FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Věnná města Českých Královen - Import 3D modelů do Unreal Engine 4
Student:	Tadeáš Bušek
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Počítačová grafika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Věnná města Českých Královen je projekt zabývající se zobrazením historického prostředí ve virtuální realitě.

1. Analyzujte aktuální stav projektu, zaměřte se zejména na import modelů a REST komunikaci.
2. Proveďte analýzu zásuvných modulů s podobnou funkčností z pohledu komunikace s databází, importu 3D modelů a funkčnosti s Unreal Enginem.
3. Zhotovte obecný postup k rozšiřování funkčností Unreal Enginu.
4. Na základě předchozích analýz definujte požadavky VR aplikace.
5. Pomocí metod softwarového inženýrství navrhněte a implementujte prototyp zásuvného modulu.
6. Výsledný zásuvný modul podrobte vhodným testům.

Elektronicky schválil/a Ing. Radek Richtr, Ph.D. dne 3. února 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Věnná města Českých Královen - Import 3D modelů do Unreal Engine 4

Tadeáš Bušek

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Chludil

13. května 2021

Poděkování

Rád bych věnoval poděkování vedoucímu mé práce Ing. Jiřímu Chludilovi za cenné odborné rady a průběžné konzultace, které mi neúnavně poskytoval po celou dobu tvorby práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Bušek, Tadeáš. *Věnná města Českých Královen - Import 3D modelů do Unreal Engine 4*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zabývá vývojem zásuvného modulu, který bude zajišťovat import 3D modelů z databáze projektu Věnná města českých královen do Unreal Engine 4. Součástí práce je analýza projektu Věnná města českých královen a dále analýza možností vývoje pro Unreal Engine 4. Kromě analýzy práce obsahuje přehled požadavků kladených na vyvíjený zásuvný modul, návrh uživatelského rozhraní a návrh řešení, který vyplývá z předchozí analýzy. Výstupem práce je popis způsobu realizace zásuvného modulu a jeho funkční implementace.

Projekt Věnná města českých královen obsahuje 3D modely historických objektů, které je možné díky vyvinutému zásuvného modulu použít při práci v Unreal Engine 4. Předpokládá se využití především ze strany historiků, kteří tak budou mít možnost prezentovat objekty, které byly dříve vytvořeny v rámci projektu Věnná města českých královen. Hlavním přínosem této práce je tedy zpřístupnění 3D modelů historických objektů pro zobrazení v Unreal Engine 4.

Klíčová slova Věnná města českých královen, REST API, Unreal Engine 4, Zásuvný modul, Import 3D modelů, Blueprint třída

Abstract

This study describes the development of the plugin, which will provide import of the 3D models from the database of the project Dowry Towns of the Queens of Bohemia into the Unreal Engine 4. This document contains analysis of the project Dowry Towns of the Queens of Bohemia and also analysis of the development possibilities for the Unreal Engine 4. Besides of the analysis document contains overview of the requirements, which should be fulfilled by the developed plugin, design of the user interface and also solution overview as the result of the previous analysis. As the outcome of the study there is description of the plugin realization and its implementation.

Project Dowry Towns of the Queens of Bohemia contains 3D models of the historical objects. Thanks to the developed plugin it is possible to use these models during the work in the Unreal Engine 4. It is supposed, that the historians will be the main users of the plugin, which will give them possibility to present objects created in the project Dowry Towns of the Queens of Bohemia. The main benefit of this work is access to the 3D models of the historical object for the presentation in Unreal Engine 4.

Keywords Dowry Towns of the Queens of Bohemia, REST API,Unreal Engine 4, Plugin, Import of 3D models, Blueprint class

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Unreal Engine 4 [2] (dále jen Uneral Engine) je velmi používaný herní engine, sloužící převážně k vývoji her a následně k umožnění jejich běhu. Popularita Unreal Engine stále roste a nyní patří spolu s Unity [3] mezi nejpoužívanější enginy vůbec [4].

Právě pomocí Unreal Engine se vyvíjí i část rozsáhlého projektu, zabývajícího se výzkumem, prezentací a digitalizací infrastruktury a historického vzezření Věnných měst českých královen. Celý projekt pak nese název Věnná města českých královen (dále jen VMČK). [5] Na vývoji projektu se podílí i fakulta informačních technologií ČVUT v Praze, na které vznikalo a vzniká velké množství bakalářských prací a projektů úzce spjatých s tématem VMČK.

Jednou z obhájených bakalářských prací je práce na téma VMČK jádro [6]. Tato práce se zabývá návrhem databáze modelů a přilehlého API, skrz které půjde s databází manipulovat.

Dalším projektem spadajícím pod projekt VMČK a vyvíjeným na fakultě informačních technologií ČVUT v Praze je projekt s názvem Virtuální muzeum. Ten je vyvíjen právě v Unreal Engine a zabývá se vytvořením virtuálního muzea pro 3D modely. Muzeum mělo být původně prostředím ve virtuální realitě, ve kterém by bylo možné pomocí VR (Virtual reality) headsetu prozkoumávat muzejní exponáty a případně s nimi i interagovat. Exponáty by bylo možné zvětšovat, různě rozřezávat, otáčet a mnoho dalších věcí.

V rámci předmětu softwarový projekt se mi naskytla příležitost se na tomto projektu také podílet. Po roce stráveném na jeho vývoji za pomoci Unreal Engine jsem se rozhodl pokračovat s projektem i v mé bakalářské práci.

V nedávné době se však od původního záměru upustilo a nyní se pracuje na myšlence virtuálního muzea sloužícího jako platforma pro historiky s využitím k prezentaci vymodelovaných objektů ostatním historikům.

I přesto, že se původní myšlenka projektu změnila, u obou podob virtuálního muzea chybí funkcionalita, která by napojila vyvíjený projekt v Unreal Engine na API databáze VMČK. Z tohoto nedostatku vzniklo také téma mé

bakalářské práce.

Má bakalářská práce bude tedy do projektu VMČK přispívat vývojem prototypu zásuvného modulu pro Unreal Engine, který bude zajišťovat import 3D modelů z API databáze do projektů vytvářených v Unreal Engine. Tato funkcionalita bude dostupná i při spuštěném projektu.

Práce je určena jak pro čtenáře zabývající se problematikou projektu VMČK, tak pro čtenáře toužící dozvědět se více o vývoji zásuvných modulů pro Unreal Engine.

Práce bude tedy zkoumat Unreal Engine a možnosti pro rozšíření jeho funkcionalit za pomocí takzvaných zásuvných modulů. Dále pak bude obsahovat následnou implementaci zásuvného modulu, který pomocí REST technologie, bude řešit import 3D modelu za běhu aplikace z API do Unreal Engine.

Kapitola 1

Cíl práce

Hlavním cílem této práce je umožnit import 3D modelů z databáze VMČK do Unreal Engine. Pro tyto účely bude vyvinut prototyp zásuvného modulu pro Unreal Engine, který bude tuto funkcionalitu zajišťovat. Prototyp bude schopen importovat 3D modely (nejlépe za běhu aplikace). Import bude probíhat prostřednictvím REST komunikace přes API databáze VMČK. Pro dosažení cíle práce bude provedena analýza možností importu objektů do Unreal Engine, dále analýza již existujících zásuvných modulů s podobnou funkcionalitou, analýza možností REST komunikace v Unreal Engine a v neposlední řadě analýza struktury API poskytovaného VMČK.

Důležitým prvkem práce bude také analýza a nalezení vhodné technologie pro vývoj, protože Unreal Engine nabízí veliké množství nástrojů a zásuvných modulů, ať už oficiálních nebo přidaných komunitou.

Kapitola 2

Analýza

2.1 Projekt VMČK

Jedná se o rozsáhlý projekt, který má za cíl seznámit veřejnost s kulturním dědictvím Věnných měst českých královen. Mimojiné toho dosahuje za pomoci moderních nástrojů počítačové grafiky a informačních technologií. Nyní již existuje jádro projektu, skládající se z databáze naplněné 3D modely a komunikačních API. Jádro projektu je důkladně popsáno v bakalářské práci [6].

Další již existující, ale nedokončenou částí, je virtuální muzeum, jež je vytvořené v Unreal Engine 4. Funkcí virtuálního muzea mělo být původně přenesení vystavovaných exponátů do čistě virtuálního prostředí, po rozhovoru s ředitelem muzea východních Čech v Hradci Králové panem doc. Mgr. Petrem Grulichem, Ph.D., se však od původního záměru upustilo, více se této problematice budu věnovat v kapitole 2.3. I přesto, že z původního plánu virtuálního muzea sešlo, je tato část projektu hlavním důvodem vzniku mé bakalářské práce.

Zbývajícími částmi projektu jsou zásuvné moduly pro blender[7] a 3D StudioMax[8], které slouží k propojení výše zmíněných aplikací a projektu VMČK, prototyp mobilní aplikace pro zobrazování historických 3D modelů v rozšířené realitě, virtuální aplikace vytvořená v Unreal Engine pro úpravu modelů, která bude v budoucnosti sloužit jako nástroj pro tvorbu virtuálních měst a vznikající 3D digitální repliky historických objektů. Tato témata však nejsou hlouběji spojená s mojí prací a proto se jim v této práci již nebudu věnovat.

2.2 Jádro projektu VMČK

Mimo jiné je úkolem mé práce umožnit import 3D modelů získaných z databáze jádra projektu VMČK. Ke komunikaci s databází slouží REST API

vytvořené kolegou Danielem Vančurou v rámci bakalářské práce [6].

V jádru projektu VMČK je kladen veliký důraz na možnost ukládání 3D modelu ve více podobách (v řádu desítek až stovek). Důvodem je, že projekt VMČK se snaží o co největší podobu s realitou, proto je důležité, aby byl model dostupný v podobách odpovídajících podmínkám, do kterých je umístěn. Pokud je například model umisťován do rozbořeného města, je žádoucí, aby byla v databázi dostupná i rozbořená podoba umisťovaného 3D modelu nebo pokud je model umisťován do zasněžené krajiny, požadujeme po databázi zasněženou podobu 3D modelu, atd.

V mé práci bude tedy velmi důležité při přístupu do databáze počítat s možnou existencí až stovek podob modelů a nalézt způsob, jak dané podoby reprezentovat a zobrazit uživateli.

Kromě existence vícero podob 3D modelu umožňuje jádro projektu VMČK i existenci vícero verzí 3D modelu.

> *„Pro podporu modelování a snadnější realizaci zpětné vazby byl vytvořen systém verzování 3D objektů. V API jsou tedy navrženy příslušné endpointy pro úpravu a vytváření nových verzí, případně pro získání celé historie daného záznamu.“ [6]*

Tato funkce je implementovaná z důvodu co největšího ulehčení případných oprav a vylepšování již uložených 3D modelů.

Protože verze slouží hlavně pro modeláře a jejich úpravy, v mé práci se bude zobrazovat pouze verze nejnovější. Uživatel však bude upozorněn, pokud jeho importovaná verze bude zastaralá a bude mu umožněna aktualizace 3D modelu.

Další implementovanou funkcí v jádru projektu VMČK je možnost jednoduchého vyhledávání mezi 3D modely. Vyhledávání probíhá pomocí GET requestu na API obsahujícího vyhledávácí parametry.

Poslední funkcionalitou zde zmíněnou je povinnost využívání uživatelských účtů, které mají rozdílná práva podle přidělených rolí.

> *V projektu jsem navrhl pro začátek a jednoduchost jen pár endpointů pro správu uživatelů a přístupových práv. Momentálně se při vytvoření nového uživatele nastaví jeho status na „new“ a dokud není „schválen“ (admin pošle PUT request na příslušný endpoint se změněným stavem a/nebo seznamem práv) tak mu není přidělena ani role ani žádná přístupová práva. [6]*

2.3 Virtuální muzeum

Virtuální muzeum měl být původně projekt vytvářený pro česká muzea, který by umožňoval vytváření virtuálních expozic pro veřejnost, které by mohly nahradit expozice reálné. Původní návrh byl předveden panu doc. Mgr. Petrovi Grulichovi Ph.D., který je ředitelem muzea východních Čech v Hradci

Králové. Ten v rozhovoru oznámil, že Virtuální muzeum je velmi zajímavá myšlenka, ale organizátoři muzeí chtěji především nalákat veřejnost do budovy muzea. Muzea by měla být funkční i bez digitálních technologií a expozice z reálných exponátů by měla být vždy hlavním prvkem. Dalšími problémy původní myšlenky virtuálního muzea je absence fyzického kontaktu a fakt, že často je i samotná budova muzea historickým exponátem.

I přesto, že pro veřejnost je tedy projekt Virtuálního muzea nezajímavý, cílovou skupinu projektu je možno přesunout na historiky.

Podle poznatků vedoucího této práce pana Ing. Jiřího Chludila, který učí na katedře pomocných věd historických Filozofické fakulty Univerzity Hradec Králové předmět 3D modelování, jsou historici schopni vytvářet 3D modely ať už za pomoci 3D modelování, či pokročilejších technik jako je 3D sken nebo fotogrammetrie. Historiky k používání modelování motivuje možnost digitalizace historických předmětů, které nejsou snadno dostupné a je možné si je pouze zapůjčit, výsledný 3D model lze následně poskytnout studentům, historikům, či veřeknosti jakožto digitální doplněk výstavy.

S ohledem na nedostatečnou odbornost historiků v počítačové grafice však 3D modely dodané historiky obvykle obsahují větší množství polygonů, než je pro strukturu daného modelu potřeba. Z tohoto důvodu je vyšší výskyt chyb a vad ve výsledném 3D modelu. Většina chyb se v 3D modelovacím nástroji, jako je například Blender [7], neprojeví, a proto je těžké je rozeznat. Dalším velkým problémem je pravděpodobné násobení počtu vytvořených chyb při převodu 3D modelu na jiný formát, tím se chyby stávají těžšími na odstranění a je možné, že více ovlivní strukturu výsledného modelu. K částečnému řešení tohoto problému se dá využít nová myšlenka virtuálního muzea jakožto platformy pro historiky, sloužící pro kontrolu kvality 3D modelů ve virtuální realitě a jakožto sdílený virtuální prostor pro prezentaci modelů ostatním historikům.

2.4 Možnosti Unreal Engine

Unreal Engine v současné době umožňuje tři způsoby vývoje. Vývoj za pomocí takzvaných blueprint tříd, vývoj v jazyce C++ a vývoj v jazyce python. Následující sekce je věnována těmto různým způsobům a jejich výhodám a nevýhodám.

2.4.1 Blueprint třída

Takzvaná blueprint třída, občas referovaná pouze jako blueprint, je asset, který umožňuje rozšíření funkcionalit za pomoci vizuálního programování. Tohoto lze dosáhnout pomocí uzlů spojovaných v takzvaném graph editoru. Uzly jsou krabičky obsahující určitou funkcionalitu se vstupy a výstupy. Tyto krabičky se dají mezi sebou díky vstupům a výstupům spojovat takzvanými dráty a tím vytvářet složitější funkcionality. Blueprint třída může také obsahovat proměnné, eventy, funkce, či je k ní možné připojit komponentu (například

static mesh [9] nebo Procedural Mesh komponentu [10]). Bluepinrt třídy jsou také schopné principu dědění, čímž jsou schopné dědit vlastnosti od svých rodičů, či předávat své vlastnosti svým potomkům.

Výhodou blueprint třídy je, že po uživatelovi není vyžadována žádná znalost programovacích jazyků. Pokud však uživatel využívá pouze metodu blueprint tříd, je omezen na již vytvořené funkcionality a jejich kombinaci, proto se jedná o metodu krajně limitující. I přes dané nevýhody se stále jedná o velice využívanou variantu pokrývající velmi rozsáhlé spektrum funkcí.

2.4.2 C++

Unreal Engine je převážně napsán v jazyce C++ a je rozdělený do modulů. Každý modul nese určitou funkcionalitu a je pro uživatele dostupný a využitelný i v jiném modulu. Tento přístup tedy umožňuje programátorovi zahrnutím odpovídajícího modulu do jeho kódu využívat jakékoliv veřejné funkcionality, které využívá Unreal Engine [2].

Unreal Engine [2] například využívá pro své uživatelské rozhraní modul s názvem Slate, pokud je tento modul správně zahrnut do projektu uživatele, je v něm možné tvořit idnetické rozhraní, jako obsahuje Unreal Engine.

Velkou výhodou vývoje za pomoci C++ je možnost přidávat do Unreal Engine [2] chybějící funkcionality napsaným kódem, či využitím externích knihoven.

Skrze psaní C++ kódu je možné vytvořit nové typy assetů, přidat v rámci uzlů nové funkcionality do blueprint třídy, či pomocí zásuvných modulů upravovat funkcionality samotného Unreal Engine editoru.

2.4.2.1 Zásuvné moduly

Unreal Engine umožňuje rozšiřování svých funkcionlit za pomocí zásuvných modulů.

> *„V Unreal Engine 4, jsou zásuvné moduly souborem kódů a dat, které vývojáři mohu jednoduše povolit, či deaktivovat pomocí editoru na projektové bázi. Zásuvné moduly mohou přidat runtime herní funkcionality, modifikovat vestavěné Engine funkcionality (nebo přidat nové), vytvářet nové typy souborů a rozšířit vlastnosti editoru s novými menu, tool bar příkazy a sub-mody. Mnoho existujících UE4 subsystému bylo navrženo tak aby se dali rozšiřovat pomocí zásuvných modulů.“ [1]*

V Unreal Engine editoru je pro zásuvné moduly přímo sekce v menu. V této sekci jsou vidět všechny dostupně zásuvné moduly a jak je výše zmíněno, Unreal Engine zde umožňuje různé zásuvné moduly aktivovat či deaktivovat. Mimojiné umožňuje taktéž pomocí tlačítka „New Plugin“ vytvořit novou šablonu pro zásuvný modul podle zvoleného typu. Dostupné šablony jsou pro

Typ Zásuvného modulu	Vyhledávací cesta
Unreal Engine	/[UE4 Root]/Engine/Plugins/[Plugin Name]/
Hra/Projekt	/[Project Root]/Plugins/[Plugin Name]/

Tabulka 2.1: Tvary cest, ve kterých Unreal Engine vyhledává zásuvné moduly, přebráno z Unreal Engine dokumentace [1]

prázdný zásuvný modul (pouze základ), pro vytvoření nové knihovny bluperintů, pro vytvoření samostatného okna dostupného v menu editoru, pro přidání nástroje pro úpravu scény, pro vytvoření nového módu pro editor, pro zahrnutí externí knihovny a pro zahrnutí pouze dodatečného obsahu. Poslední šablona narozdíl od ostatních neumožňuje dodatečné psaní kódu a hodí se pouze pro předávání například assetů mezi projekty.

Aby však bylo zásuvné moduly možno nalézt, je potřeba je umísit do správné složky, ve kterých pak Unreal Engine dané zásuvné moduly vyhledává. Tyto složky jsou dvě a na volbě složky závisí, zda bude zásuvný modul dostupný pouze v určitém projektu nebo napříč všemi projekty. Pokud je zvolena možnost napříč všemi projekty, je potřeba vybrat složku pro zásuvné moduly přímo v kořenovém adresáři Unreal Engine. Pokud je zvolena možnost pouze pro daný projekt, je potřeba zásuvný modul umístit do složky nacházející se v kořenovém adresáři konkrétního projektu. Bližší pohled na strukturu složek nabízí následující tabulka 2.1.

Každý zásuvný modul má vlastní zdrojovou složku obsahující předem danou strukturu. Ve struktuře lze najít například složku „Content“, která obsahuje assety přidružené k zásuvnému modulu, složku „Resources“, která obsahuje pouze ikonku zobrazovanou u zásuvného modulu při jeho aktivování, velmi důležitou je složka „Source“, která obsahuje všechen kód uspořádaný do struktury modulu. Do zbylých složek programátor z velké části případů není nucen zasahovat, dané složky jsou následující:

Složka „Binaries“ obsahuje zkompilovaný kód, ve složce „Intermediate“ jsou uložené dočasné zbuildované soubory a do složky „Config“ se ukládají soubory konfigurační. [1]

Většina zásuvných modulů obsahuje ve složce „Source“ pouze jeden modul, ale je možné, aby jich měl zásuvný modul více. Aby však UnrealBuildTool věděl, z jakých modulů se zásuvný modul skládá, je potřeba dané moduly zapsat do souboru *.uplugin*. Soubor *.uplugin*se nachází taktéž ve složce „Source“ a obsahuje základní informace o daném zásuvném modulu. Dále je velmi důležité, aby každý modul obsahoval svůj vlastní *Build.cs*soubor. Tento soubor obsahuje informace o závislostech na ostatních modulech, externích knihovnách a říká UnrealBuildTool, jak má daný modul sestrojit a zkompilovat.

2.4.3 Python

Unreal Engine [2] umožňuje ke skriptování používat Python 3.

> „*Python může být v Unreal Engine použit k věcem jako:*
>
> - *Vytvoření pipeline, sloužících k řízení assetů větších rozměrů, či workflow, které napojují Unreal Editor k ostatním 3D aplikacím, používaných k organizaci.*
> - *Automatizování zdlouhavých úloh řízení assetů v Unreal Editoru, například generování úrovně detailu (LODs) pro static mesh.*
> - *Procedurálně rozvrhnout obsah v úrovních.*
> - *Kontrolování Unreal Editoru pomocí uživatelského rozhraní vytvořeného v Python.*
>
> “ [11]

Aby bylo možné využít Python v Unreal Engine [2] je nutné aktivovat k tomu sloužící zásuvný modul se jménem Python Editor Script Plugin [11]. Tento zásuvný modul je dostupný již v základní instalaci Unreal Engine [2].

2.5 Již existující možnosti importu do UE4

2.5.1 Import skrze uživatelské rozhraní Unreal Engine

Tato metoda patří mezi ty nejzákladnější a také nejjednodušší k použití, i když nenabízí žádné možnosti automatizace a napojení na již vzniklý projekt VMČK. Jedná se o základní metodu, ke které postačí pouze základní Unreal Engine bez dalších přidaných nástrojů. Při použití této metody vytvoří Unreal Engine dialogové okno, které nabízí mnoho možností, jako například importovat model i s texturami a materiály, vytvořit kolize s ostatními modely ve scéně, určit počáteční trasnfomraci modelu, importovat více modelů najednou a další pokročilejší možnosti. I přesto, že tento způsob je díky uživatelskému rozhraní k uživatelům přívětivý, import 3D modelů do Unreal Engine je stále složitá činnost a je potřeba již nějaká zkušenost se strukturou 3D modelů a prací s Unreal Engine. Projekt, pro který je můj zásuvný modul určen, bude však používán převážně historiky s ne příliš vysokými zkušenostmi v oblasti informačních technologií. Proto tento způsob importu není vyhovující, může však posloužit jako inspirace pro můj vyvíjený zásuvný modul.

2.5.2 Runtime Mesh Loader

Jedná se o jednoduchý Unreal Engine zásuvný modul napsaný v C++, který ke své funkcionalitě využívá knihovnu Assimp [12]. Assimp je open-source

C++ knihovna, která umožňuje import téměř všech používaných formátů 3D modelů.

Runtime Mesh Loader zpřístupní po aktivování v Unreal Engine 4 nový typ blueprintu. Daný blueprint dále umožňuje podle zadané cesty k souboru načíst daný 3D model a získat informace o jeho struktuře. Tyto informace jsou použity k sestrojení 3D modelu ve scéně Unreal Engine projektu za běhu aplikace.

Po analýze ostatních řešení a možností je tato metoda jediná, která umožňuje import modelu za běhu aplikace, proto se v mém řešení bude z tohoto zásuvného modulu vycházet. Tento zásuvný modul má však stále problémy. Mezi jeho nedostatky pro mé řešení je neumožnění komunikace s REST API a k jeho využití nutnost složitého napojování pomocí blueprintů.

2.5.3 UnrealEnginePython

Některé funkcionality zabudované v Unreal Engine využívají ke svému běhu Python. Z tohoto důvodu má Unreal Engine přímo v sobě integrovaný Python 3. K použití Unreal Engine Python z pozice programátora je však potřeba použít oficiální zásuvný modul se jménem Python Editor Script Plugin [11]. Po aktivaci zásuvného modulu je umožněn přístup do python konzole, kde je možno provádět python příkazy, či spustit python script. K práci s Unreal Engine existuje přímo Unreal Engine knihovna, ke které je na stránkách Unreal Engine oficiální dokumentace[11].

2.5.3.1 UnrealEnginePython import script

K importu 3D modelů do Unreal Engine existuje python script [13]. Tento script pro svou práci využívá Unreal Engine knihovnu Asset Tools [14], díky které je schopen se postarat o import 3D modelů a jeho animací. Tento script je však funkční pouze na model, pro který byl napsán, protože se v kódu odkazuje na části konkrétního modelu a textury. K použití scriptu na jiný model by bylo potřeba zjistit strukturu daného modelu a v kódu se odkazovat na ní. Z důvodu již zmíněného výše je však přepisování kódu a práce s python konzolí pro cílové uživatele příliš složitá. Script také neumožňuje komunikaci s REST API ani import za běhu aplikace, tudíž se jedná o nevyhovující řešení.

2.5.4 Blender zásuvný modul s názvem Blender for Unreal

Existuje velice vydařený zásuvný modul pro blender[7], nesoucí jméno Blender for Unreal[15], který zaručuje export modelů z Blender přímo ve formátu a struktuře vhodné pro Unreal Engine. Spolu s exportovaným modelem vytvoří zásuvný modul python script, zajišťující import konkrétního modelu do Unreal Engine. Skript je pro import nutné spustit skrze UnrealEnginePython. I přesto, že i tato metoda nesplňuje komunikaci s REST API, import

modelu za běhu aplikace ani jednoduchost pro uživatele, jedná se o velmi kvalitní zásuvný modul s funkčností blížící se řešení problematiky této bakalářské práce. Mezi problémy tohoto řešení ale patří fakt, že tento zásuvný modul je vytvořený v programu Blender, tudíž znemožňuje napojení na Unreal Engine část projektu VMČK a k použití totoho zásuvného modulu je potřeba mít 3D modely v programu Blender. Jedná se však o kvalitní inspiraci pro mé řešení.

2.6 Obsažené funkcionality a použité technologie

2.6.1 Registrace a přihlašování

Pro přístup k databázi je potřeba mít vytvořený uživatelský účet, proto můj zásuvný modul umožní registraci i přihlášení. K implementaci registrace a přihlašování budou využity k tomu určené endpointy přístupné na API jádra projektu VMČK. Jádro projektu VMČK také bude zajišťovat správu rolí a práv, tudíž v mé práci bude pouze potřeba vytvořit odpovídající uživatelské rozhraní, napojit ho na již vytvořené API a ošetřit nesprávné vstupy.

2.6.2 Komunikace s REST API

Aby komunikace s databází byla vůbec možná, je potřeba, aby vytvářený zásuvný modul disponoval technologií schopnou REST komunikace. V tomto případě funguje databáze s API jakožto server, na který jsou odesílány požadavky na potřebná data mým zásuvným modulem. Výhodou této komunikace je, že vývoj klienta může být na serveru naproso nezávislý a ke komunikaci stačí pouze znát a dodržovat formát posílaných zpráv.

REST komunikace je tedy využívána ke snadnému přístupu k určeným datům, přičemž existují čtyři typy požadavků. Požadavek GET, který zajišťuje získání potřebných dat, požadavek POST, zajišťující vytvoření nových dat, požadavek DELETE, který smaže již existující data a požadavek PUT, který slouží k aktualizaci určených dat.

Můj zásuvný modul bude využívat pouze GET požadavek na získávání dat o modelech a samotných modelů. K požadavkům mého zásuvného modulu na REST komunikaci postačí zásuvný modul VaRest[16].

2.6.3 Zásuvný modul VaRest

VaRest[16] je nejvíce stabilní a nejznámější způsob řešení REST komunikace se serverem v Unreal Engine [2]. Jedná se o zásuvný modul, který poskytuje nástroje k vytváření všech typů požadavků (GET, POST, DELETE nebo PUT) a k jejich odesílání serveru. VaRest také umožňuje vytvářet a spravovat JSON objekty, v těch je pak možno v případě požadavků upravit hlavičku, což se může hodit například pro autentizaci a tělo objektu. V případě odpovědi od serveru je z objektu možné získávat pomocí klíčových názvu potřebná data v

podobě JSON pole, stringu a dalších formátů. Vše je vytvořeno jakožto funkcionalita pro takzvané blueprint třídy (více ohledně Unreal Engine a blueprint třídách v sekci 2.4.1).

Zásuvný modul je vyvíjen nezávislým vývojářem Vladimirem Alyamkinem, který do dnešního dne zásuvný modul aktualizuje a spravuje. K dostání je pak na oficiálním Unreal Engine marketplace [17] zcela zdarma.

2.6.4 Zobrazení 3D modelů

Před započetím importu samotného modelu do Unreal Engine [2] je potřeba uživateli zprostředkovat nástroje, kterými si bude moci konkrétní model k importu vybrat. Problémem je, že 3D modely jsou v API obsažené pouze jako součást takzvaného 3D objektu. 3D objetkům jsou pak nadřazené takzvané struktury, které reprezentují určitou budovu.

S nynějším stavem API se tedy nabízejí dvě možnosti, jak zobrazení 3D modelů řešit. První možností je pomocí požadavku na API zobrazovat v uživatelském rozhraní všechny struktury, pod ty zařadit příslušné 3D objekty a z 3D objektů získávat požadované modely. Druhou možností je struktury z této práce vynechat a využít možnost API získávat všechny dostupné 3D objetky přímo. Pro rozhodnutí, jaká možnost bude pro práci vhodnější, bude nutné počkat na stav API v době implementace. Po analýze stavu API a po konzultaci s vedoucím práce vyšlo najevo, že se API stále vyvíjí a je možné, že vazby mezi strukturami a 3D objekty budou podrobeny změnám.

Důležitou součástí problému je určení reprezentace struktur, 3D objetků a modelů v uživatelském rozhraní. Tím je myšleno jaké informace získané z API bude vhodné zobrazit v uživatelském rozhraní jakožto reprezentaci daného objektu. V případě 3D objektu bude po analýze k reprezentaci využito jméno 3D objektu, stav 3D objektu, jméno obsaženého modelu a verze. Podobnou reprezantaci vuyžívá ve svém uživatelském rozhraní i zásuvný modul pro software Blender [7], který je nyní vyvíjen vedoucím této práce a který zajišťuje propojení mezi API jádra projektu VMČK[6] a softwarem Blender[7]. Verzi bude možno v uživatelském rozhraní přepínat a tím měnit verzi importovaného objektu.

2.6.5 Import 3D modelů

Poté, co bude vybrán konkrétní model k importu, bude potřeba daný model importovat do Unreal Engine [2]. Z analýzy možností Unreal Engine[2] a již existujících řešení vyplynuly dvě rozdílné cesty implementace, ze kterých si bylo nutné vybrat. Obě cesty mají své výhody a úskalí a tak nebylo jednoduché rozhodnout, kterou z cest zvolit.

2.6.5.1 Vytvoření nového typu assetu - cesta 1

První cesta by přidala do Unreal Engine [2] pomocí C++ zásuvného modulu nový typ assetu. Tento nový typ assetu by držel všechna potřebná data a zajišťoval by podobnou funkcionalitu jako asset, který vznikne základním importem 3D modelu poskytovaným od Unreal Engine [2]. Pro umožnění importu z API by bylo potřeba vytvořit uživatelské rozhraní za pomoci Unreal Engine[2] modulu Slate a vytvořit za pomoci Unreal Engine [2] HTTP modulu možnost komunikace s REST API.

Pro vytváření instancí assetu by bylo nutné vytvořit takzvané factories. Factories slouží k vytváření nových instancí nového typu assetu. Unreal Engine [2] poskytuje již jejich předdefinované typy, kterým je možné za pomoci override upravit funkcionalitu dle potřeby. Mezi předdefinované factories patří například factory pro Drag & Drop nebo factory pro vytvoření assetu z context menu menu. K samotné funkcionalitě importu by byla využita třída FFbxImporter, která danou funkcionalitu poskytuje.

Výhodou této implementace je import do assetu, se kterým se dá následně v Unreal Engine [2] editoru dále pracovat. Problémem je však nemožnost importu za běhu aplikace a nemožnost využití již vytvořených funkčních řešení pro REST komunikaci.

2.6.5.2 Runtime Import - cesta 2

Druhá cesta by využila pro import takzvané Procedural Mesh komponenty [10]. Tato komponenta poskytuje možnost procedurálního generování meshe pomocí zadaných dat o modelu. Toto zvládá procedurální mesh komponenta za běhu aplikace, a proto zde vyvstává první výhoda tohoto způsobu, kterou je umožnění importu přímo za běhu. Data nutná k vygenerování meshe, kterými jsou například informace o trojúhelnících, vrcholech či normálách, by se získala pomocí mírně upraveného zásuvného modulu RuntimeMeshLoader [18]. Tento zásuvný modul využívá ke své funkcionalitě C++ knihovnu Assimp [12] a věnuje se mu kapitola 2.5.2.

Poté, co by se pomocí blueprint tříd napojila funkcionalita zásuvného modulu RuntimeMeshLoader [18] na Procedural Mesh komponentu, by bylo nutné vše napojit na vytvořené uživatelské rozhraní. Ke komunikaci s API projektu VMČK by se využily funkcionality, přidané do blueprint tříd, které zpřístupňuje zásuvný modul VaRest. [16].

2.6.5.3 Zvolená cesta

Po analýze obou cest a diskusi s vedoucím o záměrech této práce bylo rozhodnuto vydat se cestou druhou, která obnáší práci s Procedural Mesh komponentou [10]. Důvodem pro toto rozhodnutí je z hlavní části možnost importovat model za běhu aplikace. Dalšími důvody je již existující zásuvný modul pro

REST komunikaci a základní import za běhu aplikace, oba dva zásuvné moduly však lze využít pouze za použití blueprint tříd, které jsou s první cestou neslučitelné.

Důvodem, proč si nevybrat cestu první, je mimo jiné nedostatečná dokumentace využívaných Unreal Engine [2] funkcí a tudíž i nejistota zdárného výsledku.

2.7 Požadavky

Tato sekce se zabývá funkčními a nefunkčními požadavky na vyvíjený zásuvný modul. Požadavky byly sestaveny za pomoci konzultací s vedoucím práce a budou sloužit jako jeden z podkladů pro vytvoření návrhu.

Požadavky vznikaly v době, kdy ještě nebylo přímo jasné, jak bude stále se vyvíjející API vypadat, a že bude potřeba zvolit pouze jednu ze dvou možností importu 3D modelu, zmíněné v sekci 2.6.5. Z tohoto důvodu jsou zde zmíněné i požadavky, o kterých již nyní víme, že nebudou splněny.

2.7.1 Funkční požadavky

Funkční požadavky definují, jaké funkcionality by měl zásuvný modul splňovat, to jest co by měl zásuvný modul umět. Tato sekce je rozdělena na dvě podsekce, kde se první věnuje funkcionalitám spojeným s uživatelským rozrhraním a druhá vnitřní funkcionalitě samotného zásuvného modulu.

2.7.1.1 Uživatelské rozhraní

Funkční požadavky zmíněné v této podsekci jsou úzce spojeny s tvorbou uživatelského rozhraní pro vytvářený zásuvný modul.

F1: Přihlášení uživatelů Zásuvný modul umožní registraci a následný přístup do databáze jádra projektu Věnná města českých královen. Zadané přihlašovací údaje budou ověřovány v databázi registrovaných uživatelů za pomoci API jádra projektu VMČK [6], které následně bude také zajišťovat vygenerování tokenu využívaného k následnému přístupu a nastavení příslušných přístupových práv.

F2: Uložení tokenu Zásuvný modul umožní uložení tokenu na disku. To zajistí automatizaci přihlášení do jádra projektu VMČk [6] a přístup do jeho databáze.

F3: Zobrazení dostupných modelů Zásuvný modul umožní zobrazení seznamu modelů, které jsou dostupné v databázi jádra projektu VMČK [6]. Pro tuto úlohu zobrazí zásuvný modul uživatelské rozhraní, kde bude jednotlivé modely v seznamu reprezentovat pomocí jména, náhledu v podobě obrázku a aktuální verze modelu. V seznamu bude možno nastavit

počet zobrazovaných modelů a mezi nimi se bude procházet pomocí posuvníku.

F4: Filtrování a vyhledávání Zásuvný modul umožní v seznamu dostupných modelů filtrovat a vyhledávat. Tohoto bude docíleno filtrováním a vyhledáváním implementovaném v API jádra projektu VMČK [6]. Filtrovat bude možné podle názvu modelů, ale také podle verze, či podle specifikací modelu, které jsou reprezentovány vyhledávacími tagy.

F5: Zobrazení více variant modelu V databázi jádra projektu VMČK [6] se může model vyskytovat ve více variantách, například bude model poničený, zasněžený a tak dále. Proto zásuvný modul bude schopen po výběru modelu v seznamu dostupných modelů zobrazit i všechny varianty daného modelu.

F6: Zobrazení importovaných modelů Zásuvný modul si bude schopen na disku držet informaci o již importovaných modelech a bude přes uživatelské rozhraní informovat, pokud se v databázi jádra projektu Věnná města českých královen s daným modelem něco stane (bude dostupná novější verze modelu nebo bude model smazán).

2.7.1.2 Funkce zásuvného modulu

Funkční požadavky zmíněné v této podsekci nemají žádné větší vazby na tvorbu uživatelského rozhraní nýbrž na samotné možnosti importu.

F7.1: Import modelu do Unreal Engine 4 Zásuvný modul bude schopen importovat vybraný model ve formátu fbx jakožto nový asset do Unreal Engine [2]. Tímto způsobem bude model ihned připraven k následnému použití v projektu. Model bude možné vybrat buď z disku počítače, či z databáze jádra projektu VMČK [6].

F7.2: Runtime import modelu do Unreal Engine 4 Zásuvný modul bude schopen importovat vybraný model i přímo do spuštěného projektu. K tomuto účelu bude jako základ využíván již hotový Unreal Engine [2] zásuvný modul se jménem RuntimeMeshLoader [18].

F7.3: Import více formátů Zásuvný modul umožní import všech formátů, které podporuje C++ knihovna Assimp [12]. Podporovanými formáty jsou následující

> „*3DS, BLEND (Blender), DAE/Collada, IFC-STEP, ASE, DXF, HMP, MD2, MD3, MD5, MDC, MDL, NFF, PLY, STL, X, OBJ, OpenGEX, SMD, LWO, LXO, LWS, TER, AC3D, MS3D, COB, Q3BSP, XGL, CSM, BVH, B3D, NDO, Ogre Binary, Ogre XML, Q3D, ASSBIN (Assimp custom format), glTF (partial), 3MF.*“ [18]

F8: Import modelu do různých složek Zásuvný modul umožní výběr destinace v Unreal Engine [2] projektu, do které bude model importován.

F9.1: Vložení modelu do Unreal Engine 4 Pokud se bude v destinaci importu již nacházet model se stejným názvem jako nově importovaný model, umožní zásuvný modul nahrazení starého modelu novým.

F9.2: Aktualizace modelu v Unreal Engine 4 Pokud bude v databázi jádra projektu VMČK [5] dostupná nová verze již importovaného modelu, zásuvný modul umožní aktualizaci zastaralého modelu.

F9.3: Odebrání modelu z Unreal Engine 4 Zásuvný modul umožní odebrání modelu, uloženého do Unreal Engine [2] v rámci funkcionality zásuvného modelu.

2.7.2 Nefunkční požadavky

Nefunkční požadavky určují požadované kvality a omezení zásuvného modulu.

NF1: Technologie Úkol bude vypracován jakožto zásuvný modul do Unreal Engine 4 [2]. Kód záusvného modulu bude napsán v C++, případně doplněn využitím technologie blueprint.

NF2: Dokumentace Kód zásuvného modulu bude náležitě okomentován a bude dostupná technická dokumentace a uživatelská příručka k použití zásuvného modulu.

NF3: Rozšiřitelnost zásuvného modulu Protože se tato práce zabývá pouze vytvořením prototypu, který se bude v budoucnu vylepšovat a rozšiřovat o další potřebné funkce, bude zásuvný modul lehce rozšiřitelný.

Kapitola 3

Návrh

Tato kapitola se zabývá návrhem vyvíjeného zásuvného modulu na základě požadavků stanovených v sekci 2.7.1. Návrh byl vytvářen s ohledem na omezení, která vyplynula z analýzy možností nabízených Unreal Engine [2].

3.1 Definice pojmů

V této sekci jsou definovány základní pojmy pro snazší pochopení následujících myšlenek a postupů.

Asset Asset je pro Unreal Engine [2] soubor s koncouvkou .uasset, obsahující jakýkoliv obsah ve formátu, se kterým umí Uneal Engine [2] pracovat. Assety je možné najít v takzvaném content browser a například se může jednat o importované obrázky, modely, či o vytvořené blueprint třídy, které jsou zmíněné níže.

Blueprint třída Blueprint třída je speciální typ assetu, který slouží k rozšíření funkcionalit za pomoci vizuálního programování. Více se tématu blueprint třída věnuje skece 2.4.1.

Parent blueprint třída Při vytváření blueprint třídy je možné si vybrat parent třídu, pod kterou je nový blueprint vytvořen. Z této parent třídy pak blueprint dědí vlastnosti.

SaveGame object SaveGame object je speciální typ blueprint třídy, která je potomkem SaveGame. Zděděné vlastnosti parent třídy SaveGame slouží pak k ukládání dat, která setrvají uložena i po vypnutí přístroje.

Widget Widget je speciální typ blueprint třídy sloužící k vytváření uživatelského rozhraní.

Procedural Mesh komponenta Procedural Mesh komponenta[10] je komponenta, kterou je možné přidat k vybrané blueprint třídě. Tato kompo-

nenta slouží k sestrojení mesh využitím vlastních informací o trojúhelníkách modelu.

3.2 Vybrané technologie vývoje

Jak vyplývá ze sekce 2.6.5 v analýze, k vývoji bude využita primárně technologie blueprint a jeho vyizuální programování. Pro rozšíření potřebných funkcionalit dosuptných v blueprint třídách budou využity zásuvné moduly VaRest [16] a RuntimeMeshLoader [18]. Sekundárně pak bude k vývoji využit jazyk C++ k úpravě funkcionalit, poskytovaných zásuvným modulem RuntimeMeshLoader[18]. Funkcionality budou přizpůsobeny tak, aby je bylo možné využít k řešení problematiky této práce.

3.3 Diagramy

Tato sekce zobrazuje diagramy, sloužící k reprezentaci případů užití. Protože byly některé původní diagramy příliš rozsáhlé, byly určité podúlohy přesunuty do vlastního diagramu a původní diagram se na dané podúlohy pouze odkazuje.

3.3.1 Import z externího zdroje

Tato podsekce ukazuje možný průběh importu modelu z API či ze souboru, nacházejícího se na disku počítače. Průběh importu je zachycen na activity diagramu 3.1 a na něj navazujících activity diagramů 3.2, 3.4 a 3.3.

V tomto diagramu uživatel otevře projekt obsahující můj zásuvný modul, umístí do scény asset schopný importu (více v sekci 3.9) a otevře uživatelské rozhraní 3.12.1. Uživateli je pak nabídnuta možnost volby importu z API, či importu z disku počítače. Podle volby se přechází na poddiagram importu z disku nebo na poddiagram přihlášení a následně importu z API.

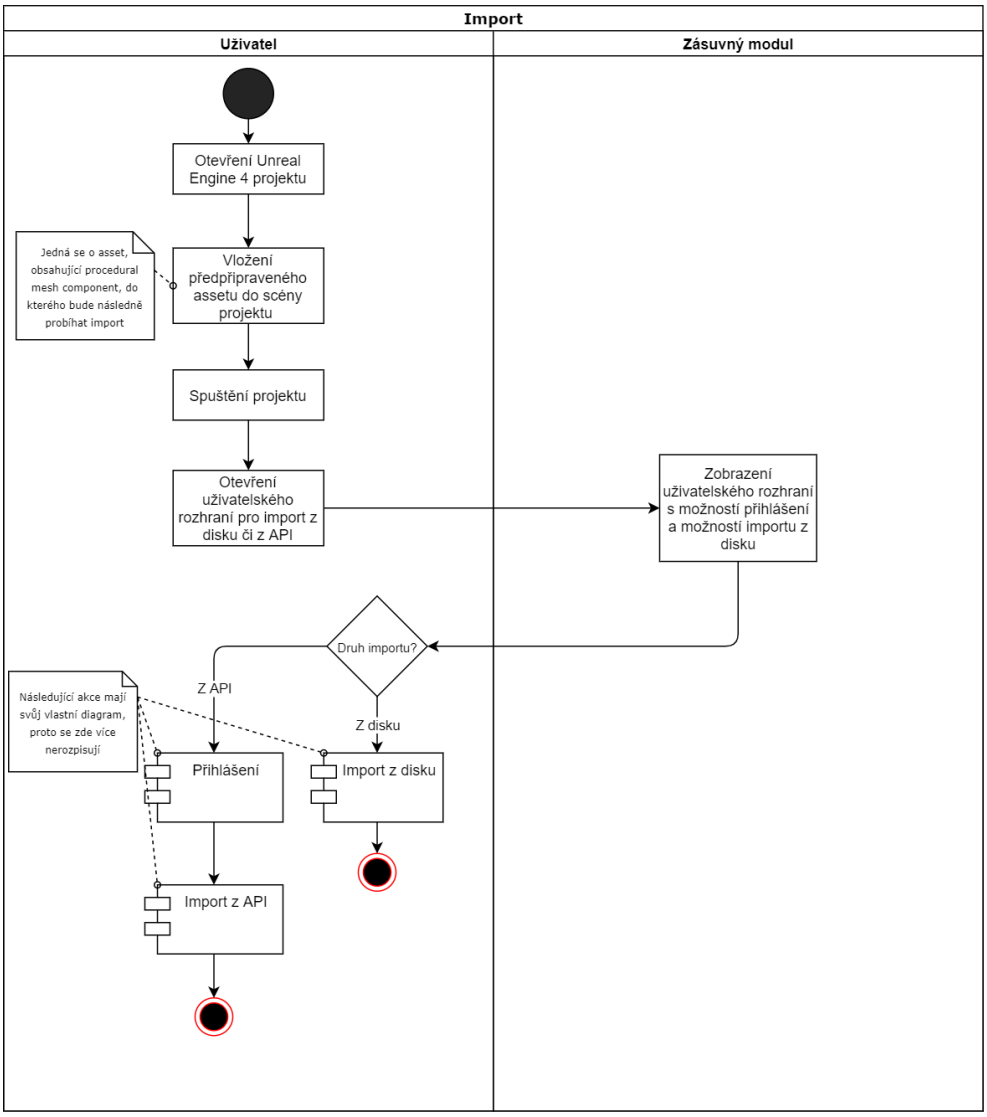
3.3.1.1 Přihlášení

Přihlášení v diagramu 3.2 probíhá podle popisu v sekci 3.5.

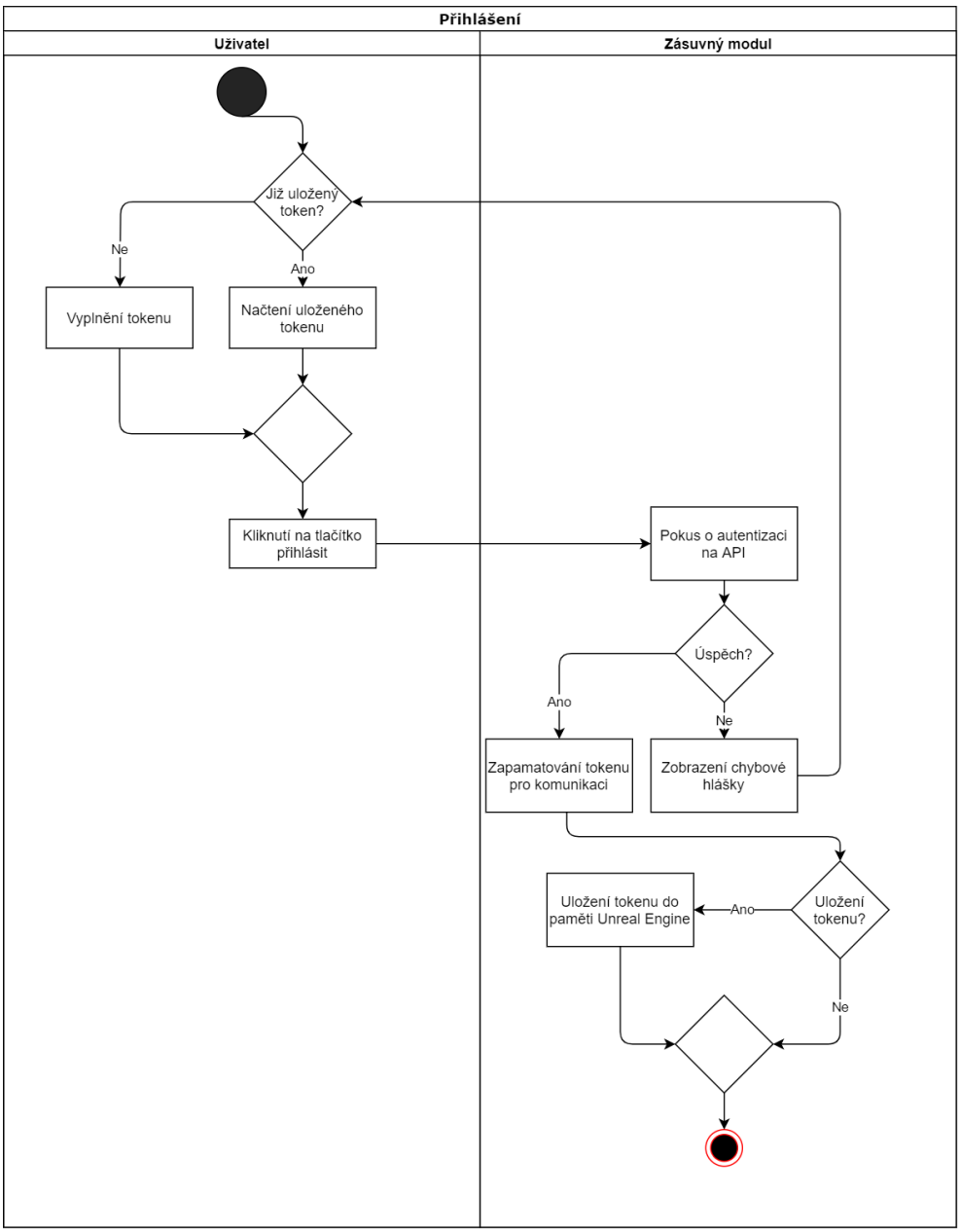
3.3.1.2 Import z API

Zásuvný modul v diagramu 3.3 zobrazí uživateli všechny struktury 3.12.3, po výběru jedné ze struktur zobrazí uživateli 3D Objekty 3.12.4, nacházející se pod danou strukturou.

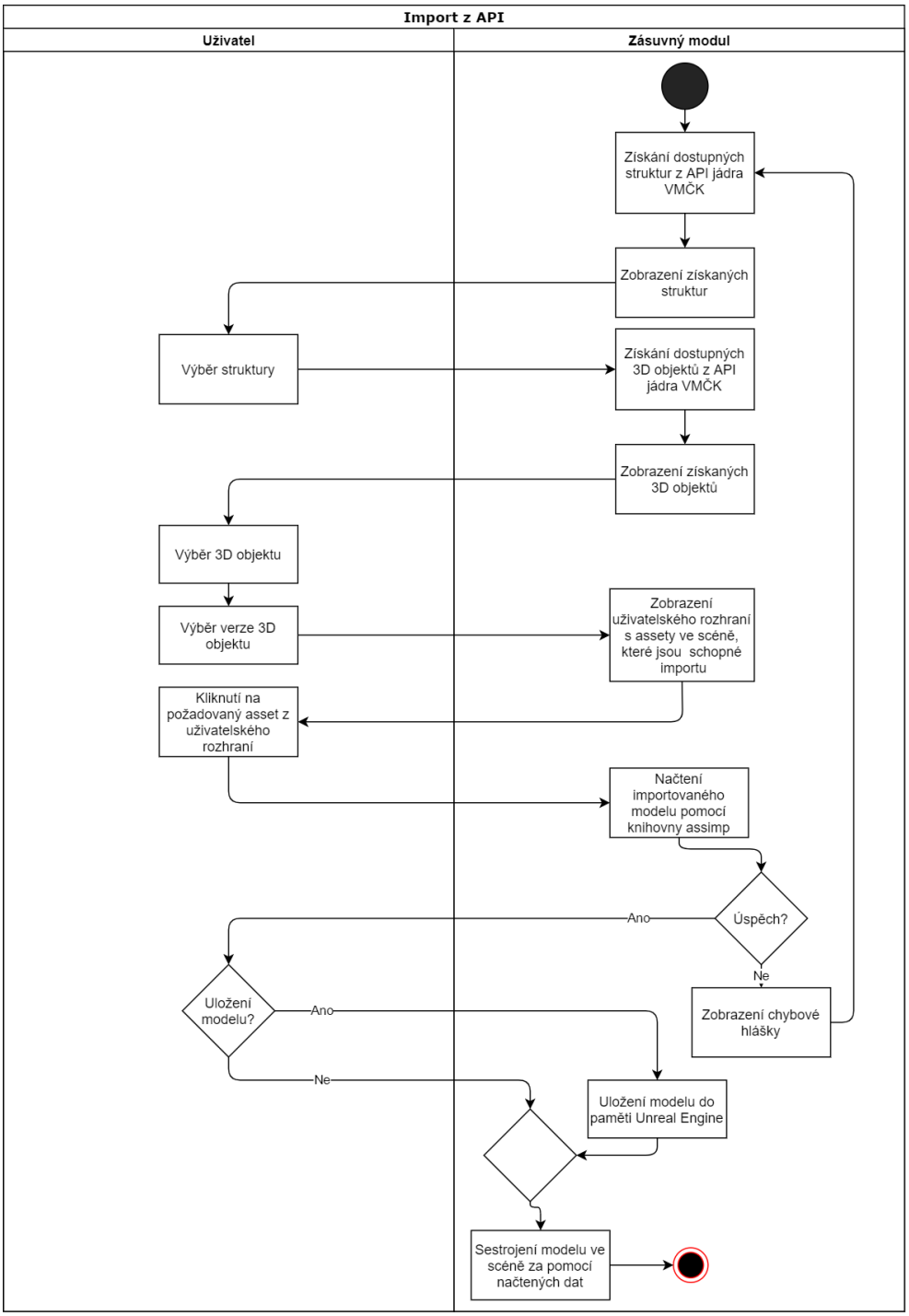
Po výběru 3D objektu uživatelem zobrazí zásuvný modul všechny assety ve scéně, které jsou schopny importu 3.12.5, ze zobrazených assetů si uživatel vybere cíl svého importu a následný import probíhá podle popisu v sekci 3.8.



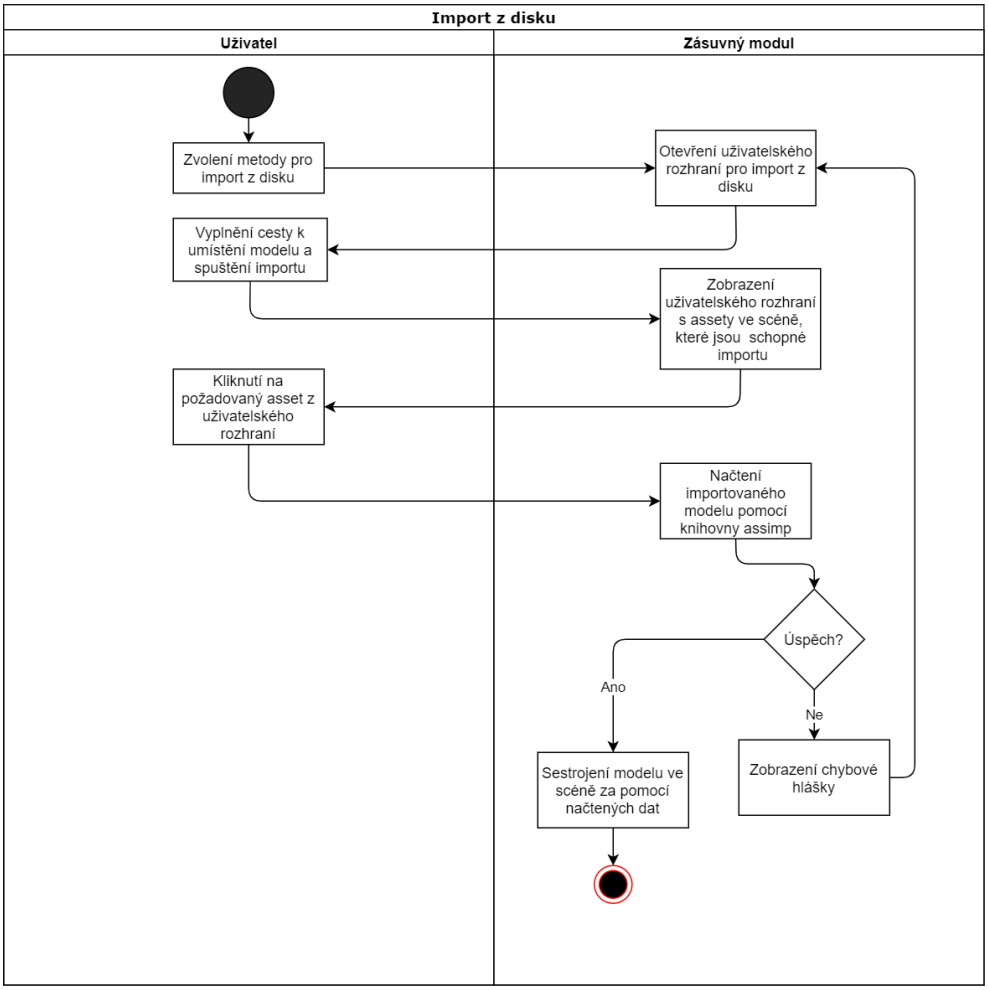
Obrázek 3.1: Diagram zobrazující import modelu do Unreal Engine pomocí zásuvného modulu



Obrázek 3.2: Diagram zobrazující přihlášení pomocí zásuvného modulu k API



Obrázek 3.3: Diagram zobrazující import z API za pomoci zásuvného modulu

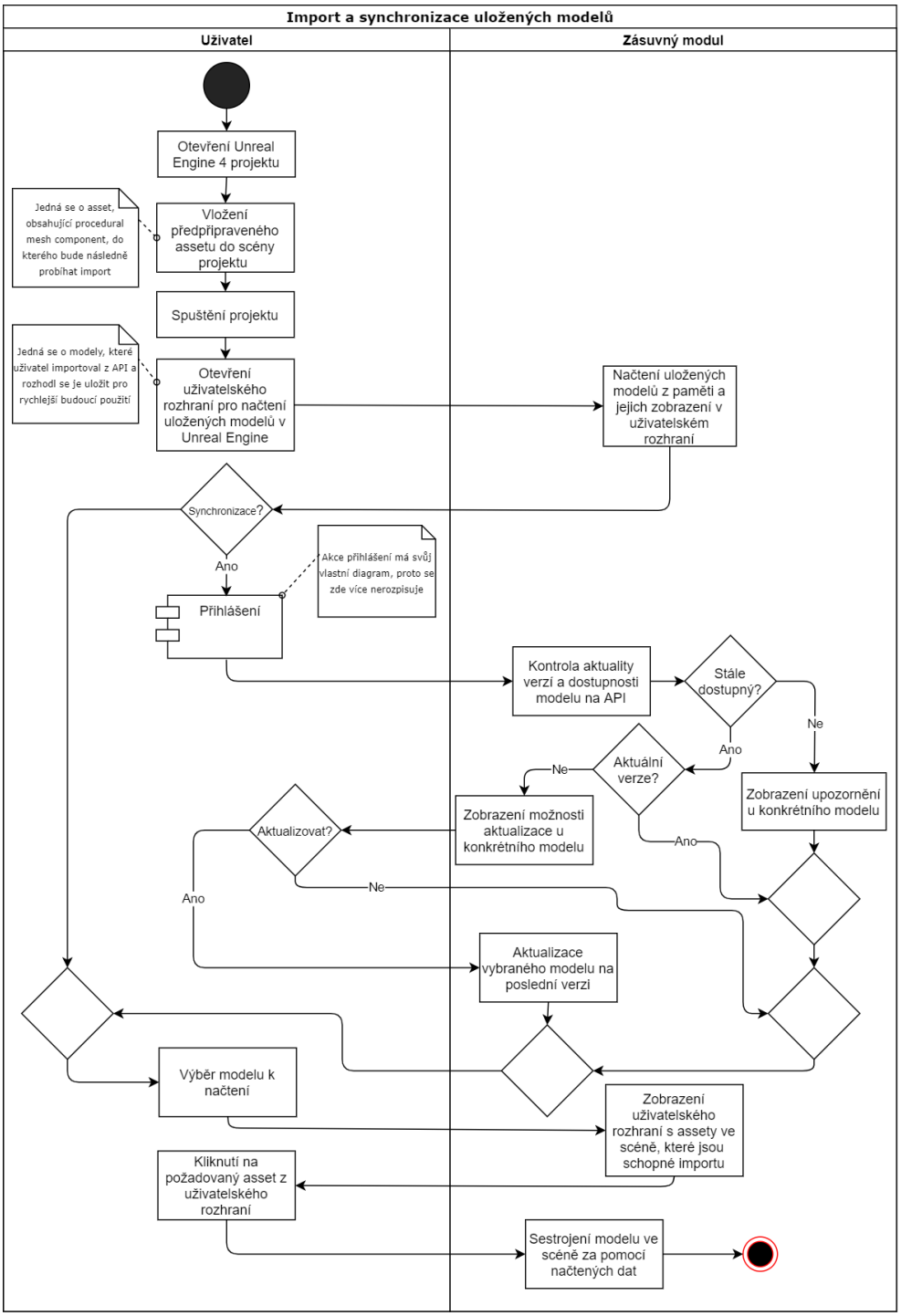


Obrázek 3.4: Diagram zobrazující import z disku počítače za pomoci zásuvného modulu

3.3.2 Import a synchronizace uložených modelů

Tato podsekce ukazuje možný průběh sestrojení modelu ve scéně z již uložených modelů. V rámci této ukázky je demonstrován i průběh synchronizace uložených modelů. Vše je zachyceno na diagramu 3.5.

V tomto diagramu uživatel otevře projekt obsahující můj zásuvný modul, umístí do scény asset schopný importu (více v sekci 3.9) a otevře uživatelské rozhraní 3.12.8. Toto uživatelské rozhraní je naplněno modely uloženými v paměti 3.12.9. Uživateli je pak nabídnuta volba, zdali se chce přihlásit a synchronizovat své modely dle sekce 3.11, či chce některý z uložených modelů pouze sestrojit ve vybraném assetu ve scéně.



Obrázek 3.5: Diagram zobrazující sestrojení a synchronizaci uložených modelů

3.4 Log

Všechny části uživatelského rozhraní budou obsahovat log, ve kterém budou zobrazeny informace o aktuálních procesech zásuvného modulu.

3.5 Přihlašování a autentizace

Přihlašování je navrženo pouze v rámci přístupu k API databáze VMČK [6]. API využívá k autentizovanému přístupu token. Návrh tedy počítá s možností zadávání tokenu do uživatelského rozhraní. Aby token nebylo nutné zadávat znovu při každém dotazu na API, návrh umožňuje pomocí dočasného SaveGame objektu token uložit. Token se uloží pouze po úspěšném ověření dotazem na API, aby nedocházelo k nadbytečným ukládáním.

Pokud přihlášení selže, uživatelské rozhraní zůstane stejné, pouze se informace o neúspěchu zmíní do logu. Do logu se pak zapisuje i úspěšné přihlášení, či úspěšné uložení tokenu.

Uložený token bude za pomoci zásuvného modulu VaRest [16] při každém následujícím dotazu na API automaticky vkládán do hlavičky jakožto autentizace. Po zavření užívatelského rozhraní bude však nutné při dalším spuštění projít přihlášením znovu.

Při přihlášení bude uživateli poskytnuta v uživatelském rozhraní možnost uložení tokenu. Toho bude dosaženo za pomoci SaveGame objektu a pouze za podmínky úspěšného přihlášení.

Pokud již bude v SaveGame objektu token uložený, bude uživateli umožněno daný token načíst a vyhnout se tím jeho opakovanému vyplňování. Při uložení jiného tokenu se stávající uložený token přepíše, bude se tedy ukládat pouze poslední použitý.

3.6 Hierarchie zobrazovaných modelů

Návrh hirearchie zobrazovaných modelů je odvozen od podoby nacházející se na API databáze VMČK [6]. Nejvýše je postavená takzvaná struktura, která reprezentuje určitou budovu. Pod strukturou jsou obsaženy takzvané 3D objekty, které obsahují již samotný model a jeho textury.

Protože 3D objekt vždy obsahuje pouze jeden model a import textur není obsahem této práce, bude 3D objekt v uživatelském rozhraní zároveň brán jakožto reprezentace samotného modelu. 3D objekt je pak nutné rozvětvit pouze na jeho verze.

3.7 Komunikace s API

Komunikace bude probíhat skrze GET požadavky na API. Tyto GET požadavky budou včetně autentizace sestavovány a odesílány za pomocí zásuvného mo-

dulu VaRest [16].

V následujících podsekcích jsou popsány požadavky na API, které budou využity pro fungování mého zásuvného modulu.

3.7.1 Požadavky k přihlášení

Při požadavku na přihlášení bude odeslán GET požadavek na url API. Tento požadavek nebude žádat žádné specifické informace, pouze bude sledovat odpověď API na uživatelem zadaný token přiložený v hlaviččce dotazu. Tento token bude po úspěšném přihlášení přikládán i do hlavičky všem následujícím dotazům.

3.7.2 Požadavky k zobrazení modelů

Pokud uživatel bude chtít zobrazit modely dostupné na API, budou nejprve zobrazeny pouze dostupné struktury. Zásvuný modul tak odešle GET požadavek na všechny struktury dostupné na API. API jakožto odpověď navrátí pole přehledu všech struktur, které pak zásuvný modul zobrazí v uživatelském rozhraní. Do každé zobrazené struktury je pak z dotazu uloženo její ID pro použití v pozdějších požadavcích přímo na danou strukturu.

V přehledu všech struktur však chybí podstatná informace pro správnou funkcionalitu zásuvného modulu. Chybějící informací je pole 3D objektů, které pod danou strukturu patří. Při výběru požadované struktury v uživatelském rozhraní je tedy nutné sestavit nový GET požadavek na konkrétní strukturu. K tomuto požadavku je potřeba přidat ID dané struktury.

Přijatá odpověď bude obsahovat pole „allVariants“, ve kterém lze najít přehled 3D objektů, nacházejících se v dané struktuře. Z přehledu 3D objektů bude pak možné získat všechny potřebné informace k jejich zobrazení a případnému stažení modelu.

3.7.3 Požadavky ke stažení modelu

Ke stažení modelu bude potřeba sestrojit GET požadavek na model, který bude obsahovat ID požadovaného modelu. API pak v odpovědi vrátí string, který bude načten do paměti a zpracován pomocí knihovny Assimp [12].

3.7.4 Požadavky k synchronizaci

Uživateli bude umožněno uložit informace o importovaném modelu v SaveGame objektu, aby bylo možné model načíst do scény bez komunikace s API. Pokud by chtěl uživatel synchronizovat uložená data s daty na API, bude nejprve proveden GET požadavek na ID konkrétního 3D objektu, zdali stále exituje. Pokud byl požadavek úspěšný a 3D objekt stále existuje, je z odpovědi na požadavek získaná aktuální verze 3D objektu na porovonání s verzí uloženého modelu.

3.8 Import a sestrojení modelu

K importu bude využíván upravený zásuvný modul RuntimeMeshLoader [18]. Tento zásuvný modul obsahuje funkcionalitu umožňující načíst model podle zadané cesty k souboru. Z načteného modelu pak získá data potřebná k jeho sestrojení pomocí Procedural Mesh komponenty. Načtení a získání potřebných dat zajišťuje v zásuvném modulu knihovna Assimp [12].

Původní funkcionalita zásuvného modulu je dostačující pro načtení modelu ze souboru pomocí zadané cesty, při stahování modelu z API je však model předán jako string a rovnou načten do paměti. Z tohoto důvodu je nutné zásuvný modul upravit, aby byl schopen načítat model přímo z paměti.

Ke konstrukci modelu je nutné využít funkcionalitu uzlu Create Mesh Section [19], která vytvoří novou sekci Procedural Mesh komponenty. V nové sekci je procedurálně zkonstruován mesh na základě zadaných vrcholů (vertices), trojúhelníků (triangles), normál (normals), UV a tečen (tangents). Data o vrcholech a trojúhelnících jsou povinná a jejich umístění slouží k sestrojení tvaru celého modelu. UV data, normály a tečny jsou nepovinná data a slouží ke správnému namapování textur a ke správné funkci shaderu.

Data sloužící jako vstup do uzlu Create Mesh Section je schopen za pomoci knihovny Assimp [12] získat právě zásuvný modul RuntimeMeshLoader [18]. Proto pro sestrojení modelu stačí zásuvný modul RuntimeMeshLoader [18] napojit na uzel Create Mesh Section.

3.9 Assety schopné importu

Jak již bylo zmíněno v minulé kapitole, k sestrojení modelu je potřeba Procedural Mesh komponenta a uzel Create Mesh Section. Aby tedy bylo možné importovaný model sestrojit ve scéně, je nutné, aby se v ní nacházel asset, který obsahuje kompontentu Procedural Mesh a je schopný využívat funkci Create Mesh Section.

V návrhu této práce je „asset schopný importu“ vytvořená parent blueprint třída. Tato parent třída bude obsahovat všechny vlastnosti nutné pro správný chod importu a bude z ní možné vytvářet potomky. Tito potomci zdědí veškeré vlastnosti nutné ke komunikaci s mým zásuvným modulem, a proto po vytvoření potomka ho může uživatel ihned přesunout do scény a využít k importu. Výhodou tohoto přístupu je možnost vytvořit potomka parent třídy a případně upravit určité funkcionality pro rozšíření možností importu.

V návrhu této práce obsahuje tedy parent třída Procedural Mesh komponentu, funkci na sestrojení modelu, funkci na import modelu a funkci pro znovunačtení modelu. Funkce na sestrojení modelu bude mít jako vstup data nutná k sestrojení modelu pomocí uzlu Create Mesh Section [19], zmíněného

v sekci 3.8. Tato data následně funkce využije jako vstup do zmíněného uzlu Create Mesh Section a tím model sestrojí.

Funkce na import modelu bude přijímat jako své vstupy ID modelu, ID 3D objektu a hodnotu říkající, zdali model uložit či nikoliv. Model bude následně získán z API a sestrojen za pomoci funkce k sestrojení modelu. Poté, co se model ve scéně sestrojí, uloží se do SaveGame objektu pouze ID modelu či ID 3D objektu a data nutná k sestrojení modelu. Informace se uloží pod referencí instance blueprint třídy a tvar uložených informací závisí na tom, zdali se uživatel rozhodnul daný model uložit, či nikoliv.

Funkce pro znovunačtení nasledně využívá informace uložené předešlou funkcí, a zajišťuje, aby se po vypnutí a zapnutí projektu modely již jednou do assetů importované automaticky načetly znovu. Tímto se zásuvný modul bude snažit docílit, aby nebylo nutné po každém zapnutí projektu manuálně importovat již dříve importované modely znovu. Znovunačtení pak bude moci probíhat jak z API, tak z uložených modelů a způsob bude možné zvolit pro každou instanci blueprint třídy zvlášť.

3.10 Ukládané modely

Uživateli bude umožněno si impportované modely uložit. Při načítání uloženého modelu nebude již třeba komunikace s API ani zpracování modelu za pomoci knihovny Assimp [12], a proto bude načtení modelu výrazně rychlejší.

Modely budou ukládány pomocí SaveGame objektu, kde se budou ukládat informace o vrcholech (vertices), trojúhelnících (triangles), normálách (normals), UV a tečen (tangents). Uložené modely bude pak možné zobrazit skrze uživateslské rozhraní a následně je sestrojit v assetech schopných importu.

Uložené modely bude také možné skrze uživatelské rozhraní pro uložené modely z paměti odstranit.

3.11 Aktualizace uložených modelů

Uložené modely bude možné synchronizovat s API způsobem zmíněmým v podsekci 3.7.4. Pokud se po synchronizaci ukáže, že uložený model na API již neexistuje, bude na to uživatel upozorněn v uživatelském rozhraní uložených modelů. Aby bylo upozornění snadné odhalit, bude reprezentováno změnou barvy určeného modelu v uživatelském rozhraní na červenou.

Pokud se po synchronizaci ukáže, že uložený model existuje na API v novější verzi, bude na to uživatel upozorněn v uživatelském rozhraní uložených modelů. Aby bylo upozornění snadné odhalit, bude reprezentováno změnou barvy určeného modelu v uživatelském rozhraní na oranžovou. Uživateli pak bude umožněno uložená data aktualizovat na nejnovější verzi.

3.12 Uživatelské rozhraní

Tato sekce se věnuje návrhu uživatelského rozhraní. K vývoji uživatelského rozhraní bude využita speciální blueprint třída, která se nazívá Widget. Widget kromě graph editoru, sloužícího pro vytváření funkcionalit, obsahuje i takzvaný designer. Designer je možné využít k vytváření uživatelského rozhraní a to za pomoci grafických nástrojů, které poskytuje Unreal Engine [2].

Grafické nástroje obsahují například komponenty jako tlačítko, text, textové pole nebo vertikální box. Tyto komponenty mohou mít pak za potomky jiné komponenty a tím vytvářet složité struktuyr. Například je možné vložit text jako potomka tlačítka a tím lze získat tlačítko s popiskem.

3.12.1 Přihlašování

Uživatelské rozhraní pro přihlášení bude obsahovat:

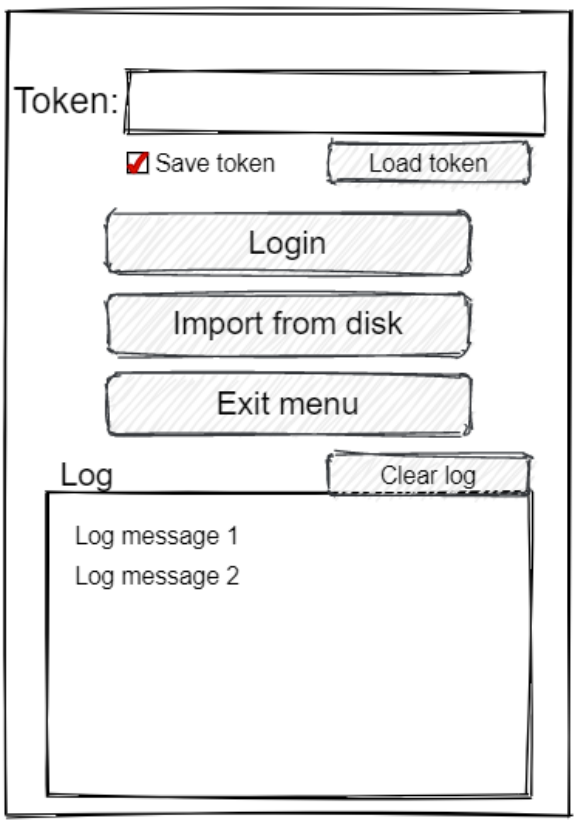
- textové pole pro vyplnění tokenu
- zaškrtávací políčko pro volbu uložení tokenu
- tlačítko pro načtení uloženého tokenu
- tlačítko pro přihlášení
- tlačítko pro opuštění menu
- tlačítko pro otevření uživatelského rozhraní importu z disku
- log zmíněný v sekci 3.4

Celkový návrh uživatelského rozhraní pro přihlačování je ukázán na obrázku 3.6. Po přihlášení, které probíhá podle 3.5, je uživatel přesměrován na uživatelské rozhraní, ve kterém se zobrazují reprezentace dostupných struktur.

3.12.2 Zobrazení struktur, 3D objektů a assetů schopných importu

Uživatelské rozhraní pro zobrazení struktur, 3D objektů a assetů schopných importu bude zůstávat identické, ať už zobrazuje kteroukoliv reprezentaci z výše zmíněných. Obsahuje:

- sekci s posuvníkem určenou pro zobrazování reprezentací objektů
- tlačítko pro opuštění menu
- tlačítka pro navrácení se o krok zpět
- log zmíněný v sekci 3.4



Obrázek 3.6: Návrh uživatelského rozhraní pro přihlášení

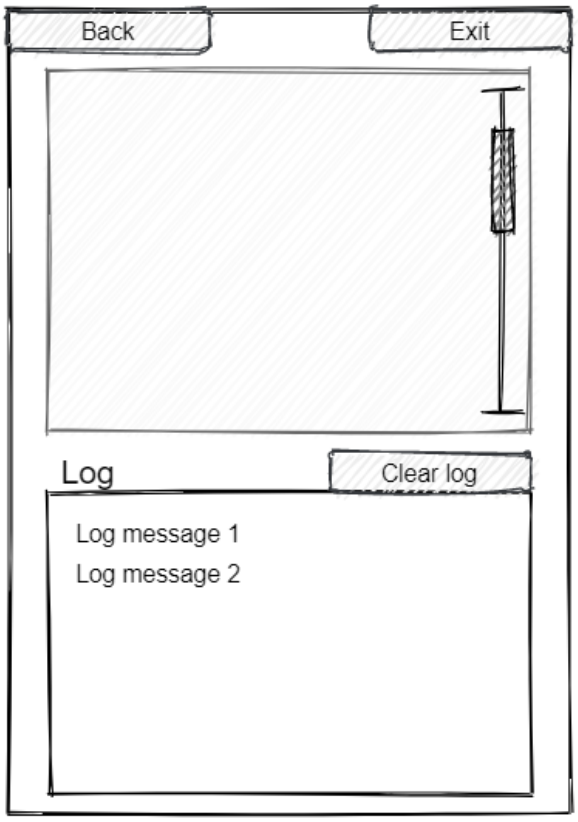
Celkový návrh uživatelského rozhraní pro zobrazení struktur, 3D objektů a assetů schopných importu se řídí obrázkem 3.7. Po výběru objektu ze seznamu se zobrazí další v pořadí podle hierarchie probrané v sekci 3.6. Poté, co uživatel vybere požadovaný model k importu, zobrazí uživatelské rozhraní všechny assety schopné importu ve scéně. Uživatel si tak bude schopen vybrat do jakého assetu bude chtít model importovat. Po výběru assetu k importu se zobrazí uživatelské rozhraní pro import.

3.12.3 Reprezentace struktur

Reprezentace struktury, zobrazované v uživatelském rozhraní, bude obsahovat:

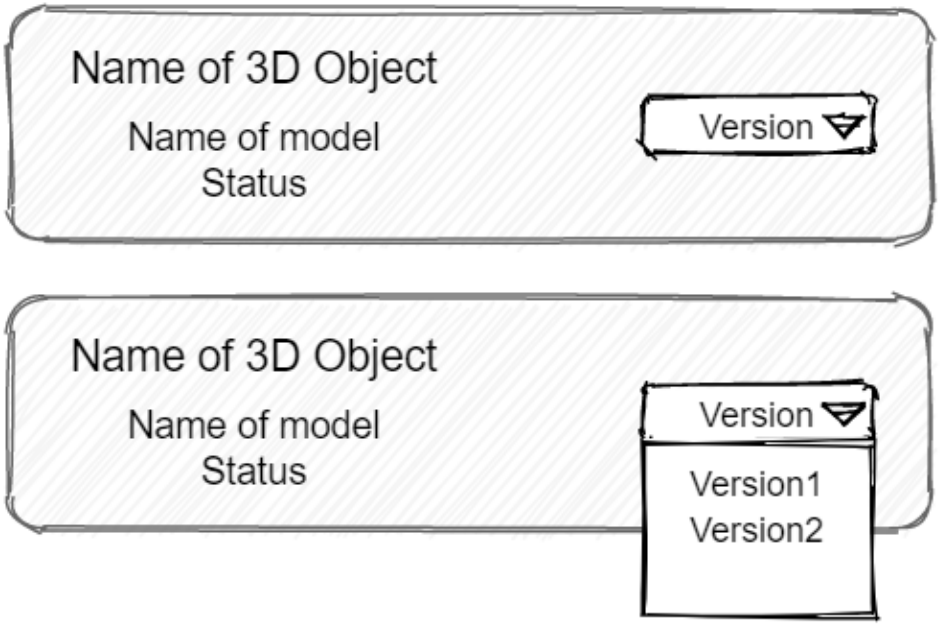
- jméno struktury
- popis struktury

Celkový návrh reprezentace struktury se řídí obrázkem 3.8. Po kliknutí na strukturu, se zobrazí všechny 3D objekty, které struktura obsahuje.



Obrázek 3.7: Návrh uživatelského rozhraní pro zobrazení struktur, 3D objektů a assetů schopných importu

Obrázek 3.8: Návrh reprezentace struktur pro zobrazení v uživatelském rozhraní



Obrázek 3.9: Návrh reprezentace 3D objektů pro zobrazení v uživatelském rozhraní

3.12.4 Reprezentace 3D objektů

Reprezentace 3D objektu, zobrazovaného v uživatelském rozhraní, bude obsahovat:

- jméno 3D objektu
- jméno obsaženého modelu
- status 3D objektu na API
- rozbalovací menu s možností výběru verzí

Celkový návrh reprezentace 3D objektu se řídí obrázkem 3.9. Po výběru verze a kliknutí na strukturu se zobrazí všechny assety schopné importu, nacházející se ve scéně.

3.12.5 Reprezentace assetů schopných importu

Reprezentace assetu schopnéhoho importu, se bude skládat pouze z reference na jeho instanci, převedené na text. Celkový návrh reprezentace assetu schopnéhoho importu se řídí obrázkem 3.10. Více o návrhu assetu schopného importu lze najít v sekci 3.9. Po jeho výběru se zobrazí uživatelské rozhraní spravující import.

Obrázek 3.10: Návrh reprezentace assetů schopných importu v uživatelském rozhraní

3.12.6 Uživatelské rozhraní pro import

Uživatelské rozhraní pro import bude obsahovat:

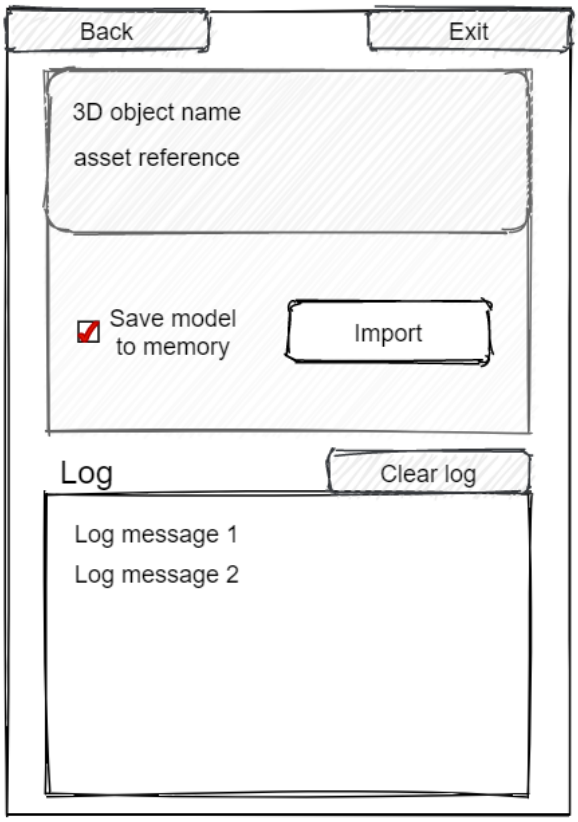
- jméno importovaného 3D objektu
- textový tvar reference cíle importu
- zaškrtávací políčko pro volbu uložení modelu
- tlačítko pro import
- tlačítko pro opuštění menu
- tlačítka pro navrácení se o krok zpět
- log zmíněný v sekci 3.4

Celkový návrh uživatelského rozhraní pro import se řídí obrázkem 3.11. Ve vrchní části uživatelského rozhraní se bude nacházet stručný přehled co a kam se importuje. Dále je možné zvolit si pomocí zaškrtávacího políčka, zdali import uloží model pouze jako odkaz (při znovu načítání se bude opět importovat z API), či jako celý model (při znovu načítání se model pouze sestrojí pomocí dat v paměti). Po kliknutí uživatele na tlačítko importu se již spustí samotný proces importu.

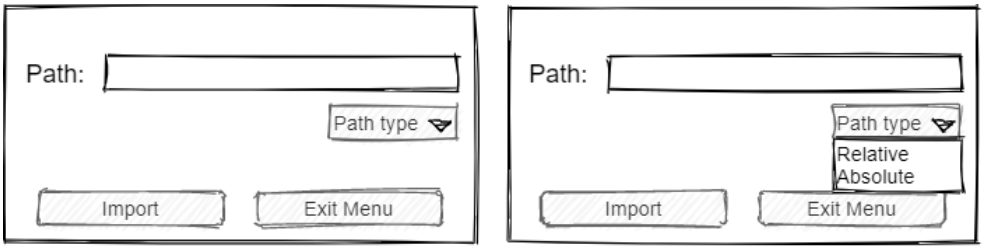
3.12.7 Uživatelské rozhraní pro import z disku

Uživatelské rozhraní pro import z disku bude obsahovat:

- textové pole pro vyplnění cesty k souboru
- rozbalovací menu s možností výběru typu cesty k souboru
- tlačítko pro import
- tlačítko pro opuštění menu



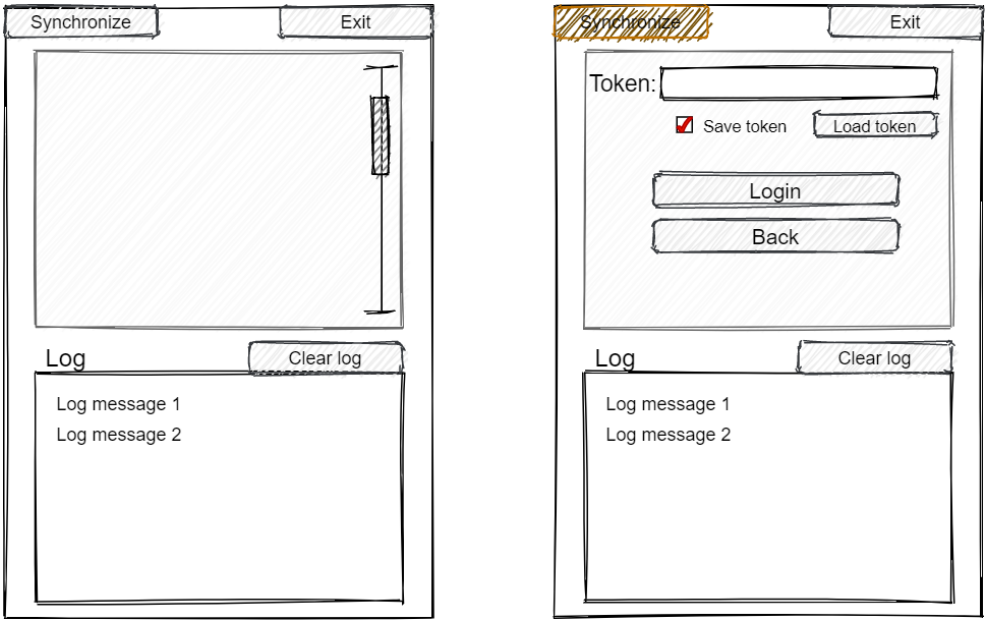
Obrázek 3.11: Návrh uživatelského rozhraní pro import modelu



Obrázek 3.12: Návrh uživatelského rozhraní pro import z disku počítače

- log zmíněný v sekci 3.4

Celkový návrh uživatelského rozhraní pro import z disku se řídí obrázkem 3.12. Rozbalovací menu určuje typ cesty k souboru (relativní, či absolutní). Po kliknutí uživatele na tlačítko importu se již spustí samotný proces importu.



Obrázek 3.13: Návrh uživatelského rozhraní pro zobrazení uložených modelů a pro přihlášení v případě synchronizace

3.12.8 Uživatelské rozhraní pro zobrazení uložených modelů

Uživatelské rozhraní pro zobrazení uložených modelů bude obsahovat:

- sekci s posuvníkem určenou pro zobrazení reprezentací uložených modelů
- tlačítko pro synchronizaci s API
- tlačítko pro opuštění menu
- log zmíněný v sekci 3.4

Celkový návrh uživatelského rozhraní pro import z disku se řídí obrázkem 3.13. Tlačítko synchronizace bude zobrazovat zmenšenou verzi uživatelského rozhraní pro přihlášení, pokud následné přihlášení bude úspěšné, bude spuštěna synchronizace uložených modelů s modely na API. Tématice synchronizace se více věnuje sekce 3.11.

Sekce s posuvníkem bude naplněna reprezentací uložených modelů. Po kliknutí uživatelem na vybraný uložený model se zobrazí všechny assety schopné importu, nacházející se ve scéně.

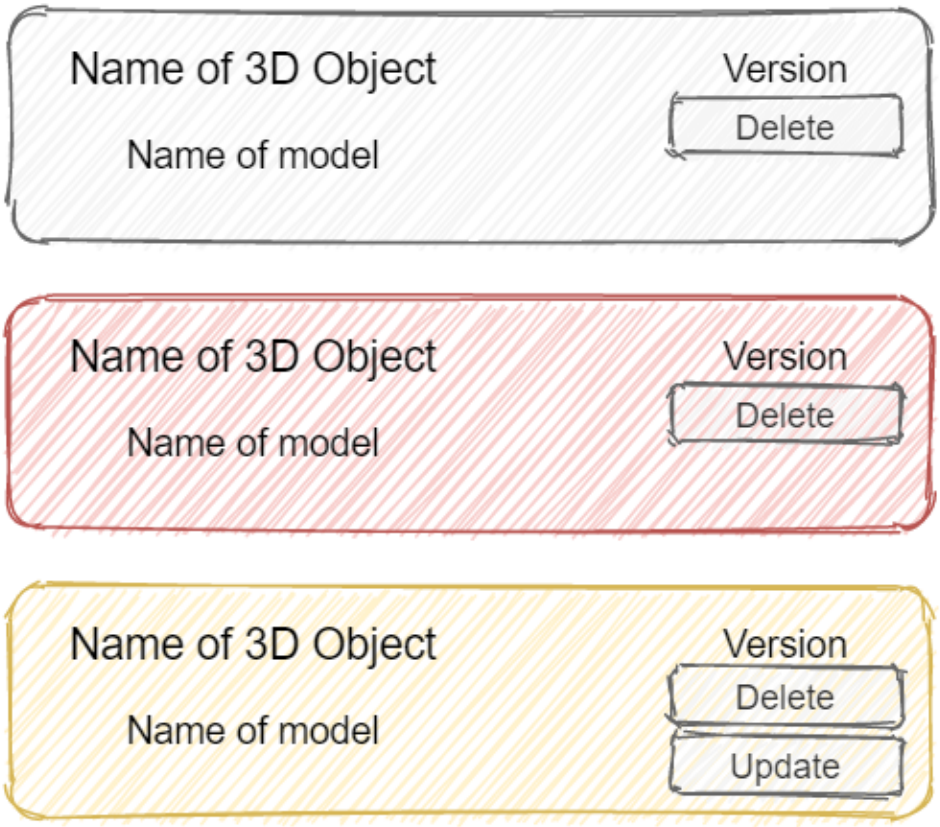
3.12.9 Reprezentace uložených modelů

Reprezentace uloženého modelu, zobrazovaného v uživatelském rozhraní, bude obsahovat:

- jméno nadřazeného 3D objektu
- jméno modelu
- verze 3D objektu
- tlačítko pro odstranění modelu
- v případě zastaralého modelu tlačítko pro aktualizaci

Celkový návrh reprezentace uloženého modelu se řídí obrázkem 3.14. Jak je již řečeno v sekci 3.11, synchronizace bude porovnávat uložené objekty se shodnými objekty vrácenými z API a na základě výsledku porovnání budou uložené modely převáděny do různých stavů (aktuální, neaktuální, smazaný). Pokud synchronizace neproběhla, či je model aktuální, zůstane barva reprezentce světle šedá. Pokud synchronizace proběhla a daný model se na API již nenachází, změní reprezentace svou barvu na červenou. Pokud synchronizace proběhla a na API se bude nacházet nová verze, reprezentace dostane oranžovou barvu a zpřístupní se tlačítko, které po kliknutí uživatelem stažený model aktualizuje.

Po kliknutí na tlačítko odstranění bude daný model ihned odstraněn z paměti. Po kliknutí na samotnou reprezentaci uloženého modelu se zobrazí všechny assety schopné importu, nacházející se ve scéně, aby v nich mohl být model případně zkonstruován.



Obrázek 3.14: Návrh reprezentace uložených modelů pro zobrazení v uživatelském rozhraní

Kapitola 4

Realizace

4.1 Příručka pro uživatele

Tato sekce je věnována implementovaným funkcionalitám a návodu pro jejich použití.

4.1.1 Předpoklady pro úspěšné zprovoznění zásuvného modulu

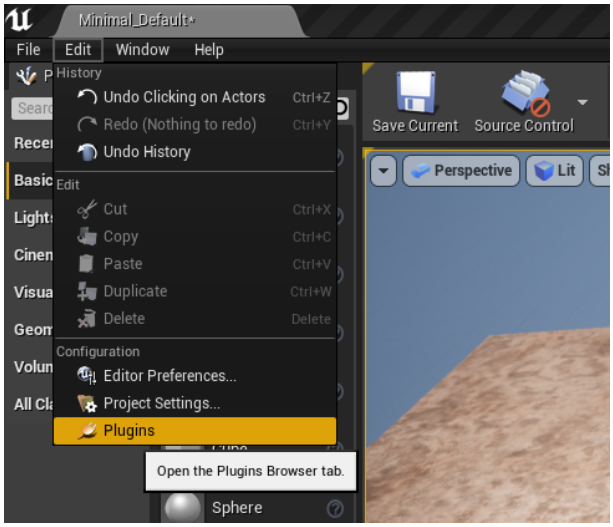
Zásuvný modul RuntimeImportPlugin, jenž je výsledkem této práce, vychází ze dvou dalších zásuvných modulů VaRest[16] a RuntimeMeshLoader[18]. Zatímco zásuvný modul RuntimeMeshLoader je dodáván společně s RuntimeImportPlugin, zásuvný modul VaRest si uživatel musí zadarmo stáhnout z Unreal Engine MarketPlace [17] sám.

Doporučeno je používat verzi Unreal Engine 4.25.3 a je doporučena deaktivovace všech magic leap [20] zásuvných modulů z důvodu nekompatibility se zásuvným modulem RuntimeMeshLoader.

4.1.2 Instalace zásuvného modulu

Pro použití mého zásuvného modulu je nejdřívě nutné přesunout jeho složku do cílového projektu. Složka obsahující vše potřebné pro můj zásuvný modul se nazývá *RuntimeImportPluginPackage* a je nutné ji přesunout do složky Plugins v kořenovém adresáři cílového projektu. Pokud složka Plugins v kořenovém adresáři projektu neexistuje, je třeba ji vytvořit.

Po přesunutí složky se zásuvným modulem je možné projekt zapnout. Pokud se při zapínání objeví hláška, upozorňující na nesestavený zásuvný modul RuntimeImportPlugin či RuntimeMeshLoader [18], stačí pouze zásuvný modul sestavit kliknutím na tlačítko Ano. Unreal Engine [2] se pak pokusí sestavit zásuvný modul sám a spustí daný projekt s nainstalovaným zásuvným modulem.



Obrázek 4.1: Obrázek ukazující otevření prohlížeče zásuvných modulů

Můj zásuvný modul by se měl postarat o aktivaci ostatních zásuvných modulů, které používá k chodu. Je ale možné, že zásuvný modul RuntimeImportPlugin samotný nebude po instalaci aktivní. Pokud tato situace nastane, uživatel bude muset zásuvný modul aktivovat skrze takzvaný Plugin browser. Ten se dá vyvolat pomocí tlačítka edit 4.1. V Plugin browser bude pak nutné vyhledat zásuvný modul RuntimeImportPlugin a kliknout na enable 4.2. Pro zaznamenání změny se Unreal Engine bude muset restartovat.

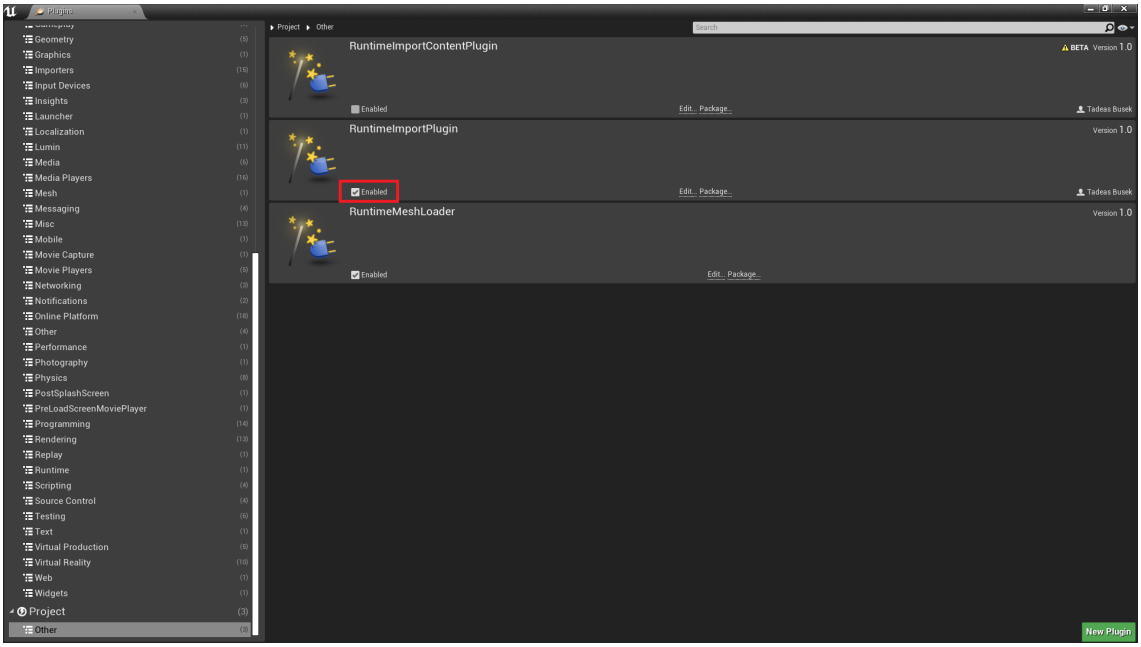
4.1.3 Blueprint třídy

Blueprint třídy jsou pro uživatele dostupné v RuntimeImportPlugin content složce. Pro uživatele bez záměru upravovat či přidávat funkcionality jsou určené pouze blueprint třídy, nacházející se v kořenovém adresáři. Blueprint třídy nacházející se v podadresářích zajišťují funkcionality zásuvného modulu, a proto není manipulace s nimi doporučována.

4.1.3.1 RuntimeImportPluginHandler

Pro využití funkcionalit zásuvného modulu je nejprve nutné umístit jednu instanci RuntimeImportPluginHadnler do scény. Tato blueprint třída se nachází v kořenovém adresáři RuntimeImportPlugin content složky a zajišťuje eventy k otevírání uživatelského rozhraní uživatelem.

Při otevření graph editoru RuntimeImportPluginHandler je možné otevírání uživatelského rozhraní přemapovat na jinou klávesu. V základu se však uživatelské rozhraní otevírá klávesou *I*.



Obrázek 4.2: Obrázek ukazující aktivaci zásuvného modulu RuntimeImport-Plugin

Pokud uživatel nechce umisťovat blueprint třídu RuntimeImportPluginHandler do scény, je možné volat blueprint funkci *OpenMenu* 4.3 z jiného místa v projektu. Funkce *OpenMenu* je díky zásuvnému modulu všude v projektu dostupná a její úlohou je zobrazení základního menu zásuvného modulu na obrazovce. Z tohoto menu se pak dále spouští veškeré další funkcionality.

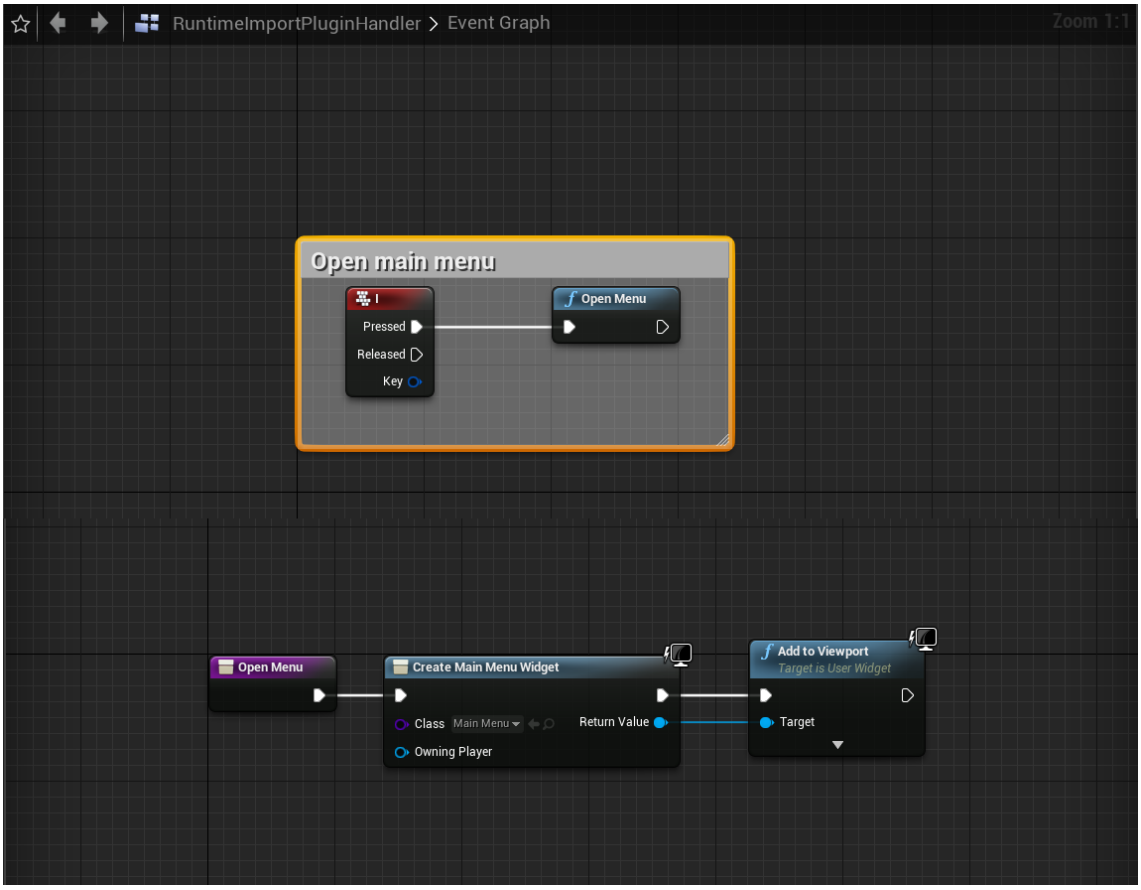
4.1.3.2 RuntimeMeshImportParent

Další blueprint třída, která se nachází v RuntimeImportPlugin content složce, je RuntimeMeshImportParent. Tato třída obsahuje takzvanou Procedural Mesh komponentu a implementuje funkcionalitu, která je za pomoci dané Procedural Mesh komponenty [10] schopná sestrojit model při spuštěném projektu 4.4.

RuntimeMeshImportParent je tedy asset, který je možný přesunout do scény a za běhu mu posílat data, ze kterých sestrojí model.

Od RuntimeMeshImportParent lze pak vytvářet potomky, kteří jsou schopni stejné funkcionality. I když je používat RuntimeMeshImportParent k sestrojení modelu nezávadné, je doporučeno si vytvořit a používat potomky. Různé potomky si uživatel může nezávisle na sobě upravovat bez ovlivnění funkcionality ostatních.

Zkušenější uživatelé mohou pak využít techniku override k přepsání funkcionalit poskytnutých potomkovi rodičem, a tím si upravit funkcionalitu v



Obrázek 4.3: Obrázek ukazující implementaci a použití funkce OpenMenu

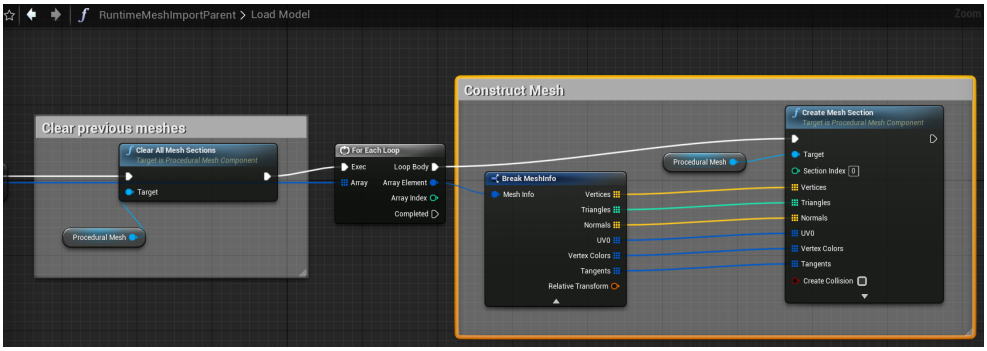
rámci zásuvného modulu.

4.1.3.3 Vytváření potomka

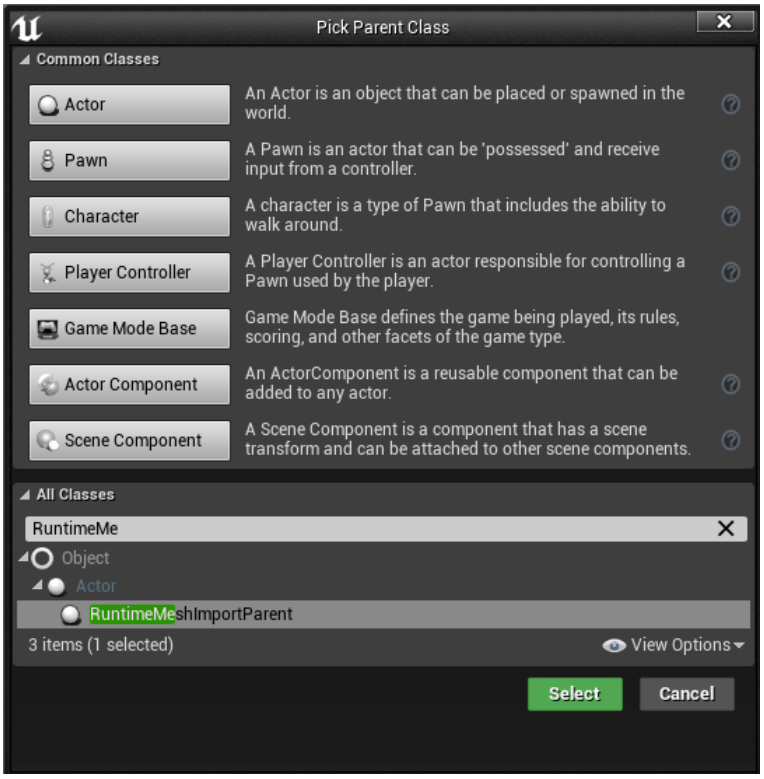
Pro vytvoření potomka určité třídy je nutné při vytváření nového blueprint rozkliknout *All Classes*. Zde je pak nutné vyhledat třídu, která bude rodičem vytvářeného blueprint 4.5.

4.1.4 Implementované funkcionality

Všechny funkcionality zásuvného modulu jsou dostupné skrze uživatelské rozhraní. Uživatelské rozhraní je pak vytvářeno pomocí widget tříd a většinu funkcionalit obsahuje přímo v sobě. Pro využití funkcionalit zásuvného modulu je důležité, aby projekt obsahoval možnosti interakce s widget třídou.



Obrázek 4.4: Obrázek ukazující implementaci části funkce pro sestrojení modelu pomocí Procedural Mesh komponenty



Obrázek 4.5: Obrázek ukazující vytvoření potomka parent třídy RuntimeMeshImportParent

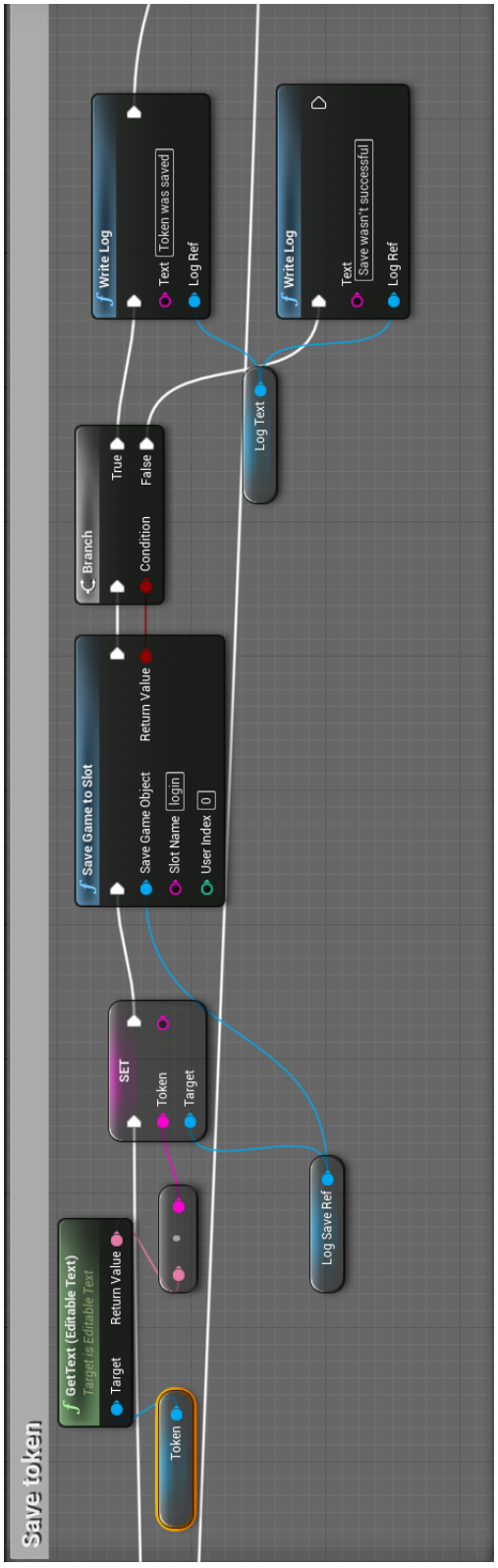
Obrázek 4.6: Obrázek ukazující uživatelské rozhraní pro přihlášení vytvořené v Unreal Engine

4.1.4.1 Přihlašování

Zásuvný modul poskytuje po otevření hlavního menu možnost přihlášení. Po zvolení této možnosti se otevře uživatelské rozhraní 4.6 vytvořené podle návrhu 3.6. Z konečné implementace byl nakonec odstraněn návrh na import z disku počítače, a to z důvodu neslučitelných funkcionalit s importem z API. Proto i uživatelské rozhraní pro přihlášení již neobsahuje tlačítko pro možnost importu z disku.

Přihlášení pak funguje na stejném principu uvedeném v návrhu 3.5. Uživatel zadá do textového pole token, který je po úspěšném ověření s API uložen do dočasného SaveGame objektu pro použití při budoucí komunikaci s API. Dále je uživateli poskytnuta možnost správně zadaný token při přihlášení uložit 4.7 pro možnost budoucího předvyplnění tokenu. Token je ukládán pomocí SaveGame objektu, a to pouze ten poslední (předešlý uložený token bude vždy nahrazen novým uloženým tokenem).

Pomocí tlačítka *LoadToken* je pak možné načíst poslední uložený token a tím předvyplnit textový box určený pro vložení tokenu.



Obrázek 4.7: Obrázek ukazující ukládání tokenu do SaveGame objektu

4.1.4.2 Výběr modelu

Zásuvný modul po úspěšném přihlášení zobrazí uživatelské rozhraní vytvořené podle návrhu 3.7. Důležitou částí zobrazeného uživatelského rozhraní je sekce pro zobrazení objektů. Tuto část tvoří takzvaný widget přepínač. Ten umožňuje přepínat mezi zobrazováním jeho potomků. Po zobrazení uživatelského rozhraní odešle widget GET požadavek na API 4.8, ve kterém žádá přehled všech struktur pro zobrazené uživatelské rozhraní. Po obdržení odpovědi, kterou tvoří pole všech struktur, projde widget všechny prvky pole a pro každý vytvoří reprezentaci struktury s daty získanými v odpovědi.

Reprezentace struktury je další widget, vytvořený podle návrhu 3.8. Vytvořené reprezentace struktur se ihned po vyvtoření vkládají do potomka výše zmiňovaného widget přepínače.

Po kliknutí na některou ze struktur se změní zobrazovaný potomek widget přepínačem a tím vytvoří místo na nové objekty, bez toho, aby se musely mazat ty staré. Následně je pak odeslán nový GET požadavek na strukturu, která byla vybrána. Po úspěšném obdržení odpovědi je z odpovědi získané pole *AllVariants*, které obsahuje seznam všech 3D objektů obsažených v dané struktuře.

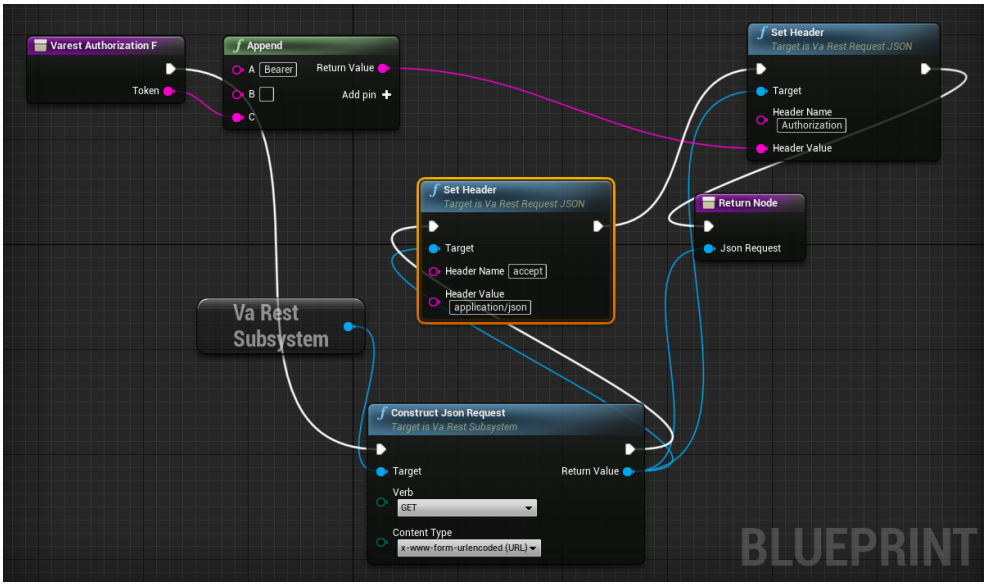
Widget projde všechny prvky pole a za každý prvek vytvoří novou reprezentaci 3D objektu. Skutečná reprezentace 4.9 je odlišná od toho, jak byla popsána v návrhu 3.9. Důvodem je nevyhovující reprezentace verzí na API a nemožnost získání informací o předešlých verzích. Dalším důvodem je také nemožnost získat jméno modelu, proto je ve skutečné realizaci jméno nahrazeno ID modelu.

Vytvořené reprezentace se pak ihned vkládají do nově zobrazovaného potomka widget přepínače a tím se zobrazují uživateli. Ze zobrazovaných 3D objektů si pak uživatel již může vybrat konkrétní model, verze se však vždy stahuje ta nejnovější a z důvodu nevhodné reprezentace verze na API není verze modelu z API ani získávána ani uživatelům zobrazována.

4.1.4.3 Výběr assetu schopného importu

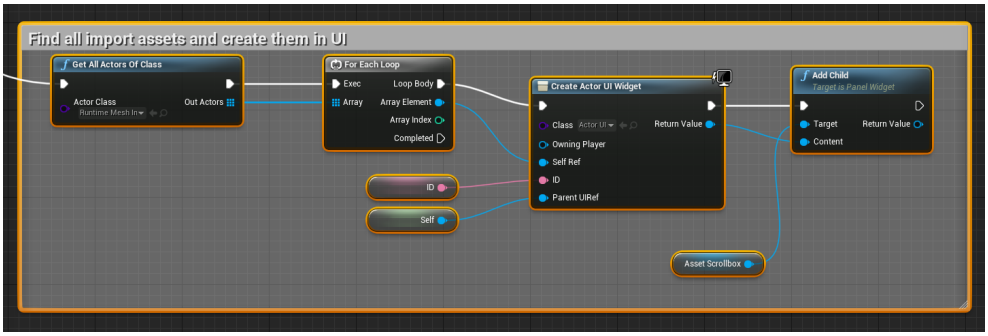
Uživatelské rozhraní pro výběr assetu je stále ten samý widget, jako pro výběr modelu, pouze se opět přepne zobrazovaný potomek widget přepínače. Po navolení určitého 3D objektu a jeho verze se tedy přepne zobrazovaný prvek widget přepínače, který je již naplněn reprezentacemi assetů.

Seznam všech assetů, které jsou schopny importu, je získán Unreal Engine funkcí *Get All Actors Of Class* [21], která vrátí pole všech assetů typu Actor patřících do specifické třídy 4.10. Tato funkce se volá ihned při konstrukci rodičovského widgetu a tím se uživatelské rozhraní plní reprezentacemi assetů ihned při vzniku.



Obrázek 4.8: Obrázek ukazující sestavení GET požadavku s autorizací v hlavičce

Obrázek 4.9: Obrázek ukazující reprezentaci 3D Objektu vytvořenou v Unreal Engine. I když je reprezentace připravená na zobrazování verzí, z důvodu nevyhovující reprezentace na API je tento argument při zobrazení skrytý.



Obrázek 4.10: Obrázek ukazující vyhledání všech assetů schopných importu a následné vytvoření jejich reptezentace. Nakonec je asset přidán do rodičovského widgteu.

4.1.4.4 Import z API

Po výběru assetu, ve kterém se bude model sestrojovat, se přepne zobrazovaný potomek widget přepínače a zobrazí se uživatelské rozhraní importu 4.11, vytvořené podle návrhu 3.11. Uživatelské rozhraní poskytne uživateli základní informaci o zdroji a cíli importu a uživatel si bude schopen zvolit, zdali bude model při importu uložen do paměti.

Ukládání do paměti probíhá pomocí SaveGame objektu. V SaveGame objektu je vytvořená mapa s ID 3D objektu jakožto klíčem a speciálně vytvořenou strukturou jakožto hodnotou. Vytvořená struktura pak obsahuje informaci o verzi 3D objektu, jméno 3D objektu, ID 3D objektu, ID modelu a data nutná k sestrojení daného modelu (informace o vrcholech, trojúhelnících, normálách, UV, barvách vrcholů a tečnách modelu).

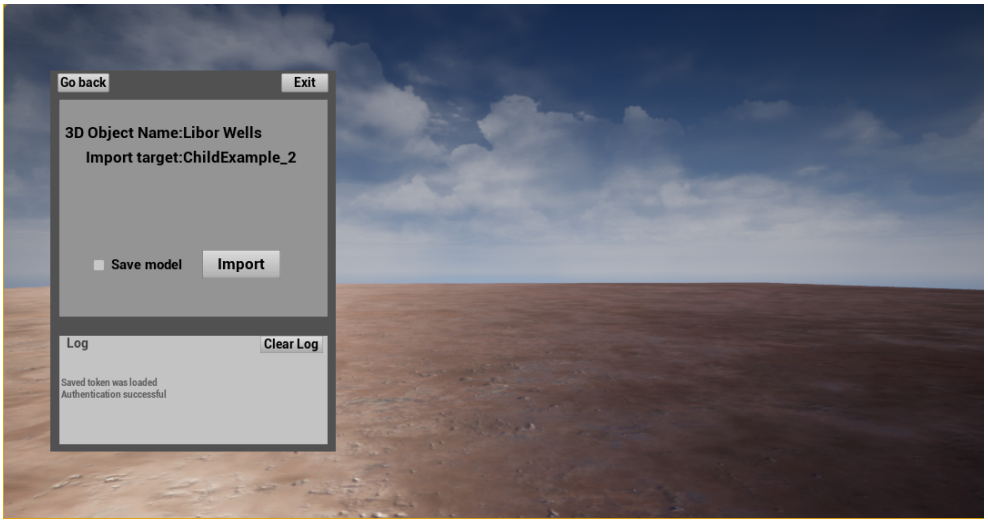
Při samotném importu získá zásuvný modul pomocí GET požadavku pro konkrétní ID modelu model data. Data jsou zpracována pomocí zásuvného modulu RuntimeMeshLoader [18], rozšířeného o assimp funkci *ReadFileFromMemory* [22]. Nová funkcionalita byla do kódu RuntimeMeshLoader přidána, aby bylo možné modely importovat rovnou z paměti, bez nutnosti dočasného ukládání dat do souboru.

Pokud proběhne zpracování dat úspěšně, budou výstupní data využita v implementované funkcionalitě pro sestrojení modelu v assetu schopného importu. Tato funkcionalita pak sestrojí v běžící scéně importovaný model.

Nakonec je do instance uloženo ID importovaného 3D objektu pro účely znovu sestrojení při opětovném zapnutí aplikace.

4.1.4.5 Uložení informací o sestrojených modelech

Uživateli je po otevření hlavního menu umožněno uložit do SaveGame objektu informaci o assetech a jejich zkonstruovaných modelech. Toho je docíleno implementovanou funkcionalitou v assetech schopných importu, která poskytne



Obrázek 4.11: Obrázek ukazující uživatelské rozhraní importu při spuštěné aplikaci

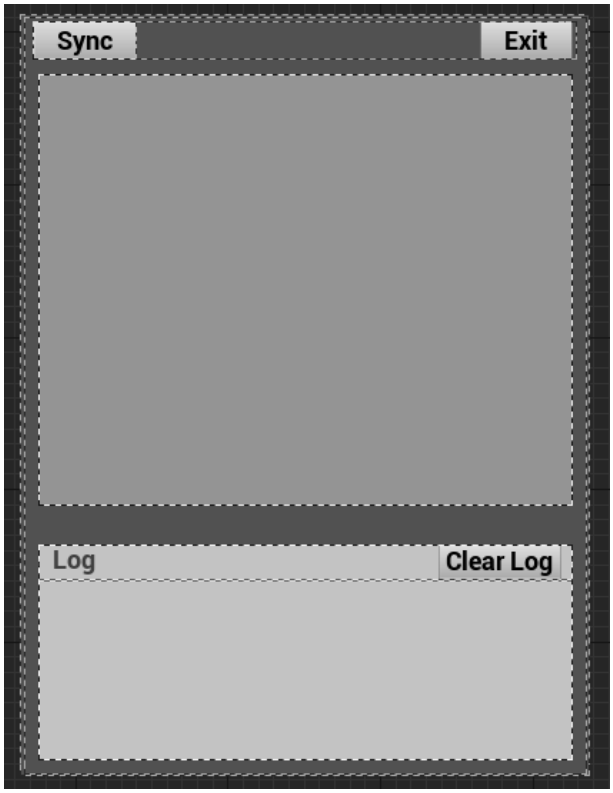
potřebné informace pro uložení. Data získaná funkcionalitou jsou pak uložená do mapy vytvořené v SaveGame objektu. Reference na asset je uložená jako klíč a ID sestrojeného 3D objektu je uložené jako hodnota. Tato funkcionalita slouží pro znovu sestrojení modelů, které je zmíněno níže.

4.1.4.6 Znovu sestrojení modelů na základě uložených informací

Pokud uživatel sestrojí ve scéně model a následně aplikaci vypne, při příštím zapnutí aplikace se model již znovu nesestrojí. Aby uživatel nemusel pokaždé po zapnutí aplikace znovu všechny modely importovat a sestrojovat, implementuje můj zásuvný modul funkcionalitu uložení 4.1.4.5 a následné znovu sestrojení modelů.

Uživatel tuto funkcionalitu nalezne v hlavním menu. Po aktivaci bude uživatel nucen k přihlášení z důvodu nutnosti komunikace s API. Pokud přihlášení proběhne úspěšně, získá zásuvný modul za pomoci Unreal Engine funkce *Get All Actors Of Class* [21] všechny instance assetů, které jsou schopné importu a nachází se ve scéně. U všech získaných assetů je pak volána implemetnovaná funkcionalita pro znovu načtení.

Funkcionalita zkontroluje, zdali se reference instance assetu nachází v uložených datech. Pokud se asset v uložených datech nenajde, neprovádí nic, pokud se v datech nachází, získá z dat ID 3D objektu, který by měl sestrojit. ID 3D objektu je následně vyhledáno v mapě uložených modelů. Pokud je 3D objekt nalezen, je pouze načten z paměti, pokud však 3D objekt v datech nalezen není, je automaticky proveden import daného modelu z API.

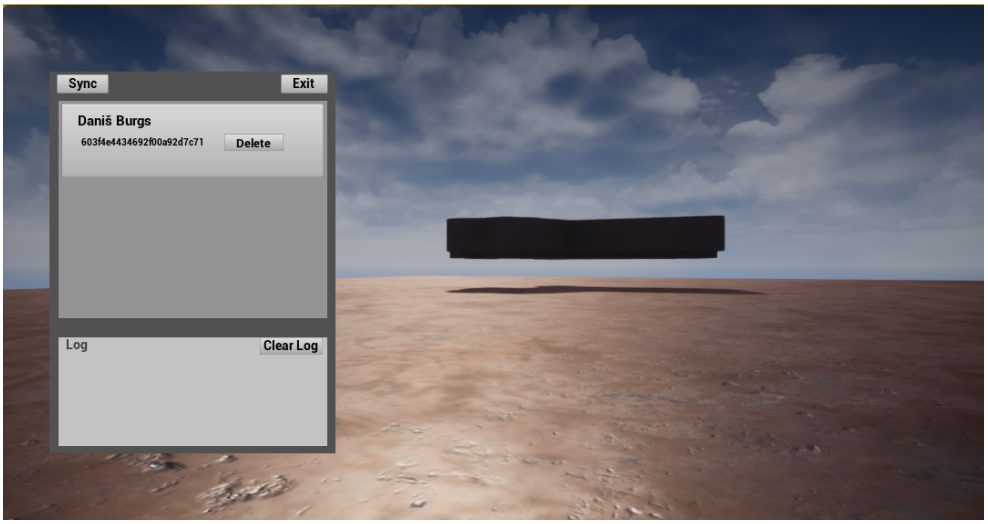


Obrázek 4.12: Obrázek ukazující uživatelské rozhraní zobrazení uložených modelů, které je vytvořeno v Uneral Engine

4.1.4.7 Uložené modely

V hlavním menu se nachází také funkcionalita pro zobrazení přehledu uložených modelů. Po zvolení této funkcionality je sestrojeno uživatelské rozhraní 4.12 podle návrhu 3.13. Sestrojený widget následně projde všechny uložené modely v SaveGame objektu a sestrojí jejich reprezentaci v uživatelském rozhraní 4.13. Reprezentace je tvořena podle návrhu 3.14. Všechny reprezentace jsou sestrojeny ve stejném stavu a tím je stav *akutální*.

Stav se však může změnit po provedené synchronizaci, která však vyžaduje přihlášení. Pokud uživatel zvolil synchronizaci a přihlášení proběhlo úspěšně, je odeslán GET požadavek na ID každého uloženého 3D objektu. Odpovědi od API jsou pak srovnávány s uloženými daty. Protože, jak již bylo řečeno, reprezentace verzí není na API plně funkční, může se v rrealizaci reprezentace modelu nacházet pouze ve dvou stavech. Pokud se uložený 3D objekt na API stále nachází, zůstává reprezentace stále ve stejném stavu. Pokud se 3D objekt na API nepodaří najít, je pak stav reprezentace změněn na *odstraněný* a její barva je změněna na červenou. Data o snychronizaci jsou ztracena po zavření uživatelského rozhraní.



Obrázek 4.13: Obrázek ukazující reprezentaci uloženého objektu při běhu aplikace

4.1.4.8 Aktualizace uloženého modelu

Tato funkcionalita v realizaci nebyla implementovaná z důvodu nevyhovujívího stavu verzí na API.

4.1.4.9 Odstranění uloženého modelu

Objekt je možné za pomoci uživatelského rozhraní odstranit. Tato funkcionalita pouze odstraní prvek nalezaný za pomoci ID 3D objektu z mapy SaveGame objektu.

4.1.4.10 Vložení uloženého modelu

Pokud dojde k ukládání 3D objektu se stejným ID, které už se v mapě nachází, dojde k přepsání starého uloženého modelu.

4.1.4.11 Sestrojení uloženého modelu

Pokud dojde k výběru určitého uloženého modelu, zobrazí uživatelské rozhraní všechny dostupné instance assetů schopných importu. Tato operace probíhá velmi podobným způsobem, jako v podsekci 4.1.4.3.

Po výběru assetu se spustí sestrojení modelu. K tomu jsou využita uložená data a implementovaná funkcionalita pro sestrojení modelu v assetu schopném importu.

4.1.5 Obsah složek v RuntimeImportPlugin content

4.1.5.1 Save

Složka *Save* obsahuje všechny SaveGame objekty využívané k ukládání potřebných informací a struktury, které jsou v SaveGame objektech použity.

4.1.5.2 UI

Složka *UI* obsahuje všechny widget třídy využívané k sestrojení uživatelských rozhraní. Složka také obsahuje podsložku *Subwidgets*, která obsahuje widget reprezentace využívaných objektů.

4.1.5.3 BlueprintFunction a BlueprintMacroLibrary

Složky *BlueprintFunction* a *BlueprintMacroLibrary* obsahují vytvořené funkcionality a takzvaná makra, které využívá zásuvný modul.

Závěr

Cílem práce bylo vyvinout prototyp zásuvného modulu sloužícího k importu 3D modelů z databáze projektu Věnná města českých královen do Unreal Engine. Byla provedena analýza projektu Věnná města českých královen a dále analýza vývojových možností pro Unreal Engine. Z analýzy vyplynulo, jakým způsobem je možné cíl práce realizovat a na základě těchto zjištění bylo navrženo řešení. Návrh řešení byl použit pro implementaci prototypu zásuvného modulu, a to jak jeho funkční části, tak uživatelského rozhraní pro jeho ovládání. Popis způsobu realizace byl zachycen v tomto dokumentu. Závěrem tedy mohu říci, že cíle práce bylo dosaženo.

Literatura

[1] Epic Games: *Plugins. How to create Unreal Engine plugins. [online].* [cit. 4.4.2021]. Dostupné z: https://docs.unrealengine.com/en-US/ProductionPipelines/Plugins/index.html

[2] Epic Games: Unreal Engine. Dostupné z: https://www.unrealengine.com/en-US/

[3] Unity Technologies: Unity. Dostupné z: https://unity.com/

[4] The 10 Best Video Game Engines: 2021 Edition. *GameDesign [online]*, březen 2021, [cit. 13.5.2021]. Dostupné z: https://www.gamedesigning.org/career/video-game-engines/

[5] doc. Mgr. Petr Grulich, Ph.D.: Věnná města českých královen. Dostupné z: https://www.kralovskavennamesta.cz/index.html

[6] Vančura, D.: *Věnná města českých královen - jádro.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 7 2020.

[7] Blender Foundation: Blender. Dostupné z: https://www.blender.org/

[8] Autodesk: 3D Studio Max. Dostupné z: https://www.autodesk.cz/products/3ds-max/overview?term=1-YEAR

[9] Epic Games: *Static Mesh Components [online].* [cit. 7.5.2021]. Dostupné z: https://docs.unrealengine.com/en-US/Basics/Components/StaticMesh/index.html

[10] Epic Games: *Procedural Mesh [online].* [cit. 7.5.2021]. Dostupné z: https://docs.unrealengine.com/en-US/BlueprintAPI/Components/ProceduralMesh/index.html

[11] Epic Games: *Scripting the Editor using Python. Describes how to use Python in the Unreal Editor to script content production tasks. [online].* [cit. 4.4.2021]. Dostupné z: https://docs.unrealengine.com/en-US/ProductionPipelines/ScriptingAndAutomation/Python/index.html

[12] Kulling, K.: Assimp. Dostupné z: https://github.com/assimp/assimp

[13] 20tab: *Your First Automated Pipeline with UnrealEngine-Python [online].* [cit. 10.12.2020]. Dostupné z: https://github.com/20tab/UnrealEnginePython/blob/master/tutorials/YourFirstAutomatedPipeline.md

[14] Epic Games: Unreal Python 4.26 (Experimental) documentation, Asset tools[online]. Dostupné z: https://docs.unrealengine.com/en-US/PythonAPI/class/AssetTools.html

[15] BleuRaven, xavier150: Blender for UnrealEngine. Dostupné z: https://github.com/xavier150/Blender-For-UnrealEngine-Addons

[16] Alyamkin, V.: VaRest. Dostupné z: https://github.com/ufna/VaRest

[17] Unreal Engine - MarketPlace. Dostupné z: https://www.unrealengine.com/marketplace/en-US/store?sessionInvalidated=true

[18] GameInstitute: RuntimeMeshLoader. Dostupné z: https://github.com/GameInstitute/RuntimeMeshLoader

[19] Epic Games: *Create Mesh Section [online].* [cit. 7.5.2021]. Dostupné z: https://docs.unrealengine.com/en-US/BlueprintAPI/Components/ProceduralMesh/CreateMeshSection/index.html

[20] Abovitz, R.: Magic leap, inc. Dostupné z: https://www.magicleap.com/en-us

[21] Epic Games: *Get All Actors Of Class [online].* [cit. 7.5.2021]. Dostupné z: https://docs.unrealengine.com/en-US/BlueprintAPI/Utilities/GetAllActorsOfClass/index.html

[22] Kulling, K.: Assimp::Importer Class Reference - documentation [online]. Dostupné z: http://assimp.sourceforge.net/lib_html/class_assimp_1_1_importer.html

Příloha **A**

Seznam použitých zkratek

3D Three dimensional (trojrozměrný)

VR Virtual reality (virtuální realita)

VMČK Věnná města českých královen)

ID Identification (identifikace)

UI User interface (uživatelské rozhraní)

API Application programming interface

REST Representational state transfer

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
  - `BPDemo.exe` ..... spustitelná verze Unreal Engine 4 projektu
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `RuntimeImportPluginPackage` ..... Unreal Engine 4 projekt
    - `BPDemo` ..... Unreal Engine 4 plugin
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `media` ..... video z práce
  - `showcase.mp4` ..... videoukázka funkcí zásuvného modulu