tabulce** - Tento podprogram se spustí ve chvíli, kdy aplikace běží a existuje tedy instance tohoto Blueprintu ve scéně. Po zahájení podprogramu se načtou data se souboru **projects.txt** a názvy projektů se přemístí do přiřazeného pole. Velikost pole je uložena v lokální proměnné **Count Rooms**. Poslední část algoritmu prochází cyklus, ve kterém vytvoří vždy **Text Render Component** a přiřadí mu podle indexu příslušný text a barvu reprezentující konkrétní helmu.

3. **Kompletní instrukce** - všechny předchozí podprogramy jsou spuštěny přes Blueprint MotionControllerPawn po spuštění aplikace, kdy je volána událost **Event Beginplay**, která spustí příslušné procedury.

7.1.4 Teleportační zařízení mezi projekty

Pro přesunutí se mezi načtenými projekty z kapitoly 7.1.3 je připraven systém teleportace. Cílem této procedury je rozmístit na předpřipraveném stole helmy podle toho, kolik je momentálně načteno projektů a následně po nasazení helmy provést přenesení uživatele do konkrétního projektu. Každá helma je barevně odlišena, aby se uživatel zorientoval, která helma patří ke kterému projektu.
Statický model, který jsem potřeboval k vytvoření tohoto systému, je model helmy. Tento model jsem se rozhodl získat od externího autora. Konkrétní model helmy jsem obdržel na serveru zabývajícím se distribucí 3D modelů mezi autory a zákazníky, *www.turbosquid.com*. Helmu, kterou jsem si objednal lze používat v jakémkoli projektu bez žádných poplatků či dalších omezení.

Blueprint HelmetVisor

HelmetVisor slouží jako uchopitelný předmět pro uživatele. Pokud uživatel uchopí tuto helmu a přiblíží její model do dostatečné blízkosti své hlavy (v enginu je hlava reprezentována kamerou), přenese se do příslušné místnosti. HelmetVisor vychází z Blueprintu BP-Pickup-Cube, kde musí být podle kapitoly 7.1.2 nastavené nutné parametry. Blueprint HelmetVisor obsahuje tyto vlastnosti a metody:

1. **Atributy Blueprintu** - mezi základní atributy Blueprintu patří: idRoom, initLocation, isPickedUp, deletePreviousLevel, countDown. *idRoom* definuje id místnosti, do které se má přesunout. *initLocation* označuje globální (world) pozici helmy ve scéně pro uchování počáteční pozice helmy, pokud uživatel už nechce držet helmu a rozhodl se jinak. *isPickedUp* říká, jestli je konkrétní helma držena uživatelem či ne. *deletePreviousLevel* představuje informaci, jestli je nutné provést operaci odstranění místnosti ze scény (vysvětleno později). Poslední *countDown* definuje odpočet, za jak dlouho se má provést odstranění místnosti ze scény.

2. **Test nasazení helmy** - test nasazení helmy je typ algoritmu, který probíhá neustále pokud je helma držena uživatelem. Pokud uživatel přiblíží helmu dostatečně blízko své hlavy dojde k průchodu tohoto testu a spustí další část popsanou níže.

3. **Teleportace uživatele, vytvoření konkrétní místnosti a helem** - principem algoritmu je přenést uživatele na konkrétní místo v rámci herní plochy včetně vytvoření místnosti a příslušných helem v této místnosti, aby se uživatel mohl dále přesouvat mezi ostatními projekty.

   Úkolem první části algoritmu je uživatele přenést do konkrétní místnosti, to je zajištěno atributem idRoom, který říká, jakou místnost chceme navštívit po nasazení helmy. Po přemístění uživatele algoritmem do nové místnosti dojde k odstranění všech předmětů týkajících se starého projektu a místnosti, vyjma základní. Následně proběhne načtení nutných předmětů a vytvoření nové místnosti. Na dynamickému načítání místností a všech objektů jsem musel přistoupit z důvodu udržení stabilní snímkové frekvence. Při konstrukci Blueprintu symbolizujícího strop jsem narazil na optimalizační problém, v momentu kdy jsem chtěl umístit pět světelných zdrojů na svůj model stropu do připravených míst. Při této implementaci nastavení osvětlení jsem naměřil pouze 40 FPS. Po různých změnách nastavení a počtu zdrojů jsem nakonec došel k závěru, že pokud budou zobrazeny všechny místnosti a objekty projektů, cílové hranice 90 FPS se nepřiblížím. Proto jsem navrhl dynamickou formu načítání místností a objektů projektu. Po této aplikaci jsem uskutečnil umístění pouze tří světelných zdrojů na strop pro osvětlení místnosti, které postačily k osvětlení scény.

4. **Odstranění staré místnosti, helem a objektů projektu** - Posledním zásadním algoritmem je systém odstranění místnosti, ve kteréz zse uživatel před teleportací nacházel a všech jejich součástí. Procedura se spustí pouze tehdy, pokud je konkrétní atribut deletePreviousLevel nastaven na true a dojde k odpočtu countDown na stanovenou hodnotu. Pokud dojde ke splnění obou podmínek, proběhne smazání a nastaví se deletePreviousLevel na false.

Blueprint AddableRoomBP

Tento Blueprint slouží k sloučení modelů místnosti UC 335, vstupních dveří, stolu pro helmy a stropu. Strop je reprezentován Blueprintem CeilingBP, kde je statický mesh stropu plus tři zdroje světla. AddableRoomBP obsahuje pouze jednu funkci pro přidávání instancí Blueprintů HelmetVisor, které jsou umisťovány na stůl.
Algoritmus přidání helmy vypočte na základě umístění místnosti a počtu helem konkrétní místo pro novou helmu. Následně z předaných parametrů ve funkci nastaví barvu helmy a přiřadí helmu do seznamu helem místnosti.

Podprogramy v MotionControllerPawn

Do tohoto Blueprintu byly přidány funkce pro řešení teleportace uživatele: Spawn Room, Despawn Room with Helmets a Teleporting to Room.

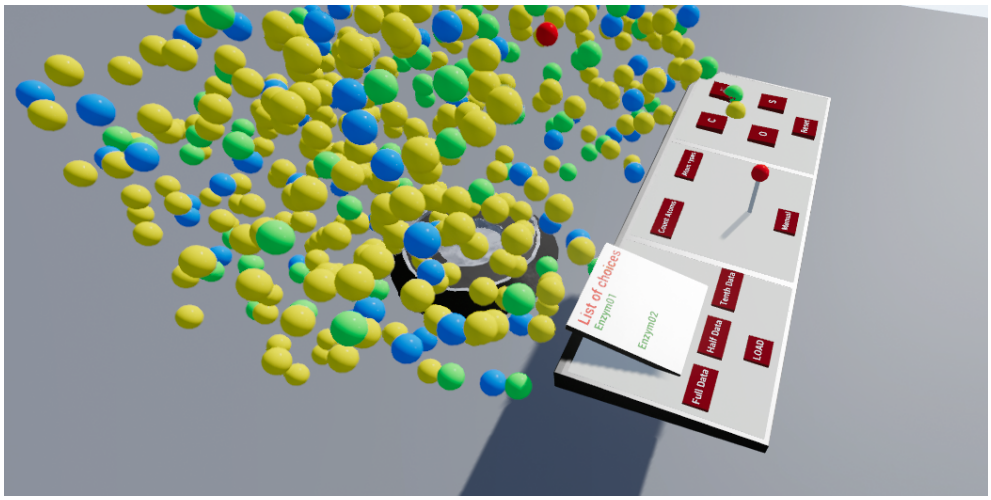
1. **Spawn Room** - na základě předaného parametru definujícího id místnosti provede vytvoření instance Blueprintu **AddableRoomBP**. Poté podle počtu načtených všech místností přidá do nově vygenerované místnosti helmy s příslušnými indexy místností a barev.

2. **Despawn Room with Helmets** - algoritmus provede na základě atributu **Added Room**, který byl nastaven při první teleportaci mimo základní místnost, odstranění všech helem z místnosti. Na konci celého procesu odstraní místnost samotnou a nastaví addedRoom na novou místnost do které se momentálně uživatel přesunul.

3. **Teleporting to Room** - procedura provede na základě předaného parametru identifikace místnosti výpočet, na jakou pozici se má uživatel přesunout. Následně provede přesun kamery do konkrétní místnosti. Poslední části procedury je odstranění objektů z místnosti, ze které se přesouváme a načtení objektů z projektu, do kterého se právě teleportujeme.

7.2 Projekt enzym

Pro vytvoření projektu enzym bylo zapotřebí připravit sadu funkcí, které provedou načtení dat z konkrétního datového souboru a na jejich základě se vygenerují atomy. Po zobrazení určitého enzymu lze u přiloženého kontrolního panelu obsluhovat ovládání rotace enzymu, zobrazení dodatečných informací a filtrace konkrétních atomů. V souvislosti s kontrolním panelem je nutné také zmínit jeho příslušné Blueprinty: **ButtonBP**, **ControlHandleRotationBP** a **ChoosableListBP**, které jsou umístěny na kontrolním panelu. Obrázek 7.5 zachycuje vzezření místnosti.



Obrázek 7.5: Obrázek místnosti s enzymem a kontrolním panelem.

7.2.1 Statické modely

Modely, které byly použity pro tento typ projektu jsou: Podstavec pro enzym a deska kontrolního panelu. Oba tyto modely jsem vytvořil v programu SketchUp a importoval do enginu (nacházejí se ve složce **CustomModels** v projektu Unreal enginu). Pro vykreslení atomů, z kterých se skládá konkrétní enzym je použit model koule (Sphere Static Mesh) přímo ze základního balíčku enginu.

7.2.2 Načtení dat a vytvoření modelu enzymu

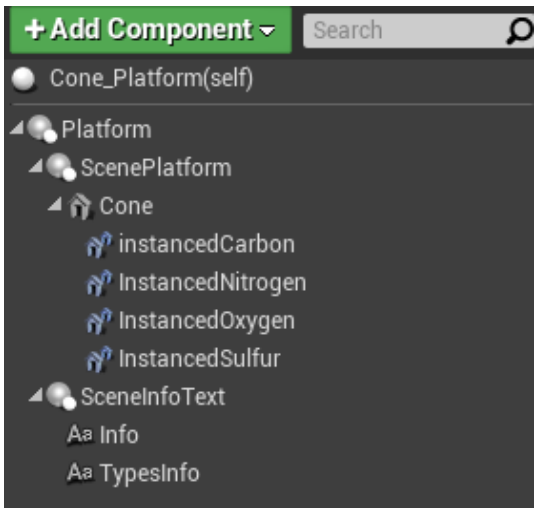
Při spuštění aplikace se provede načtení souboru s konkrétními daty enzymu. V kapitole 6.2 lze získat představu, jak vypadá originální soubor s daty enzymu. Na základě představených optimalizací jsem napsal program v jazyce Java, který zkrátí a upraví data (algoritmus bude přiložen také na

DVD). Data následně předá proceduře, která provede vytvoření všech atomů a umístí je na předem vytvořenou platformu v prostoru. K ovládání enzymu slouží připravený kontrolní panel umístěný naproti enzymu viz. obrázek 7.5.

Blueprint Cone_Platform

Blueprint Cone_Platform představuje konkrétní platformu, nad kterou se enzym umístí. Platforma provádí rotaci enzymu podle osy Z (rotace kolem geometrického středu) a upravuje rotaci zobrazovaného textu tak, aby byl neustále směrem k uživateli přístupný. Na obrázku 7.6 je schéma komponent této „třídy“.

Blueprint je rozdělen mezi scény ScenePlatform a SceneInfoText na dva



Obrázek 7.6: Komponenty Cone_Platform.

díly, kde první zprostředkovává zobrazení enzymu a druhý zařizuje zobrazení informací o enzymu. Ve scéně ScenePlatform přibyla nová komponenta **Instanced Static Mesh Component** (v našem případě kupříkladu **instancedCarbon**). Jedná se o komponentu, která se chová jako statický mesh s tím rozdílem, že vytváří instance zvoleného meshe optimalizované pro snížení nároků na operační paměť. Tímto krokem ale ztrácíme možnost změnit vlastnosti konkrétní instance, proto je nutné všechny její vlastnosti nastavit předem. V mém případě se jednalo pouze o dvě nastavení: Nastavit konkrétní texturu a zrušit renderování stínů. Z tohoto důvodu jsem vytvořil čtyři Instanced Static Mesh komponenty.

Blueprint dále obsahuje osm funkcí plus **Event graph** (základní rozcestník Blueprintu, který definuje, jak se má na které podněty z ovladačů či systému enginu zachovat), ale popíši tu pouze tří důležité a Event graph vlastní.

1. **addAtoms** - funkce AddAtoms provede z předaného parametru názvu souboru načtení kompletního souboru a rozdělení řetězců podle znaku ';' a uloží je do pole. Dalším krokem je aplikace cyklu, který projde všechny prvky načteného pole a rozdělí příslušné informace na konkrétní elementy. Výsledné hodnoty následně uloží do vybraných proměnných: **EnzymDataX, EnzymDataY, EnzymDataZ, Elements**. S každým načteným elementem tímto způsobem je prováděna úprava výpočtu geometrického středu. Po načtení všech atomů ze souboru do příslušných proměnných se učiní finální výpočet geometrického středu a uloží se do proměnných **ConeCenterAvgX a ConeCenterAvgY**. Na závěr všechny proměnné připraví jako návratovou hodnotu.

2. **makeAtomComponents** - funkce na svém počátku provede inicializaci všech nutných lokálních proměnných. Následně se aplikuje cyklus, který projde všechny atomy, přičemž na počátku definuje o jaký druh atomu se jedná. Po identifikaci elementu přidá instanci statického meshe na příslušné místo z předaných dat funkce addAtoms. S každým vytvořeným atomem přiřadí do lokálního pole typ přidaného atomu. Po vytvoření všech instancí se provede i nastavení všech druhů informací a přiřadí se do Text Renderer komponentů Blueprintu (Info a TypesInfo).

3. **unloadAllAtomComponents** - funkce provede vyčištění všech instancí každé z komponent. Tato funkce probíhá před načtením nového enzymu, pokud již byl nějaký načten.

4. **Event graph** - zde probíhá každý „tik enginu“ (při každé realizaci vykreslení scény) rotace enzymu (pokud je platný atributu **isTurning**) a rotace všech textů momentálně zobrazených ve scéně.

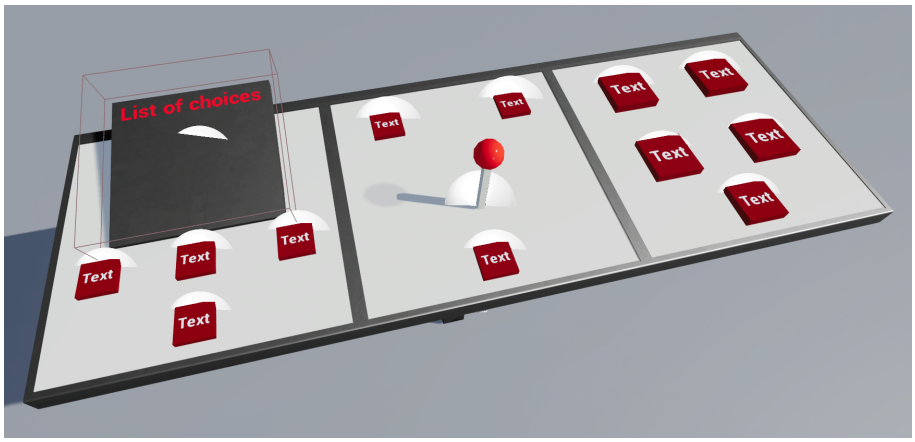
Nová funkce v Blueprintu MotionControllerPawn

Do tohoto Blueprintu přibyla nová funkce, která má za účel provést vytvoření Cone_Platform ve scéně, následně provést načtení, vytvoření enzymu a zahájit rotaci enzymu ve scéně.

7.2.3 Kontrolní panel a interakce

Druhou částí projektu je zprostředkovat uživateli možnost pracovat nad konkrétním modelem enzymu, popřípadě přejít na jiný model. Kontrolní panel

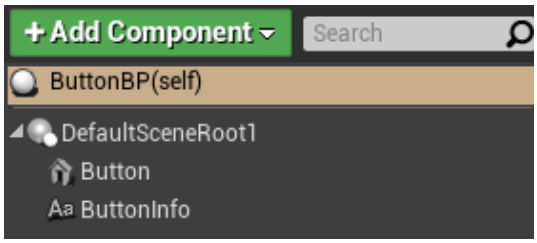
který jsem naprogramoval se skládá ze tří částí: Načtení enzymu, zobrazení dodatečných informací včetně ovládání rotace a poslední filtrace enzymu podle typů atomů. Pro pochopení **Blueprintu ControlPlatformEnzymBP** je nyní nutné nejprve představit Blueprinty, ze kterých se skládá. Převážně se jedná o systém interakce s uživatelem ve VR.



Obrázek 7.7: Kontrolní panel pro Enzym.

Blueprint ButtonBP

Blueprint **ButtonBP** je „třída“, která slouží k reprezentaci tlačítka. Blueprint hlídá, jestli nedošlo k interakci s ovladačem (virtuální rukou), popřípadě jestli hráč tlačítko neaktivoval. Ostatní funkce již mají za účel vykreslit animaci při aktivaci tlačítka uživatele. Na obrázku 7.8 je zobrazeno schéma komponent, kde můžeme vidět, že tlačítko se skládá pouze ze statického meshe vytvářející samotný model a textu pro identifikaci, o jaké tlačítko se jedná. Nyní uvedu základní funkci a schéma Event graphu:



Obrázek 7.8: Schéma komponent pro ButtonBP.

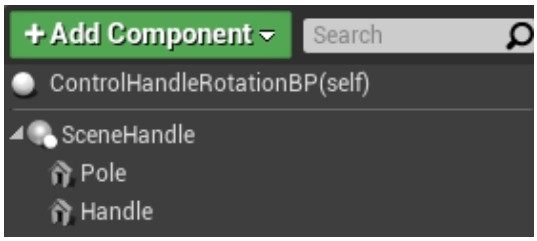
1. **pushedButton** - tato funkce zařídí stisknutí tlačítka a aktivování animace, za předpokladu, že animace je už ukončena, pokud není, stisk-

nutí tlačítka se neprovede. Z atributu **isActivated** pak následně můžeme přečíst, jestli je tlačítko aktivní či není.

2. **Event graph** - v této sekci probíhají tři hlavní části. Zaprvé při spuštění aplikace proběhne událost **BeginPlay**, která spustí inicializační část pro nastavení atributů „třídy“. Události **ActorBeginOverlap a ActorEndOverlap** detekují, jestli došlo k interakci s ovladačem (virtuální rukou). Pokud ano, dojde k navázání konkrétního ovladače s tlačítkem a umožní se aktivování tlačítka ovladačem. Pokud ovladač opustí hranice tlačítka, proběhne deaktivace možného stisknutí ovladačem a zrušení navázání s ovladačem. Tento systém jsem navrhl pro zajištění interakce pouze jednoho ovladače, kde by jinak mohlo docházet k tomu, že při vniknutí druhého ovladače a jeho opětovném opuštění by první ovladač ztratil přístup k aktivování tlačítka. Poslední událost je **EventTick**, která testuje, zdali se nemá provést animace tlačítka. Pokud ano, provede se transformace tlačítka podél osy Z a změní se jeho textura podle toho, jestli tlačítko bylo aktivováno nebo deaktivováno.

Blueprint ControlHandleBP

ControlHandleBP představuje řídící páku, která je v dalších projektech primárně určena k manuální rotaci přidruženého modelu. Řídící páka se otáčí pouze v jedné rovině. Po jejím uchopení uživatel pohyby ve směru otáčení provádí příslušné natočení. Pro použití k jiným účelům, než je natočení jsem navrhl systém, který převede původní stav natočení a nynější na hodnotu od -80 do 80, kdy základní stav představuje 0. Jinými slovy, pokud uživatel pohne řídící pákou do jedné ze stran, tato konkrétní hodnota bude nabývat mezi (-80, 80). Schéma komponent, ze kterých se konkrétní Blueprint skládá jsou pouze dva statické meshe, kde jeden zaujímá formu tyče a druhý tvar koule, která je nasazena na tyči.
Blueprint obsahuje dvě důležité funkce a Event graph, které nyní popíši:



Obrázek 7.9: Schéma komponent pro ControlHandleBP.

1. **initialiseRotation** - funkce, které jsou předány parametry ovladače, konkrétně kolizní komponenty (Sphere Collision) a specifikaci ovladače. Na základě ovladače nastaví atribut **isHeld**, který určuje jaký ovladač právě drží konkrétní řídící páku. Uloží si základní pozici páky a její rotaci plus nastaví inicializaci na false.

2. **setHandleRotation** - procedura provede výpočet vzdálenosti od počáteční hodnoty umístění páky (získané z funkce initialiseRotation) pro konkrétní ovladač. Na základě získané vzdálenosti upraví natočení páky a to následně přenastaví. Pokud by vzdálenost byla větší než 80 nebo menší než -80 stupňů, dojde k omezení nastavení otočení na tyto hraniční podmínky, tedy maximální možný úhel je (-80,80). Dále pak lokální proměnná **precisionCoeficient** definuje, jak rychle se bude páka otáčet. Poslední částí procedury je uložení konkrétního stavu natočení do proměnné **handleRotation**.
   Důležitým poznatkem pro tento druh problému je stanovit si, na kterých osách se bude měřit vzdálenost. To definuje hodnota atributu **xOriented**, které říká, jestli se bude počítat x-ová nebo y-ová složka vzdálenosti. Dále také záleží na tom, jestli je páka orientována k uživateli čelem nebo zády. Atribut **invertedOrientation** tento stav definuje. Posledním faktorem je druh natočení, které má řídící páka provádět, jestli v ose x či y. To můžeme definovat pomocí atributu **rollRotation**, který udává, jestli jde o rotaci podle x nebo y.

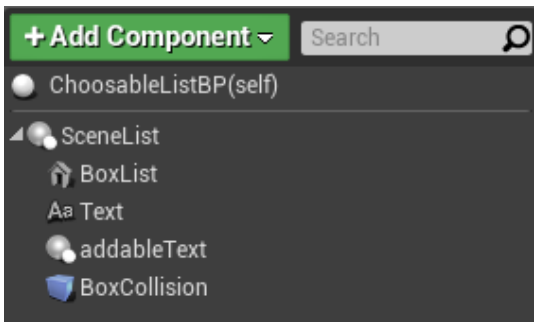
3. **Event graph** - tato sekce je velmi podobná jako v případě Blueprintu ButtonBP, kde opět podle událostí **ActorBeginOverlap a ActorEndOverlap** bude provedeno přidružení ovladače nebo zrušení navázaní a posléze nastavení aktivace či deaktivace řídící páky. Další část pak zaujímá samotné spuštění ovládání řídící páky, kdy pokud uživatel stiskne a drží akční tlačítko ovladače (Trigger Buton) dojde k uchopení řídící páky. Následně do doby zrušení uchopení páky bude uživatel držet akční tlačítko a provádět změnu natočení páky.

Blueprint ChoosableListBP

ChoosableListBP je seznam, se kterým lze interagovat pomocí ovladače, kde po kliknutí akčním tlačítkem ovladače na konkrétní řádek seznamu se text zvýrazní. Tento text pak následně vnitřně Blueprint uloží do atributu, ke kterému potom jiné třídy mohou přistupovat. Tento druh komponenty jsem použil při nahrávání nových druhů modelů či jejich verzí, kupříkladu u enzymu, kde bylo nutné vypsat uživateli konkrétní možné druhy enzymů,

ze kterých si může vybrat. Pro vybrání konkrétního druhu enzymu se změní index řádku seznamu a na jeho základě se identifikuje jaký enzym se má načíst do projektu.

Schéma komponent, ze kterých se Blueprint skládá jsou: Model desky, na



Obrázek 7.10: Schéma komponent pro ChoosableListBP.

které je konkrétní seznam posléze zobrazen, nadpis seznamu, komponenta **BoxCollision** a scéna primárně určená pro další texty. Komponenta BoxCollision slouží podobně jako Sphere Collision pro detekce kolize s objekty, s tím rozdílem, že se jedná o krychli, která detekuje interakci s ovladači. Pro tento Blueprint jsem se rozhodl popsat detailněji dvě funkce a Event graph:

1. **addTexts** - tato funkce na základě předaného pole řetězců vytvoří do scény pro **addableText** (Text Render Component) a přiřadí mu příslušný text a barvu. Dále ke konkrétnímu Text Render Componentu přiřadí svůj vlastní Box Collision Component pro detekci kolize textu a ovladače. Je důležité zmínit, že Box Collision Component ve větším množství může zapříčinit značné zpomalení aplikace. Z tohoto důvodu jsem se omezoval jak počtem textů v konkrétním seznamu, tak i v rámci globálního počtu instancí tohoto Blueprintu. Další z důležitých informací je, že algoritmus na základě velikosti pole vypočte nutný rozměr písma a mezery mezi samotnými prvky seznamu, aby zabíraly co největší plochu desky, nad kterou jsou zobrazeny. Je nutné si uvědomit, že s přibývajícím počtem prvků v seznamu dochází k zmenšování písma, tím pádem se bude zhoršovat i detekce samotného textu, jelikož mezi jednotlivými texty nebude taková mezera.

2. **testOverlapingTexts** - procedura provede načtení všech komponent, které obsahuje scéna **addableText**. Následně vnitřní programový cyklus provede pro všechny komponenty proces: Pokud je komponenta typu Text Renderer uloží si jí do lokálního pole, jestliže se jedná o

komponentu typu Box Collision provede se test, zdali je konkrétní aktivovaný ovladač uvnitř Box Collision. Při splnění, dojde k vracení všech zvýrazněných textů do původní pozice a zvýrazní se nový text a procedura se ukončí s návratovou hodnotou true k identifikaci, že došlo ke změně. V opačném případě se cyklus opakuje do té doby, až proces dokončí kontrolu všech Box Collision komponent a po dosažení konce cyklu je pak návratová hodnota false. Zvýraznění prvku spočívá pouze v transformaci umístění Textu o několik jednotek vzhůru od tabulky.

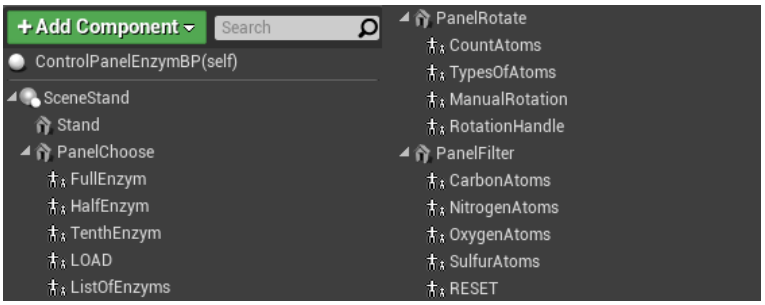
3. **Event graph** - Event graph pro tento Blueprint opět obsahuje část pro detekci ovladače a jeho přiřazení či uvolnění ovladače při vniknutí či opuštění prostoru Box Collision. Dále obsahuje událost **BeginPlay**, ve které po zahájení aplikace provede načtení všech textů ze seznamu. Seznam je možné předat externě z enginu, jelikož je atribut **allTexts** nastaven public. To umožňuje upravit seznam přímo v editoru scény, tedy přidat či ubrat nové prvky seznamu. Druhou možností je tuto část kódu upravit nebo vytvořit funkci, která předá proceduře addTexts nový parametr s novým seznamem za předpokladu, že ten starý bude smazán. Poslední částí Event graphu je potom kontrola, jestli hráč nestiskl akční tlačítko ovladače (Trigger button) a následně provést funkci testOverlapingText.

Blueprint ControlPlatformEnzymBP

Blueprint ControlPlatformEnzymBP slouží k vytvoření kompletního panelu pro projekt enzym. Součástí tohoto panelu jsou Blueprinty ButtonBP, ControlHandleBP a ChoosableListBP. Společně se statickými meshi tvoří kostru, se kterou následně může uživatel interagovat. Blueprinty výše zmíněné vyžadují externí implementaci, jelikož jejich vnitřní procedury slouží pouze pro aktivaci či deaktivaci komponenty. Proto ControlPlatformEnzymBP každému z tlačítek, řídících ovladačů a seznamu přiřazuje i jejich vlastnosti a kontroluje jejich stavy.

Schéma panelu je rozděleno do tří částí, jak již bylo naznačeno v kapitole 7.2.3. Uvnitř schématu nám přibylo mnoho nových komponent typu **Child Actor Component**. Jedná se o druh instance Actora. Tento druh komponenty slouží jako univerzální druh objektu, který na základě nastavení příslušného Blueprintu převezme na sebe jeho podobu, vlastnosti a schopnosti. Kupříkladu **CountAtoms** je instance Blueprintu ButtonBP, která je umístěna pod panelem **PanelRotate**.

Pro tento Blueprint jsem se rozhodl upřesnit Event graph. Pro jeho hlavní skript nebudu moci bohužel předvést detailní popis kódu z důvodu nad-



Obrázek 7.11: Schéma komponent pro ControlPlatformEnzymBP.

měrné rozsáhlosti celého skriptu. Proto se zaměřím pouze na vysvětlení kódu obecně a jak jsou konkrétní procedury spuštěny na základě interakce uživatele. Event graph je rozdělen do tří pasáží, které probíhají každý „tik enginu“. Tyto části jsou: Detekce provedení akce nad konkrétním tlačítkem/řídící ovladačem/seznamem a provedení příslušné operace, rotace podstavce enzymu a načtení nového enzymu ze vstupů od konzole.

1. **Aktivace komponent a provedení konkrétních instrukcí** - proces nejdříve identifikuje, zda-li uživatel použil akční tlačítko ovladače (levého či pravého). Pokud bylo aktivováno, musí nalézt jakou komponentu chce uživatel aktivovat a použít. To se provede kontrolou, zda-li komponenta na kontrolním panelu nekoliduje s aktivním ovladačem. Jestliže takovou komponentu nenašel, žádnou akci neprovádí a proces je ukončen. Pokud ale komponenta byla nalezena, dojde k nastavení animace tlačítka a provede se příslušná operace spjatá s tímto tlačítkem. Po vykonání je proces ukončen a čeká se na další „tik enginu“.

2. **Rotace podstavce enzymu** - pokud bylo aktivováno tlačítko vázané k zajištění manuální rotace podstavce, bude tento proces aktivní. Z **ChoosableList** komponenty, která se ve schématu komponent jmenuje RotationHandle, získáme hodnotu **HandleRotation**, která bude upřesňovat platformě enzymu rotaci okolo své osy. Pro zajištění většího pokrytí úhlů rotace (HandleRotation pokrývá pouze -80 až 80 stupňů) je tato hodnota zdvojnásobena.

3. **Načtení nového enzymu** - pokud je stisknuto tlačítko „LOAD“ dojde k nastavení atributu **prepLoadNew**, které má za úkol v tomto třetím bodě spustit načtení nového enzymu. Načtení nového enzymu zavolá nejprve funkci pro smazání konkrétního enzymu. Následně provede načtení enzymu, který byl nastaven pomocí tlačítek na kontrolním panelu, ze souboru a umístí jej nad platformu. Po načtení dojde k re-

setování všech tlačítek, které byly aktivovány včetně zvoleného prvku ze seznamu enzymů. **ListofEnzyms**.
Je vhodné zmínit náročnost konkrétní procedury načtení nového enzymu. Pokud si uživatel zvolí plnou verzi zobrazení enzymu, bude počet načítaných atomů v řádech tisíců a to už je náročné načíst a vytvořit, kde během zpracování se engine jeví ve stavu „zamrznutí“. Implementovat jsem zkoušel různé druhy nahrazení statických meshů a načítání dat včetně variantních nastavení zobrazení atomů, kde se ve výsledku toto konkrétní řešení projevilo jako nejvhodnější. Pokud si tedy uživatel zvolí enzym s 5000 atomy, může očekávat přibližnou délku načtení enzymu okolo 15 vteřin.

7.3 Projekt dekompozice modelů

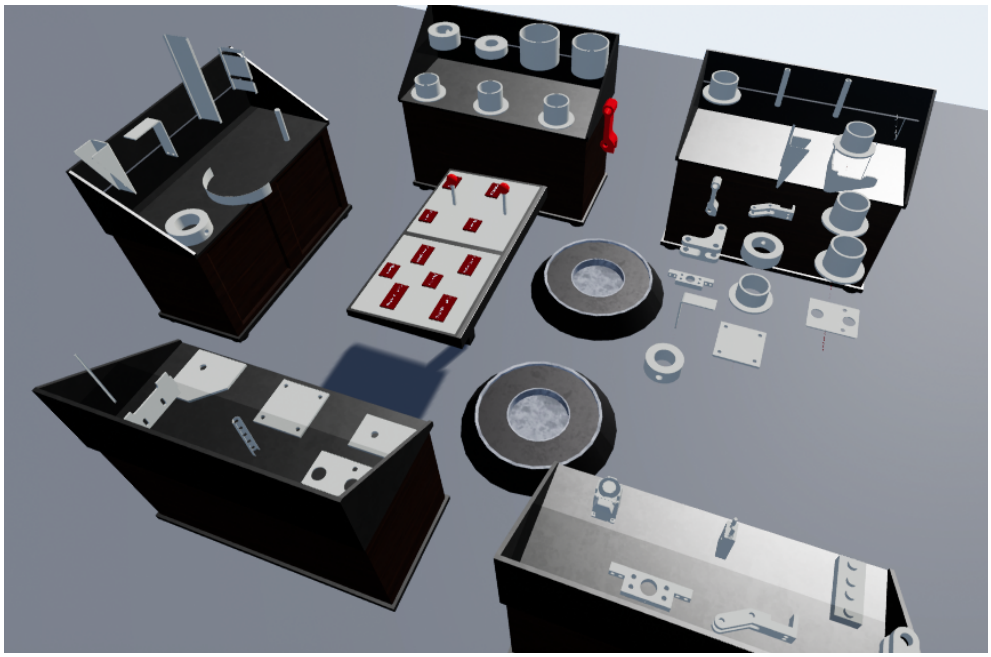
Pro projekt zabývající se dekompozicí modelů bylo zapotřebí naprogramovat systém, který připraví všechny druhy modelů a zobrazí je na pultech, které budou umístěny okolo uživatele. Dále je nutné implementovat interakci uživatele mezi těmito modely, aby je bylo možné přenést na konkrétní platformu a nadále s nimi pracovat. Posledním prvkem je implementace kontrolního panelu, kde lze nad konkrétním modelem získat řadu informací, či si ho lépe prohlédnout a také zrušit jeho výběr pro další porovnávání, pokud to uživatel požaduje.

7.3.1 Statické modely

Modely, které zde byly použity mi předal pan doc. Ing. Petr Vaněček, Ph.D. Modely byly importovány do těla projektu v Unreal enginu a je možné je použít při výběru komponenty statického meshe. Z důvodu rozsáhlosti modelů nejsou k práci přiloženy. Mezi další modely patří skříň pro umístění vlastních modelů pro dekompozici. Skříň jsem vytvořil v externím programu *SketchUp* a importoval do enginu.

7.3.2 Načtení a rozmístění modelů

Po přenesení se teleportací na tento typ projektu se nejdříve musí načíst konkrétní modely. Pro každý model je také nutné přiřadit jeho výslednou zprávu a všechny informace o podobných modelech, které jsou uloženy v souboru **SimilaritiesUE4.txt**, přičemž jeho formát byl popsán v kapitole 6.3.1. Po načtení těchto dat se následně musí vytvořit příslušné modely do scény projektu. To provedeme za pomoci Blueprintu **ShowShelf2**, který



Obrázek 7.12: Obrázek místnosti s modely kategorizace a kontrolním panelem.

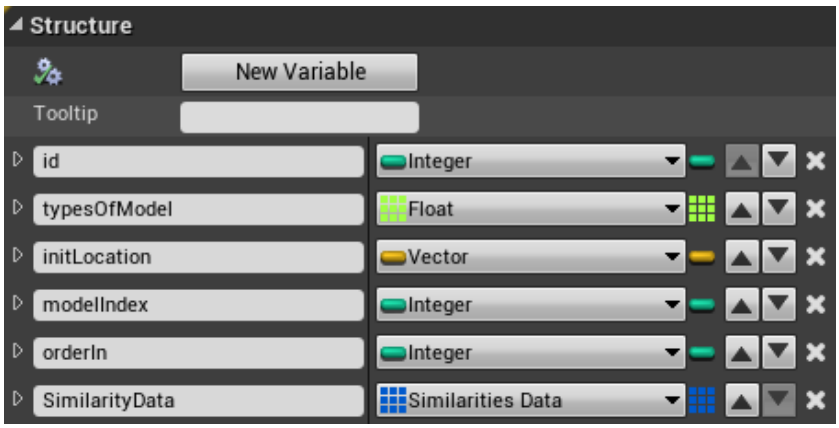
vytvoří model skříně a na ní umístí určitý počet modelů. Dále je nutné vytvořit platformu nad kterou budou modely zobrazeny. K tomu využijeme Blueprint **Platform_Category**.

Blueprint ShowShelf2

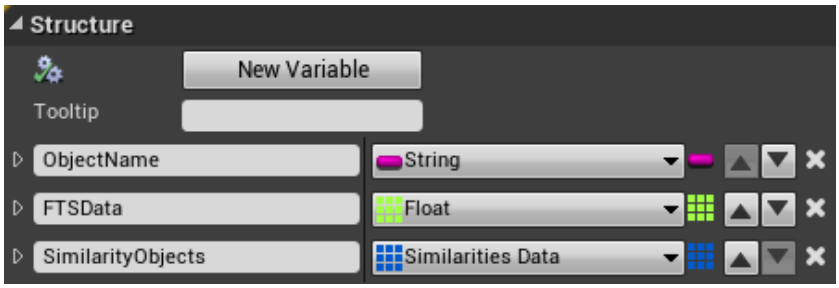
Blueprint ShowShelf2 slouží pro vytvoření skříně, na které se následně z jejich vnitřních funkcí vytvoří příslušné modely. Každá taková skříň má dvě patra, na která lze umístit určitý počet modelů. Mezi další důležité funkce patří nalezení a příprava podobných modelů a jejich dodatečných informací. Schéma komponent, ze kterých se skládá konkrétní Blueprint je pouze základní scéna a model skříně uložený ve statickém meshi. Není potřeba zde přidávat obrázek.

1. **addItems** - tato funkce provede na základě předaných parametrů, kterými jsou globální index modelů, maximální počet modelů na skříň a fts data včetně informací o podobných modelech ve struktuře **ObjectData**. Následně proběhne výpočet, zjišťující kolik na každé patro připadne modelů a jaké vzdálenosti budou mít mezi sebou. Dalším krokem je podle počtu modelů vytvořit statický mesh a přiřadit konkrétní model pomocí předaného parametrů. Poslední částí je do struktury **RandomItemTypes** přiřadit globální id modelu, fts data a informace

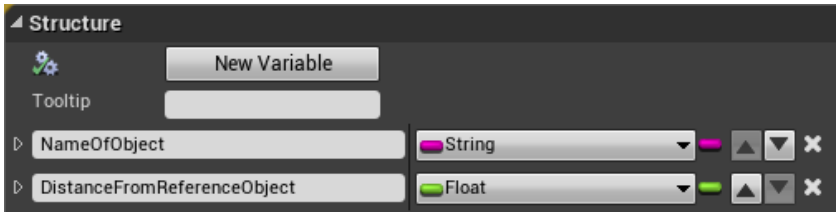
o podobných modelech. Struktury RandomItemTypes, ObjectData a SimilarityData slouží jako přepravky, které jsem vytvořil pro potřeby tohoto Blueprintu. Komponenty uvnitř struktur jsou na obrázku 7.13, 7.14 a 7.15.



Obrázek 7.13: Přepravka RandomItemTypes.



Obrázek 7.14: Přepravka ObjectData.



Obrázek 7.15: Přepravka SimilarityData.

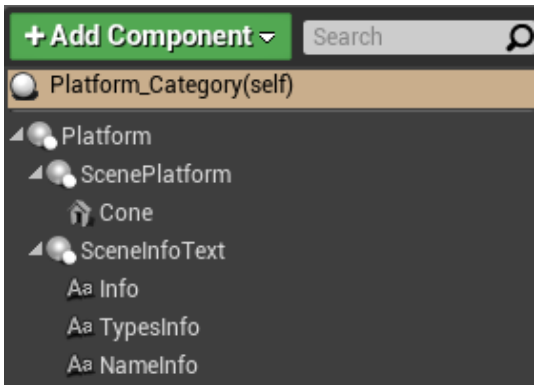
2. **findDominantCategory** - procedura původně sloužila v raných fázích vývoje jako funkce, která na základě předaného indexu modelu u konkrétní skříně nalezla dominantní prvek z fts dat pro vybraný model a dále provedla vytvoření informací o tomto modelu pro další využití

v jiných Blueprintech či funkcích. Nyní tato funkce momentálně slouží pro přípravu fts dat k zobrazení uživateli. Mezi informace patří název modelu, samotná fts data a do jaké skupiny algoritmus zařadil konkrétní model.

3. **highlightModels2** - procedura požaduje parametry, mezi které patří index pozice modelu a instanci konkrétní platformy, a na jejich základě přenese konkrétní model na příslušný podstavec, změní jeho barvu a nastaví u skříně, že tento model je již zvýrazněn.

Blueprint Platform_Category

Tento konkrétní Blueprint slouží primárně jako místo k přenesení vybraného modelu a také pro zobrazení informací tohoto modelu. Dále umožňuje rotaci modelu okolo své vlastní osy. Schéma komponent je na obrázku 7.16.



Obrázek 7.16: Schéma Blueprintu Platform_Category.

1. **setTypesInfoText** - připraví řetězec a naplní ho daty z hodnot vektoru dekompozice a následně je uloží do textu **TypesInfo** komponenty.

2. **Event graph** - Event graph slouží pro tento Blueprint pouze k rotaci objektu nebo rotaci zobrazeného textu. K oběma rotacím dochází každý „tik enginu“ (Event Tick).

Nové funkce v Blueprintu MotionControllerPawn

Pro načítání a vytvoření modelů byly v tomto Blueprintu přidány další funkce: **loadCategory**, **unloadCategory** a **loadAllShelfs**. Load a unload slouží pro dynamické načítání projektu kategorizace včetně všech propriet.

1. **loadCategory** - funkce spustí načtení a vytvoření všech skříní a příslušných modelů ze souboru. Následně provede vytvoření dvou instancí Platform_Category a u každé z nich provede nastavení zarovnání textu.

2. **unloadCategory** - funkce provede odstranění všech skříní včetně jejich modelů umístěných na nich a také odstraní přilehlé platformy.

3. **loadCategoryObjects** - načtení souboru **SimilaritiesUE4.txt** podle formátu souboru z kapitoly 6.3.1. Výsledná data jsou pak uložena ve struktuře ObjectData pro využití funkce loadAllShelfs.

4. **loadAllShelfs** - procedura, která provede podle počtu načtených modelů z funkce loadCategoryObjects vytvoření skříní. Po jejím vytvoření se upraví data z pole tak, aby mohla být předána funkci addItems z Blueprintu ShowShelf2 pro vytvoření modelů konkrétní skříně. Po vytvoření všech skříní a modelů je proces ukončen.

7.3.3 Interakce uživatele s modely

Interakce uživatele s modely umístěnými na skříních je implementována tak, že pokud se bude ovladač nacházet uvnitř konkrétního modelu a bude stisknuto akční tlačítko ovladače, přenese se konkrétní model na platformu. Dále podle toho, jaký ovladač byl aktivován při stisknutí (levý či pravý) se rozhodne, jestli se má model přemístit na levou či pravou platformu.

Nové funkce v Blueprintu MotionControllerPawn

Mezi nové funkce pro tento Blueprint přibyly **getItemsOverlapLeft/Right**, **highlightLeft/RightItem**, **spawnSimilarityItems/despawnSimilarityItems** a **resetLeft/RightItem**. Pro demonstraci předvedu pouze funkce pro levou ruku, jelikož pro pravou ruku (ovladač) jsou funkce stejné, ale používají se jiné proměnné.

1. **getItemsOverlapLeftItem** - procedura provede pro každou vytvořenou skříň kontrolu, jestli pro nějaký konkrétní model zvolené skříně uživatel spustil akční tlačítko společně s kolizí ovladače a testovaného modelu. Pokud k této situaci došlo, proces předá index skříně a index modelu v konkrétní skříní a proces se ukončí. V opačném případě oznámí návratovou hodnotou nenalezení kolize s akčním tlačítkem ovladače.

2. **highlightLeftItem** - tato funkce má za účel přemístit konkrétní vybraný model levým ovladačem na určitou platformu v místnosti. První částí algoritmu je provedení kontroly vstupních parametrů. Pokud proběhla validace správně, proběhne detekce, zda-li již už není nad pravou platformou vybraný model. Pokud ano, provede se přemístění vybraného modelu na levou platformu a zvýrazněné modely od předchozího modelu (pravého), který byl vybrán již dříve, se nyní vrátí na své původní místo. Pokud ještě nebyl vybrán pravý model, proběhne přemístění konkrétního modelu na levou platformu a zobrazí se za platformou všechny modely, které jsou mu podobné na základě dat z načteného souboru **SimilaritiesUE4.txt**, které vygeneruje funkce spawnSimilarityItems.

3. **spawnSimilarityItems** - provedení vykreslení podobných modelů, které jsou funkci předané parametrem na vstupu včetně i konkrétních informací, jako absolutní vzdálenost od porovnávaného modelu, kde je tato hodnota zobrazena pod každým modelem. Čím je hodnota nižší, tím je konkrétní model blíže porovnávanému modelu. Funkce despawnSimilarityItems zaručuje pouze odstranění vykreslení podobných modelů ze scény.

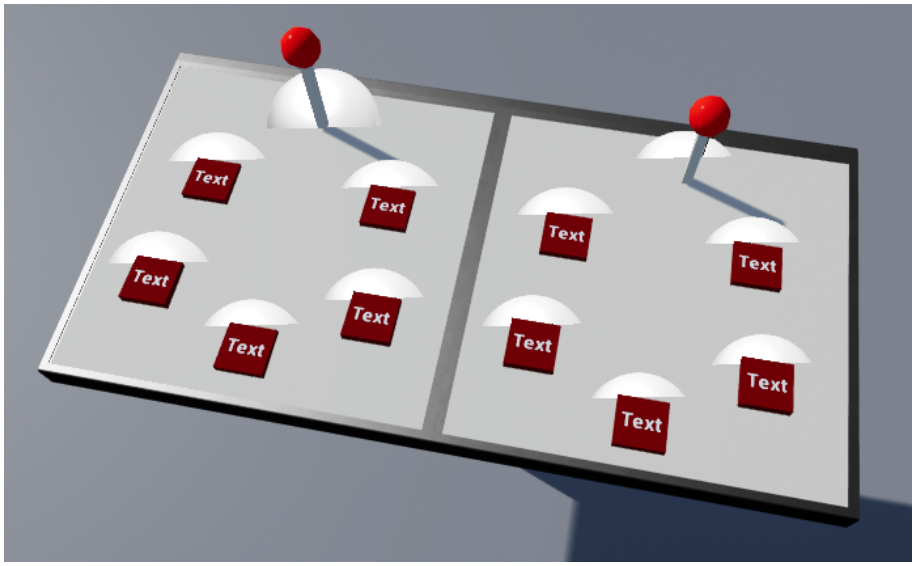
4. **resetLeftItem** - tato funkce má za úkol vrátit na původní pozici vybraný model na levé platformě. Pokud na pravé platformě je stále vybraný model, je nutné provést opětovné vykreslení všech podobných modelů. Pokud již byl pravý model přemístěn na své původní místo, stačí pouze přemístit levý model.

5. **upravení skriptu v Event graphu** - v tomto schématu skriptů bylo nutné mírně doplnit procedury zodpovídající za stisk akčního tlačítka, kdy po jeho aktivaci se musí provést detekce kolize pomocí funkce getItemsOverlap a pokud tato funkce navrátí hodnotu true, pokračuje dále spuštěním funkce highlightItem.

7.3.4 Kontrolní panel a interakce

Kontrolní panel byl původně rozdělen na dvě části, první část se skládala z ovládání rotace každého z modelů, které byly přemístěny na podstavec, a možnost vrácení modelu zpět na své původní místo. Druhá část pak definovala jaké konkrétní informace o vybraném modelu si chceme nechat zobrazit(informaci o dominantní kategorie, název modelu atd.). Po otestování kolegou mi bylo doporučeno toto schéma předělat. Kde každá sekce kont-

rolního panelu bude sloužit pro konkrétní vybraný model, viz. kapitola 6.5.



Obrázek 7.17: Kontrolní panel projektu kategorizace modelů po její úpravě na základě testování uživatelského rozhraní.

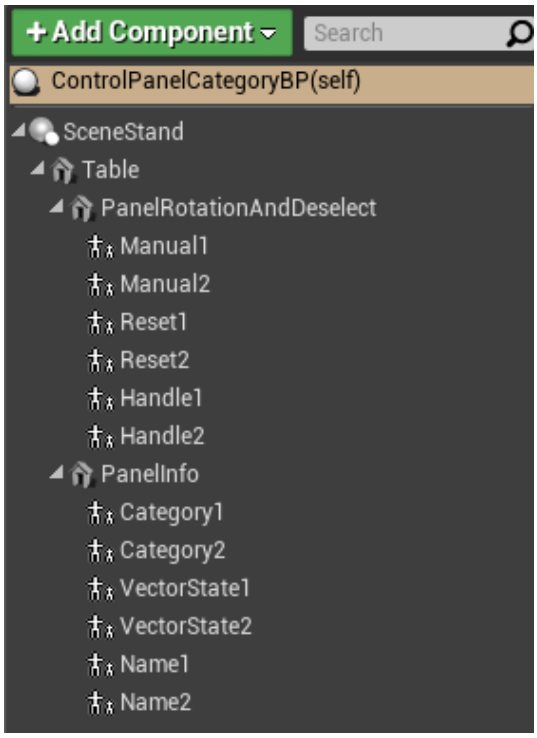
Blueprint ControlPanelCategoryBP

Blueprint **ControlPanelCategoryBP** slouží pro seskupení všech potřebných komponent a funkcí k vytvoření funkčního kontrolního panelu. Kontrolní panel obsahuje Blueprinty ButtonBP a ControlHandleRotationBP. Zbylé komponenty slouží k vytvoření modelu kontrolního panelu.

Koncept Blueprintu je velmi podobný ControlPanelEnzymBP z kapitoly 7.2.3. Z tohoto důvodu jej nebudu dále podrobně rozebírat a pro případné detaily lze nahlédnout do zmíněného Blueprintu nebo nahlédnout do kapitoly 6.3.2 pro ujasnění implementovaných schopností.

7.4 Projekt komprese modelů

Součástí projektu komprese modelů jsou v příslušné místnosti připravené dvě platformy. Platformy slouží pro umístění vybraných modelů k porovnání, které si vybral uživatel. K platformám uvnitř místnosti přísluší také dva kontrolní panely, jeden pro každou platformu zvlášť, k načtení nového modelu, zobrazení dodatečných informací a v neposlední řadě k ovládání rotace modelu.



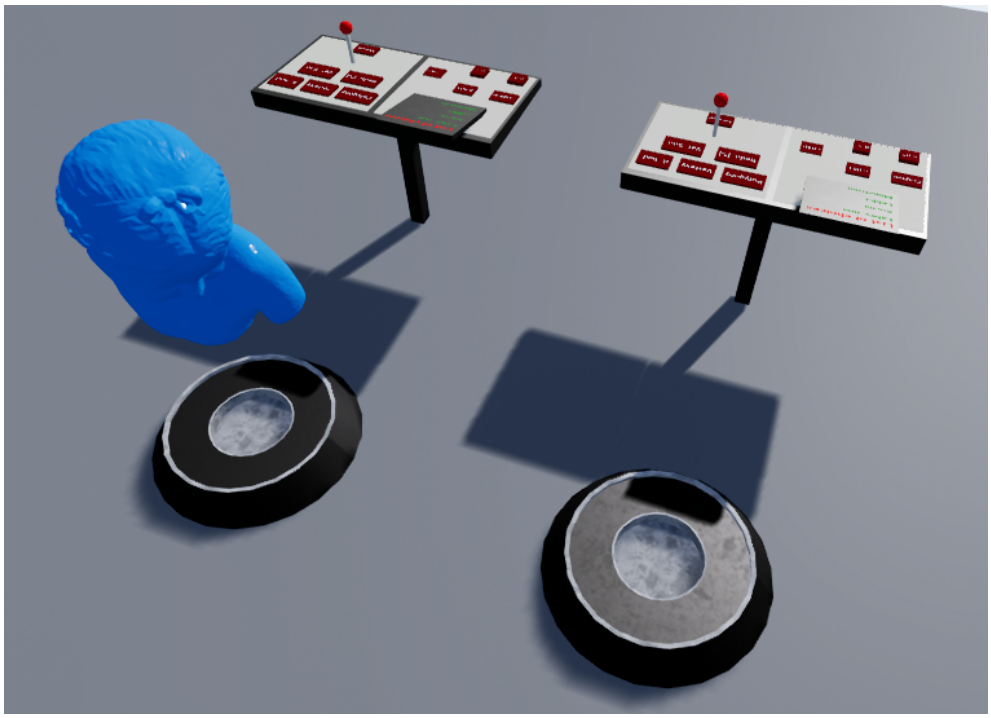
Obrázek 7.18: Schéma komponenta kontrolního panelu ControlPanelCategoryBP.

7.4.1 Statické modely

Modely, použité pro tento projekt jsem získal od pana doc. Ing. Váši, Ph.D., které jsem doplnil o modely z webové stránky *www.free3d.com*. Po vyzkoušení několika modelů následované jejich kompresí a dekompresí, jsem se rozhodl pro mojí práci vybrat tři externí modely a jeden model pana Váši. Těmito modely jsou: láhev vína, socha, lebka a zlobr. Každý z použitých modelů byl pomocí algoritmu komprese zredukován a opět dekomprimován pro hodnoty 0.001, 0.01 a 0.1. Konkrétní modely jsou ve formátu **OBJ** a byly importovány do Unreal enginu pro snadnější přístup. Lze je nalézt ve složce **CustomModels** ve všech konkrétních variantách komprese.

7.4.2 Umístění a zobrazení modelů

Podle předchozího odstavce budou modely umisťovány přímo nad samotnou platformou pro snadnější zobrazení konkrétního modelu a jeho varianty. Princip načtení modelů spočívá v zadání požadovaného modelu a jeho komprese na kontrolním panelu a pak stlačením tlačítka „LOAD“. Funkce následně provede detekci na základě indexu modelu a indexu komprese, který



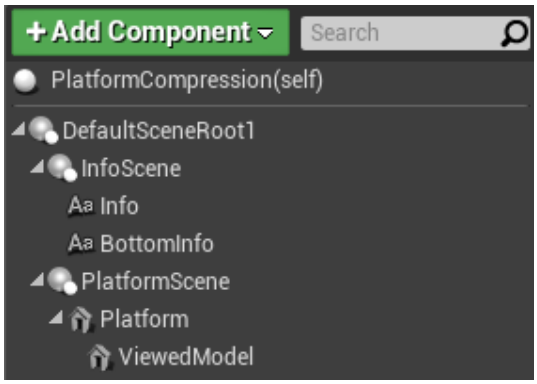
Obrázek 7.19: Obrázek místnosti s modely komprese a kontrolním panelem.

statický model ze složky **CustomModels** má zvolit.

Blueprint PlatformCompression

Tento Blueprint zprostředkovává platformu, nad kterou bude následně model umístěn na základě vybraných kritérií. Dále umožňuje zobrazení informací o modelu (počet polygonů, počet vertexů, velikost souboru na disku atd.). Na obrázku 7.20 je konkrétní schéma komponent tohoto Blueprintu.

Schéma této „třídy“ je rozděleno na část informační, kde jsou připra-



Obrázek 7.20: Schéma Blueprintu PlatformCompression.

vené Text Renderery pro zobrazení příslušných informací a druhou část pro zobrazení konkrétního modelu (komponenta **ViewedModel**). Mezi zásadní funkce patří **addOneModel**, **prepareCompressionText** a Event graph.

1. **addOneModel** - funkce provede na základě předaných parametrů (index modelu a index komprese) detekci zjišťující, který model a jeho kompresní variantu má použít. Následně přiřadí komponentě **ViewedModel** konkrétní statický mesh.

2. **prepareCompressionText** - úprava dat ze souboru **CompressionData.txt** do čitelného formátu. Výsledek je následně předán jako výstupní hodnota funkce k dalšímu zpracování.

3. **Event graph** - v této sekci se provádí každý „tik enginu“ kontrola, jestli je aktivní automatické otáčení modelu. Pokud ano, bude se otáčet model okolo své osy, jinak bude model pozastaven.

Nové funkce v MotionControllerPawn

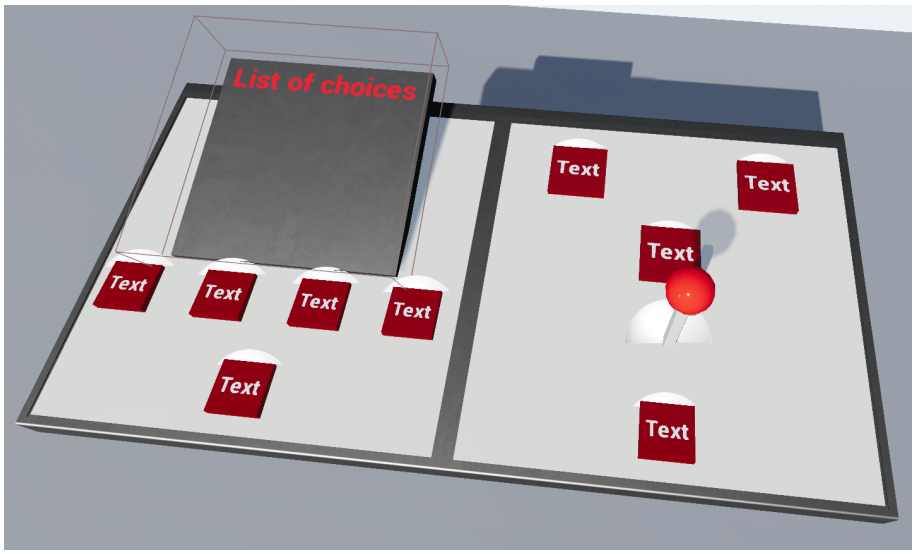
Pro tento Blueprint přibyly nové funkce, které mají za účel po aktivaci projektu komprese načíst konkrétní místnost a všechny komponenty. V případě teleportace do jiné místnosti jsou všechny komponenty odstraněny.

1. **loadCompression** - funkce, která vytvoří dvě platformy, a dva kontrolní panely a spáruje platformu s příslušným kontrolním panelem. Všechny načtené prvky uloží do atributů Blueprintu k pozdější možnosti manipulacemi s nimi. Posledním krokem algoritmu je provedení inicializace seznamu všech položek, ze kterých si potom bude moci uživatel vybrat typ modelu.

2. **unloadCompression** - procedura odstranění obou platforem a kontrolních panelů tak, že z atributů získá na objekty referenci a zavolá vnitřní funkci enginu **destroy**, která konkrétní objekt odstraní ze scény a následně i z paměti aplikace.

7.4.3 Kontrolní panel a interakce

Kontrolní panel k tomuto projektu se skládá ze dvou částí. První část umožňuje uživateli si zvolit, který druh modelu by si chtěl nechat zobrazit a v jakém kompresním poměru by model měl být(Originální, 0.001 atd.). Druhá část umožňuje zobrazení dodatečných informací o konkrétně vybrané instanci modelu a dále možnost zvolení automatické či manuální rotace (ovládáním řídící páky).



Obrázek 7.21: Kontrolní panel projektu komprese modelů.

Blueprint ControlPanelCompressionBP

Blueprint **ControlPanelCategoryBP** slouží pro seskupení všech potřebných komponent a funkcí k vytvoření funkčního kontrolního panelu. Kontrolní panel obsahuje Blueprinty ButtonBP a ControlHandleRotationBP. Zbylé komponenty slouží k vytvoření modelu kontrolního panelu.

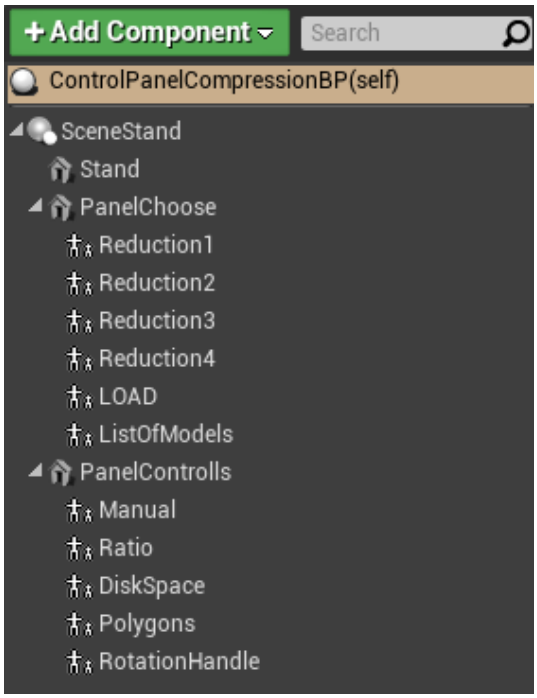
Koncept Blueprintu je velmi podobný ControlPanelEnzymBP z kapitoly 7.2.3. Z tohoto důvodu jej nebudu dále podrobněji rozebírat a pro případné detaily lze nahlédnout do zmíněného Blueprintu.

7.5 Rozšířitelnost aplikace

V rámci zadání bakalářské práce bylo nutné navrhnout a aplikovat různé vrstvy možného rozšíření vytvořené aplikace, aby další programátoři, kteří by chtěli obohatit mojí práci, měli svojí úlohu zjednodušenou. Z tohoto důvodu jsem navrhl tři základní pilíře, které umožní snadnější přístup ke změně napsaného kódu, jednoduchému vytvoření nových místností a rozšíření komponent i funkcionalit kontrolních panelů.

Komentáře tříd, funkcí, atributů a proměnných

Prvním pilířem k snadnějšímu přístupu a pochopení kódu je kompletní okomentování všech Blueprintů, jak mnou vytvořených tak vygenerovaných enginem, při jejich vytváření. Uvnitř Blueprintů jsem dále připravil komentáře pro všechny funkce i pro jejich vnitřní lokální proměnné. Komentáře funkcí



Obrázek 7.22: Schéma komponent pro Blueprint ControlPanelCompressionBP.

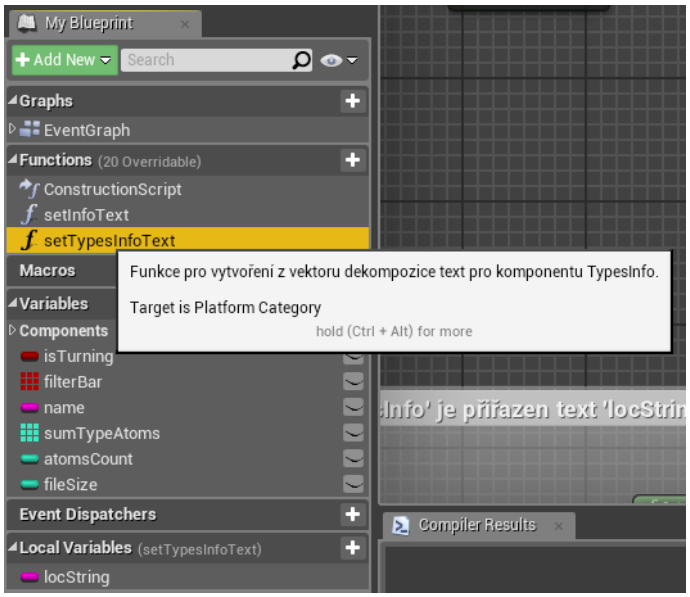
jsou rozděleny po jednotlivých, relativně uzavřených částech, tedy podle toho, co která část kódu představuje a provádí. Posledním krokem byly komentáře u všech použitých atributů a komentář Event graphu, pokud se v této sekci nějaké skripty nacházely. Na obrázcích 7.23 až 7.26 zobrazuji příklady komentářů pro Blueprint Platform_Category, které mají znázorňovat, jak takové popisky vypadají pro funkce, lokální proměnné či atributy. Případ bude ukázán pro funkci **setTypesInfoText**.

V jakémkoli Blueprintu si lze po přesunutí kurzoru myši na název funkce, atributu nebo lokální proměnné přečíst její význam, užití či rozmezí hodnot (atributy, proměnné).

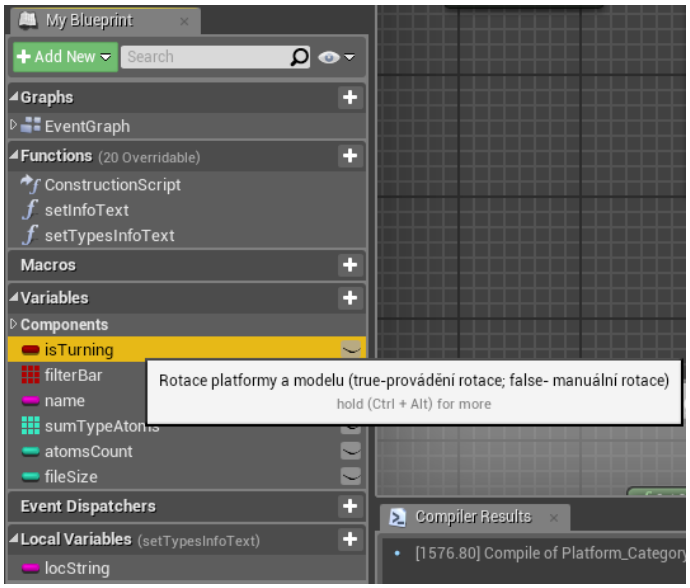
Poslední částí je pak uspořádání všech Blueprintů, struktur, modelů atd. do příslušných složek podle jejich typu či použití. Mezi důležité složky patří: CustomModels, CustomMaterials, CustomTextures, CustomBP, VirtualRealityBP a Maps.

Inicializační soubor

V kapitole 7.1.3 jsem představil, jak vypadá inicializační soubor **projects.txt**, který slouží k zajištění specifikace počtu místností a jejich názvů, kde na jejich základě připraví helmy pro teleportaci mezi místnostmi. Tento systém

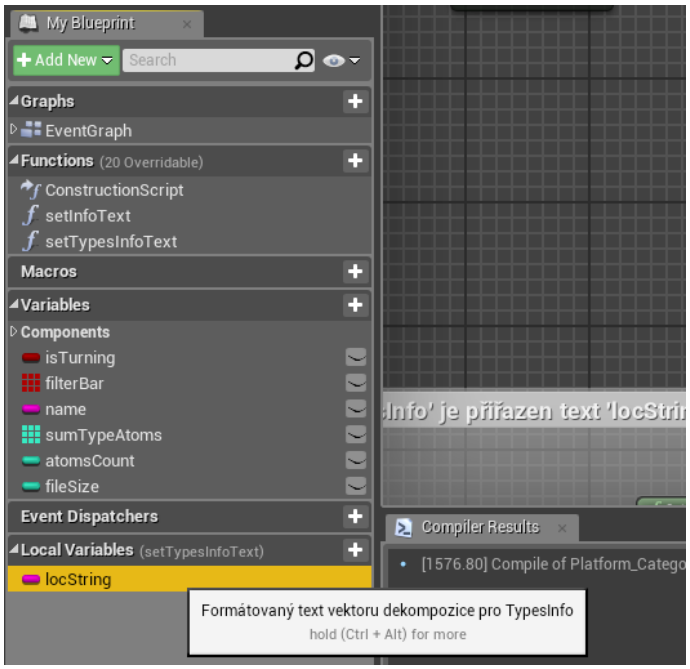


Obrázek 7.23: Po najetí kurzoru myši na funkci ze zobrazí popisek.

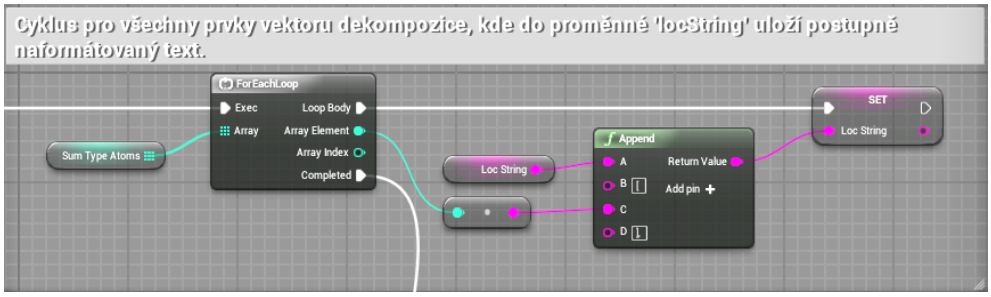


Obrázek 7.24: Po najetí kurzoru myši na atribut se zobrazí jeho použití, popřípadě rozmezí hodnot.

jsem navrhl, aby programátor nemusel v editoru Unreal enginu ručně přidávat každou místnost a všechny potřebné helmy včetně jejich obarvení a nastavení jejich atributů pro správnou teleportaci mezi projekty. Místo toho vše zařídí jednoduchý systém, který načte konkrétní soubor a všechno připraví automaticky. Jak soubor vypadá je již popsáno v kapitole 7.1.3, nyní se zaměřím na to, jak konkrétní místnost přidat do souboru a jaká omezení



Obrázek 7.25: Po najetí kurzoru myši na lokální proměnnou funkce se zobrazí její význam a užití.



Obrázek 7.26: Ukázka okomentované části kódu funkce.

s tím souvisí.
Do souboru **projects.txt** lze novou místnost jednoduše přidat vytvořením nového řádku s názvem místnosti (bez diakritiky) a ukončeným znakem '!'. Příkladem takového zápisu může být kupříkladu:

NewProject!

Důležitá informace ohledně ukončení souboru je, že projects.txt nesmí mít žádný prázdný řádek. Pokud by po posledním názvu místnosti byl ještě jeden prázdný řádek, mohlo by dojít k chybě načtení, program se sice normálně spustí, ale nenačte konkrétní data o místnosti a nevytvoří helmy v základní místnosti.

Mezi omezení tohoto systému patří především počet místností, který by neměl přesahovat cifru osm. Po deklaraci více než osmi místností nestačí „Zemská plocha“, po které by měl uživatel chodit, jinými slovy by se po přesunu do místnosti systém nemohl rozhodnout, kde je umístěna podlaha konkrétní místnosti a uživatel by se ocitl nad nekonečnou propastí, ze které by již nebylo úniku. Pokud by programátor chtěl rozšířit počet místnosti na větší počet než je osm, musel by zasáhnout do mapy, ve které se momentálně nachází a rozšířit komponenty **Box Brush** a **NavMeshBoundVolume**, které definují kde se může uživatel pohybovat a kam má přístup. V našem případě by pouze stačilo tyto dvě komponenty zvětšit ve směru přidávání místností.
Stejný problém se týká i počtu helem, kdy helmy jsou v místnostech umisťovány na předpřipravený stůl vedle sebe. Pokud by programátor neustále přidával nové místnosti i s upravenými komponentami z předchozího odstavce, helmy by se už nevešly na stůl a začaly by se přidávat vně stolu, kam už by uživatel neměl přístup. Pro upravení této části by následně bylo nutné se podívat do funkce, která definuje rozmístění helem v konkrétní místnosti (addHelm v Blueprintu AddableRoomBP) a upravit funkci tak, aby helmy byly umisťovány v jiném uspořádání k dosažení většího počtu místností. Poslední omezení je tabulka s nápovědou pro uživatele, kde po přidání více místností by text již nebyl čitelný. Pro úpravu bych doporučoval navrhnout systém, který bude cyklovat mezi všemi místnostmi, čímž se zachová velikost textu a také vzdálenost mezi samotnými prvky seznamu.

Komponenty a jejich rozšíření

Posledním pilířem je možnost vygenerovat si na základě mých vytvořených Blueprintů nové a následně upravit jejich vlastnosti podle své vlastní potřeby či dopsat nové funkcionality. Příkladem takových Blueprintů mohou být kontrolní panely, uchopitelné objekty (HelmetVisor či BP_PickupCube), struktury místností atd. Smyslem je připravit programátorovi různé druhy komplexnějších komponent a zjednodušit mu práci při jejich editaci nebo přidávání nových prvků podle potřeby. Dobrým příkladem je Blueprint BP_PickupCube, kde si můžeme v enginu vytvořit kopii tohoto Blueprintu, následně lze pozměnit statický model, doplnit jej nějakým novým efektem (kupříkladu změnou barvy objektu po jeho zvednutí) a získat tak nový Blueprint, který bude mít již všechny funkce nutné pro uchopení předmětu ve VR. Tímto způsobem může programátor přistupovat i ke všem ostatním Blueprintům.
Pro některé Blueprinty (viz ControlPanel...BP), lze přidat či ubrat mnou vytvořené komponenty. Například pokud bych chtěl pro novou místnost mít

stejný kontrolní panel jako má projekt kategorizace, ale současně přidat tři další tlačítka, potom to lze realizovat přidáním komponenty **Child Component** a nastavením jejího „rodiče“ na třídu ButtonBP. Nyní je nutné upravit tlačítku v sekci Blueprintu **Viewport** atribut **idButton** tak, aby nekolidovala s id ostatních tlačítek. Posledním krokem je přidat do Event graphu v sekci detekce a aktivace tlačítek (viz kapitola 7.2.3) nový stav podle id nového tlačítka a přiřadit konkrétní operace, které chceme, aby po stisknutí tlačítka byly provedeny. Kombinací, jak rozšířit konkrétní Blueprinty je spousta a je jen na programátorovi, jaké použije Bluerpinty a co v nich upraví či změní.

8 Závěr

Zadání práce jsem zpracoval do celkem sedmi kapitol. Pro první kapitolu jsem vzal v úvahu dostupná vývojářská prostředí a zpracoval jejich výhody a nevýhody k datu zpracování bakalářské práce. S tímto poznatkem jsem postupně podle zvolených kritérií možnosti implementace na virtuální realitu, dostupnosti vývojářského prostředí atd. vyřadil ty, které je nesplňovaly. Ve druhé kapitole mi pro detailní rozbor zbyly z osmi vývojářských prostředí pouze tři, konkrétně Unreal engine od Epic Games, Unity engine od Unity Technologies a poslední CryEngine od firmy Crytek. Pro každé z těchto prostředí (dále jen herních enginů) jsem uvedl jejich historii a konkrétní technologie, které byly během let vývoje implementovány.
Třetí kapitolu jsem věnoval rozboru výsledků osobního testování herních enginů, přičemž jsem testoval kvalitu uživatelského rozhraní, možnosti nastavení modelů, osvětlení, efektů atd. a v neposlední řadě také to, jak se vytvářejí ovládací skripty. V Závěru této kapitoly jsem shrnul přednosti a nedostatky každého z těchto herních enginů. Z konečného výběru jsem snadno vyřadil CryEngine a rozhodoval jsem se mezi Unity a Unreal enginem. Nakonec jsem vybral definitivně pro svoji bakalářskou práci Unreal engine.
Ve čtvrté kapitole jsem popsal své nabyté zkušenosti při seznamování se se zařízeními pro virtuální realitu, které mi Katedra informatiky a informační techniky umožnila k vyzkoušení. Mezi testované zařízení patřily Oculus Rift ve verzi DK2 od firmy Oculus a HTC Vive od firmy Valve. Obě zařízení jsem mezi sebou detailně porovnal na úrovni hardwarové i softwarové včetně dostupných aplikací, které pro ně byly vytvořeny k demonstraci jejich schopností. Výsledný dojem u obou z nich vedl k velmi porovnatelnému zhodnocení, ale pro zpracování bakalářské práce jsem vybral nakonec HTC Vive. V Závěru kapitoly jsem popsal historii firmy Valve a také detailněji rozebral jejich produkt HTC Vive.
V rámci kapitoly pět jsem uvedl představení několika výzkumných projektů, se kterými jsem se seznámil na Katedře informatiky a výpočetní techniky v oboru grafiky. Prvním tématem je výzkum a analýza enzymů. Smyslem výzkumu je studovat enzymy a hledat v jejich modelech dutiny (vnitřní prostory enzymu mezi atomy). Druhé téma se zabývá klasifikací modelů v průmyslu z hlediska jejich typu (trubka, plát atd.). Na fakultě je vyvíjen algoritmus, který identifikuje různé části modelů a na jejích základě stanoví, do jaké kategorie model nejvíce patří. Posledním projektem je výzkum zaměřený na kompresi datových souborů modelů a animací, přičemž smyslem

výzkumu je nalézt druh algoritmu, který provede vysokou kompresi modelu s co nejmenší ztrátou dat. Po domluvě s konzultantem jsme se rozhodli navrhnout implementaci všech těchto tří projektů do prostředí virtuální reality. V kapitole šest jsem pro každý projekt definoval, z jakých komponent se bude prezentace skládat včetně funkcionální části. Jelikož jsou projekty dosti rozdílné, bylo třeba navrhnout, jak všechny projekty spojit do jednoho celku a libovolně mezi nimi přecházet. Výsledné řešení jsem stanovil umístěním každého projektu a všech jeho komponent do virtuální místnosti navržené přesně podle laboratoře na Katedře informatiky a výpočetní techniky UC335, kde se nachází instalace virtuální soupravy HTC Vive. Všechny projekty pak bude spojovat tzv. „Základní místnost“. Pro projekt analýzy enzymů jsem navrhl zobrazení enzymu na podstavci v různých variantách počtů atomů. Společně s enzymem je v projektu přítomen i kontrolní panel, který slouží k ovládání prezentovaného enzymu nebo načtení nového enzymu. K projektu kategorizace modelů jsem navrhl systém, který zobrazí uživateli v prostoru několik desítek modelů. Uživatel si bude moci nechat zvýraznit dva modely k porovnání nebo pouze jeden model, u kterého se také vykreslí další modely jenž algoritmus vyhodnotil jako podobné. K zobrazení dodatečných dat a ovládání modelu slouží opět kontrolní panel. U projektu komprese modelů jsem navrhl prezentaci modelů vedle sebe k jejich vzájemnému přímému porovnání a u každého z nich jsem umístil kontrolní panel. Závěrečnou částí kapitoly je specifikace rozšířitelnosti, která spočívá v přidávání nových místností pomocí konfiguračního souboru. V rámci interakce s prostředím jsem navrhl systém tlačítek, řídících ovladačů a výběrových seznamů, které lze použít pro jakýkoliv jiný další typ projektu nebo zcela nové téma.
Poslední kapitolu jsem věnoval popisu implementace všech komponent z předchozí kapitoly do Unreal enginu, přičemž jsem se soustředil při vytváření na možnost opětovné použitelnosti každé komponenty zajištěním jednoduchého přístupu ke změně parametrů či přidávání nových funkcí. Kód jsem psal v jazyce skriptů Unreal Engine nativní formou systému tzv. Blueprintů, které jsem skládal do jednoduchých struktur pro lepší přehlednost a také pro vytvoření vzoru, který poslouží dalšímu programátorovi jako předloha k vývoji podobného produktu. Dalším aspektem zajištění snadné rozšířitelnosti bylo řádné zpracování dokumentace a okomentování kódu včetně lokálních proměnných. V příloze lze nalézt příklady, jak správně vytvořit nový Blueprint od úplného počátku.

9 Přehled zkratek

VR (Virtual Reality)	Zkratka pro virtuální realitu.
ID (Identification)	Volně přeloženo jako identifikátor, v textu použito jako unikátní označení.
CPU (Central Procesing Unit)	Základní výpočetní jednotka počítače (procesor).
OLED (Organic light-emitting diode)	Zkratka pro technologie displeje založenou na principu LED s využitím organických diod.
Blueprint	Grafické vytváření kódu v Unreal Enginu pomocí uzlů a spojnic, kde uzel symbolizuje změnu stavu, proces či funkci a spojnice definují závislosti nebo postup zpracování uzlů. Blueprintem se také označuje kompletní třída reprezentující komplexnější komponentu (třída skládající se z funkcí, atributů a nastavení).
HMD (Head Mounted Display)	Zkratka pro zobrazovací zařízení spojené s hlavou nebo helmou.
FPS (Frames per Second)	Zkratka pro počet zobrazovaných snímků za sekundu.
Event	Akce či událost, která provede na základě podnětu ze systému enginu, vstupů z periferií atd. příslušné operace.
DK (Develompent Kit)	Vývojářský nástroj.

10 Literatura

[1] List of game engines - Wikipedia. *Wikipedia, the free encyclopedia* [online]. Ashburn: Wikimedia Foundation, 2019 [cit. 2019-01-05]. Dostupné z: https://en.wikipedia.org/wiki/List_of_game_engines

[2] Unreal Engine – Wikipedie. *Wikipedia, the free encyclopedia* [online]. Ashburn: Wikimedia Foundation, 2019 [cit. 2018-06-25]. Dostupné z: https://cs.wikipedia.org/wiki/Unreal_Engine

[3] About Unreal Engine 4. *Unreal Engine* [online]. Cary (North Carolina): Epic Games, 2019 [cit. 2019-01-05]. Dostupné z: https://www.unrealengine.com/en-US/features

[4] CryEngine – Wikipedie. *Wikipedia, the free encyclopedia* [online]. Ashburn: Wikimedia Foundation, 2019 [cit. 2019-01-05]. Dostupné z: https://cs.wikipedia.org/wiki/CryEngine

[5] CryEngine - Wikipedia EN. *Wikipedia, the free encyclopedia* [online]. Ashburn: Wikimedia Foundation, 2019 [cit. 2019-01-05]. Dostupné z: https://en.wikipedia.org/wiki/CryEngine

[6] CRYENGINE Features. *CRYENGINE | The complete solution for next generation game development by Crytek* [online]. Frankfurt am Main: Crytek, 2019 [cit. 2019-01-05]. Dostupné z: https://www.cryengine.com/features

[7] CRYENGINE Roadmap. *CRYENGINE | The complete solution for next generation game development by Crytek* [online]. Frankfurt am Main: Crytek, 2019 [cit. 2019-01-05]. Dostupné z: https://www.cryengine.com/roadmap

[8] List of Unity games - Wikipedia. *Wikipedia, the free encyclopedia* [online]. Ashburn: Wikimedia Foundation, 2019 [cit. 2019-01-05]. Dostupné z: https://en.wikipedia.org/wiki/List_of_Unity_games

[9] Unity (game engine) - Wikipedia. *Wikipedia, the free encyclopedia* [online]. Ashburn: Wikimedia Foundation, 2019 [cit. 2019-01-05]. Dostupné z: https://en.wikipedia.org/wiki/Unity_(game_engine)

[10] Fast Facts - Unity. *Unity* [online]. San Francisco (California): Unity Technologies ApS, 2019 [cit. 2019-01-05]. Dostupné z: https://unity3d.com/public-relations

[11] HTC Vive - Wikipedia. *Wikipedia, the free encyclopedia* [online]. Ashburn: Wikimedia Foundation, 2019 [cit. 2019-01-05]. Dostupné z: https://en.wikipedia.org/wiki/HTC_Vive

[12] Oculus Rift - Wikipedia. *Wikipedia, the free encyclopedia* [online]. Ashburn: Wikimedia Foundation, 2019 [cit. 2019-01-05]. Dostupné z: https://en.wikipedia.org/wiki/Oculus_Rift

[13] VIVE™ | Vive Ready Computers. *VIVE™ / Discover Virtual Reality Beyond Imagination* [online]. New Taipei City (Xindian): HTC Corporation, 2019 [cit. 2019-01-05]. Dostupné z: https://www.vive.com/us/ready/

[14] Oculus Rift DK2 - Virtual Reality and Augmented Reality Wiki. *Virtual Reality and Augmented Reality Wiki* [online]. Ashburn: Wikimedia Foundation, 2016 [cit. 2019-01-05]. Dostupné z: https://xinreality.com/wiki/Oculus_Rift_DK2

[15] Oculus Rift DK2 - High In The Sky. In: *High In The Sky* [online]. Praha: High In The Sky, 2016 [cit. 2019-01-05]. Dostupné z: http://www.highinthesky.cz/3d-bryle/oculus-rift-dk2/

[16] Vive la best VR system coming to Thailand. In: *THE NATION* [online]. New York: The Nation Company, L.P., 2017 [cit. 2019-01-05]. Dostupné z: http://www.nationmultimedia.com/detail/Startup_and_IT/30328344

[17] Unity - Manual: Input for OpenVR controllers. In: Unity - Manual: Unity User Manual (2018.1) [online]. San Francisco (California): Unity Technologies ApS, 2017 [cit. 2019-01-5]. Dostupné z: https://docs.unity3d.com/560/Documentation/Manual/OpenVRControllers.html

[18] Adaptér Vive Wireless odstraní z VR brýlí HTC Vive kabely [online]. Praha: https://otechnice.cz/adapter-vive-wireless-odstrani-vr-bryle-htc-vive-kabelu/, 2018 [cit. 2019-01-05]. Dostupné z: https://www.vive.com/us/wireless-adapter/

[19] MCCAFFREY, Mitch. *Unreal Engine VR Cookbook: Developing Virtual Reality with UE4*. Boston (Massachusetts): Addison-Wesley, 2017. ISBN 978-0134649177. 0134649176, B01MV6ZKU2.

[20] KOZLÍKOVÁ, Barbora, Eva ŠEBESTOVÁ, Vilém ŠUSTR a Jan BŘEZOVSKÝ. CAVER Analyst 1.0: Graphic tool for interactive visualization and analysis of tunnels and channels in protein structures. *Bioinformatics* [online]. 2014, 2014, , 1-2 [cit. 2019-05-05]. DOI: 10.1093/bioinformatics/btu364. Dostupné z: https://www.researchgate.net/publication/262787114_CAVER_Analyst-_10_Graphic_tool_for_interactive_visualization_and_analysis_of_-tunnels_and_channels_in_protein_structures

[21] ŠIGUT, Pavel, Petr VANĚČEK a Libor VÁŠA. *Analytic Surface Detection in CAD Exported Models* [online]. Plzeň, 2019 [cit. 2019-05-05]. Dostupné z: https://www.researchgate.net/publication/331776208_Analytic_Surface-_Detection_in_CAD_Exported_Models. Konferenční práce. Západočeská univerzita v Plzni, Fakulta informatiky.

[22] VÁŠA, Libor a Guido BRUNNETT. Exploiting Connectivity to Improve the Tangential Part of Geometry Prediction. *IEEE Transactions on Visualization and Computer Graphics*. 2013, **19**(9), 1467 - 1475. DOI: 10.1109/TVCG.2013.22. ISSN 1077-2626.

11 Příloha

Vytvoření nového Blueprintu

V rámci přílohy se pokusím nastínit základní proces vytváření nového Blueprintu včetně jeho součástí a nutných nastavení. Cílem této kapitoly tedy bude vytvořit Blueprint, který po jeho přidání do scény a spuštění aplikace náhodně vybere model a začne se otáčet podle jedné z os (nebo více), které si programátor předem vybere v globálním nastavení Blueprintu.

Založení Blueprintu

Základní editor Unreal enginu se skládá z pěti panelů: Models, Editor Viewport, World Outliner, Details Panel a Content Browser. **Models** (sekce A na obrázku 11.1) představuje panel nástrojů, ve kterém můžeme přidávat základní komponenty Unreal enginu jako modely, osvětlení, efekty, ale také úprava terénu, přidání rostlinné flóry a mnoho dalších. **Editor Viewport** (B) prezentuje umístění všech komponent, které byly přidány do konkrétního projektu (scéna prezentující konkrétní stav projektu). **World Outliner** (C) obsluhuje seznam všech přidaných komponent v projektu. Seznam umožňuje třídění komponent do složek a zobrazení či skrytí komponent ve Viewportu pro snadnější nalezení a upravení hledané komponenty. **Details Panel** (D) zprostředkovává zobrazení nastavení vybrané komponenty z Viewportu. **Content Browser** (E) slouží jako souborový prohlížeč všech součástí projektu. Content Browser umožňuje operace se složkami a jejich manipulaci včetně editace všech komponent nalézajících se uvnitř projektu.
Pro založení nového Blueprintu si v Content Browseru připravíme složku s libovolným názvem. Po vytvoření složky se přesuneme do jejího obsahu a tlačítkem **+AddNew** (sekce 1 na obrázku 11.2) provedeme přidání nové komponenty. Ze seznamu možností vybereme *Blueprint Class* (2). Dalším krokem je provedení výběru typu Blueprint Class. Pro naše účely zvolíme možnost **Actor** (sekce 3 na obrázku 11.3).

Schéma Blueprintu a nastavení

Pokud při vytváření Blueprintu nedojde k žádné chybě, zobrazí se nyní nové okno reprezentující *Blueprint Class* jako na obrázku 11.4. Okno si lze rozdělit na čtyři části: Components Tab, My Blueprint, Viewport Panel a Details Panel. **Components Tab** (F) zobrazuje aktuální sestavení všech komponent,

ze kterých se Blueprint momentálně skládá. Komponenty jsou uspořádány do obecného stromu. Hierarchie umožňuje u prvků patřících skupině provádět společné operace pohybu, rotace či změny měřítka. V **My Blueprint** (G) jsou umístěny všechny události, atributy (Variables) včetně odkazů na komponenty z Components Tab a definice všech funkcí, které Blueprint obsahuje. **Viewport** (H) zobrazuje, jak jsou komponenty z Components Tab uspořádány v prostředí. **Details Panel** (I) zobrazuje všechna nastavení pro zvolený model z Viewportu.

Pro naše účely nyní přidáme komponentu **Static Mesh Component**, která bude sloužit k zobrazení náhodně vybraného modelu. V panelu Components Tab zvolíme možnost **+AddComponent** (sekce 4 na obrázku 11.5), která rozbalí seznam všech možných komponent. Najdeme si komponentu Static Mesh Component (5) a kliknutím provedeme její přidání mezi komponenty třídy. Po zvolení jména statického meshe by se měly zobrazit v Details Panel informace a nastavení tohoto statického meshe. V tomto stavu by statický mesh neměl mít nastavený žádný geometrický model, který ho reprezentuje, proto ve Viewportu není zobrazen. Nastavení konkrétního modelu provedeme pomocí funkcí.

Přidání funkcionality

Pro tento Blueprint vytvoříme dvě funkce. První funkce zajistí po spuštění aplikace načtení náhodného modelu a jeho přiřazení komponentě statického meshe uvnitř naší třídy. Druhá funkce bude každý „tik enginu“ provádět natočení statického meshe.

Pro založení nové funkce Blueprintu je nutné uvnitř panelu My Blueprint v sekci **Functions** přidat novou funkci stisknutím symbolu **+Function** (sekce 6 na obrázku 11.6), který nás přenese do jejího zápisu. Po stanovení názvu funkce (7) se vytvoří počáteční uzel (8), který představuje počátek funkce. Pro počáteční uzel si můžeme stanovit, jaké vstupní parametry bude funkce vyžadovat v Details Panel. Pro ukončení funkce je možné přidat konečný uzel (9), kterému lze nastavit návratové hodnoty. Systém zápisu programu funguje na principu uzlů a spojnic. Uzly symbolizují operace nebo objekty a spojnice slouží pro navázání vztahů mezi objekty nebo postup vykonávání procesů. Z obrázku 11.6, na kterém je vyobrazen algoritmus pro načtení modelu popíši význam uzlů a jak spojnice definují postup celého algoritmu.

V prvotní situaci funkce vede z počátečního uzlu spojnice do uzlu **Switch on Int** (10). Switch on Int představuje proces, který ze vstupní hodnoty pro tento uzel definuje jaký další podproces se má vykonat. V našem případě vstupní hodnota uzlu je předána uzlem **Random Integer in Range** (11),

který vygeneruje na základě vstupních parametrů náhodné číslo a hodnotu umístí do návratové části. Pro zajištění správné funkce Switch on Int tedy musíme navázat spojení návratové hodnoty uzlu Random Integer in Range se vstupním parametrem uzlu Switch on Int. Tímto docílíme realizace procesu zpracování výstupní hodnoty z Random Integer in Range uzlem Switch on Int. V procesu následujícím za uzlem Switch on Int dále budeme potřebovat lokální proměnou pro uložení příslušné reference na konkrétní model. V My Blueprint panelu nám přibyla nová sekce **Local Variables** (12). Pro přidání nové lokální proměnné provedeme stisknutí tlačítka **+LocalVariable**. Po vytvoření lokální proměnné musíme stanovit jakého typu má proměnná být a jak se bude nazývat, to nastavíme v Details Panel (13). V našem případě typem proměnné bude Static Mesh Reference. Nyní pro každou větev Switch on Int nastavíme hodnotu proměnné na konkrétní geometrický model ze základního balíčku modelů (lze pochopitelně navolit i jiné modely). Dalším krokem je přidružení konkrétního geometrického modelu komponentě Blueprintu. To zajistíme uzlem **Set Static Mesh** (14). **Set Static Mesh** přiřadí komponentě Static Mesh Component příslušný předaný model. U našeho algoritmu se nastaví komponentě RandomModel geometrický model z lokální proměnné locStaticMesh. Ukončení funkce provedeme spojením výstupu posledního uzlu s uzlem **Return Node**. Posledním krokem je spuštění překladu Blueprintu tlačítkem (15). Přeložit Blueprint je nutností po jakékoliv změně uvnitř funkcí, nastavení komponent, ale i po změně atributů či lokálních proměnných.

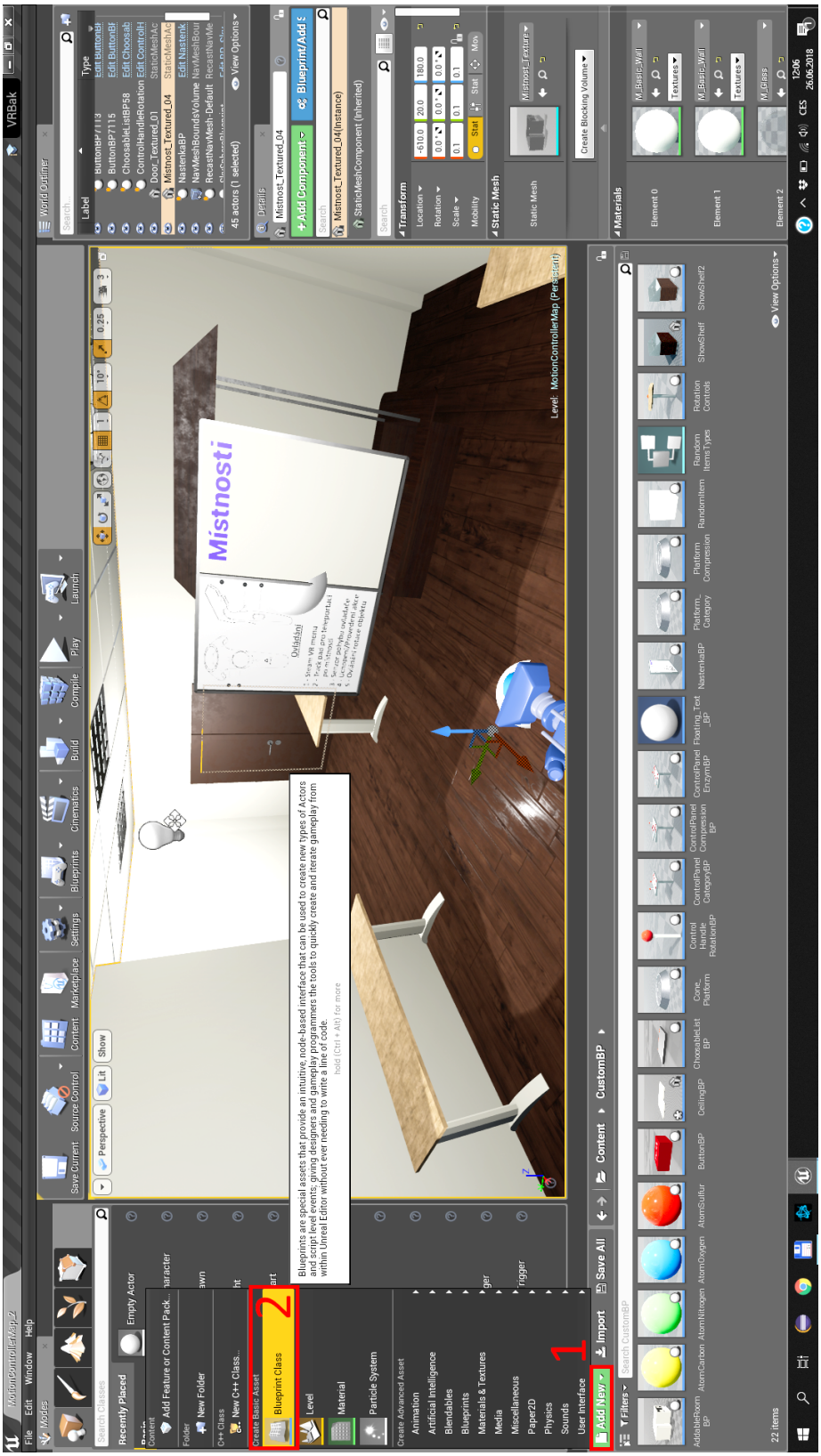
Pro druhou funkci budeme chtít připravit proces, který na základě nastavení přímo v editoru po spuštění aplikace začne otáčet modelem v jedné z os (nebo více osách). Nejprve si musíme vytvořit atributy Blueprintu pro stanovení, v jaké ose se má konkrétní model otáčet. To provedeme panelem My Blueprint v sekci Variables, kde tlačítkem **+AddVariable** (16) přidáme nový atribut. Pro naše účely si vytvoříme tři atributy symbolizující každou osu, pro kterou budeme chtít provádět rotaci (typ proměnné bude boolean) (17). Důležitým nastavením, aby atribut byl přímo ovlivněn z editoru, je u všech tří atributů změnit hodnotu viditelnosti na **public**. To provedeme pomocí stisknutí symbolu oka (18) vedle atributu. Na obrázku 11.8 je vyobrazen konkrétní algoritmus rotace objektu na základě tří atributů.

Nyní máme všechny funkce připravené a musíme propojit příslušné funkce s událostmi, jinak nebudou vlastně aktivní. Události nastavíme v Event graphu (19), kde vytvoříme dvě události **Event BeginPlay** a **Event Tick**. Událost BeginPlay bude aktivována po spuštění aplikace, které přiřadíme funkci pro načtení modelu. Event Tick bude spuštěn při každém vykreslení scény aplikace, kterému bude přiřazena funkce natočení modelu.

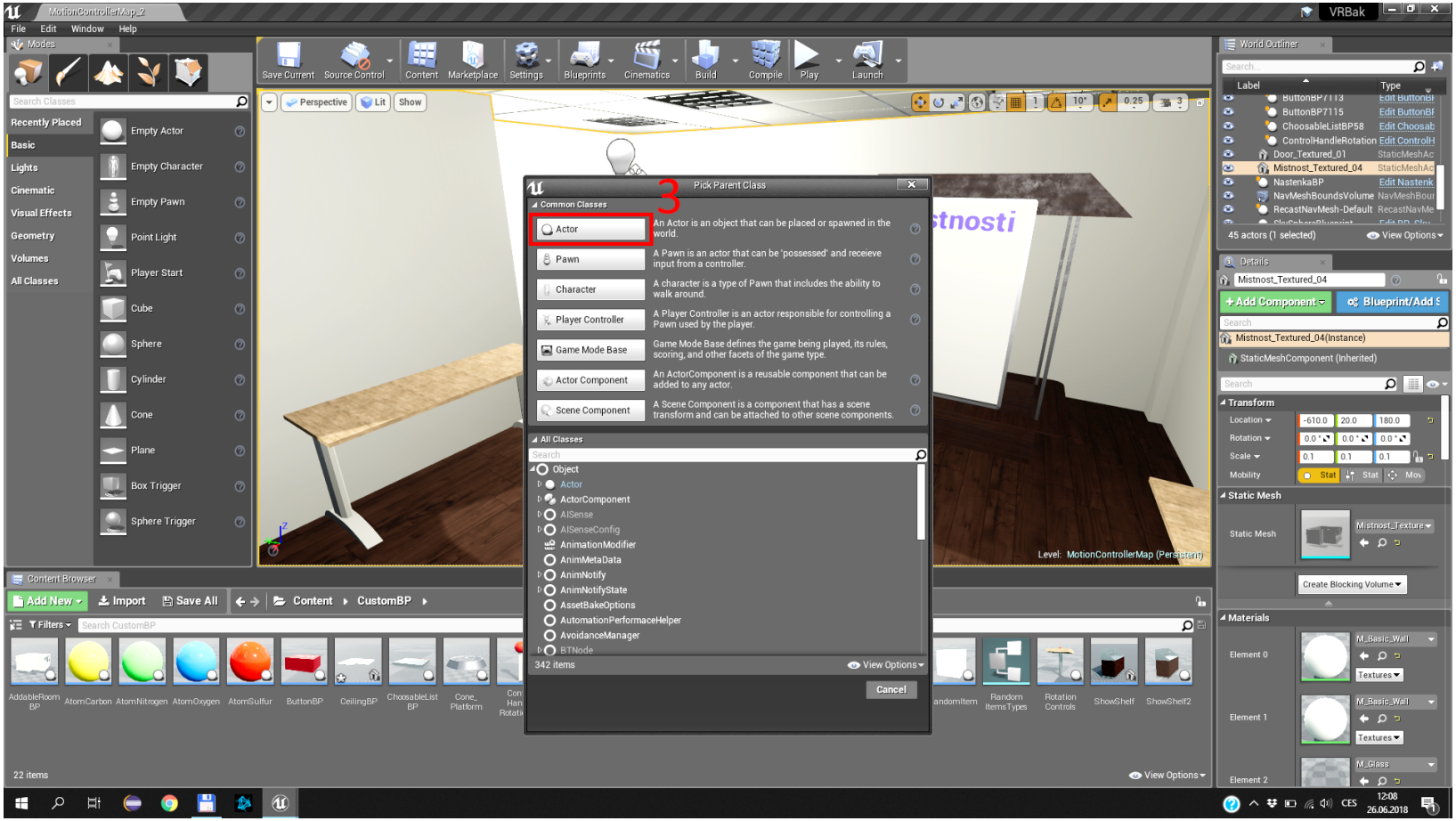
Explicitní nastavení rotace v základním editoru

Finální částí Blueprintu je v tento moment umístění jeho instance do scény. To provedeme uchopením Blueprintu uvnitř Content Browseru, přetáhnutím do Viewportu a následně jeho upuštěním. Náš Blueprint se nyní nachází ve scéně a po spuštění aplikace se načte konkrétní model, ale nespustí se rotace modelu! To je způsobeno tím, že ještě nebyly nastaveny atributy zodpovědné za rotaci podle konkrétní osy. Tyto atributy nastavíme v Details Panelu našeho Blueprintu v sekci **Default** (20). Po zvolení jednoho (nebo kombinace atributů) se konkrétní model začne pomalu otáčet při spuštění aplikace.

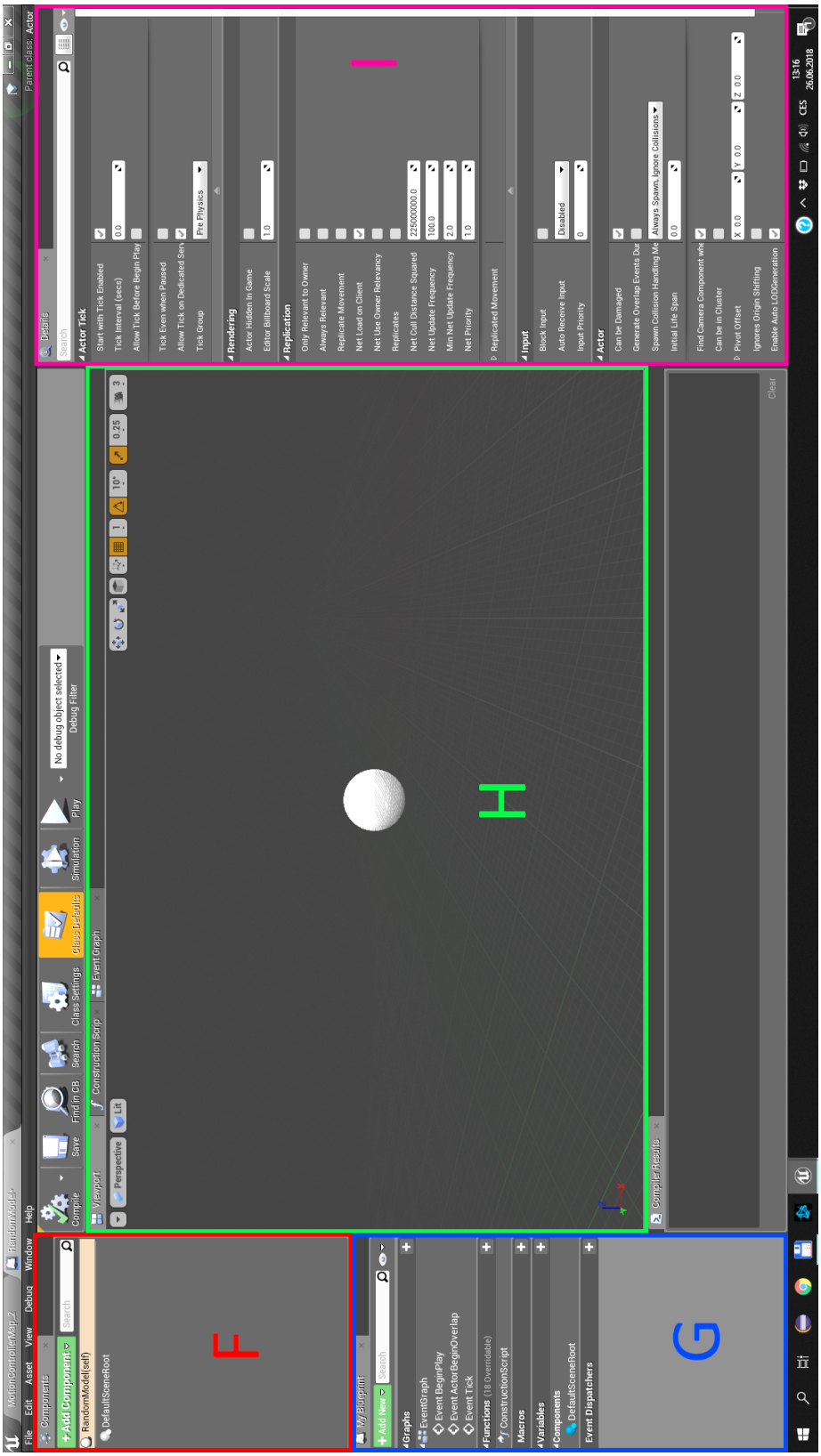
Obrázek 11.1: Základní rozložení všech panelů editoru Unreal enginu.



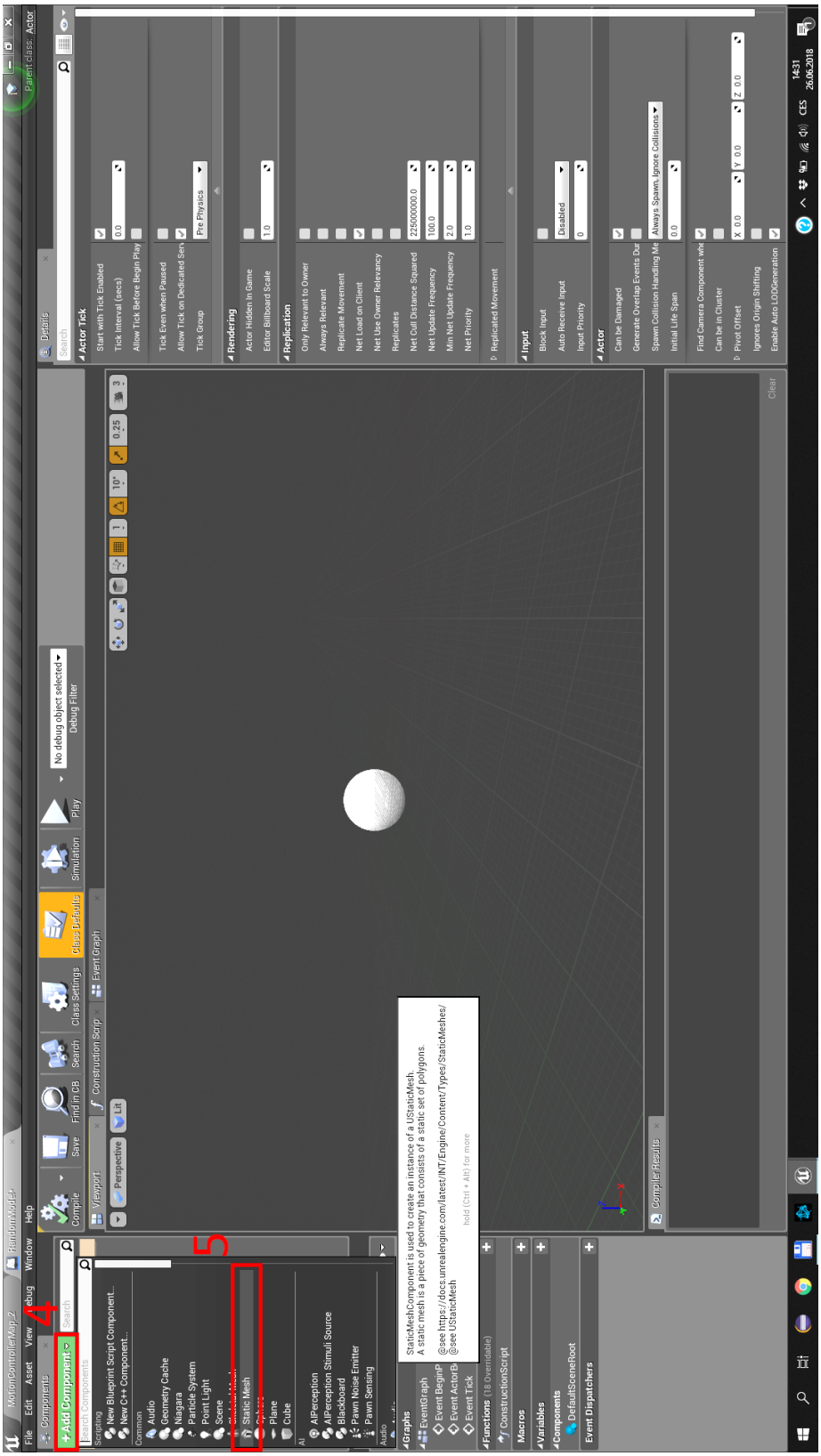
Obrázek 11.2: Postup vytvoření nového Blueprintu část 1.



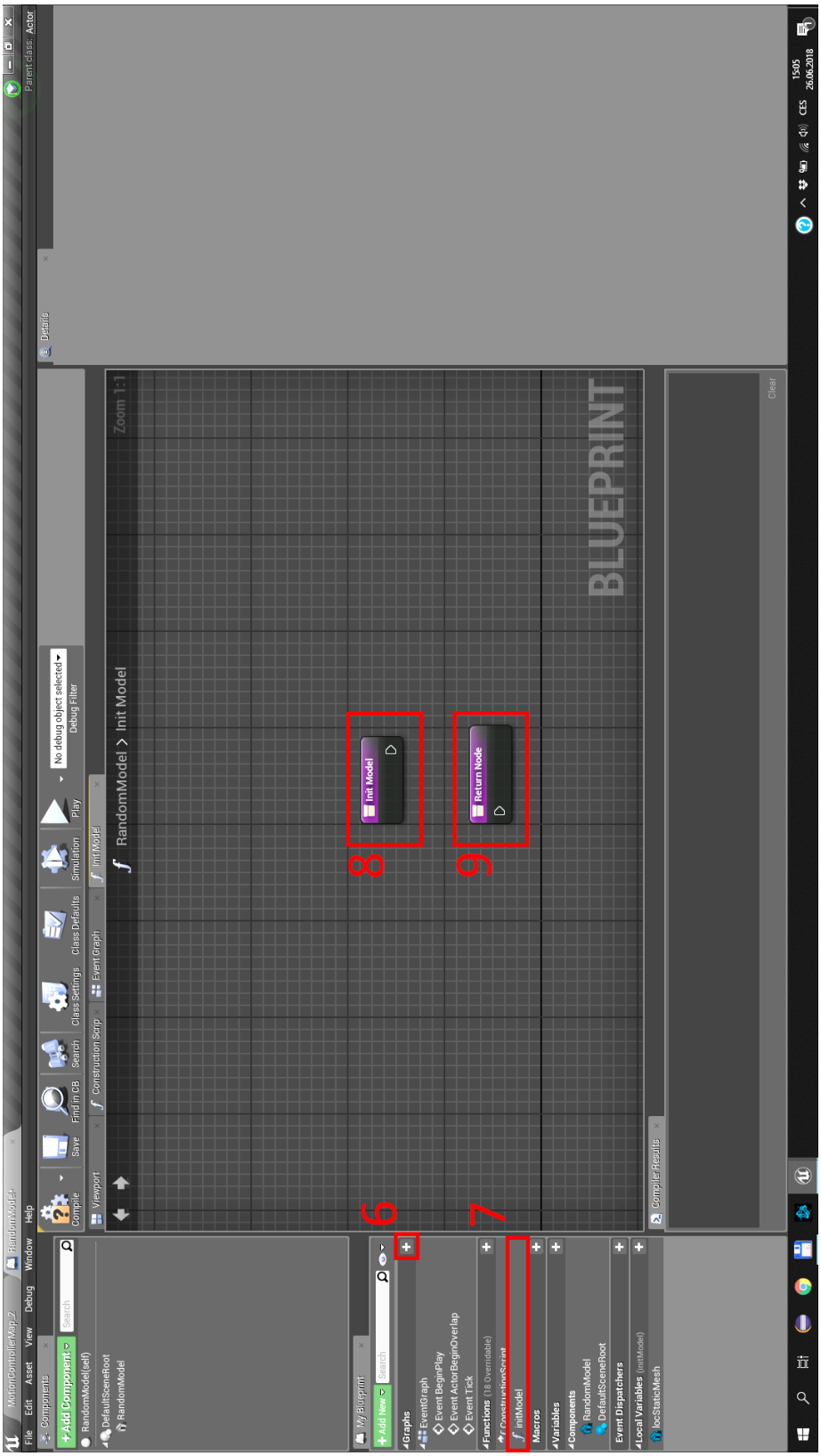
Obrázek 11.3: Postup vytvoření nového Blueprintu část 2.



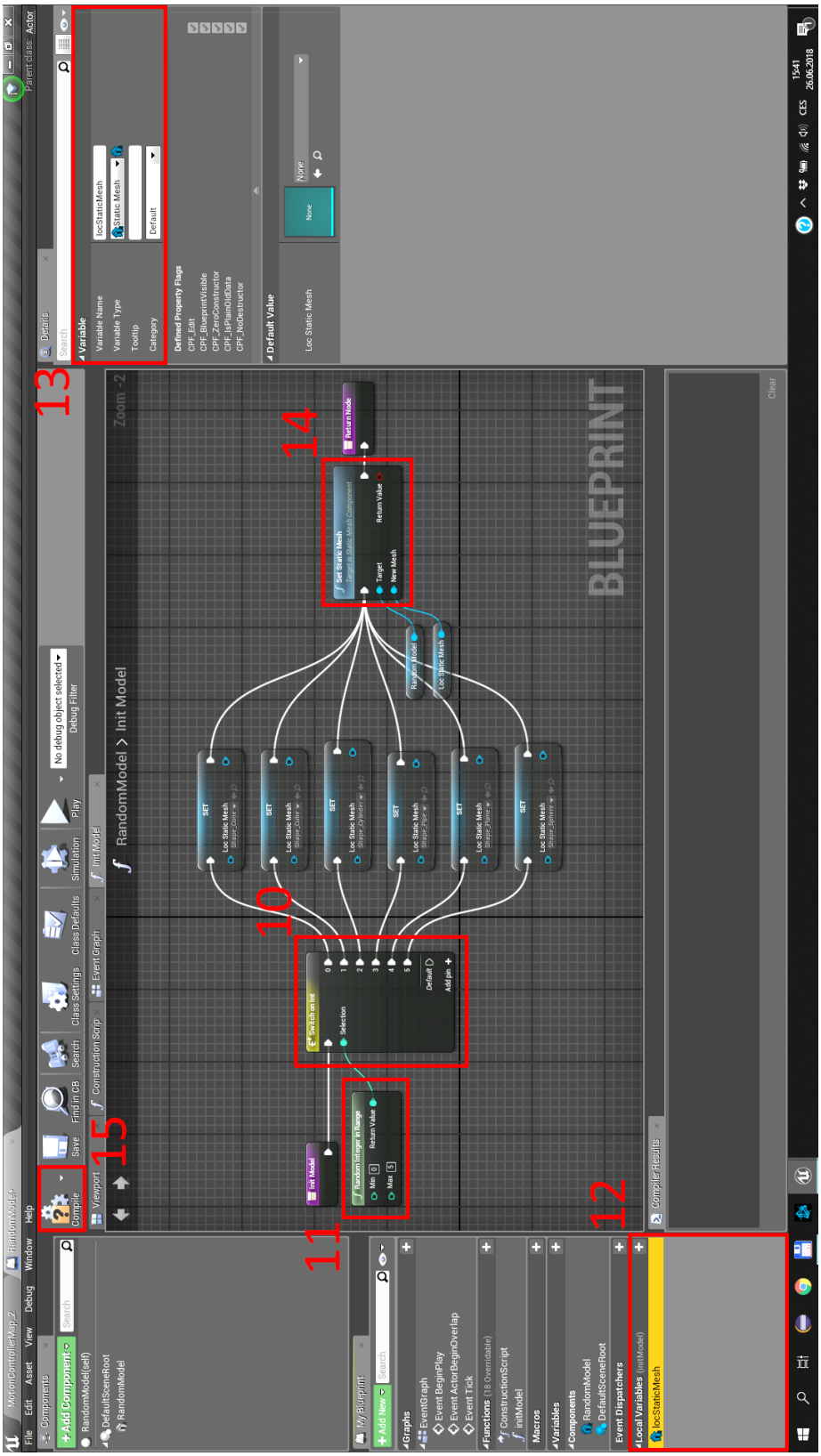
Obrázek 11.4: Rozložení sekcí uvnitř Blueprintu.



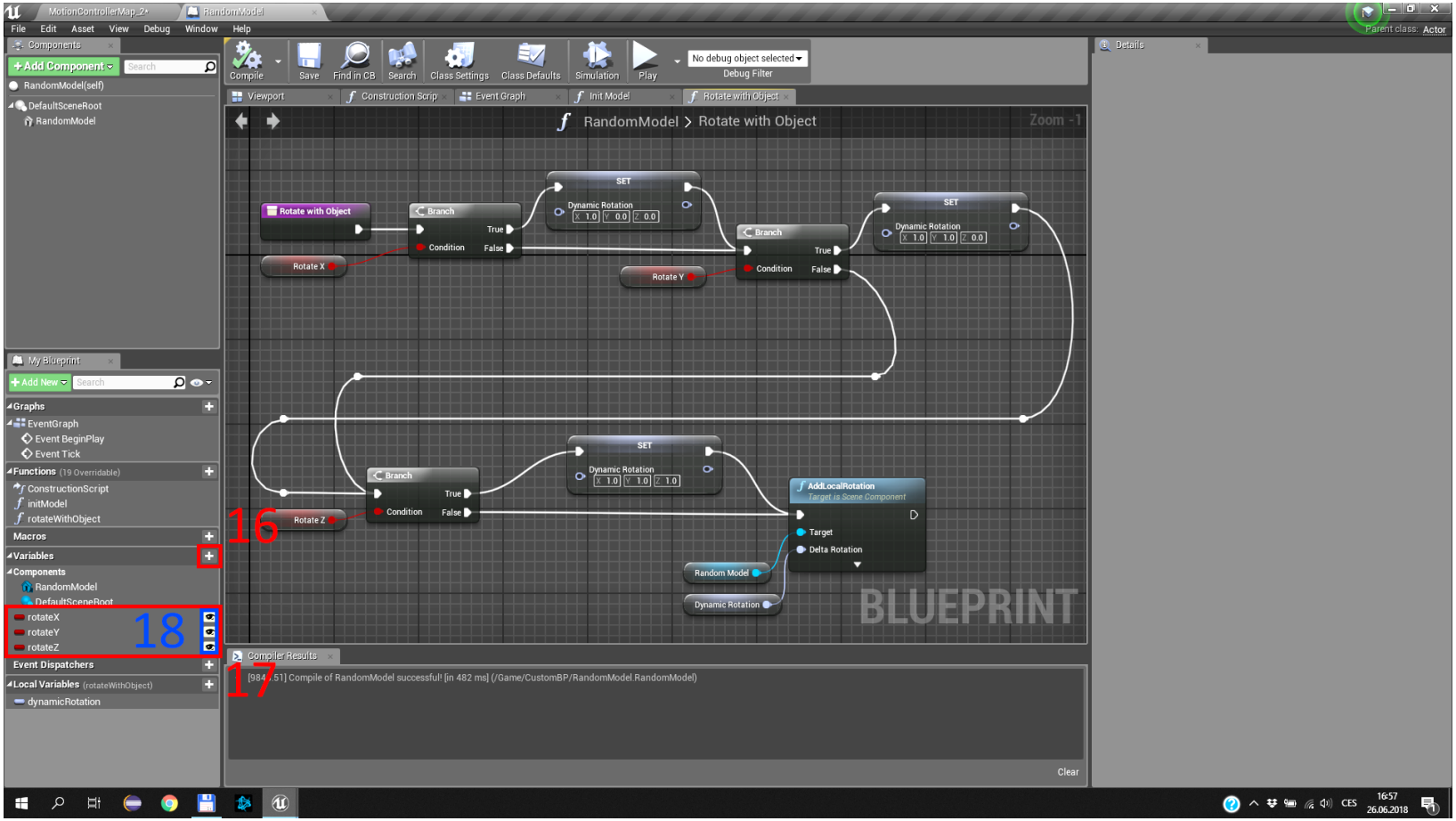
Obrázek 11.5: Přidání komponenty Static Mesh Component do Blueprintu.



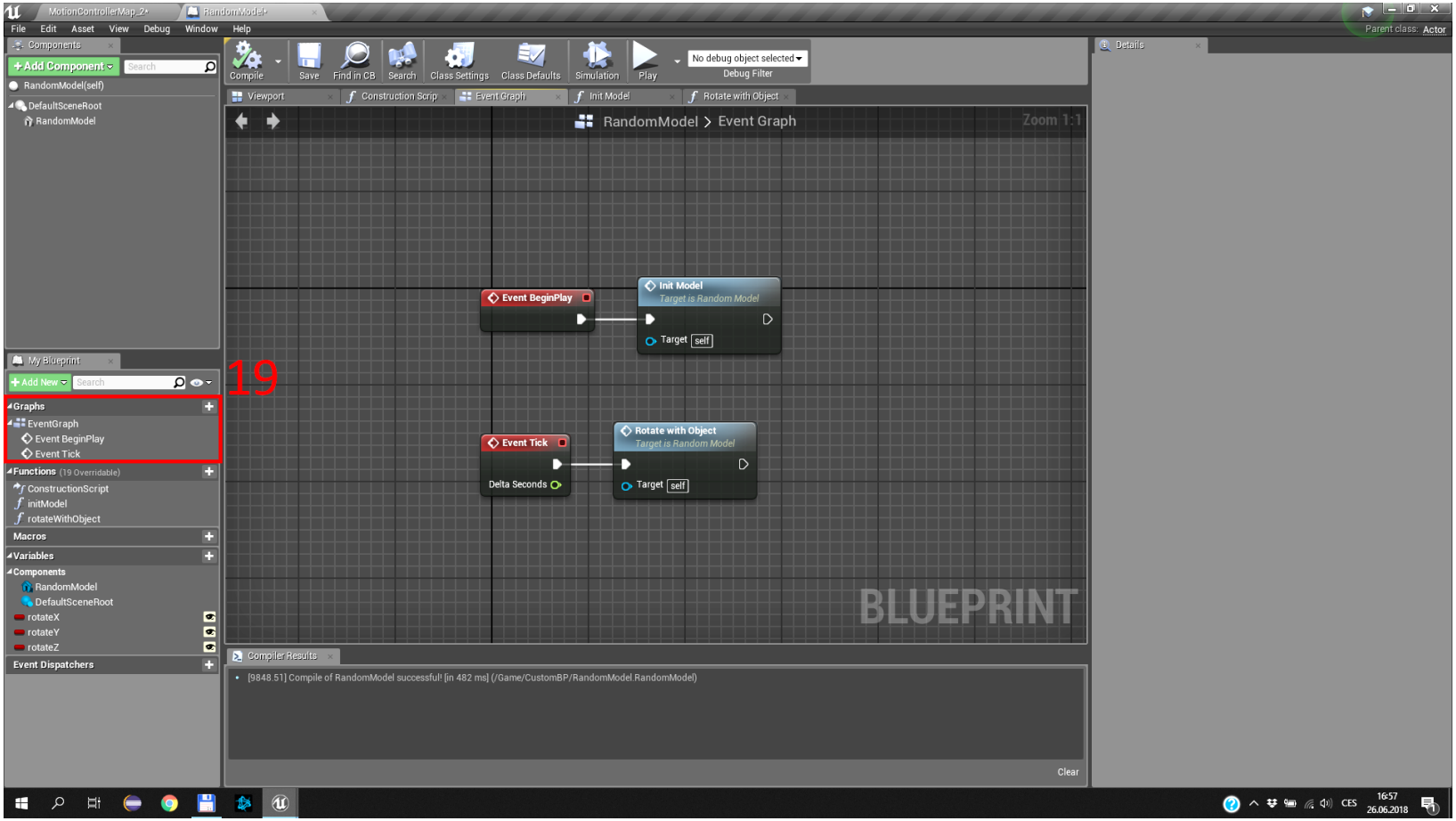
Obrázek 11.6: Založení funkce a vysvětlení základních uzlů.



Obrázek 11.7: Funkce pro vytvoření nového modelu a jeho přiřazení komponentě RandomModel.



Obrázek 11.8: Funkce pro natočení modelu.



Obrázek 11.9: Události Blueprintu RandomModel.

Obrázek 11.10: Upravení atributů Blueprintu RandomModel k zajištění rotace po spuštění aplikace.

Seznam obrázků

Seznam tabulek