Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Interaktivní průvodce Fakultou elektrotechniky a informatiky Univerzity Pardubice

Bakalářská práce

2021 Lukáš Milar

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Lukáš Milar**
Osobní číslo:	**I17123**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Interaktivní průvodce Fakultou elektrotechniky a informatiky Univerzity Pardubice**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem této práce bude vytvořit aplikaci umožňující interaktivní prohlídku modelu budovy Fakulty elektrotechniky a informatiky Univerzity Pardubice.
Aplikace bude umožňovat tyto funkcionality:

- pohyb po areálu fakulty vně i uvnitř budovy
- zobrazení informací na průhledovém displeji o současné poloze a názvu dané místnosti a doplňující informaci o místnosti.

Teoretická část se bude zabývat problematikou herních enginů, aplikačním softwarem z oblasti 3D počítačové grafiky, modelovacími technikami a skriptováním v herním enginu.
Dále bude popisovat samotné vytváření modelů, import do herního enginu a následné sestavení celé scény a dokončení celé aplikace.
Pro vytvoření aplikace bude využit herní engine Unity 3D, skriptovací jazyk C# a aplikační software z oblasti 3D počítačové grafiky (Blender , Autodesk Maya, ?).

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

UNITY 3D. Unity User Manual. Unity3d.com [online]. 2019 [cit. 2019-10-12]. Dostupné z: https://docs.unity3d.com/Manual/index.html
HOCKING, Joseph. *Unity in action: multiplatform game development in C#*. Second edition. Shelter Island, NY: Manning Publications, 2018. ISBN 978-1617294969.
VAUGHAN, William. *Digital modeling*. Berkeley, CA: New Riders, 2012. ISBN 978-0321700896.

Vedoucí bakalářské práce: **Ing. Zbyněk Kopecký**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2020**
Termín odevzdání bakalářské práce: **14. května 2021**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 26. února 2021

Prohlašuji:

Práci s názvem Interaktivní průvodce Fakultou elektrotechniky a informatiky Univerzity Pardubice jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 11. 5. 2021

Lukáš Milar

PODĚKOVÁNÍ

Chci poděkovat vedoucímu mé bakalářské práce, Ing. Zbyňkovi Kopeckému za veškerou pomoc při tvorbě této práce včetně odborných rad. Dále chci poděkovat mé rodině za veškerou pomoc a morální oporu po celou dobu mých studií. Mé sestře chci obzvláště poděkovat za veškeré rady v průběhu studia.

ANOTACE

Tato bakalářská práce se zabývá tvorbou interaktivní prohlídky Fakulty elektrotechniky a informatiky Univerzity Pardubice v herním enginu Unity3D. Nejprve je projednána problematika herních enginů a vysvětleny důvody pro volbu zmíněného enginu. Poté jsou představeny modelovací techniky a aplikace, ze kterých je následně podrobněji popsán Blender. V další části je popsána analýza vytvářeného prostředí, práce s referenčními materiály a samotná tvorba modelů s texturami. Následně je navržena struktura samotného projektu. Na jejím základě je v Unity vytvořena konečná scéna a implementována logika pro pohyb, interakci a zobrazování informací na průhledovém displeji. Nakonec je projekt sestaven na platformu WebGL.

KLÍČOVÁ SLOVA

Blender, 3D modelování, Unity3D, WebGL, HUD, informační systém, tvorba textur, herní engine, C#

TITLE

An interactive guide of the Faculty of Electrical Engineering and Informatics, University of Pardubice.

ANNOTATION

This bachelor thesis deals with the creation of an interactive tour of the Faculty of Electrical Engineering and Informatics, University of Pardubice in the game engine Unity3D. First, the topic of game engines is discussed and the reasons of choosing the mentioned engine are explained. Then the modeling techniques and applications are introduced and Blender is the one from the list which is subsequently described in more detail. The next part describes the analysis of the created environment as well as the work with reference materials and the creation of the models and textures. The structure of the project itself is then designed. On its basis, the final scene is created in Unity along with the implementation of logic for movement, interaction and displaying of information using a heads-up display. Finally, the completed project is built for the WebGL platform.

KEYWORDS

Blender, 3D modeling, Unity3D, WebGL, HUD, Information System, Texture Creation, Game Engine, C#

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK

PDF	Portable Document Format
VR	Virtual Reality
AR	Augmented Reality
PBR	Physically Based Rendering
FPS	First Person Shooter
NURBS	Non-Uniform Rational B-Splines
PBS	Physically Based Shading
XML	Extensible Markdown Language
JSON	JavaScript Object Notation
FBX	Filmbox
PNG	Portable Network Graphics
CORS	Cross-Origin Resource Sharing
UI	User Interface
DXF	Drawing Exchange Format

ÚVOD

Autor si toto téma zvolil na základě svých zkušeností s modelováním architektonických modelů. Volba tématu dále proběhla též na základě zkušeností s herním enginem Unity3D a s tím svázanými znalostmi s tvorbou skriptů pomocí jazyka C# v tomto enginu.

Cílem práce je v teoretické části stručně představit základní požadavky na finální aplikaci a s tím spojenou problematiku herních enginů a jejich možných způsobů využití v 3D interaktivních aplikacích. Dále bude na základě metody komparace vybrán herní engine Unity3D, který bude podrobněji popsán a ve kterém bude tato práce vyhotovena. Teoretická část bude rovněž popisovat modelovací techniky, a aplikace které je možné využít pro tyto účely. Z těchto aplikací bude věnována pozornost Blenderu, ve kterém budou později potřebné modely vytvořeny. Mimo jiné zde bude osvětlena problematika tvorby textur dle principů tzv. PBR a jejich následné použití na 3D modelech. V závěru této části bude popsán celý proces tvorby modelů pro použití k této činnosti.

Hlavním cílem praktické části práce je vytvoření aplikace, která bude schopná v reálném čase zobrazit model budovy Fakulty elektrotechniky a informatiky Univerzity Pardubice a umožní uživateli navigovat scénu, ve které se tento model bude nacházet. Tato aplikace bude též schopna zobrazit informace o navštěvované místnosti či současné poloze uživatele uvnitř scény. Zmíněné informace budou zobrazeny formou průhledového displeje.

Tato aplikace může prakticky sloužit kupříkladu jako možnost uchazečů prohlédnout si budovu fakulty odkudkoliv a případně se dozvědět více o studiu na této fakultě, čímž též může posloužit k účelům propagace. Aplikaci též ocení nově příchozí studenti, kterým pomůže orientovat se uvnitř budovy a tím i snáze nalézt učebny, ve kterých se mají v danou dobu nacházet.

1 PŘEHLED DOSTUPNÝCH HERNÍCH ENGINŮ A JEJICH KOMPARACE

Pro vytvoření interaktivní aplikace umožňující pohyb po budově fakulty je potřeba zajistit především vykreslování scény a reakci na vstup uživatele. Vzhledem k interaktivnímu aspektu je též potřeba oba zmíněné požadavky vykonávat v reálném čase. K těmto účelům lze využít některý z existujících herních enginů, či vytvořit si vlastní.

Jak uvádí Ward, základní účel herního enginu je umožnit vývojářům věnovat pozornost unikátním detailům jejich her prostřednictvím abstrakce obecných úkonů jako jsou fyzika, zpracování vstupu či vykreslování. [1]

Jak ve svém článku uvádí Charbonneau, vytváření vlastního enginu mnohdy vede ke ztrátě motivace a následnému opuštění projektu [2]. Zároveň by vytvoření vlastního herního enginu již ze své podstaty vyžadovalo vynaložení značného úsilí pro vytvoření dostatečně stabilní vývojové platformy a tato možnost je tak pro tuto práci nevhodná. Na základě těchto informací můžeme dojít k závěru, že vhodnou cestou pro tento projekt je využít již existující engine.

Před vytvářením prototypu by měl být vytvořen co nejkompletnější seznam vlastností a je lepší určit na jaké platformy se má vyvíjet hned po jejich vymezení, jelikož následné dodatečné přepisování na další platformy je nákladné. [2]

Analýzou požadavků na tento projekt je zřejmé, že výsledná scéna bude sestavena ze 3D modelů a tudíž v enginu musí být možné scénu vykreslovat ve 3D. Dále je nutné, aby zvolený engine poskytoval podporu pro vytváření interaktivního uživatelského rozhraní. Vzhledem k účelu aplikace musí zvolený enigne umožňovat jeho užití a publikování výsledného projektu na stanovenou platformu bez nutnosti zakoupení licence. Musí být také schopen sestavení výsledného projektu na platformu WebGL a případně také další platformy. V neposlední řadě musí být dostatečná podpora skriptování herní logiky pomocí vhodného jazyka a dostatečná dokumentace pro tuto činnost.

1.1 Unreal Engine 4

Jak na svých stránkách uvádí společnost Epic Games, „*Unreal Engine je kompletní balík vývojových nástrojů pro kohokoliv, kdo pracuje s real-time technologií. Od návrhové vizualizace a filmové zážitky po hry vysoké kvality napříč PC, konzolemi, VR a AR, Unreal Engine Vám poskytne vše, co potřebujete k začátku, vydání, růstu a vystoupení z davu.*“ [3] (překlad autora)

Vydat výsledný projekt na platformu HTML5 je též podporováno, avšak od verze 4.24 byla tato možnost přesunuta do externího rozšíření spravovaného komunitou, a tak již není součástí samotného enginu [4]. Vzhledem ke zmíněnému přesunu podpory WebGL ze samotného enginu lze konstatovat, že tento engine nemusí nutně v tomto ohledu odpovídat účelu tohoto projektu.

Uživatelské rozhraní zde lze tvořit prostřednictvím Unreal Motion Graphics UI Designer (UMG) pomocí upravitelných blueprint widgetů. Pomocí blueprintů lze také vytvářet herní logiku jako například interakci s objekty. Zároveň je však možné příslušnou logiku vytvářet pomocí jazyka C++, kterým lze také rozšiřovat funkce samotného enginu. [3]

Unreal Engine lze používat pod licencemi **Unreal Engine End User License Agreement for Publishing**, **Unreal Engine End User License Agreement for Creators** či pod licencí vytvořenou na míru. První uvedená je zdarma, ale při překročení zisku 1 000 000 $ ze zpeněženého běžně dostupného produktu je vyžadováno odvádění licenčních poplatků ve výši 5 % z daného zisku. Druhá uvedená je též zdarma a nevyžaduje odvádění žádných licenčních poplatků. Nelze s ní však publikovat běžně dostupné nabídky a hodí se tak spíše například pro speciální projekty pro konkrétní klienty nebo obsah zdarma. [5]

1.2 CryEngine V

Jak se lze dočíst v dokumentaci, jazyk LUA je zřejmě primárním jazykem na psaní skriptů [6]. Dále jsou pro tyto účely podporovány jazyky C++ a C# [7]. K těmto účelům však lze využít také systém pro vizuální skriptování jménem Flowgraph [8].

Prostřednictvím modelu založeném na licenčních poplatcích je možné získat přístup k celému CRYENGINE včetně jeho zdrojových kódů napsaných v jazyce C++. Při překročení zisků ve výši 5 000 $ za rok za projekt je nutné z těchto odvádět licenční poplatky ve výši 5 %. Nezávisle na tom, zda je v plánu výsledný produkt zpeněžit je nutné jej u Cryteku registrovat. V případě komerčního produktu pak minimálně 3 měsíce před jeho vydáním. Dle licenčního ujednání je také nutné zahrnout logo Cryteku a větu připisující zásluhy v projektu, přičemž ve všech významných marketingových materiálech musí být viditelné. [9]

Dle oficiálních stránek tohoto enginu lze výsledný projekt publikovat na konzole, PC, VR a v budoucnu také na mobilní zařízení [10]. Vzhledem ke zjevné absenci podpory platformy WebGL zvolené pro tento projekt lze tudíž konstatovat, že tento engine není vhodnou volbou pro tento projekt.

1.3 Unity 3D

Unity Editor umožňuje vytvářet cutscény, 3D či 2D scény nebo animace. Během vývojového cyklu lze provádět rychlé úpravy a iterace s předběžným náhledem v reálném čase díky základní platformě Unity [11]. Díky těmto vlastnostem lze tedy na této platformě vytvořit potřebný typ scény pro tento projekt.

Aktuální verze v době psaní této práce jsou Unity 2019.4 LTS a Unity 2020.2 TECH. První uvedená bude po dobu 2 let dostávat pravidelné aktualizace a disponuje stejnou sadou funkcí jako Unity 2019.3. Hodí se tak pro projekty připravené pro produkci. Druhá zmíněná je aktualizovaná po týdnu do doby, než bude vydána další verze TECH a obsahuje nejnovější funkce [11]. Pro účely tohoto projektu je zřejmě vhodnější verze 2019.4 LTS vzhledem k dlouhodobější podpoře a vyšší stabilitě.

Jak je uvedeno v dokumentaci, komponenty řídí chování herních objektů (GameObjects) a implementace vlastní herní logiky lze na této platformě dosáhnout vytvořením nových komponent pomocí skriptů. Pro psaní těchto skriptů je jako výchozí možnost používán jazyk C# [12, s. Creating and Using Scripts]. Můžeme dojít k závěru, že implementace potřebné logiky je na této platformě v dostatečné míře možná.

Pro používání Unity lze zvolit z vícero typů licencí, přičemž už žádného typu nabízených licencí není potřeba odvádět licenční poplatky. Pro individuální použití jsou nabízeny plány Student a Personal a na týmy jsou cíleny plány Plus, Pro a Enterprise. Plán Personal mohou zdarma používat ti, jejichž obrat či dosažené financování v posledních 12 měsících nepřekročily částku 100 000 $. Tento plán umožňuje kromě přístupu k nejnovější verzi vývojového prostředí Unity také přístup ke zdrojům pro studium a začátku práce s Unity [13]. Pro realizaci tohoto projektu jej lze dle informací uvedených výše používat zdarma, čímž splňuje další z bodů určených na začátku této kapitoly. Pro více informací o ostatních licencích viz stránku obchodu ([13]).

Pro vydání výsledného projektu je podporováno více než 20 platforem včetně mobilních zařízení, AR a VR, stolních PC, konzolí či WebGL [14]. Vzhledem k této skutečnosti lze tedy konstatovat, že tento engine splňuje výše uvedený požadavek na podporované platformy.

1.4 Shrnutí

Na základě informací popsaných výše lze konstatovat, že pro účely tohoto projektu je vhodné zvolit herní engine Unity 3D. Podobně jako ostatní zmíněné možnosti jej lze používat pro projekt této velikosti zdarma, avšak jako jediný poskytuje oficiální podporu publikace projektu na platformu WebGL přímo v samotném enginu. Na rozdíl od Unreal Engine také již v základu umožňuje psaní herní logiky prostřednictvím jazyka C#, se kterým má autor více zkušeností než s jazyky C++ či LUA.

2 MODELOVACÍ TECHNIKY A APLIKACE PRO VYTVÁŘENÍ MODELŮ

Pro vytvoření modelů vhodných pro účely tohoto projektu je nutná znalost technik, kterými je lze vytvářet. V této kapitole jsou možné techniky pro vytváření 3D modelů popsány a následně je zhodnoceno, které z nich je vhodné použít. Na základě těchto vybraných technik jsou poté prozkoumány dostupné aplikace pro tvorbu 3D modelů umožňující tyto techniky aplikovat.

Digitální modely lze rozdělit na polygonové modely, NURBS povrchy a subdivision surafes. První zmíněné jsou tvořeny polygony, hranami a body. Objekty z druhé kategorie jsou založeny na křivkách a hladkých površích mezi nimi. Třetí typ je také tvořen polygony, hranami a body, ale zároveň sdílí určité výhody NURBS [15, s. 102]. V této práci bude věnována pozornost pouze polygonovým modelům.

Bod, zvaný též vrchol (vertex), je základním prvkem 3D modelu. Samotné body nemohou být vykresleny, jelikož postrádají výšku, šířku a hloubku. Přesto však existují ve 3D prostoru na konkrétních souřadnicích X, Y a Z. **Hrana** vznikne propojením dvou bodů, zatímco spojením třech bodů vznikne polygon. V nepřerušené polygonové síti mohou být dané body součástí vícero polygonů. [15, s. 102–103]

Povrch 3D objektu je definován **polygony** neboli ploškami. Vytváření polygonů z jednoho či dvou bodů je v některých 3D aplikacích umožněno, ale častěji jsou vyžadovány alespoň 3 body. Polygony tvořené třemi body se označují jako trojúhelníky (tris). Označení čtyřúhelník (quad) se používá pro polygony skládající se ze 4 bodů. Označení n-úhelník (n-gon) značí polygon s n stranami a náleží každému polygonu s počtem bodů vyšším než 4. [15, s. 108–109]

2.1 Přehled modelovacích technik

Tvůrce může pro tvorbu modelu využít libovolných technik dle toho, jak se mu s nimi pracuje. Jakkoliv je snadné ustálit se v určitém způsobu práce, tato strategie může vést ke snížené efektivitě modelování kvůli omezování vlastní sady nástrojů [15, s. 121–122]. Na základě této skutečnosti lze konstatovat, že výhodná strategie spočívá v kombinaci technik uvedených níže.

Nejstarší a v průmyslu stále široce používaný přístup je tzv. **build out**. V rámci tohoto přístupu se již od prvního polygonu vytváří terciální detaily a po dokončení dané části modelu se stejným způsobem postupuje na další, dokud není model dokončen. Nejčastěji se používá pro retopologii (předělávání topologie objektu) či tvorbu realistických modelů hlavy. Pod tento přístup patří

techniky **point by point** a **edge extend**. V první zmíněné jsou polygony stvořeny až po vytvoření příslušného tvaru objektu pomocí jednotlivých bodů a hodí se kupříkladu pro tvorbu písmen při absenci fontu. Druhá uvedená technika spočívá v tvorbě nových polygonů pomocí vytažení hran existujícího polygonu a je vhodná pro tvorbu široké škály objektů. Nejčastěji se však využívá při tvorbě realistických hlav [15, s. 122–123]. Ačkoliv se v tomto projektu nevyskytuje žádný model postav, stále najde tato technika využití například při tvorbě architektonických prvků.

Přístup **primitive modeling** spočívá v kombinování jednoduchých geometrických tvarů, kupříkladu kostky, a jejich úpravy do požadované podoby. Mezi digitálními modeláři se jedná o populární techniku díky mnoha nástrojům umožňujícím s takto vytvářenými tvary pracovat v rámci 3D aplikací. Primárně nalezne využití pro tvorbu modelů pevných povrchů (hard surface) jako jsou například stoly, klávesnice či monitor, jejichž tvar je možné zachytit pomocí jednoduchých tvarů [15, s. 124]. Vzhledem k těmto informacím lze konstatovat, že tato technika v tomto projektu nalezne využití například při tvorbě nábytku.

Podobně jako u předešlé techniky začíná **box modeling** s jednoduchými tvary. Namísto kombinace vícero těchto tvarů se však pracuje pouze s jedním (jak napovídá název, jedná se zpravidla o kostku) a pro vytvoření požadovaného tvaru je dotvářena potřebná geometrie pomocí zkosení či vytažení jednoho či více polygonů. Obvykle je tato technika použita v kombinaci subdivision surfaces, pomocí kterých je zjemněna původní polygonová síť s málo polygony (low-poly), kterou se však i nadále ovládá celkový konečný tvar. Nejčastější využití nachází pro organické modely, jako jsou kupříkladu postavy [15, s. 126]. S ohledem na absenci organických modelů v této práci tato technika nebude zřejmě příliš hojně využívána. Stále však nalezne určité uplatnění při tvorbě modelů nábytku v kombinaci s předešlou technikou u těch částí, které mají být logicky součástí jednotné polygonové sítě (polygon mesh).

V rámci techniky **patch modeling** je tvar definován křivkami, mezi kterými vznikají povrchy zvané patches. Kontrolními body, které tvoří dané křivky jsou tyto povrchy kontrolovány a s minimálním počtem křivek lze kombinací těchto povrchů tvořit složité organické tvary. Použití této techniky bývá v zásadě stejné nezávisle na tom, zda jsou použity NURBS či polygony a první zmíněný způsob je často využíván například při CAD a výrobě [15, s. 127–128]. Tuto techniku lze tak hojně využít pro složitější tvary vyžadující kupříkladu určitou deformaci podle tvaru dané křivky.

Nejblíže tradičnímu sochaření je technika **digital sculpting**. V rámci této techniky lze pomocí systému založeném na štětcích manipulovat s polygonovou sítí o libovolném množství polygonů a bez nutnosti věnovat pozornost topologii. Primárně se tato technika hodí pro vytváření organických modelů, ale lze s ní tvořit také modely pevných ploch. Využití nalézá napříč celým 3D průmyslem včetně herního, kde takto vytvářené modely slouží jako základ pro normálové mapy modelů využívaných ve hrách [15, s. 128–129]. Vzhledem k podstatně menším zkušenostem autora s touto technikou a absence organických modelů v rámci tohoto projektu lze konstatovat, že tato technika zde nenalezne využití a efektivnější řešení představuje kombinace výše popsaných technik pro dosažení potřebných tvarů.

Techniku **3D skenování** lze označit za ekvivalent například použití fotoaparátu, avšak v případě 3D objektů, jelikož s ní lze povrchová data skutečného objektu snímat. Polygonová síť převedená z těchto dat obvykle obsahuje miliony bodů, ale lze ji využít na cokoliv od malých sošek po krajinu a nachází tak uplatnění v celém 3D průmyslu. Pro aplikaci této techniky je potřeba mít k dispozici 3D skener, kterým se provede sběr dat [15, s. 131]. Vzhledem k absenci potřebného zařízení nebude možné v tomto projektu tuto techniku použít.

Při modelování lze využít také nástroje určené pro texturování a animaci. Kupříkladu pomocí techniky **texture displacement** lze pomocí textur manipulovat s body polygonové sítě a s minimální námahou tvořit například modely terénu [15, s. 134]. S ohledem na typ většiny tvořených modelů lze konstatovat, že tato technika zde také nenalezne využití. V případě tvorby terénu by však byla zřejmě nepostradatelná pro dosažení požadovaných výsledků.

2.2 Přehled modelovacích aplikací

Autodesk Maya je se svou širokou škálou nástrojů a funkcí považována za průmyslový standard. Vzhledem k velkému množství nástrojů v ní zahrnutých je možné, že některé z nich daný uživatel nikdy nevyužije. Je poskytována prostřednictvím modelu předplatného [16]. S touto aplikací má autor předešlé zkušenosti z předmětu 3D Grafika vyučovaného na této fakultě.

3ds Max pro modelování obsahuje mnoho nástrojů a využívá se například pro architektonickou vizualizaci. Proces modelování je usnadněn rozsáhlou knihovnou modifikátorů a lze kombinovat procedurální modelovací techniky s modelovacími technikami založenými na přímé manipulaci. Získat jej lze prostřednictvím předplatného či zdarma pokud je daný uživatel student [16]. Vzhledem ke skutečnosti, že hlavní využití těchto aplikací v tomto projektu spočívá v mo-

delování architektonických modelů, je možné konstatovat, že tato aplikace poskytuje více relevantních výhod než první uvedená. Zároveň však autor nemá s touto aplikací příliš mnoho zkušeností, což může vést k přílišnému zpomalení vlivem nutnosti se s tímto programem naučit pracovat.

SketchUp Free je dostupný zdarma přímo v prohlížeči a používat jej je snadné [16]. Autor má s touto aplikací předešlé zkušenosti, ale zároveň lze vzhledem k použití přímo v prohlížeči vyjádřit jisté obavy z hlediska použitelnosti na scénu o komplexitě modelu fakulty a jejího vybavení.

Blender je otevřený software dostupný zdarma a se svou širokou škálou modelovacích nástrojů se jedná o dobrou alternativu placených aplikací pro modelování. Lze s ním dosáhnout výsledků na úrovni ostatních modelovacích aplikací a mnoho dříve známých problémů spojených s neortodoxním přístupem bylo již napraveno [16]. Autor má s touto aplikací ze všech zde uvedených nejvíce zkušeností v oblasti modelování a lze tak konstatovat, že nejlepší strategií je pro účely této práce zvolit právě tuto aplikaci.

2.3 Shrnutí vybraných technik a aplikací

Jak je dokázáno v této kapitole, existuje mnoho modelovacích technik využitelných pro tuto práci. Vzhledem k typu vytvářených modelů však zřejmě naleznou uplatnění především modelovací techniky build out a primitive modeling a v trochu menší míře také techniky box modeling a patch modeling. Pro realizaci těchto technik je zvolena aplikace Blender z důvodu dostatečně široké škály nástrojů a značným zkušenostem autora s touto aplikací.

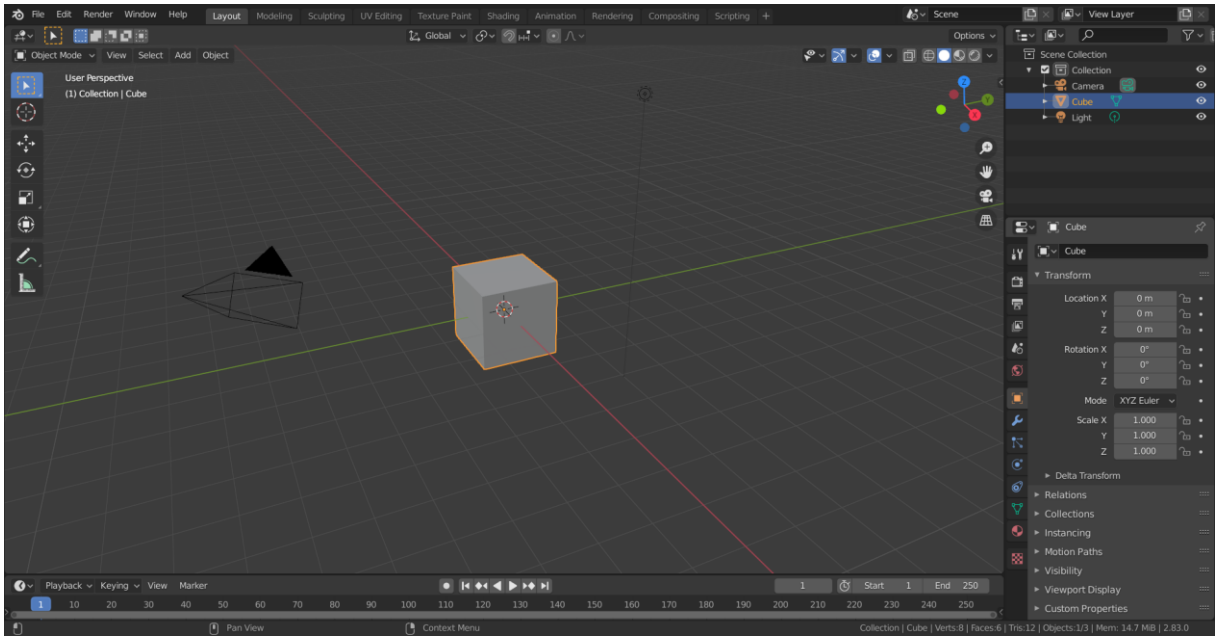
3 MOŽNOSTI PRÁCE S BLENDEREM

Blender ve verzi 2.80 prošel mnoha změnami, ze kterých lze zmínit kupříkladu zcela nové uživatelské prostředí či kolekce nahrazující starý systém vrstev [17, v menu: Getting Started > About Blender > Blender's History]. Pro účely tohoto projektu byla zvolena verze 2.83 a následující sekce tak popisují informace vztahující se k této verzi.

3.1 Základní používání Blenderu

V úvodní obrazovce, která se po spuštění zobrazí ve středu okna lze nalézt možnosti pro vytvoření nových souborů či otevření souborů, se kterými se nedávno pracovalo. Kliknutím kdekoliv v okně Blenderu mimo úvodní obrazovku či stiskem klávesy Esc dojde k zavření této obrazovky. [17, v menu: User Interface > Splash Screen]

Uživatelské prostředí Blenderu je tvořeno třemi hlavními částmi. Navrchu je Topbar, v prostředku Areas a ve spodní části se nachází Status Bar. Pro urychlení práce jsou značně využívány klávesové zkratky, které lze upravovat v nastavení Keymap Editor [17, v menu: User Interface > Window System > Introduction]. Výběr položek se ve výchozím nastavení provádí levým tlačítkem myši, ale lze jej také přenastavit na pravé tlačítko myši [17, v menu: User Interface > Tools > Selecting].



Obrázek 1: Uživatelské rozhraní Blenderu. Zdroj: Autor

Sekci uživatelského prostředí **Topbar** lze dále rozdělit do podsekcí Menus, Workspaces a Scenes & Layers. Sekci Menus lze dále rozdělit na položky App, File, Edit, Render, Window a Help. Všechny tyto položky skrývají další operace. V sekci Workspaces lze pomocí záložek vybrat předdefinované rozvržení okna označované jako Workspace. V poslední jmenované podsekci lze vybrat současnou aktivní scénu a pohledovou vrstvu. [17, v menu: User Interface > Window System > Topbar]

Možností položek menu v části Topbar je mnoho, a proto budou vyjmenovány pouze ty nejzákladnější. Pro kompletní seznam a vysvětlení všech položek menu viz příslušnou sekci dokumentace ([17, v menu: User Interface > Window System > Topbar]). Jak uvádí dokumentace, soubor lze otevřít pomocí File > Open. Uložit jej lze pomocí File > Save či File > Save As… pro specifikování jména a lokace pro uložení. Provést operace Undo a Redo lze pomocí Edit > Undo a Edit > Redo [17, v menu: User Interface > Window System > Topbar].

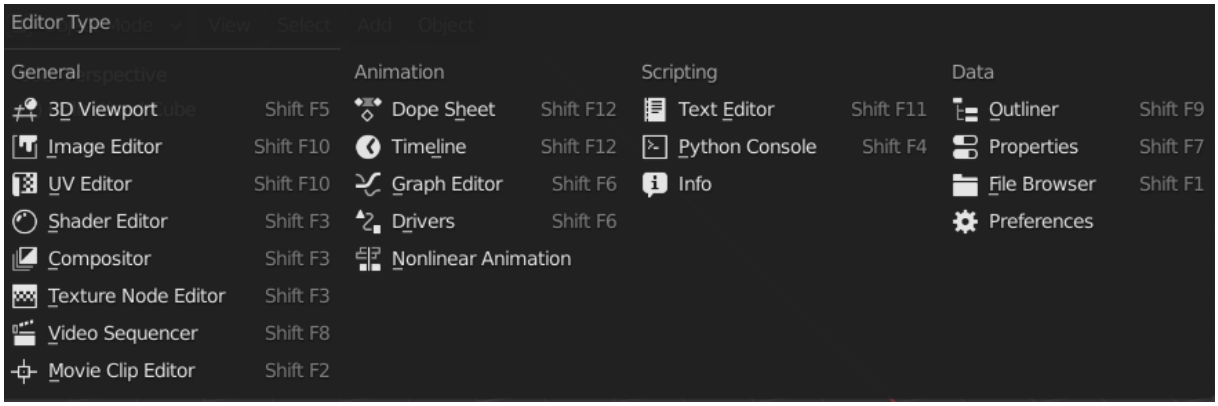
V sekci **Status Bar** se zobrazují kontextuální informace jako klávesové zkratky či varovné zprávy [17, v menu: User Interface > Window System > Status Bar]. Lze tudíž konstatovat, že může být pomocníkem při řešení dílčích kontextuálních problémů či provádění úkonů specifických pro daný kontext. Pro podrobnější popis viz příslušnou stránku dokumentace ([17, v menu: User Interface > Window System > Status Bar]).

Sekce **Areas** je rozdělena do obdélníků označovaných jako oblasti (areas). Tyto obdélníky vyhrazují editorům prostor na obrazovce a jsou ohraničeny zaoblenými rohy. Při najetí myší na hranici mezi dvěma oblastmi lze držením levého tlačítka myši a tažením měnit jejich velikost. Veškeré klávesové zkratky se vykonají pro editor, v jehož oblasti se nachází kurzor myši. Editor obecně slouží pro náhled a úpravu práce přes konkrétní část Blenderu. [17, v menu: User Interface > Window System > Areas]

Všechny editory jsou složeny z regionů, ve kterých mohou být menší prvky jako záložky či panely s tlačítky a widgety. V editoru je vždy viditelný alespoň jeden region, který je označován jako Main region. Všechny editory mají kontejner pro menu a často používané nástroje, které se liší dle typu editoru, zvoleného objektu a režimu. Tento kontejner je nazýván hlavička (header) a může být navrchu či vespod oblasti v podobě malého horizontálního proužku. V levé části oblasti editoru lze klávesou T měnit viditelnost lišty nástrojů s interaktivními nástroji. V pravé části lze klávesou N zobrazit či skrýt postranní panel obsahující panely s nastavením

pro editor a objekty v editoru. Tažením prostředního tlačítka myši lze regionem rolovat v horizontálním i vertikálním směru a pokud region nemá úroveň přiblížení, lze k tomuto úkonu použít kolečko myši. [17, v menu: User Interface > Window System > Regions]

Typ editoru v dané oblasti lze změnit pomocí selektoru, prvního tlačítka zleva v hlavičce editoru [17, v menu: Editors]. Jak je patrné z obrázku níže, těchto editorů existuje vskutku mnoho, a proto bude v této práci věnována pozornost pouze těm, které jsou pro ni relevantní.



Obrázek 2: Typy editorů uvedené v selektoru. Zdroj: Autor

Prostřednictvím editoru **3D Viewport** lze interagovat se 3D scénou například za účelem modelování či práce s animacemi. Jeho hlavička je rozdělena do skupin tlačítek Mode & Menus, Transform Controls a Display & Shading. Možnostmi v první skupině lze přepínat mezi jednotlivými režimy tohoto editoru, používat nástroje pro výběr, vytváření a úpravu objektů v závislosti na současném režimu či používat nástroje pro navigaci ve 3D prostoru. Možnosti v druhé skupině zahrnují kupříkladu úpravu a výběr aktivních orientací transformací, možnost nastavení referenčního vztažného bodu (pivot point) pro nástroje a nastavení nástrojů pro přichytávání či proporční úpravy. Ve skupině Display & Shading lze kupříkladu určit viditelnost jednotlivých typů objektů, nastavit zobrazení pro jednotlivé manipulátory (gizmos), nastavit zobrazení překrytí či měnit stínování v tomto editoru. Lze zde také zprůhlednit celou scénu. [17, v menu: Editors > 3D Viewport > Introduction]

Režimy v editoru 3D Viewport jsou orientovány na objekty a jejich dostupnost se tak může v závislosti na typu vybraného objektu lišit. Výchozí režim je Object Mode a pro většinu typů je také jediný dostupný, jelikož je určen editaci dat jako jsou pozice, rotace či velikost. Režim Edit Mode je určen pro editaci tvaru objektů, které lze vykreslit a je tudíž pro tyto typy objektů též dostupný. Režimy mohou ovlivnit mnoho věcí od dostupných zkratek až po chování celého

editoru [17, v menu: Editors > 3D Viewport > Object Modes]. Pro informace o zbylých režimech viz dokumentaci ([17, v menu: Editors > 3D Viewport > Object Modes]).

Pomocí klávesových zkratek a myši se lze pohybovat 3D scénou. Podržením prostředního tlačítka myši a tažením lze kroužit pohled kolem bodu zájmu a posunu pohledu lze dosáhnout stejnou akcí při držení klávesy Shift. Přiblížení lze ovládat kolečkem myši či držením klávesy Ctrl a tažením prostředního tlačítka myši. Stisknutí klávesy Home zobrazí v pohledu všechny objekty a pomocí klávesy NumpadPeriod lze pohled přiblížit na vybrané objekty [17, v menu: Editors > 3D Viewport > Navigating > Navigation]. Pro řádnou navigaci ve 3D scéně je vhodné znát ve kterém systému souřadnic pracuje. Jak je uvedeno v dokumentaci, souřadnicový systém v Blenderu je pravotočivý a osa Z míří vzhůru [17, v menu: Editors > 3D Viewport > Navigating > Aligning].

Stiskem klávesy Numpad0 se lze přepnout do pohledu aktivní kamery. Klávesovou zkratkou Ctrl+Alt+Numpad0 lze aktivní kameru zarovnat dle běžného pohledu. Aktivní kameru lze přenastavit výběrem požadované kamery a klávesové zkratky Ctrl+Numpad0. [17, v menu: Editors > 3D Viewport > Navigating > Camera View]

V rámci scény lze využívat bod ve 3D prostoru zvaný 3D cursor. Pro každý nově vytvořený objekt je použit jako jeho středový bod (origin point) a lze jej také využít například jako vztažný bod. Pokud je aktivní nástroj Cursor, lze jej umístit na požadovanou pozici pomocí levého tlačítka myši. [17, v menu: Editors > 3D Viewport > 3D Cursor]

Všechny objekty mají středový bod, který určuje jejich pozici ve 3D prostoru. Při provádění operací posunu, rotace či škálování je jeho pozice důležitá. Jeho pozici lze měnit nejen relativně ke geometrii objektu či 3D kurzoru, ale též přímo povolením možnosti Origins ve vyskakovacím okně. [17, v menu: Scenes & Objects > Objects > Object Origin]

Pro stínování v editoru 3D Viewport lze volit z vícero režimů. V režimu Wireframe jsou vykreslovány pouze hrany objektů a lze mu nastavit kupříkladu jakou barvou mají být objekty vykreslovány či jak má vypadat pozadí. Při použití režimu Solid jsou objekty vykreslovány jako pevné objekty pomocí enginu Workbench. Režim Material Preview je vhodný například pro náhled materiálů a scéna je v něm vykreslována pomocí enginu Eevee. V režimu Rendered je scéna vykreslována vykreslovacím enginem scény. [17, v menu: Editors > 3D Viewport > Display > Viewport Shading]

V rámci tohoto editoru lze také spravovat překrytí (Viewport Overlays), ve kterých lze nastavit například zobrazení mřížky, zobrazení 3D kurzoru či zobrazení orientace plošek. Při zapnutí poslední zmíněné možnosti jsou všechny plochy směřující ke kameře zbarveny modře, zatímco plochy opačného směru jsou zbarveny červeně. Hodí se tak pro určení orientace normál daných ploch [17, v menu: Editors > 3D Viewport > Display > Viewport Overlays].

Editor **Outliner** slouží k organizaci dat v souboru. Je to seznam, ve kterém lze například zobrazovat a vybírat data ve scéně, měnit viditelnost objektů či spravovat kolekce. Pomocí levého tlačítka myši či kontextové nabídky lze vybírat jednotlivé položky, které jsou po této akci zvýrazněny modře, přičemž aktivní položka je označena světlejší modrou barvou. Výběr lze rozšířit pomocí kombinace klávesy Ctrl a levého tlačítka myši. Ve výchozím nastavení jsou tyto výběry synchronizovány s editorem 3D Viewport. Scény jsou organizovány pomocí kolekcí, které mohou obsahovat vše ve scéně včetně dalších kolekcí. V rámci tohoto editoru je lze kupříkladu vytvářet, duplikovat, mazat či upravovat jejich viditelnost. [17, v menu: Editors > Outliner]

V pravé straně editoru Outliner se nachází přepínače umožňující určitým způsobem omezit jednotlivé položky. Většinu těchto možností je potřeba povolit v Restriction Toggles pod Outliner filter a ve výchozím nastavení je povolen pouze přepínač s ikonkou oka pro dočasnou změnu viditelnosti v editoru 3D viewport. Jedna z možností, kterou lze povolit je Global Viewport Visibility označená ikonkou obrazovky. Použitím tohoto přepínače budou objekty či kolekce vykreslovány, ale všechny editory 3D Viewport je budou ignorovat. [17, v menu: Editors > Outliner]

Pomocí editoru **Properties** lze upravovat mnoho dat scény a aktivního objektu. Lze zde spravovat například nastavení vykreslování, nastavení scény, vlastnosti objektu, modifikátory objektu, materiály objektu, data polygonových sítí objektu či data křivek v závislosti na typu objektu [17, v menu: Editors > Properties Editor]. Pro kompletní seznam všech upravitelných dat viz dokumentaci ([17, v menu: Editors > Properties Editor]).

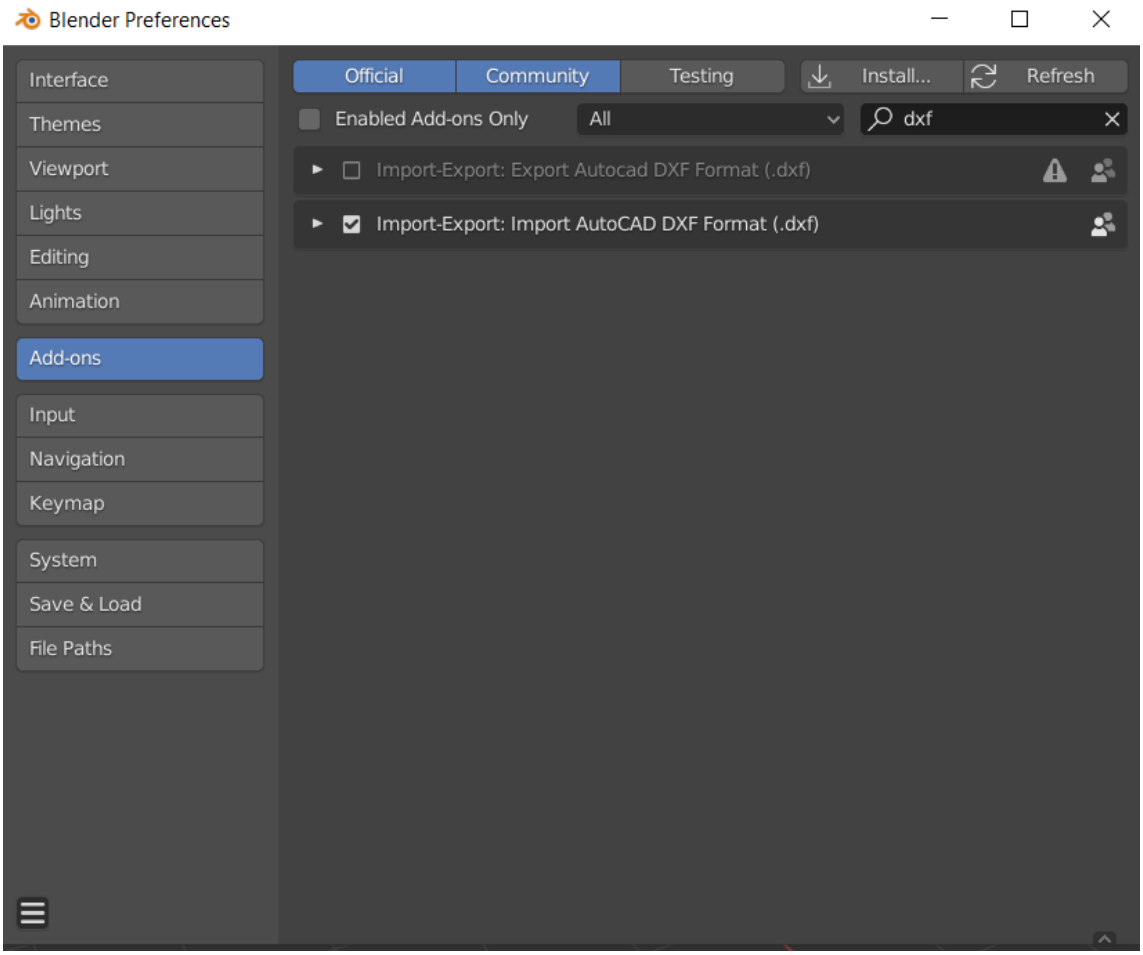
Veškeré objekty ve scéně jsou součástí tzv. kolekce scény a zároveň mohou být pro lepší organizaci součástí kolekcí vytvořených uživatelem, které lze podobně jako složky umístit do hierarchie pod jiné uživatelsky vytvořené kolekce [17, v menu: Scenes & Objects > Collections > Introduction]. Seznam kolekcí, ke kterým byl objekt přiřazen lze najít pod záložkou Object v panelu Collections. Lze zde také například kolekce přejmenovat či změnit hodnotu odsazení od středového bodu původního objektu pro instance [17, v menu: Scenes & Objects > Collections > Collections].

Prostřednictvím **UV** editoru lze provádět UV mapování 2D textur na 3D objekty. Během tohoto procesu jsou trojrozměrné polygonové sítě rozbaleny do dvourozměrného obrázku. Označení UV vychází z pojmenování dvou směrů v tomto prostoru, kde U značí směr X a V značí směr Y. [17, v menu: Editors > UV Editor > Introduction]

V **Shader** editoru probíhá pomocí stromu uzlů úprava materiálů používaných vykreslovacími enginy Cycles a Eevee. V postranním panelu se nachází panel Options obsahující stejná nastavení jako záložka Material v editoru Properties. [17, v menu: Editors > Shader Editor]

Pomocí **Text** editoru lze psát a upravovat text. Lze jej využít kupříkladu pro sepsání vysvětlení obsahu souboru či tvorbu a provádění skriptů [17, v menu: Editors > Text Editor]. Editor **Python Console** lze využít pro rychlé provádění menších částí kódu například pro jejich otestování před použitím ve větším skriptu [17, v menu: Editors > Python Console].

V editoru **Preferences** lze vyvolat klávesovou zkratkou F4 + P a spravovat v něm nastavení chování Blenderu, která jsou seskupena do sekcí v levé straně tohoto editoru. Lze zde spravovat například uživatelské prostředí (Interface) či rozšíření (Add-ons) [17, v menu: Editors > Preferences > Introduction]. Pomocí rozšíření lze rozšiřovat funkce Blenderu a lze je vyhledávat a spravovat v sekci Add-ons. Lze využít některé z předinstalovaných rozšíření či přidávat nová z internetu. Rozšíření jsou rozdělena na oficiální (official) od vývojářů Blenderu a komunitní (community), která vytvořila komunita Blenderu [17, v menu: Editors > Preferences > Add-ons]. Jedním z předinstalovaných rozšíření je kupříkladu AutoCAD DXF, který používá DXF knihovnu dxfgrabber [17, v menu: Add-ons > Add-ons Category Listings > Import-Export > AutoCAD DXF]. Toto rozšíření může být v této práci využito pro importování plánů budovy.



Obrázek 3: Editor Preferences s aktivovaným rozšířením. Zdroj: Autor

3.2 Nástroje pro práci s modely

V režimu Object Mode lze na všech objektech provádět základní transformace pro změnu pozice, rotaci či škálování [17, v menu: Scenes & Objects > Objects > Editing > Transform > Introduction]. Stiskem klávesy G lze volně měnit pozici zvoleného prvku či objektu pohybem myši, kterou lze následně potvrdit stiskem levého tlačítka myši. Otáčet zvoleným prvkem lze po stisku klávesy R dle vybraného vztažného bodu. Po stisku klávesy S lze pohybem myši měnit proporce objektu. Tyto transformace je namísto potvrzení možné zrušit stiskem klávesy Esc či pravého tlačítka myši [17, v menu: Scenes & Objects > Objects > Editing > Transform > Basic Transformations]. Hodnoty je během provádění transformací možné zadat také číselnými hodnotami a je možné je také vyjádřit jednotkami jako například centimetry [17, v menu: Scenes & Objects > Objects > Editing > Transform > Transform Control > Numeric Input]. Transformace provedené v editoru 3D Viewport je možné po jejich potvrzení upravovat v panelu Adjust Last Operation daného nástroje a lze tak kupříkladu měnit hodnoty či omezit akce

na vybrané osy [17, v menu: Scenes & Objects > Objects > Editing > Transform > Basic Transformations]. Zkratkou Ctrl+A lze transformace objektu aplikovat, čímž dojde k jejich resetování bez vizuální změny objektu [17, v menu: Scenes & Objects > Objects > Editing > Apply].

Prováděné transformace je možné bez ztráty možnosti přímého zadávání číselných hodnot omezit na konkrétní osu například jednou z kláves X, Y, Z či podržením prostředního tlačítka myši a jeho následným puštěním na požadované ose. Podržením klávesy Shift v kombinaci s výše uvedenými tlačítky je možné omezit transformace změny pozice a škálování na zvolené ose. Prvním stisknutím jedné z uvedených kláves dochází k omezení dle globální osy. Po druhém stisku stejné klávesy jsou transformace omezeny dle současného výběru orientace transformací, či lokální orientace pokud je vybrána možnost Global. Pro odstranění těchto omezení stačí stisknout stejnou klávesu potřetí. [17, v menu: Scenes & Objects > Editing > Transform > Transform Control > Axis Locking]

Objekt lze učinit rodičem jiných objektů. K tomu je potřeba všechny tyto objekty vybrat, následně vybrat objekt, který se má stát rodičem a zvolit možnost Object v nabídce vyvolané zkratkou Ctrl+P. Transformace prováděné na rodiči jsou obvykle aplikovány na potomky, zatímco transformace prováděné na potomcích neovlivňují rodiče. [17, v menu: Scenes & Objects > Objects > Editing > Parenting Objects]

K provádění úprav v geometrii je potřeba se přepnout do režimu Edit Mode, což lze provést například klávesou Tab. Stisknutí této klávesy v režimu Edit Mode přepne zpět do režimu Object Mode [17, v menu: Scenes & Objects > Objects > Editing > Transform > Introduction]. K úpravám existuje v režimu Edit Mode mnoho nástrojů, ze kterých jsou níže krátce popsány pouze ty nejzákladnější.

V režimu Edit mode lze přepínat mezi režimy výběru pro body, hrany a plošky a téměř všechny nástroje jsou ve všech režimech dostupné. Lze pracovat ve vícero režimech výběru podržením klávesy Shift při výběru daných režimů [17, v menu Modeling > Meshes > Selecting]. Pro podrobnější popis možností výběru viz dokumentaci ([17, v menu Modeling > Meshes > Selecting]).

Pomocí nástroje **Bevel** lze zaoblit zvolené hrany. Klávesovou zkratkou Ctrl+B se spustí v režimu pro hrany a tažením myši lze měnit hodnotu odsazení, zatímco kolečkem myši lze určit množství segmentů. Kombinace Shift+Ctrl+B spustí tento nástroj v režimu pro body. [17, v menu: Modeling > Meshes > Editing > Edge Tools > Bevel Edges]

Smyčku plošek lze rozdělit vložením nové smyčky hran nástrojem **Loop Cut and Slide** pomocí klávesové zkratky Ctrl+R. Použití tohoto nástroje se skládá ze dvou kroků. V prvním kroku bude po přesunu kurzoru na požadovanou hranu vytvářený řez vizualizován čárou barvy magenta. Ve druhém kroku je možné po stisku levého tlačítka myši blíže určit pozici vytvářené hrany podél existující hrany. Další stisknutí levého tlačítka hranu umístí do specifikované pozice, zatímco pravé tlačítko umístí tuto hranu tak, aby řez rozdělil plošky na přesné poloviny. Druhý krok je přeskočen v případě vytváření vícero řezů, jejichž počet lze určit kolečkem myši. [17, v menu: Modeling > Meshes > Tools > Types > Loop Subdivide]

Nástroj **Extrude Region** je vyvolán klávesou E a lze s ním vytahovat body, hrany či plošky, čímž je velmi významný při tvorbě nové geometrie. Během tohoto procesu jsou body duplikovány při zachování spojení s původními body. Hranu lze tímto nástrojem vytvořit jeho použitím na bod, zatímco při použití na hranu vznikne nová ploška. [17, v menu: Modeling > Meshes > Tools > Types > Extrude Region]

Propojit vícero smyček hran ploškami lze nástrojem **Bridge Edge Loops** přístupným v menu Edge > Bridge Edge Loops. Propojované hrany se nemusí skládat ze stejného množství bodů a je možné propojit vícero vybraných smyček jedním krokem. Výběrem plošek je možné při propojení zároveň vytvářet díry. Lze také nastavit například množství smyček, které mají vzniknout mezi dvěma smyčkami hran. [17, v menu: Modeling > Meshes > Editing > Edge Tools > Bridge Edge Loops]

Klávesovou zkratkou M lze vyvolat nástroj **Merge**, kterým lze vybrané body sloučit do jednoho. Pozici sloučeného bodu lze umístit kupříkladu na místo prvního vybraného bodu nebo posledního vybraného bodu, středu výběru či na pozici 3D kurzoru [17, v menu: Modeling > Meshes > Editing > Mesh Tools > Merge]. Lze konstatovat, že tento nástroj je značně užitečný při čištění geometrie.

Rozložit polygonovou síť do 2D prostoru lze nástrojem **Unwrap**, který lze vyvolat klávesou U. Rozložení probíhá podle švů (seams) a pokud je to možné, jednotlivé plošky jsou rozmístěny tak, aby se vzájemně nepřekrývaly. Výsledek je následně možné zobrazit v editoru UV Editor. [17, v menu: Modeling > Meshes > Editing > UV Tools]

Pomocí nástrojů ze skupiny **Normals** lze spravovat normály. Nástroj Flip Direction otočí směr normál všech vybraných ploch a lze vyvolat pomocí Mesh > Normals > Flip. Přepočítat směr vybraných normál ven či dovnitř lze nástrojem Recalculate Normals pomocí klávesové zkratky Ctrl + N či Shift + Ctrl + N [17, v menu: Modeling > Meshes > Editing > Mesh Tools > Normals]. Plochám lze také nastavit hladké či ploché stínování pod menu Mesh > Shading [17, v menu: Modeling > Meshes > Editing > Mesh Tools > Shading].

3.3 Modifikátory

Modifikátory lze nedestruktivně ovlivňovat geometrii objektu. S pomocí těchto automatických operací lze bez ovlivnění základní geometrie dosáhnout mnoha efektů bez nutnosti zdlouhavého manuálního postupu. Geometrii lze při jejich použití stále modifikovat přímo a přidáním vícero modifikátorů na jeden objekt vznikne zásobník modifikátorů. Aplikováním modifikátoru se změny jím prováděné stanou trvalými [17, v menu: Modeling > Modifiers > Introduction]. Jak je patrné z obrázku níže, existuje mnoho modifikátorů, a proto budou popsány pouze ty, které jsou pro tuto práci relevantní.

Obrázek 4: Dostupné modifikátory. Zdroj: Autor

Modifikátorem **Array** lze vytvářet pole kopií objektu s mnoha možnostmi, jak definovat vzdálenost mezi jednotlivými kopiemi. Hodí se kupříkladu pro tvorbu komplexních opakujících se tvarů [17, v menu: Modeling > Modifiers > Generate > Array Modifier]. V této práci může nalézt využití například při tvorbě zábradlí.

Modifikátor **Boolean** vyžaduje specifikaci cíle operace v podobě druhého objektu. Vybraná operace ze dvou objektů polygonových sítí vytvoří jeden a hodí se tak pro operace, které by bylo příliš složité provést manuální cestou. Možné operace jsou Difference pro zachování všeho mimo cílovou polygonovou síť, Union pro přidání cílové polygonové sítě k té původní a Intersect pro zachování všeho uvnitř cílové polygonové sítě. [17, v menu: Modeling > Modifiers > Generate > Boolean Modifier]

Pomocí modifikátoru **Mirror** lze objekt přes jeho středový bod zrcadlit po jeho lokálních osách. Jako střed zrcadlení lze také využít kupříkladu prázdný objekt a vyhnout se tak nutnosti modifikovat pozici středového bodu daného objektu. [17, v menu: Modeling > Modifiers > Generate > Mirror Modifier]

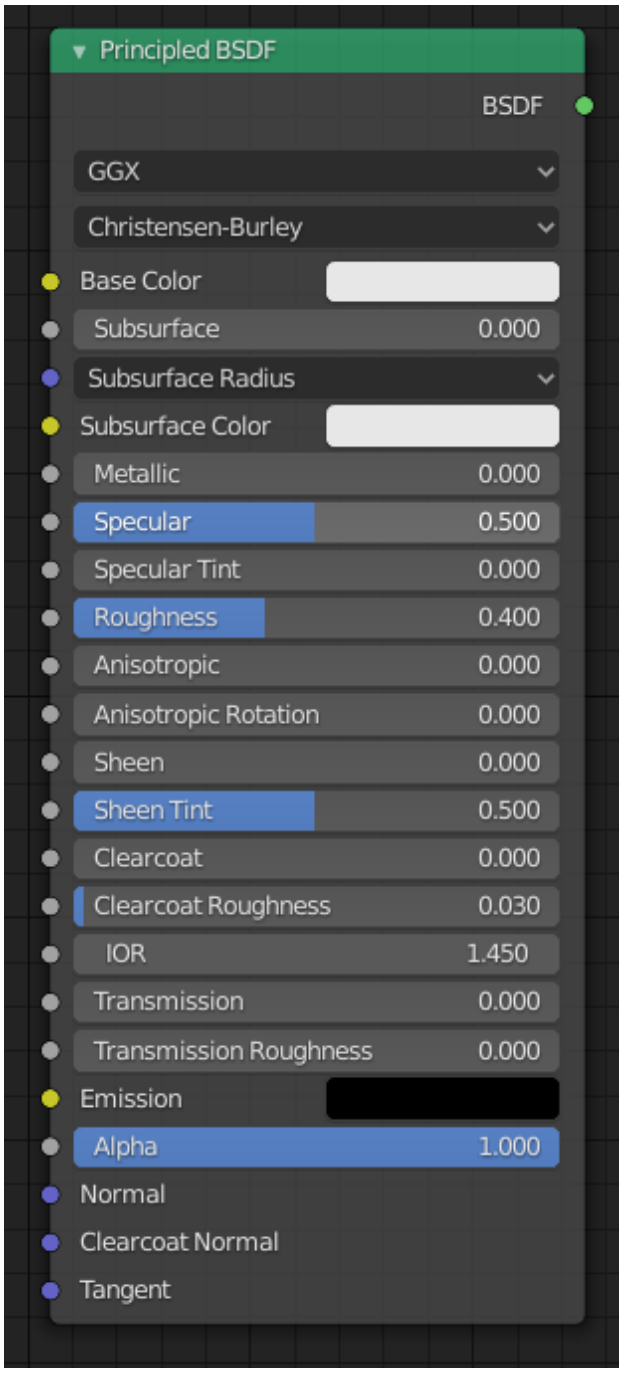
Pro případy, kdy je potřeba povrchu přidat hloubku lze využít modifikátor **Solidify**. Lze jej provozovat v režimu Simple Mode, ve kterém je pouze vytahována geometrie a v režimu Complex Mode, který se hodí na složitější geometrii. První zmíněný režim je výchozí a kvůli vyšší rychlosti také doporučený pro běžné případy. [17, v menu: Modeling > Modifiers > Generate > Solidify Modifier]

Modifikátorem **Mesh Deform** lze použít libovolnou polygonovou síť uzavřeného tvaru pro deformaci jiné polygonové sítě. Aby měl tento modifikátor efekt, je nutné spolu svázat pozice bodů deformujícího objektu a upravovaného objektu zvolením možnosti Bind [17, v menu: Modeling > Modifiers > Deform > Mesh Deform Modifier]. Využití v této práci může spočívat například v úpravě tvaru zábradlí pro schody.

3.4 Způsob tvorby materiálů

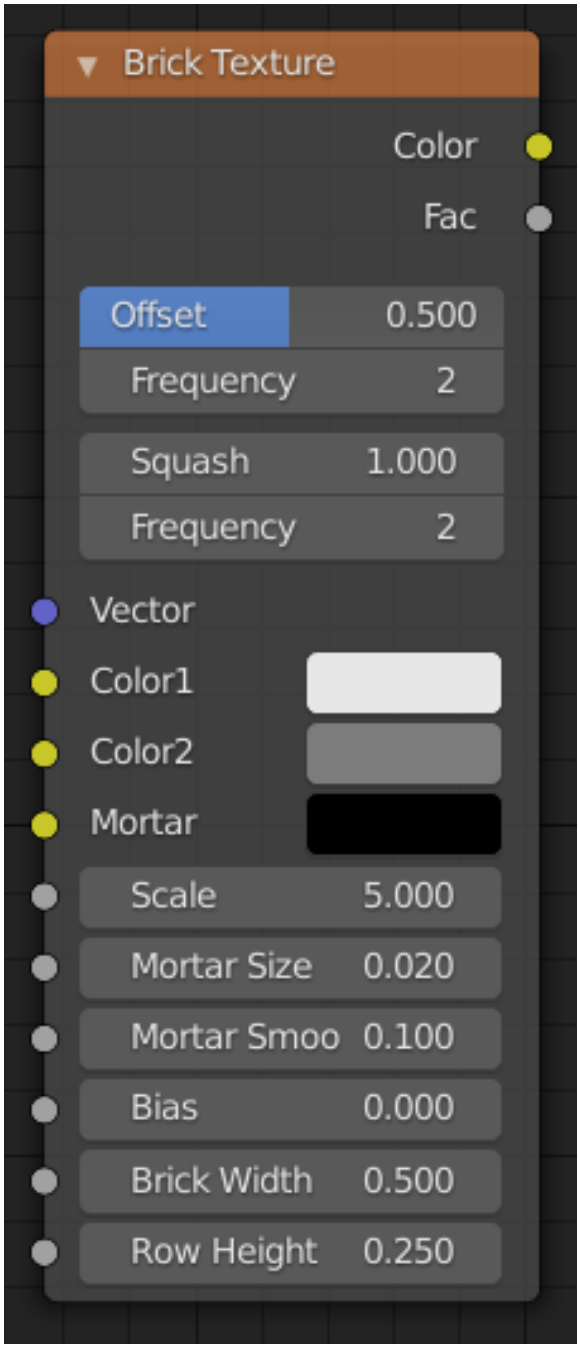
Materiály jsou tvořeny sítěmi uzlů s různými výstupními daty typu hodnota, vektor, barva či shader. Kupříkladu uzel typu shader na vstupu přijímá barvu a jeho výstupem je shader. Tímto barevným vstupem může být například výstupní hodnota uzlů typu texture, kterými lze vytvářet mnoho typů procedurálních textur [17, v menu: Rendering > Shader Nodes > Introduction]. V této sekci bude popsáno pouze pár uzlů relevantních pro tuto práci.

Uzel **Principled BSDF** je kompatibilní se softwary jako Renderman od Pixaru či Unreal Engine díky tomu, že je založen na modelu Disney principled známém také jako „PBR“ shader. Umožňuje také do příslušných parametrů přímo připojit textury vytvořené v jiných softwarech. Parametrem Base Color lze definovat barvu povrchu, parametr Metallic určuje, jak moc bude materiál kovový. Pro nastavení drsnosti povrchu slouží parametr Roughness a jeho průhlednost lze stanovit parametrem Transmission. [17, v menu: Rendering > Shader Nodes > Shader > Principled BSDF]



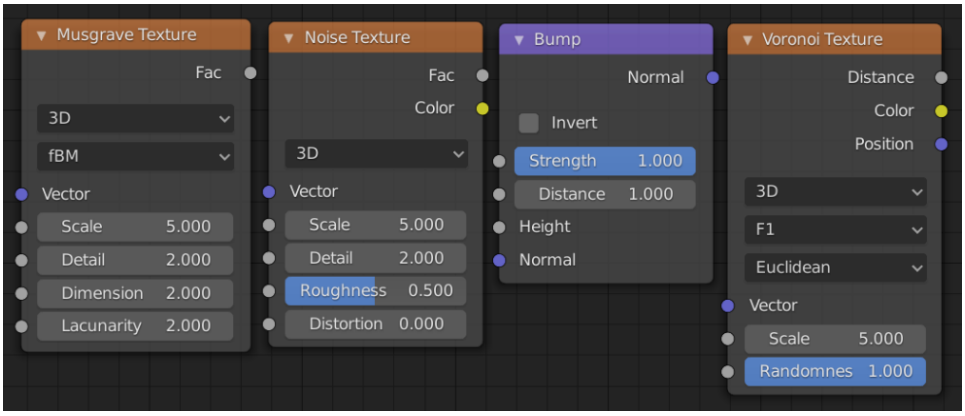
Obrázek 5: Uzel Principled BSDF. Zdroj: Autor

Pro tvorbu procedurálních textur cihel slouží uzel **Brick Texture**. Lze nastavovat kupříkladu celkovou velikost textury, šířku cihel či výšku řádku cihel [17, v menu: Rendering > Shader Nodes > Texture > Brick Texture Node]. S těmito možnostmi nastavení může být tento typ uzlu potenciálně využit nejen k vytvoření cihel, ale také pro tvorbu textur kachlí či dlaždic.



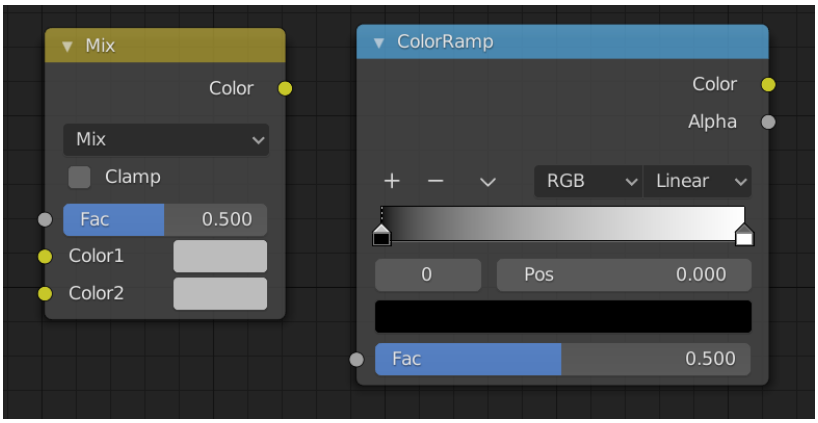
Obrázek 6: Uzel Brick Texture. Zdroj: Autor

Pro vytvoření textury fraktálního Perlinova šumu lze využít kupříkladu uzly **Musgrave Texture** či **Noise Texture** [17, v menu: Rendering > Shader Nodes > Texture > Musgrave Texture Node]. Pomocí uzlu **Bump Node** lze normály ovlivňovat výškovou texturou [17, v menu: Rendering > Shader Nodes > Vector > Bump Node]. Tyto textury mohou být v této práci využity například pro ztvárnění malých nerovností na daném povrchu či k zachycení určitých vizuálních vzorů. Uzlem **Voronoi** lze generovat Worleyho šum [17, v menu: Rendering > Shader Nodes > Texture > Voronoi Texture Node]. Tento uzel může v tomto projektu posloužit kupříkladu pro ztvárnění zbarvených skel balkónu.



Obrázek 7: Uzly Musgrave Texture, Noise Texture, Bump a Voronoi. Zdroj: Autor

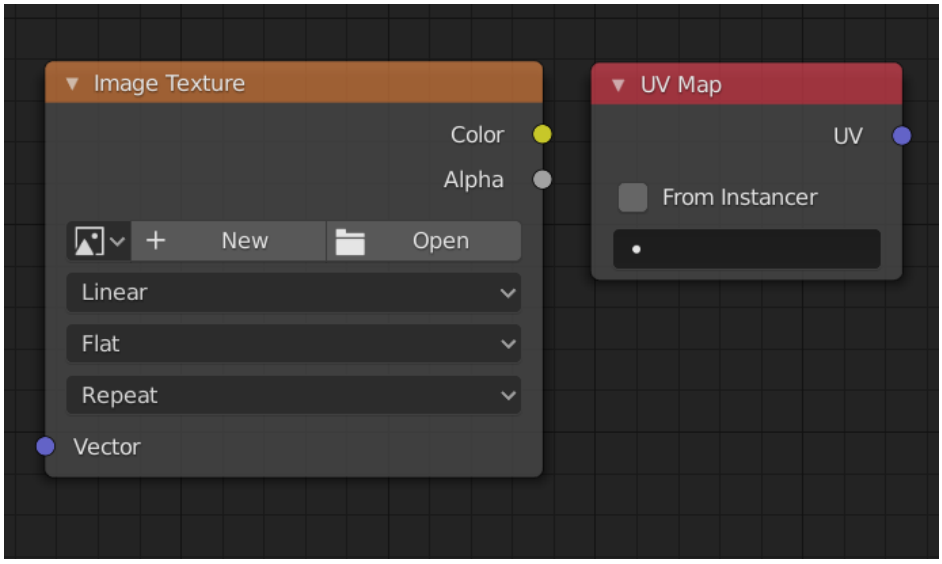
Obrázky lze míchat pomocí uzlu **Mix** v mnoha různých režimech [17, v menu: Rendering > Shader Nodes > Color > Mix Node]. Lze konstatovat, že tento typ uzlu může být značně užitečný při kombinaci vícero procedurálních textur do výsledné podoby. Uzlem **Color Ramp** je možné pomocí barevného přechodu mapovat hodnoty k barvám [17, v menu: Rendering > Shader Nodes > Converter > Color Ramp Node]. Je zřejmé, že tento uzel se hodí pro lepší kontrolu nad hodnotami barev dané textury.



Obrázek 8: Uzly Mix a uzel Color Ramp. Zdroj: Autor

Takto vytvářené textury však není možné přímo použít v jiných aplikacích. Jak je uvedeno v dokumentaci, shadery vykreslovacího enginu Cycles je možné zapéct do obrázkových textur například za účelem exportu do herních enginů. Data budou zapečena do aktivního uzlu Image a k provedení je potřeba, aby polygonová síť měla UV mapu. Lze vybrat jaká vykreslovaná data mají být do textury zapečena [17, v menu: Rendering > Cycles > Render Baking].

Obrázek lze vložit jako texturu pomocí uzlu **Image Texture**, kterému lze kupříkladu také určit jaké souřadnice má pro texturu použít [17, v menu: Rendering > Shader Nodes > Texture > Image Texture Node]. Uzlem **UV Map** lze vybrat konkrétní UV mapu objektu pro další použití [17, v menu: Rendering > Shader Nodes > Input > UV Map Node].



Obrázek 9: Uzly Image Texture a UV Map. Zdroj: Autor

3.5 Možnosti skriptování v Blenderu

Díky vestavěnému Python interpretu je v Blenderu k dispozici typické prostředí Pythonu. Tento interpret je načten při spuštění, běží až do ukončení Blenderu a kromě vykreslování uživatelského prostředí je také využíván některými vnitřními nástroji Blenderu. Python moduly jako bpy či mathutils jsou v interpetu přístupné a lze je importovat do skriptů pro další použití [18, s. Python API Overview]. Verze používaná v Blenderu je Python 3.x a kromě práce s daty editovatelných uživatelským prostředím lze s touto API také například vytvářet nové prvky uživatelského rozhraní [18, s. Quickstart Introduction].

Sripty se nejčastěji provádí přes editor Python Console či Text. Do editoru Text lze vkládat kód kupříkladu ze schránky či načtením .py souborů a následně je provádět příkazem Run Script.

Pomocí modulu **bpy.data** lze přistoupit k datům z načteného blend souboru. Ke členům kolekcí je možné přistoupit jak indexem, tak pomocí řetězce znaků a k atributům datových bloků lze přistoupit podobně jako při změně nastavení v grafickém prostředí [18, s. Quickstart Introduction]. Lze konstatovat, že tyto editory mají při psaní skriptů zásadní význam.

K datům kolekcí objektů lze přistoupit pomocí třídy **bpy.types.Collection**. Ke kolekcím, které jsou přímými potomky dané kolekce lze přistoupit atributem children a k přímým potomkům typu objekt atributem objects. Pro přístup ke všem objektům kolekce včetně těch v kolekcích potomků slouží atribut all_objects. Jméno kolekce je přístupné atributem name. [18, s. Collection(ID)]

Třídou **bpy.types.Object** jsou definovány objekty ve scéně. Jakou instanční metodu objekt používá lze zjistit atributem instance_type, který může nabývat například hodnot NONE pokud není použito instancování či COLLECTION pokud tento objekt instancuje kolekci. Přistoupit k instancované kolekci lze atributem instance_collection. Zrušit výběr objektu, či jej vybrat lze pomocí select_set a příslušnou hodnotou boolean v parametru. [18, s. Object(ID)]

Provést export scény do formátu FBX lze operátorem **bpy.ops.export_scene.fbx** a specifikací příslušných parametrů. Lze například určit cestu souboru, zvolit pro export pouze vybrané objekty, aplikovat velikost nastavených jednotek, určit jakého typu má být cesta souboru a stanovit jaká osa má mířit vpřed či vzhůru [18, s. Export Scene Operators]. Poslední jmenované možnosti najdou jistě hojné využití v případech, že je export prováděn za účelem využití v aplikaci s jiným souřadnicovým systémem. Jak je uvedeno v dokumentaci API, pro práci s cestami lze využít pomocných funkcí. Například funkce **bpy.path.abspath** vrátí pomocí prefixu // absolutní cestu relativně k současnému blend souboru [18, s. Path Utilities (bpy.path)]. Tato funkce může najít využití například při vytváření adresářové struktury kvůli lepší organizaci.

4 ZPŮSOB A MOŽNOSTI VÝVOJE V PROSTŘEDÍ UNITY

První verze vývojového prostředí Unity vyšla v roce 2005 pouze na operační systém macOS. Dnes jej lze provozovat také na Windows a výsledný projekt exportovat na mnoho různých platforem včetně webu, kde využívá WebGL. [19, s. 5]

Vývoj v Unity probíhá v rámci projektů. Pro jejich lepší pochopení je jistě vhodné přesněji definovat co se projektem rozumí. Jak ve své knize uvádí Hocking, „*Unity projekt je jednoduše řečeno složka plná různých assetů a souborů nastavení ...*“ [19, s. 18] (překlad autora)

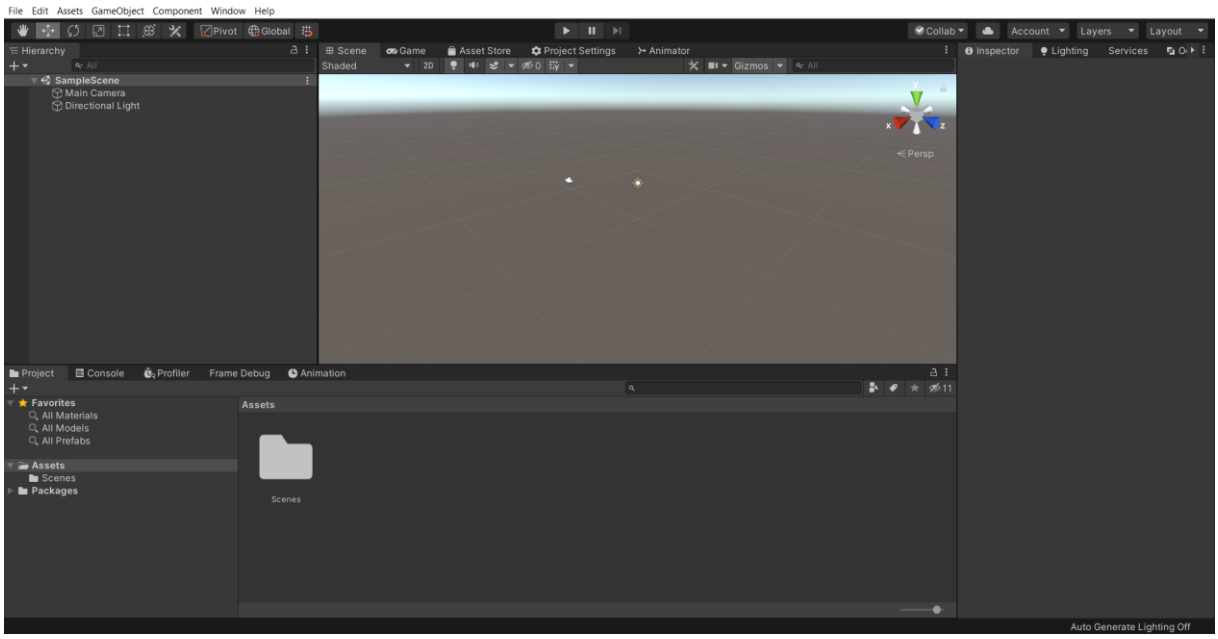
Herní objekty se v Unity vytváří na základě systému modulárních komponent, kde jsou objekty skládány z komponent pomocí kompozice. Objekty tak existují v ploché hierarchii, kde se každý objekt skládá z vlastní kolekce komponent, kterou lze flexibilně upravovat bez nutnosti refaktorování řetězce dědičnosti. S komponentami lze v Unity manipulovat nejen skrze kód, ale také prostřednictvím vizuálního editoru. V kódu není nutné používat pouze kompozici a lze také využívat dědičnosti a návrhových vzorů na ní založených. [19, s. 6]

GameObject se všemi komponentami a hodnotami vlastností lze uložit jako znovupoužitelný asset pomocí systému prefabů, které fungují jako šablony a jejich vložením do scény lze vytvořit jejich instance. Použití tohoto systému je oproti ručnímu kopírování lepší díky tomu, že změny provedené na původním prefabu jsou automaticky propsány do všech jeho instancí. Prefaby je možné do sebe vnořovat a vytvářet tak pokročilé hierarchie, které je ale snadné upravovat. [12, s. Prefabs]

V instancích prefabů je možné překrývat nastavení původního prefabu dle potřeby. Skupinu těchto překrytí je možné dle potřeby seskupit do varianty prefabu [12, s. Prefabs]. Tento systém tudíž zcela jistě nalezne v této práci široké uplatnění.

4.1 Základní používání editoru

Uživatelské rozhraní lze v Unity rozdělit do několika sekcí. V záložce **Scene** lze do otevřené scény umisťovat 3D objekty a pod záložkou **Hierarchy** lze tažením vytvářet vztahy mezi objekty. Informace o vybraných objektech včetně připojeného kódu lze zobrazit v záložce **Inspector**. V záložce **Console** lze kontrolovat chybový výstup při testování hry v záložce **Game**. Nástroje pro práci se scénou se nachází v **Toolbar**. [19, s. 13]



Obrázek 10: Výchozí podoba editoru v Unity 2019.4. Zdroj: Autor

Ve scéně se uživatel pohybuje především pomocí myši a několika kláves modifikujících její chování. Mezi tyto akce patří mimo jiné Zoom, Move a Orbit, které lze provést tažením myši při držení jejího tlačítka společně s klávesou Alt či Ctrl. [19, s. 14]

Pro řádnou orientaci ve scéně je jistě potřeba porozumět práci ve 3D prostoru. Jak uvádí Hocking, pozici a pohyb ve 3D počítačových simulacích lze shrnout jako čísla předepisující body v prostoru. Tato čísla prostřednictvím souřadnicových os korelují s prostorem [19, s. 26].

Souřadnice na osách X, Y a Z bude mít vše, co se ve scéně nachází na konkrétní pozici. V záložce **Inspector** lze pro umístění objektu tyto tři hodnoty zadat. Objekty lze umístit kódem pomocí stejných souřadnic. Souřadnice fungují nezávisle na směru os, jelikož pozitivní a záporný směr každé osy je libovolný. Pozice může být uváděna v libovolných jednotkách. Nejčastěji se však používají metry. [19, s. 27–29]

Ve většině případů míří osa Y vzhůru a osa X vpravo. Zda osa Z míří do stránky, nebo ze stránky se však mezi nástroji liší. První zmíněný směr je označován jako levotočivý, zatímco druhý jako pravotočivý. V Unity je použit levotočivý souřadnicový systém. [19, s. 28]

Všechny objekty mají svůj středový bod a vlastní směr pro osy X, Y a Z. Tato soustava souřadnic, která se pohybuje s příslušným objektem, je označována jako lokální souřadnice. Za globální souřadnice je označován nehybný souřadnicový systém skládající se ze vztažného bodu 3D scény a jejího směru pro výše zmíněné osy. [19, s. 36–37]

Objekt ve scéně lze po jeho vybrání otáčet, měnit jeho pozici či měnit jeho velikost. Mezi těmito akcemi lze přepínat kromě tlačítek v nástrojové liště také pomocí kláves W, E a R. Při zvolení dané akce se u příslušného objektu objeví tzv. Transform gizmo, které lze popsat jako barevné šipky či kruhy, kterými lze pomocí tažení příslušné transformace aplikovat. Výše zmíněné typy transformací lze ve 2D prostoru aplikovat také nástrojem **Rect tool**. V tomto nástroji jsou tyto akce z důvodu absence třetího rozměru zkombinovány. [19, s. 15]

V záložce **Hierarchy** jsou všechny objekty dané scény zobrazeny ve jmenném seznamu. Zároveň je zde pomocí vnořování těchto jmen vizuálně zachycena hierarchie těchto objektů. Tato reprezentace je vizuálně podobná reprezentaci složek a lze díky ní manipulovat s celou skupinou. Tuto záložku tak lze využít pro výběr objektů na základě jména bez nutnosti je hledat přímo ve scéně. [19, s. 15]

V záložce **Inspector** jsou uvedeny informace o vybraném objektu. Tyto informace jsou převážně tvořeny seznamem komponent, které zde lze také přidávat či odebírat. Každý objekt má však vždy minimálně jednu komponentu, kterou je konkrétně Transform. Často zde lze nalézt více komponent, mezi které se řadí také připojené skripty. [19, s. 16]

Pro zobrazení všech assetů v projektu slouží záložka **Project**. Obsah její levé části slouží k zobrazení seznamu složek podobně jako je zobrazen obsah scény v záložce Hierarchy. Pravá část umožňuje zobrazit soubory obsažené ve vybrané složce. [19, s. 16]

V záložce **Console** jsou zobrazovány zprávy z kódu, jako jsou například chybové zprávy či zamýšlený výstup debugu [19, s. 17]. Lze tedy konstatovat, že tato část editoru bude při psaní potřebné logiky jednou z nejdůležitějších.

Unity editor umožňuje pohybovat s objekty ve scéně i za běhu hry a hodí se pro rychlé iterace. Do rozhraní Unity editoru lze pomocí skriptů přidávat nové menu a funkce [19, s. 5]. Tyto funkce tudíž mohou být jistě velmi užitečné při snahách o dosažení vyšší efektivity práce.

4.2 Možnosti rozšiřování funkcí editoru pomocí skriptů

Úpravou a rozšířením Unity editoru pomocí skriptů lze usnadnit jeho použití v projektu. Rozšířením editoru o vlastní menu lze přidat funkce a poté k nim přistupovat přímo z uživatelského prostředí editoru [20]. Možností rozšíření je mnoho, a proto budou popsány pouze možnosti relevantní dále v této práci.

Skripty umístěné ve složce Editor nejsou přístupné v sestavených projektech a slouží pro rozšíření editoru. Pokud se skript nachází v této složce, nelze příslušné komponenty dědící z MonoBehaviour přiřadit ke GameObject. Lze vytvořit vícero složek Editor a umístit je kdekoliv ve složce Assets. [12, s. Special folder names]

Přidat novou položku menu do vrchní lišty nástrojů lze pomocí editorového skriptu, ve kterém jsou tyto tvořeny statickými metodami s atributem MenuItem. Často se tak přidává kupříkladu menu Tools, pod které se nové funkce vloží. Je však také možné vytvářet vnořená menu či přidávat nové možnosti pod již existující menu. [20]

V některých případech může být pro dostatečně efektivní práci nutné pod danými možnostmi zobrazit okno blíže specifikující jednotlivé akce. Jak je uvedeno v dokumentaci, k tomuto účelu lze využít třídu EditorWindow. Vytvoření nového okna spočívá ve zdědění z této třídy, přičemž poté s ním lze pracovat jako s původními okny v Unity [21, s. Editor Window].

Snadno rozšiřitelný způsob dalšího rozdělení funkcí uvnitř okna může být kupříkladu seznam položek a tlačítka, která po kliknutí provedou určitou akci na základě výběru z tohoto seznamu. Jak vyplývá z dokumentace, pro výběr možností lze využít metodu EditorGUILayout.Popup [21, s. EditorGUILayout.Popup]. Pro vytvoření tlačítka lze využít metodu GUILayout.Button, která po kliknutí na dané tlačítko vrátí hodnotu true [21, s. GUILayout.Button].

Takto vytvořené prvky lze využít například k rozmístění objektů ve scéně. Dle dokumentace je možné načíst objekt na základě cesty k assetu pomocí metody AssetDatabase.LoadAssetAtPath [21, s. AssetDatabase.LoadAssetAtPath]. Následně je možné ve scéně kupříkladu vytvářet instance prefabů pomocí metody PrefabUtility.InstantiatePrefab [21, s. PrefabUtility.InstantiatePrefab].

4.3 Způsob skriptování logiky

Pro přesnější určení možností při psaní kódu je důležité znát ve které verzi jazyka se příslušná logika tvoří. Jak je uvedeno v dokumentaci, používaná verze skriptovacího modulu runtime je ekvivalentní .NET 4.6 a používaný jazyk je konkrétně C# 7.3. Kompilace tohoto jazyka je prováděna kompilátorem Roslyn. [12, s. C# compiler]

V Unity je kód prováděn ze souborů s kódem nazývaných skripty, které jsou přiřazeny k objektům ve scéně. V rámci této terminologie tak lze tyto samotné soubory přirovnat ke třídám v objektově orientovaném programování, jejichž instance vzniká přiřazením k objektu ve scéně. [19, s. 18]

Skript lze vytvořit zvolením nabídky Create > C# Script, kterou lze vyvolat pravým tlačítkem myši v záložce Project nebo k ní přistoupit přes menu Assets. Po vytvoření skriptu jej lze dvojitým poklepáním otevřít v programu pro editaci [19, s. 19]. V rámci této práce je pro editace skriptů použit program Visual Studio 2019.

Skript je komponentou pouze v případě, že dědí ze třídy MonoBehaviour, v níž je definován základ způsobu napojení komponent k herním objektům. Z této třídy jsou také zděděny metody **Start** a **Update**, které lze ve vytvářeném skriptu implementovat. Volání první metody je provedeno pouze jednou při aktivaci objektu, která zpravidla nastane při načtení úrovně, ve které se nachází. Volání druhé metody je prováděno při každém snímku neboli každém cyklu kódu ve smyčce. [19, s. 18–19]

Ne vždy je však žádoucí, aby byl daný kód volán a vykonán během jednoho snímku. Pro tyto případy lze využít korutiny. Jak je uvedeno v dokumentaci, korutiny se mohou pozastavit, vrátit kontrolu Unity a na dalším snímku pokračovat z tohoto bodu. Pro jejich spuštění je potřeba zavolat funkci StartCoroutine [12, v menu: Scripting > Scripting concepts > Coroutines].

Podobně jako jsou všechny komponenty skriptů potomky třídy MonoBehaviour, jsou všechny objekty ve scéně instancemi třídy GameObject. Tato třída je v podstatě kontejner pro komponenty a existuje především proto, aby bylo k čemu připojit MonoBehaviour [19, s. 31]. Aby bylo možné po připojení skriptu k objektu měnit jeho hodnoty, jsou veřejné proměnné odkryty v záložce Inspector. Jinými slovy jsou tyto hodnoty serializovány [19, s. 35]. Lze serializovat také soukromé hodnoty použitím atributu SerializeField [21, s. SerializeField].

Pro řešení vstupních zařízení má třída Input několik metod. Její metoda GetAxis v parametru přijímá jméno osy a vrací například čísla odpovídající pohybu myši [19, s. 39]. Jelikož jsou hodnoty Vector3 transformací pouze pro čtení, je potřeba vytvářet nový Vector3 pro změnu těchto transformací. Kvůli tomu, že vypadají interpolace mezi hodnotami rotace pomocí kvaternionů (quaternions) více přirozeně, jsou tyto používány pro reprezentaci rotace [19, s. 41]. Podobně jako v případě rotace podle myši lze metodu GetAxis třídy Input použít pro pohyb podle stisku kláves. Jména os jsou abstrakce, jejichž seznam a přiřazení k ovládacím prvkům lze nalézt v sekci Input pod menu Edit > Project Settings [19, s. 44–45].

Při psaní vlastní logiky ve skriptech je jistě také velmi důležité znát způsoby, kterými lze přistoupit k jiným skriptům či obecně jiným komponentám dané instance třídy GameObject. Jak uvádí Hocking, pomocí metody GetComponent a definování typu v úhlových závorkách lze získat referenci ke komponentě připojené k danému GameObject [19, s. 46].

4.4 Způsob práce s materiály

Materiály jsou neodmyslitelnou součástí vizuálu scény. Jak je uvedeno v dokumentaci Unity, k vykreslení objektu je potřeba dodat informace o jeho tvaru a vzhledu povrchu. Tvar lze popsat polygonovými sítěmi, zatímco povrch je popsán materiály [12, s. Materials introduction].

Materiál může kromě samotné reference na Unity shader obsahovat také data jako barvy či textury. Vlastnosti materiálu lze upravovat v okně Inspector prostřednictvím assetu typu material, ve kterém jsou uložena data pro vlastnosti materiálu. [12, s. Materials introduction]

Shader je specializovaný grafický program, jehož prostřednictvím lze simulovat například lesklé či hrbolaté povrchy pomocí efektů barev a světel. Skrze ně jsou aplikovány textury, bitmapové obrázky aplikované na povrch polygonové sítě. Rozměry textur by měly být v mocninách čísla 2 a pro jejich zobrazení v okně Project je stačí umístit do složky Assets v projektu [12, s. Textures]. Nejlepší volbou pro většinu vykreslování je Standard Shader [12, s. Meshes, Materials, Shaders and Textures]. Z tohoto důvodu bude věnována pozornost tomuto shaderu.

Unity Standard Shader poskytuje mnoho vlastností, které mohou být použity či vynechány dle využití textur a nastavení parametrů v editoru materiálů. V rámci tohoto shaderu jsou simulovány interakce materiálů a světla dle reality pomocí pokročilého modelu osvětlení zvaného Physically Based Shading (PBS). Simulace reálného chování světla je v tomto modelu dosažena následováním fyzikálních principů jako jsou například konzervace energie (objekt nemůže odrazit více světla, než přijme) či Fresnelové odrazy (při pohledu z elevačního úhlu (grazing angle) jsou všechny povrchy odrazivější). [12, s. Standard Shader]

V závislosti na zvoleném způsobu práce mezi Metallic workflow a Specular workflow jsou mezi parametry pro Standard Shader malé rozdíly [12, s. Material parameters]. Jelikož podrobný rozbor tvorby materiálů není předmětem této práce, budou popsány pouze parametry při přístupu Metallic workflow relevantní pro tuto práci. Jak je uvedeno v dokumentaci, přístup Metallic má tento název díky tomu, že v něm lze definovat, jak moc kovový má povrch být a není určen pouze pro kovové materiály [12, s. Metallic Parameter].

Parametrem **Rendering Mode** lze určit režim prolnutí průhlednosti a zda bude vůbec použita. Výchozí hodnota je Opaque, ve které se nepracuje s průhledností. Dále lze nastavit například hodnoty Cutout a Transparent. Hodnota Cutout značí použití průhlednosti, kde existují pouze plně průhledné a plně neprůhledné oblasti a hodí se tak kupříkladu pro reprezentaci listů. V případě hodnoty Transparent se průhlednost řídí hodnotami alfa kanálu textur či hodnotou alfa

barvy odstínu, ale neovlivňuje viditelnost odlesků, čímž se hodí například pro materiál skla. [12, s. Rendering Mode]

Parametrem **Albedo** lze nastavit základní barvu povrchu přímou hodnotou či texturou. Jelikož bude světlo přidáno podle kontextu, ve kterém bude objekt viděn, neměla by albedo textura obsahovat žádné světelné informace. Alfa kanálem albedo textury lze ovlivňovat průhlednost materiálu. [12, s. Albedo Color and Transparency]

Při přístupu Metallic workflow je pomocí hodnot parametrů **Metallic** a **Smoothness** ovládána reakce na světlo a odrazivost povrchu. Spekulární odlesky se přímo nedefinují a jsou z těchto hodnot odvozeny. Hodnota prvního parametru určuje, jak moc se povrch chová jako kov a s vyššími hodnotami se snižuje viditelnost albedo barvy ve prospěch barvy založené na odrazech z prostředí [12, s. Metallic Parameter]. Parametr Smoothness určuje, jak moc budou viditelné odlesky, přičemž při vyšších hodnotách jsou odlesky jasnější [12, s. Smoothness]. Dokud se parametru Metallic nepřiřadí textura, jsou hodnoty obou parametrů ovládány posuvníky. Přiřazenou texturou lze pomocí červeného kanálu ovládat úrovně Metallic a pomocí alfa kanálu ovládat úrovně Smoothness. Zbylé kanály této textury jsou ignorovány [12, s. Metallic Parameter].

Pomocí parametru **Normal Map** lze používat normálové mapy pro ztvárnění povrchových detailů bez nutnosti použití další geometrie. S pomocí těchto textur tak lze použít objekty o nízkém rozlišení bez ztráty malých detailů [12, s. Normal map (Bump mapping)]. Více o tomto tipu textur viz [12, s. Normal map (Bump mapping)].

Parametrem **Emission** lze ovládat intenzitu a barvu světla vydávaného objektem, který takto může sloužit jako viditelný zdroj světla. Pro možnost editace hodnot tohoto parametru je potřeba nejprve zaškrtnout checkbox Emission. Takto zpřístupněnému parametru Color lze pro nastavení barvy a intenzity světla nastavit kromě barvy také texturu pro určení svítivých částí materiálu. [12, s. Emission]

Nový asset typu material lze vytvořit zvolením možnosti Assets > Create > Material přístupné z kontextového menu Project View či hlavního menu. Pro přiřazení materiálu k objektu musí mít daný objekt komponentu dědící z Renderer, ve které se nastaví daný asset typu material jako hodnota vlastnosti Material této komponenty. Nejčastěji bude touto komponentou MeshRenderer. [12, s. Materials introduction]

4.5 Princip tvorby UI

Uživateli je nutné v rámci aplikace poskytnout důležité informace, což může být provedeno pomocí HUD. Jak ve své knize uvádí Hocking, „*Heads-up display (HUD) překrývá grafiku navrch pohledu světa. Koncept HUD vznikl s armádními letouny. ... Podobně je na GUI překrývající herní pohled referováno jako na HUD.*“ [19, s. 147] (překlad autora)

V Unity lze tvořit HUD přes tzv. immediate mode GUI system či novější UI systém pracujícím v tzv. retained mode. První způsob se v Unity nachází od první verze a při každém snímku jsou explicitně vydávány vykreslovací příkazy. V tomto způsobu není UI tvořeno v editoru a je plně založeno na kódu. S tímto systémem lze snadno vytvořit tlačítka, ale tvorba ostatních prvků snadná není. U druhého způsobu jsou grafické prvky rozvrženy jednou v editoru a pro jejich vykreslování při každém snímku je není nutné neustále opětovně definovat. Ačkoliv vyžaduje tento způsob větší úsilí při prvotním sestavení, poskytuje kvalitnější výsledky. Zároveň lze při tvorbě UI pomocí tohoto způsobu již při umisťování jednotlivých prvků vidět vzhled UI a snadno pomocí obrázků upravovat jeho vzhled. [19, s. 148–149]

Vzhledem k výše uvedeným skutečnostem bude dále věnována pozornost druhému způsobu tvorby UI. Nejprve je zřejmě vhodné objasnit co je nutné pro použití tohoto systému vytvořit a nastavit. Jak uvádí Hocking, pro správné zobrazení elementů UI je potřeba, aby byly potomky objektu typu Canvas [19, s. 152]. Hocking dále uvádí, že „*Canvas je speciální typ objektu, který Unity vykresluje jako UI pro hru.*“ [19, s. 150] (překlad autora) Pro interakci s UI je vyžadován objekt EventSystem, který je při vytvoření objektu Canvas automaticky vytvořen [19, s. 151]. Jelikož objekt typu Canvas škáluje jeden pixel obrazovky na jednu jednotku ve scéně, bývá oproti 3D scéně obří [19, s. 151].

Na objektu Canvas lze nastavit například Render Mode a Pixel Perfect. Nastavení **Render Mode** nabízí tři možnosti a výchozí hodnotu má nastavenou na Screen Space – Overlay, kde je UI vykreslováno navrch pohledu kamery jako 2D grafika. V případě možnosti Screen Space – Camera lze za podobného způsobu vykreslování dosáhnout efektů perspektivy rotováním jednotlivých prvků UI. Možnost World Space slouží ke ztvárnění UI jako součásti scény a objekty typu Canvas jsou v tomto nastavení umisťovány do scény. Nastavením **Pixel Perfect** dojde k zabránění rozmazání obrázků kupříkladu kvůli jejich umístění mezi pixely tím, že se při vykreslování lehce upraví jejich pozice. [19, s. 151–152]

Obecně se při programování interaktivity UI nejprve vytvoří ve scéně objekt UI, který se následně připojí k objektu ve scéně, kterému se předtím připojil napsaný skript pro obsluhu operací s UI. [19, s. 157]

Tlačítkům lze přiřadit akce při kliknutí pomocí znaménka plus v panelu OnClick, následným přetažením příslušného objektu se skriptem do takto vytvořeného vstupu a výběrem jeho konkrétní metody z rozbalovací nabídky. Z množství interakcí, na které mohou prvky UI reagovat vystavuje komponenta tlačítka pouze událost OnClick. Komponentu EventTrigger lze využít pro interakce, které nejsou výchozí. [19, s. 157–158]

Podobně jako u jiných objektů ve scéně mohou být objekty uvnitř canvas umístěny do rodičovské hierarchie [19, s. 152]. Vrstvení obrázků UI na sebe je závislé na pořadí v náhledu Hierarchie. Objekty umístěné vespod hierarchie budou zobrazovány nad všemi ostatními [19, s. 160]. Díky těmto skutečnostem lze konstatovat, že proces tvorby uživatelského rozhraní lze dobře organizovat a zjednodušit seskupováním souvisejících prvků.

Ne všechen obsah se však může vždy vejít do vybrané oblasti uživatelského prostředí na obrazovce a je tak vhodné znát způsob, jak tento případ vyřešit. Jak je uvedeno v dokumentaci Unity UI, obsah zabírající mnoho prostoru lze na malé oblasti zobrazit pomocí komponenty Scroll Rect, která umožní daným obsahem rolovat [22, s. Scroll Rect].

Některé prvky může být žádoucí umístit nad sebou či vedle sebe. Jak je uvedeno v dokumentaci Unity UI, při použití komponenty Vertical Layout Group jsou jeho potomci umisťovány nad sebe a jejich výška odpovídá minimální, preferované či pružné výšce dle nastavení [22, s. Vertical Layout Group]. Komponenta Horizontal Layout Group své potomky rozmisťuje vedle sebe a jejich šířka odpovídá minimální, preferované či pružné šířce [22, s. Horizontal Layout Group]. Velikost prvku rozložení lze ovládat podle minimální či preferované velikosti kupříkladu prvku obrázku či textu pomocí komponenty Content Size Fitter [22, s. Content Size Fitter].

Pro snazší úpravy je vhodné držet scénu a HUD na sobě nezávislé, a proto by bylo nevýhodné zprostředkovat komunikaci mezi nimi přes skriptové reference. O událostech ve scéně lze UI informovat přes tzv. broadcast messenger system, ve kterém se mohou skripty registrovat pro naslouchání události, které jiný kód vysílá. Narozdíl od vestavěného systému událostí v C# nevyžaduje tento systém, aby kód znal původce zpráv. [19, s. 164]

V rámci uživatelského prostředí je důležité poskytnout uživateli informace také formou textu a k tomu je jistě dobré znát, jak s ním správně zacházet. Jak uvádí Hocking, Pro vylepšení textu Unity vznikl externě systém TextMesh Pro [19, s. 152]. Pro jeho použití je potřeba jej do projektu importovat pomocí Window > Package Manager > TextMesh Pro > Install. Na každé komponentě TextMesh Pro lze měnit vzhled pomocí připojeného materiálu bez nutnosti opětovné tvorby fontu. Zároveň je pod nabídkou Window > TextMesh Pro > Font Asset Creator možné vytvořit z libovolného fontu nový asset pro použití tímto systémem. Záložní glyfy pro případ absence některých znaků v atlase fontu lze nastavit pod Edit > Project Settings > TextMesh Pro Settings. Pro tyto případy lze také nastavit vícero záložních assetů typu Font [24].

5 ANALÝZA MAPOVANÉHO PROSTŘEDÍ A VYTVOŘENÍ MODELŮ

Před počátkem samotného modelování je potřeba vytvářené objekty zmapovat a rozmyslet si, jak bude příslušná struktura vypadat. Jako základ pro tuto činnost slouží plány budovy poskytnuté Technickým odborem Univerzity Pardubice.

Jak uvádí Vaughan, nelze se spoléhat na vlastní paměť a očekávat kvalitní výsledek či věrně zachycené detaily. Reference něčeho skutečného je nezbytná k dosažení realistického výsledku a referenčního materiálu není nikdy dost. [15, s. 79–84]

Z této skutečnosti plyne, že všechna potřebná data pro tvorbu modelu není možné získat pouze ze samotných plánů. Mezi takové údaje se řadí kupříkladu informace o výšce stropů či rozmístění nábytku. Z tohoto důvodu je nutné provést sběr dalších referenčních materiálů.

5.1 Způsoby získání referenčních materiálů

Kromě pořízení fotografií například mobilním telefonem lze modelovaný objekt také přeměřit a v některých případech mohou být informace o rozměrech objektu hodnotnější než pořízené fotografie. Obě metody lze zkombinovat vyfocením metru společně s pozorovaným objektem. K tomuto lze využít místo metru též každodenní objekty u kterých jsou známy jejich rozměry. Dále je vhodné si vést poznámky s důležitými detaily o objektu, mezi které patří kupříkladu naměřené rozměry. Zároveň lze tyto poznámky využít k tvorbě náčrtů při absenci fotoaparátu.

[15, s. 85–88]

Pořízení digitálních referencí se hodí v případech, kdy není možné se k danému objektu fyzicky dostat či osobně pořídit jeho fotky. Mezi největší zdroje pro tyto účely patří Google či Ebay [15, s. 89]. Tento typ reference zřejmě může posloužit v roli doplňkových informací například pro modelování monitorů.

Mnoho referencí lze též získat z filmů či videoklipů například na platformě YouTube. Obecná znalost filmů je užitečná také v oblasti komunikace a správného pochopení zadané myšlenky [15, s. 90]. Lze konstatovat, že tento typ reference lze zároveň získat prostřednictvím vlastního fotoaparátu a následně využít při modelování chodeb, které jsou dlouhé a získání potřebných informací z pouhých fotografií je tímto komplikováno.

Obecně je vhodné vyhnout se referenčním materiálům, na kterých nejsou dostatečně vidět detaily. Mezi nevhodné reference se řadí například fotografie z novin, oskenované fotografie, rozmazané obrázky či ilustrace. Důvodem je, že při absenci viditelných detailů se již nepracuje s referencí, ale pouze s její interpretací založenou především na fantazii. Při použití digitálního modelu jako reference modelář spoléhá na interpretaci objektu původním tvůrcem, a tak se také jedná o nevhodnou referenci. [15, s. 92–94]

5.2 Sběr referenčních materiálů

Jak vyplývá z řádků výše, pro získání dalších referenčních materiálů bylo nutné provést dokumentaci modelované budovy a vnitřního vybavení. Tomuto procesu předcházela příprava zahrnující vytištění reprezentace modelů založené exkluzivně na plánech či výběr vhodných psacích potřeb pro tvorbu dokumentačních poznámek.

Valná většina referenčních materiálů byla pořízena v podobě fotografií pomocí mobilního telefonu Xiaomi Mi 8. Touto formou byly dokumentovány především později ztvárněné učebny a jejich vybavení. V hojné míře bylo také pořízeno mnoho videozáznamů za pomoci stejného zařízení. Všechny tyto materiály doplňují také ručně dokumentované informace v podobě rozměrů naměřených pomocí metru, poznámek o počtu určitých objektů na daném místě či náčrtů do tištěných materiálů zmíněných výše.

Nejprve byly pořízeny fotografie vnějších prostor fakulty. Pozornost byla věnována především všeobecnému vzhledu pro snadné budoucí porovnávání a poté byly pořízeny také fotografie zaměřené na bližší detaily jako jsou například tvary parapetů či venkovních schodů. Pro lepší zachycení prostorového kontextu detailů bylo následně pořízeno také několik videozáznamů zachycujících sekce zdí po vnějším obvodu budovy.

V případě učeben byly nejdříve pořízeny fotografie ze všech rohů místnosti, po kterých bylo vždy pořízeno také video procházející celou místností pro lepší pochopení kontextu zachycených tvarů v prostoru. Následně bylo zdokumentováno vnitřní vybavení daných učeben, které se zpravidla skládá především ze stolů, židlí a tabulí. Pro některé učebny nebylo možné včas získat potřebné referenční materiály, a byly tudíž v této práci vynechány.

Dokumentace nábytku probíhala také nejprve nafocením těchto objektů z mnoha úhlů, následným pořízením videozáznamu a v neposlední řadě také pořízením fotografií bližších detailů dle potřeby. Zároveň byly změřeny rozměry daných kusů nábytku a jejich počtu v příslušných učebnách. Zejména v případě vybavení učebny H2 se vzhledem k jejich množství tyto ručně

zapsané informace prokázaly jako nepostradatelné pro efektivní vytvoření modelů a jejich řádné rozmístění.

Vzhledem k délce chodeb vnitřních prostor fakulty se tyto prokázaly jako těžce zachytitelné v uspokojitelné míře pomocí fotografií. Hlavním problémem vzniknuvším touto délkou je značně ztížená orientace v pořízených materiálech při zachycení dostatečných detailů. Z tohoto důvodu nejsou pro dokumentaci chodeb použity fotografie, nýbrž videozáznamy. Videozáznamy umožňují se snadno orientovat v pořízených datech při jejich použití v kontextu trojrozměrného prostoru a lze se tak na ně snadno odkázat při tvorbě příslušných modelů.

Ne vždy však bylo možné plně zachytit veškerá potřebná data během pořizování referencí v terénu. Pro doplnění těchto chybějících dat byly tudíž použity digitální reference získané ze stránek ke dni otevřených dveří Fakulty elektrotechniky a informatiky Univerzity Pardubice ([23]).

Na základě takto získaných referenčních materiálů bylo následně možné čerpat informace s dostatečnými detaily pro do značné míry autentické zachycení vnějších i vnitřních prostor fakulty včetně jejího vybavení. Zmíněné referenční materiály následně také nalezly uplatnění při tvorbě textur.

5.3 Návrh struktury modelů

Při vytváření modelů v projektu je klíčová organizace a je tak důležité vhodně zvolit jmennou konvenci. Vhodné pojmenování je takové, které s daným obsahem souvisí a usnadňuje tak jeho nalezení. Dále je vhodné v názvu namísto mezery používat znak podtržítka, jelikož jej na rozdíl od mezery není tak snadné přehlédnout a zároveň nezpůsobí problémy v aplikacích například při použití matematických výrazů. [15, s. 145–147]

Lze konstatovat, že přirozenou strukturou je vzhledem k povaze modelu rozdělení na budovy, dveře, okna, doprovodné architektonické prvky a vybavení. Budovy lze následně rozdělit na část interiéru a exteriéru, přičemž interiér je dále vhodné rozdělit na jednotlivá patra, která jsou následně složena z jednotlivých místností. Jména budov poté mohou nést skutečná jména CA a CB. Patra mohou název obdržet dle skutečného označení pater v plánech podobně jako místnosti. Tento způsob organizace poskytuje nejen přehlednost, ale také usnadňuje práci s plány vzhledem k identickému označení místností a pater. Další výhodou tohoto uspořádání je skutečnost, že je možné provádět úpravy na jednotlivých místnostech bez ovlivnění ostatních. Tuto strukturu lze v Blenderu implementovat za pomoci kolekcí popsaných v kapitole 3.1.

Jak se později ukázalo, při exportu do formátu FBX není hierarchie kolekcí zachována. Z tohoto důvodu je nutné tento způsob použít v kombinaci s řešením použitelným v tomto herním enginu Unity. Jak je uvedeno v tutoriálu na stránkách Unity, pro organizaci skupin objektů lze využít prázdných herních objektů [25]. Lze tedy dojít k závěru, že vhodným postupem je v Blenderu v každé kolekci vytvořit prázdný objekt pojmenovaný dle kolekce na souřadnicích odpovídajících hodnotám odsazení kolekce a následně jej příslušným objektům přiřadit jako rodiče.

Vzhledem ke skutečnosti, že cílem této práce není vytvořit zcela přesnou repliku budovy fakulty, nýbrž její dostatečně věrnou reprezentaci pro použití v aplikaci vykreslované v reálném čase, není nezbytně nutné zachytit zcela přesně výšky a lze je tudíž odvodit z informací ve výše zmíněných plánech a fotografiích. Pomocí plánů lze kupříkladu určit standardní výšku jednoho schodu na hlavním schodišti, kterou lze následně v kombinaci s celkovým počtem schodů použít pro zarovnání podlahy na každém patře. Toto zarovnání podlah lze následně využít k zarovnání výšky stropů na hlavních chodbách, jelikož je známa výška oken, která se zde nachází.

5.4 Proces tvorby modelu fakulty a vybavení

Jak již bylo zmíněno výše, základ samotného modelu budovy fakulty je založen na plánech. Plány byly převedeny pomocí aplikace Autodesk AutoCAD 2019 do formátu DXF pro následný import do Blenderu v podobě křivek. Zároveň byly tyto plány vyexportovány stejnou aplikací do formátu PDF, odkud byly následně pomocí aplikace Inkscape 1.0 vyexportovány do bitmapového formátu PNG také za účelem následného importu do Blenderu. Kombinace obou typů dat poskytuje možnost precizního přichytávání bodů k jednotlivým křivkám bez ztráty důležitých doprovodných informací jako jsou například čísla místností či rozměry dveří.

Zdi exteriéru byly nejprve vymodelovány pomocí techniky point by point, kdy jednotlivé body byly přichytávány k bodům křivek ze získaných plánů. V této fázi nebylo zcela jasné, jaké budou přesné pozice a rozměry jednotlivých otvorů oken a dveří, a proto nebyly ztvárněny přímo v topologii, nýbrž za použití modifikátoru Boolean (viz 3.3). Tento modifikátor však nutně nevytvoří čistou topologii a nelze jej tak aplikovat pro získání polygonové sítě vhodné pro použití v Unity. Ve fázi ustálení geometrie a přípravy na export byly tudíž tyto modifikátory odstraněny a příslušné otvory vytvořeny nástrojem Loop Cut and Slide (viz 3.3) a manipulací bodů a hran.

V případě tvorby místností interiéru byl zvolen stejný přístup, jelikož v obou případech poskytoval značnou flexibilitu při nutných změnách bez nutnosti následných oprav topologie. Jednou

z nejvýraznějších prováděných změn byla operace posouvání všech pater s výjimkou přízemí poté, co byla odhalena chyba s výškou schodišť a všechna patra tak bylo nutné zarovnat dle opravených hodnot. V kombinaci s výše popsanou strukturou založenou na kolekcích bylo možné tuto operaci provést velmi rychle a snadno bez rizika poškození topologie.

Vzhledem k absenci jiných referenčních dat v době tvorby schodů byly jejich základní rozměry získány na základě plánů a měření. Výška samotného schodiště byla následně získána dosazením příslušného počtu daných schodů. Pro větší efektivitu tohoto procesu byl použit modifikátor Array (viz 3.3), kterému lze nastavit počet a vzdálenost mezi jednotlivými prvky na osách X, Y a Z. Také v tomto případě bylo nutné konečnou topologii vyčistit, avšak tento přístup i zde značně usnadnil pozdější přidávání detailů, které bylo nutné aplikovat pouze na jediný schod, přičemž na ostatní se tyto změny automaticky zapsaly.

Vzhledem ke skutečnosti, že v budově se nacházejí celkem 3 schodiště, v rámci kterých se s výjimkou určitých částí nachází na každém patře stejný počet schodů, bylo by značně nevýhodné je pro každé patro modelovat znovu. V tomto ohledu se opět zvolená struktura projevila jako značně užitečná, jelikož umožnila opakované využití jednotlivých skupin objektů pomocí instancí kolekcí. Pro lepší přehlednost byly tyto instance umístěny do speciálních kolekcí určeným pouze instancím kolekcí schodů pro jednotlivá patra.

Základy modelů zábradlí pro jednotlivá schodiště vznikly pomocí techniky box modeling v kombinaci s nástrojem Bevel a modifikátory Solidify a Array. Tímto způsobem bylo možné kontrolovat tloušťku madel či příček a jejich počet při nutnosti manipulace pouze s body původního jednoduchého tvaru. Z tohoto základu bylo následně možné vytvořit konečné zábradlí pro samotné schodiště. Tento proces spočíval především v úpravě souřadnic na osách X, Y a Z v modifikátoru Array použitém na příčky dle výšky schodů a deformace zbytku zábradlí dle úhlu schodiště.

Druhá zmíněná činnost se prokázala jako větší problém, než bylo původně očekáváno. Původní řešení spočívalo v použití modifikátoru Mesh Deform a následné úpravě jednoduchého tvaru obepínajícího příslušné zábradlí dle požadovaného úhlu. Tento přístup však nedosahoval požadovaných výsledků v kombinaci s rohy, které jsou umístěny v rovině. Jako nejjednodušší řešení se ukázalo zarovnat vzdálenosti madel od země v jednotlivých patrech a pomocí modifikátoru Mesh Deform do daného úhlu deformovat pouze jeden článek výplně mezi příčkami, který bylo možné v režimu Edit Mode následně ručně rozkopírovat na příslušná místa tak, aby na sebe navzájem jednotlivé části navazovaly.

Vzhledem k výše zmíněným speciálním případům schodišť v určitých patrech by přiřazení zábradlí přímo do kolekcí schodišť působilo příliš mnoho komplikací, a tak byla zábradlí podobně jako schodiště instancována do speciálně určených kolekcí. Do stejných kolekcí byla následně umístěna také všechna ostatní zábradlí.

Jednotlivá okna a dveře vznikly metodou box modeling na základě informací z plánů a fotografií. Vzhledem k možnosti opětovného využití určitých částí jako jsou např. kliky, rámy či parapety, jsou tyto části vytvořeny v samostatných kolekcích a následně z instancí příslušných kolekcí poskládány do celku ve speciálních kolekcích určených těmto přednastaveným objektům.

V případě desek dveří byly nastaveny jejich středové body tak, aby je bylo možné otáčet jako skutečné dveře pro otevírání. Ne všechna okna a dveře však vyžadují takto detailní přístup a pro obě skupiny tak existují kolekce s plně obsáhlými modely bez výše popsaného rozdělení. Jejich rozmístění je prováděno zcela totožným způsobem jako v případě schodiště a zábradlí, neb se tento systém již pro vizualizaci osvědčil.

Při tvorbě nábytku byly využívány jak obrazové referenční materiály, tak ručně zapsané poznámky obsahující příslušné rozměry a další důležité znaky. Na tuto skupinu objektů byly aplikovány především techniky box modeling a primitive modeling, které umožňovaly rychlé zachycení potřebného tvaru. Mnoho částí vybavení sdílí určitý společný prvek, tudíž i zde byla zvolena cesta skládání instancí kolekcí pro dosažení požadovaného výsledku. Díky tomuto přístupu bylo možné zaplnit s menším množstvím objektů celou budovu vybavením. I v tomto případě byly potřebné instance umisťovány do kolekcí tomu určených. V tomto případě však byly pro lepší přehlednost tyto objekty děleny nejen dle pater, ale také dle samotných místností. Zcela stejným způsobem byly následně vytvořeny a rozmístěny také potřebné modely světel a informačních tabulí.

Díky tomuto rozvržení bylo možné přidávat detaily až ve chvíli, kdy byly zcela jasné zásadní tvary tvořených objektů. Zároveň bylo možné provést vizualizaci konečné scény bez narušení samotných objektů. Kolekce obsahující rozmístění jednotlivých instancí zároveň mohou sloužit jako zdroj dat pro souřadnice daných objektů ve scéně v Unity.

Při modelování nábytku byly hojně využity techniky box modeling a primitive modeling. Zároveň nelze opomenout nepostradatelnou roli referenčních materiálů, díky kterým bylo možné nejen přesněji zachytit zdokumentovaný nábytek, ale také značně urychlit samotný proces modelování, jelikož příslušné rozměry stačilo při manipulaci s hranami přímo zadat.

Pro doplnění scény okolí byly nakonec pomocí techniky box modeling vymodelovány jednoduché modely reprezentující okolní budovy. Jejich tvar je složen z jednoduchých tvarů kostek a uprostřed vyzvednutých plošek pro reprezentaci střechy.

5.5 Proces tvorby materiálů pro vytvořené modely

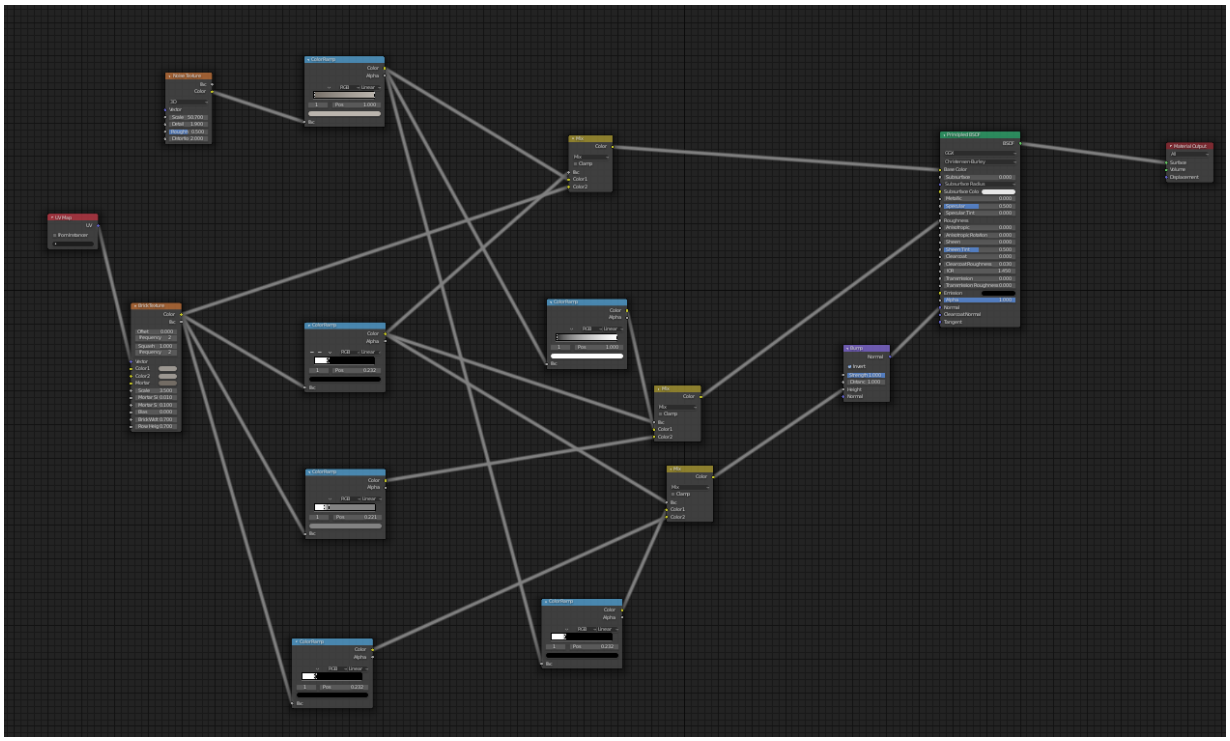
Většina materiálů byla vytvořena pouhým nastavením barvy na požadovanou hodnotu dle fotografií a nastavením hodnoty drsnosti kvůli kontrole nad odrazivostí příslušného materiálu. Tento způsob byl zvolen především z důvodu nižších nároků materiálů bez textur a skutečnosti, že v mnoha případech by detaily získané z případných textur nebyly dostatečně výrazné pro ospravedlnění dodatečné práce s tvorbou UV map.

Všechny materiály však nelze dostatečně autenticky ztvárnit pouze za pomoci barev. Mezi takovéto materiály se řadí například kachle či vzory u linolea v učebnách. Pro vytvoření potřebných textur existuje mnoho způsobů a pro tento projekt byl zvolen přístup tvorby pomocí procedurálních textur dle principů PBR a jejich případných úprav dle principů PBS v Unity.

Důvodem pro volbu procedurálních textur je dostatečná jednoduchost tvořených textur a eliminace možných problémů, které by při tradičním přístupu tvorby textur z fotografií mohly vlivem odrazivosti či obtížného dosažení návaznosti textur na sebe u některých materiálů vzniknout. Další výhoda, kterou procedurální textury poskytují spočívá v absolutní kontrole nad výsledným rozlišením textur bez ztráty kvality.

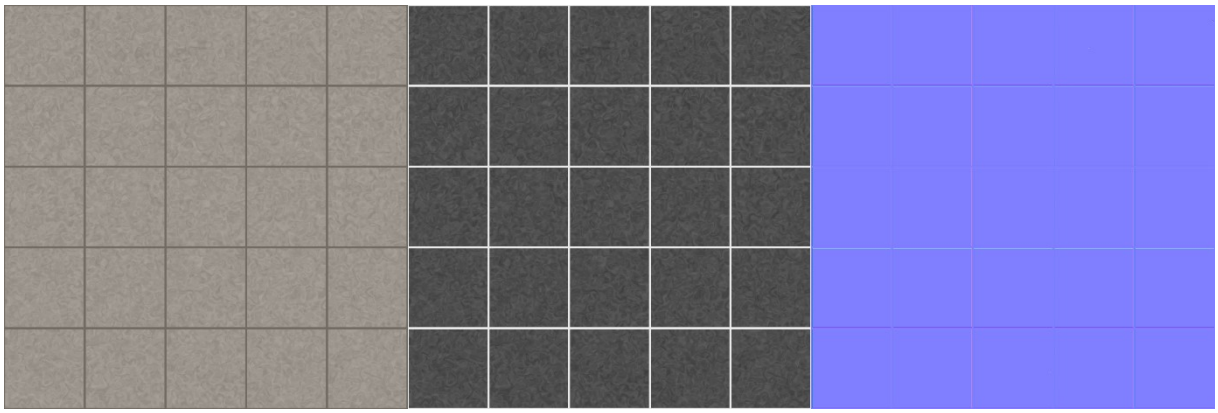
Ačkoliv je tyto textury možné vygenerovat pomocí programů k tomu určených či využít některé z existujících textur, pro účely tohoto projektu zcela postačují možnosti poskytované přímo v editoru Shader v Blenderu.

Kupříkladu pro tvorbu podlahové dlažby lze využít uzel Brick Texture a v parametrech nastavit stejné rozměry pro šířku (Brick Width) a výšku (Row Height) a jejich odsazení (Offset) do rovnoměrných sloupců a řádků. Následně lze výstup z tohoto uzlu použít a pomocí uzlu pro kombinaci barev zkombinovat s výstupem uzlu Noise generujícího Perlinův šum. Výstup této kombinace lze následně obarvit pro dosažení požadované albedo textury. Pomocí uzlu Color Ramp lze také upravit jeho kontrast pro vytvoření požadované mapy drsnosti. Černobílý výstup tohoto uzlu lze také použít v kombinaci s uzlem Bump pro tvorbu normálové mapy. Výsledný strom uzlů je znázorněn na následujícím obrázku.



Obrázek 11: Systém uzlů tvořící materiál kachlí podlahy. Zdroj: Autor

Jak však již bylo zmíněno v kapitole 3.4, takto přímo generované textury nelze přenést přímo do Unity a pro tyto účely je nutné je zapéct do souboru obrázku. Pro mapu albedo byla zvolena možnost Diffuse, pro mapu drsnosti možnost Roughness, atd. Takto zapsané obrázky lze následně uložit do vhodně pojmenovaných obrázkových souborů a zároveň je lze nyní také využít k vizualizaci konečné podoby objektů využívajících daný materiál prostým připojení uzlů s příslušnými obrázky namísto uzlů generujících procedurální textury, čímž se usnadní proces tvorby UV map.



Obrázek 12: Mapy albedo, drsnosti a normálová mapa dlaždic podlahy. Zdroj: Autor

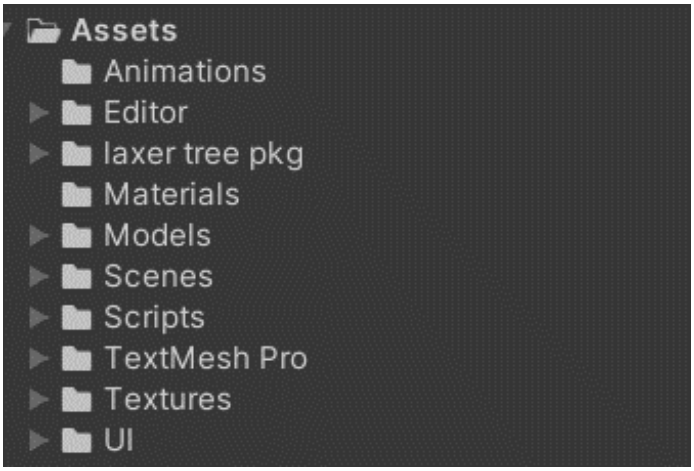
6 NÁVRH APLIKACE V PROSTŘEDÍ UNITY

Před samotnou tvorbou konečné aplikace je jistě vhodné blíže stanovit, jak má vypadat její konečná podoba nejen po sestavení, ale také v rámci vytvářeného projektu. Jak se ukázalo již při tvorbě modelů, řádná organizace má velmi pozitivní vliv na efektivitu práce.

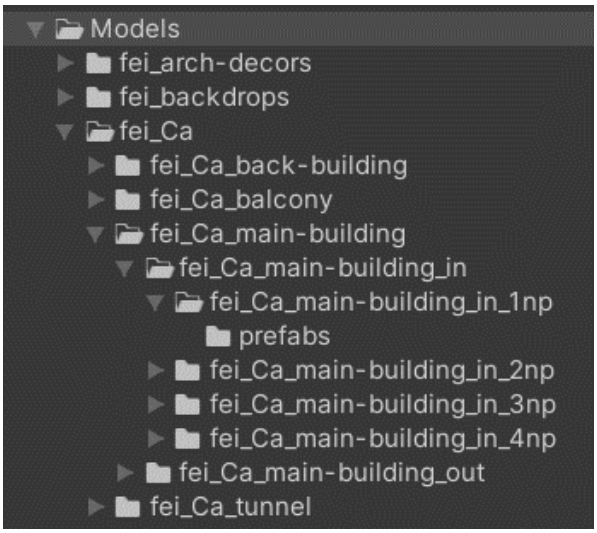
V této fázi také bylo blíže určeno, jaké další aplikace budou při tvorbě použity. Kromě již výše zmíněných aplikací Blender 2.83, Unity 2019.4, AutoCAD 2019 a Inkscape 1.0 byly také zvoleny aplikace Affinity Photo 1.8.3.641 pro dodatečné úpravy vytvářených textur a Visual Studio 2019 pro úpravu kódu. Pro testování sestaveného projektu byly zvoleny prohlížeče Mozilla Firefox 88.0 a Google Chrome 90.0.4430.93. Pro testování na lokálním serveru byl použit nástroj XAMPP 7.3.10.

6.1 Stanovení struktury projektu

Pro organizaci projektu byla zvolena struktura rozdělující modely, materiály, textury a skripty do jednotlivých složek. V rámci těchto složek je následně dle potřeby prováděno další dělení pro dosažení dostatečného rozdělení na logické části. V případě modelů se jedná především o rozdělení úzce následující organizační strukturu stanovenou dříve pro kolekce doplněné o složky určené prefabům kvůli oddělení importovaných modelů od těch, se kterými bylo již manipulováno například přidáním kolizí. Prefaby vytvářené na základě importovaných modelů jsou vytvářeny jako varianta prefabu z důvodu zachování vazeb s původním modelem a tím i lepšího propsání případných změn původního objektu.

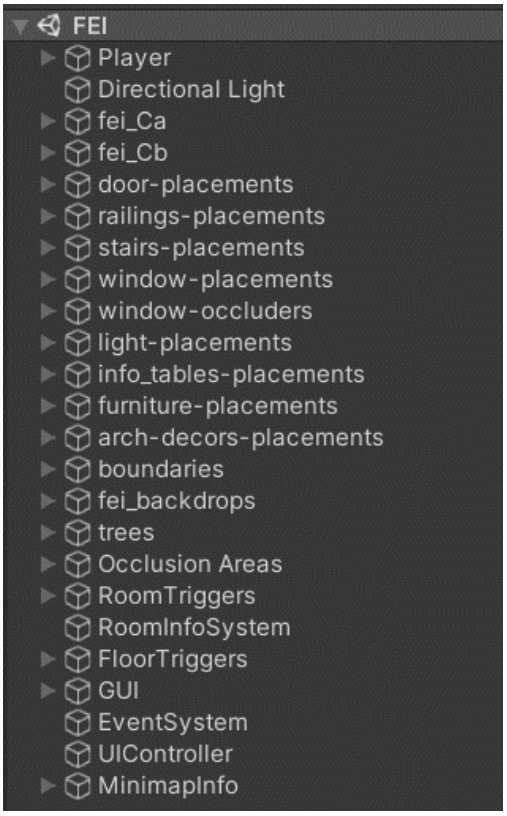


Obrázek 13: Výsledná hlavní struktura složky Assets. Zdroj: Autor



Obrázek 14: Rozbalená adresářová struktura pro modely podlaží 1NP. Zdroj: Autor

V samotné scéně se též jeví jako vhodná strategie použít pro organizaci modelů strukturu odpovídající té definované v Blenderu. Objekty uživatelského prostředí budou organizovány pod základním objektem typu Canvas a podobně tomu bude také v případě objektů týkajících se přímo postavy ovládané uživatelem. Případné pomocné objekty lze tematicky organizovat do samostatných hierarchií podobným způsobem jako samotné modely.



Obrázek 15: Implementovaná struktura scény. Zdroj: Autor

6.2 Návrh způsobu ovládání aplikace

Pro pohyb v modelu budovy byl zvolen pohled z první osoby, jelikož na rozdíl od pohledu z třetí osoby nevyžaduje tvorbu avataru a poskytuje úhel pohledu věrnější realitě. Pro samotné ovládání bylo zvoleno dnes již standardní ovládání kamery pomocí myši. Jak uvádí Hocking, pohyb pomocí kláves W, A, S, D či šipek je ve většině FPS her standardem [19, s. 45]. Z toho to důvodu byla pro pohyb v prostoru zvolena možnost pomocí těchto kláves. Pro jednodušší správu kolizí a zamezení propadů skrze objekty nebyla do tohoto schématu zahrnuta možnost skákání.

Kromě samotného pohybu je také nutné uživateli umožnit s určitými objekty jako jsou dveře či informační tabule interagovat. Vzhledem k velkému množství takovýchto objektů je zároveň nutné omezit možnost těchto interakcí dle vzdálenosti a viditelnosti. Zároveň je nutné uživatele o případné možnosti interakce vhodně informovat prostřednictvím popisu dané akce a ikonky klávesy určené pro její provedení.

V případě dveří byla pro interakci zvolena klávesa E, jelikož se nachází blízko ostatním klávesám ovládání a též se využívá často pro příslušné účely. Jako zobrazovaná zpráva při možnosti interakce byl zvolen text „Použít dveře“ a při stisku výše zvolené klávesy je možné příslušné dveře otevřít a zavřít. Vzhledem k přirozené funkci dveří jako bariér je vhodné je využít k zamezení přístupu do nezpřístupněných místností, a proto je vhodné označit výchozí stav dveří jako „zamčené“ a umožnit otevírat pouze vybrané dveře. Ačkoliv je možné problematiku zamčených dveří vyřešit odstraněním možnosti interakce na zamčených dveří, v tomto projektu bylo zvoleno řešení v podobě zobrazení textu „Zamčeno“ na průhledovém displeji z důvodu větší míry interaktivity.

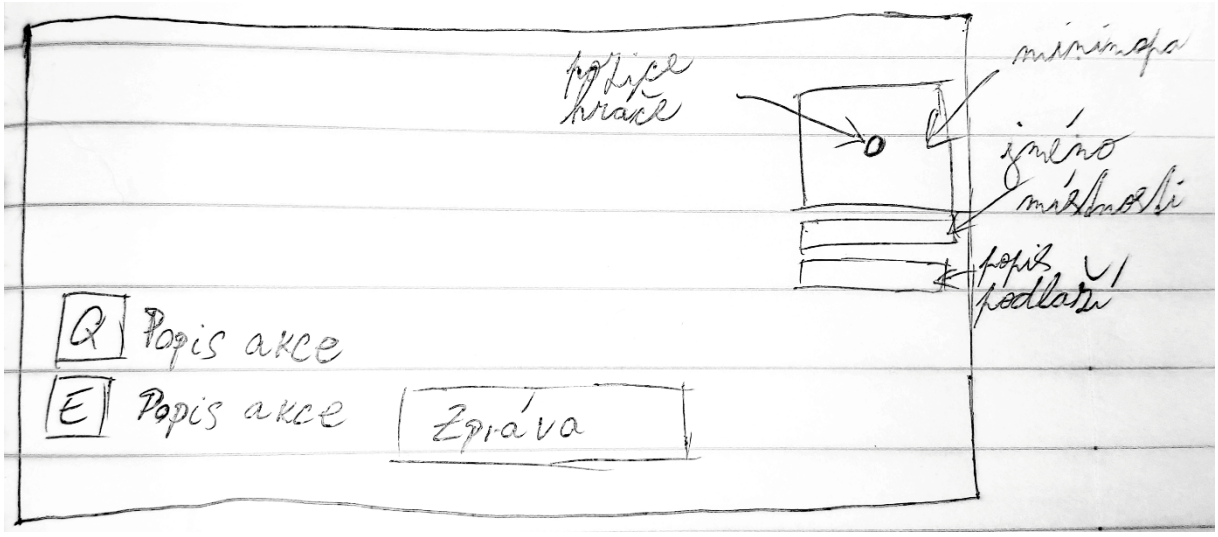
Interaktivní informační tabule lze rozdělit do dvou skupin. Účelem první skupiny je umožnit zobrazit doprovodné informace o místnosti, zatímco druhá má za cíl uživateli zpřístupnit rozvrh příslušné místnosti. V obou případech je plánovaný výsledek interakce zobrazení těchto informací na průhledovém displeji. Vzhledem ke skutečnosti, že je umístění informačních tabulí pro rozvrhy silně spjato s umístěním informačních tabulí pro doprovodné informace dané místnosti, lze konstatovat, že pro lepší uživatelský zážitek je vhodné určit jinou klávesu pro interakci s první skupinou než pro interakci s druhou. V případě první jmenované skupiny byla opět zvolena klávesa E ze stejných důvodů jako v případě dveří a jako doprovodná zpráva byl zvolen text „Zobrazit informace o místnosti“. Pro interakci s druhou skupinou byla zvolena klávesa Q také na základě blízké pozice ke zbytku kláves ovládání a pro zobrazovanou doprovodnou

zprávu byl zvolen text „Zobrazit rozvrh“. V obou případech byla pro zavření zobrazovaných informací vytvořena tlačítka.

6.3 Návrh uživatelského rozhraní

Pro lepší orientaci v prostorách budovy byla zvolena implementace minimapy, ve které jsou zobrazena jména zpřístupněných místností. Pro její umístění byl zvolen pravý horní roh obrazovky, kde jsou umístěny také informace o současné poloze uživatele v podobě jména současné místnosti a patra, na kterém se v danou chvíli nachází.

Pro informační zprávy o možnosti interakce byl zvolen levý dolní roh obrazovky, kde jsou příslušné akce v případě možnosti interakce více tlačítky vypsány pod sebou. Pro zprávy informující o zamčených dveří byl zvolen prostředek dolní části obrazovky.



Obrázek 16: Koncept HUD. Zdroj: Autor

Pro reprezentaci dodatečných informací o místnosti byla zvolena grafická podoba založena na skutečných informačních tabulích nacházejících se v budově při respektování jednotného grafického stylu Univerzity Pardubice ([26]). V případě informací o rozvrhu místnosti byl zvolen vzhled založený na grafické podobě tabulky rozvrhu dle pořízených fotografií.

Pro zobrazení dalších doprovodných informací jako kupříkladu shrnutí celého schématu ovládání byla zvolena cesta vytvoření menu pro pozastavení hry, ve kterém mohou být jednotlivé informace přístupné pomocí tlačítek. Toto menu může též obsahovat tlačítka pro návrat zpět do hry.

7 REALIZACE VLASTNÍ APLIKACE

V této fázi dochází k samotné implementaci konečného projektu na základě informací uvedených v předešlých kapitolách. Zároveň je pro úspěšné dokončení aplikace nutné vytvořené modely a textury umístit do herního enginu Unity a řádně je propojit pomocí materiálů.

7.1 Export a import modelů z Blenderu do Unity

Ačkoliv je možné do projektu vložit přímo soubory aplikací Maya, 3ds Max či Blender, má tato metoda jistá omezení. Pro její využití je nutné mít příslušnou aplikaci nainstalovanou a ve skutečnosti je načten soubor FBX či Collada, který je vyexportován na pozadí. Z těchto důvodů je preferováno export provést přímo. Export pomocí FBX zpravidla funguje nejlépe, ale Collada také funguje dobře pokud není FBX k dispozici [19, s. 90–91]. Na základě těchto skutečností byl jako formát pro export z Blenderu a následný import do Unity zvolen formát FBX.

Díky rozdělení modelů do menších celků popsaném v kapitole 5.3 bylo možné provádět export potřebných částí bez nutnosti vždy exportovat celý model. Tato skutečnost však zároveň znamená nutnost exportu mnoha objektů, což činí z prvotního exportu časově náročnou operaci. Z tohoto důvodu je důležité prozkoumat možnosti automatizace tohoto procesu.

Postup exportu jedné skupiny objektů lze shrnout do následujících kroků:

1. Vybrání hierarchie objektů k exportování.
2. Otevření dialogu pro export do požadovaného formátu.
3. Nastavení požadovaných parametrů.
4. Zvolení cílové cesty exportu.
5. Změna jména na požadovanou hodnotu.

Skutečnost že lze tento proces shrnout do několika jednoduchých kroků naznačuje, že jej lze automatizovat. Na základě dokumentace Blenderu byl proto vytvořen skript v jazyce Python, který rekurzivně projde zadané kolekce a pro každého jejich potomka provede výše popsané akce. Pomocí tohoto skriptu bylo následně možné veškeré objekty vyexportovat během pár sekund se jmény odpovídajícím jménům kolekcí bez rizika chyb z důvodu překlepů.

```
"""
Exports objects in the specified collection
"""
def export_collection_objects(object_path, collection_name):
  for obj in bpy.data.collections[collection_name].all_objects:
    obj.select_set(True)

  bpy.ops.export_scene.fbx(filepath=object_path, check_existing=True,
  filter_glob='*.fbx', use_selection=True, use_active_collection=False,
  global_scale=1.0, apply_unit_scale=True,
  apply_scale_options='FBX_SCALE_NONE', bake_space_transform=True,
  object_types={'ARMATURE', 'CAMERA', 'EMPTY', 'LIGHT', 'MESH', 'OTHER'},
  use_mesh_modifiers=True, use_mesh_modifiers_render=True,
  mesh_smooth_type='OFF', use_subsurf=False, use_mesh_edges=False,
  use_tspace=False, use_custom_props=False, add_leaf_bones=True,
  primary_bone_axis='Y', secondary_bone_axis='X',
  use_armature_deform_only=False, armature_nodetype='NULL', bake_anim=True,
  bake_anim_use_all_bones=True, bake_anim_use_nla_strips=True,
  bake_anim_use_all_actions=True, bake_anim_force_startend_keying=True,
  bake_anim_step=1.0, bake_anim_simplify_factor=1.0, path_mode='AUTO',
  embed_textures=False, batch_mode='OFF', use_batch_own_dir=True,
  use_metadata=True, axis_forward='-Z', axis_up='Y')

  for obj in bpy.data.collections[collection_name].all_objects:
    obj.select_set(False)
```

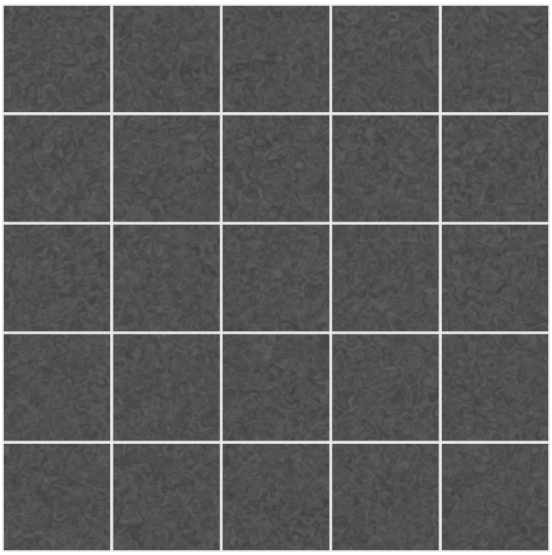
Zdrojový kód 1: Funkce pro export objektů. Kód vytvořen podle [18, s. Export Scene Operators] a upraven autorem.

Možnosti psaní skriptů lze využít také pro získání souřadnic instancí ze speciálních kolekcí popsaných v kapitole 5.4 a není je tak nutné ručně do Unity kopírovat a převádět z pravotočivého souřadnicového systému do levotočivého souřadnicového systému. Pro tyto účely byl tudíž vytvořen skript, který projde všechny objekty v zadaných kolekcích a jejich souřadnice převedené do souřadnicového systému používaného v Unity uloží společně se jménem instancované kolekce do slovníku, který přidá do seznamu těchto slovníků a následně tento seznam serializuje do souboru ve formátu JSON.

Při následném importu exportovaných modelů do Unity se strategie přiřadit jim jako rodiče prázdný objekt se jménem příslušné kolekce projevila jako výhodná. Umožňuje totiž kromě organizačních účelů zmíněných v kapitole 5.3 také manuální určení pozice vztažného bodu pro celou skupinu, která je poté v Unity zachována.

Hocking doporučuje použít pro textury v Unity formát TGA, či PNG. Dnes je formát PNG technologicky téměř totožný formátu TGA a před širokou adopcí formátu PNG na internetu byl

pro 3D grafiku oblíbený formát TGA [19, s. 82–83]. Veškeré vytvořené textury byly tudíž vyexportovány do formátu PNG. Textury drsnosti popisované v kapitole 5.5 bylo nutné pro použití v Unity upravit, jelikož vybraný typ materiálu neobsahuje parametr pro drsnost a parametr Smoothness lze v případě textur ovládat pouze mapou kovovosti. Tyto úpravy spočívaly v inverzi hodnot mapy drsnosti a jejich následném aplikování na alfa kanál vytvářené mapy kovovosti.



Obrázek 17: Výsledná mapa kovovosti materiálu dlaždic. Zdroj: Autor

Před zahájením samotného importu bylo nutné vytvořit adresářovou strukturu pro řádnou organizaci projektu. Po vytvoření adresářové struktury popsané v kapitole 6.1 byly nejprve importovány vytvořené textury do složky Textures, s pomocí kterých byly následně ve složce Materials vytvořeny materiály.

Jména těchto materiálů plně odpovídají jménům materiálů vytvořených v Blenderu kvůli usnadnění jejich přiřazování k objektům. Jak bylo nastíněno v kapitole 4.4, veškeré materiály byly vytvořeny s pomocí shaderu Standard Shader při přístupu Metallic workflow. V tomto nastavení bylo možné přiřadit vytvořené albedo mapy do slotu Albedo, případně zde nastavit barvu odpovídající té ze vzorového materiálu v Blenderu. V případě odrazivosti materiálu stačilo připojit vytvořenou mapu kovovosti či nastavit parametr Smoothness na invertovanou hodnotu Roughness materiálu v Blenderu. Také v případě ostatních textur je bylo potřeba pouze přiřadit k příslušným slotům.

Fáze importu modelů proběhla velice snadno, jelikož všechny modely se již nacházely v předepsané adresářové hierarchii, jejíž kořen tak stačilo pouze přetáhnout do složky Models v okně

editoru. U všech byly následně zaškrtnuty možnosti při generování UV map pro světelné mapy (lightmaps), vypnuto importování animací a příslušné importované materiály byly namapovány na výše zmíněné materiály vytvořené v Unity. Jak je uvedeno v dokumentaci, ve výchozím nastavení jsou materiály v importovaném modelu vloženy [12, s. Materials tab]. Bez provedení posledního zmíněného kroku by tudíž zřejmě měly všechny objekty své vlastní unikátní materiály, což by se jistě projevilo negativně nejen na možnostech kontroly vzhledu objektů, ale také na výsledném výkonu aplikace.

7.2 Sestavení modelu fakulty a rozmístění vybavení v Unity

Po importování potřebných modelů byly do scény postupně vkládány jednotlivé části budov. Vzhledem k nutnosti velké přesnosti vzájemného umístění modelů nebylo z počátku zcela jasné, jak náročný tento proces bude. Jak se však ukázalo, při vložení objektu do scény pomocí záložky Hierarchy zaujme vkládaný objekt pozici odpovídající té v Blenderu a celý základ scény bylo tak možné sestavit velmi snadno a rychle.

V každé složce s modely byla následně vytvořena složka prefabs, do které byly vkládány varianty prefabů vzniknuvší tažením objektů ve scéně ze záložky Hierarchy do této složky a zvolením možnosti prefab variant v kontextovém okně. Tento přístup poskytuje lepší možnosti úprav bez ovlivnění původních importovaných souborů a zároveň zajistí propsání změn v těchto původních souborech do příslušných prefabů.

V těchto vytvořených prefabech byly následně vytvořeny příslušné kolize. S výjimkou pár složitých tvarů, kupříkladu v místnostech H1 a H2, byly pro kolize zvoleny skupiny primitivních kolizních objektů (colliders). V případech zmíněných komplikovaných tvarů byla použita komponenta Mesh Collider. Tyto prefaby byly ve scéně následně uspořádány do hierarchické struktury odpovídající struktuře kolekcí dříve vytvořené v Blenderu.

Po umístění místností budovy byly do scény vloženy schody a zábradlí. I v tomto případě byly vytvořeny prefaby, kterým se též přiřadily kolize. Zde však již nebylo možné všechny modely přesně umístit pouhým přetažením do záložky Hierarchy. Jejich přesné souřadnice však již byly známé a jednoduše přístupné prostřednictvím výše zmíněných souborů ve formátu JSON. Tuto skutečnost by bylo možné využít při ručním umístění modelů do scény a následném ručním zkopírování těchto hodnot do komponenty Transform těchto objektů. Tento přístup by však byl značně časově náročný a náchylný na chyby.

Jako výhodnější přístup se projevilo využít rozšiřitelnosti funkcí editoru popsané v kapitole 4.2. Pro rozmístění schodů a zábradlí tudíž bylo vytvořeno několik skriptů, které je na základě dat z dříve vytvořených JSON souborů ve scéně rozmístily a zároveň umístily do odpovídající hierarchie podle původních kolekcí.

Skript PathsHolder obsahuje cesty k potřebným JSON souborům a k prefabům, které mají být do scény vloženy. K těmto cestám je následně umožněn přístup pomocí slovníku, jehož klíče odpovídají názvům odpovídajících kolekcí.

Pro řádné načtení potřebných dat bylo nutné kromě třídy PosData, reprezentující jeden prvek v serializovaném seznamu, vytvořit také třídu PosListData, do které je načten celý seznam serializovaných pozic a jméno serializovaného objektu.

Třída SceneBuilder zodpovídá za samotné rozmístění prefabů do scény a vytvoření hierarchie. Při této operaci je načten obsah příslušných JSON souborů, který je následně procházen a při nalezení shody jména s hodnotou klíče ve slovníku je příslušný prefab umístěn na požadovanou pozici.

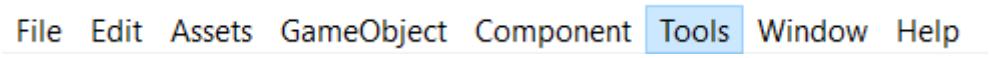
Pro vykonávání těchto skriptů byly vytvořeny nové nabídky v horní liště s názvem Tools, pod kterou byly následně umístěny možnosti Place Stairs a Place Railings. Pro snazší správu těchto položek byla také vytvořena třída WindowPopupBase, ze které příslušné třídy dědí.

```
using System.Collections.Generic;
using System.IO;
using UnityEditor;
using UnityEngine;

/// <summary>
/// Base class for all of the other
/// extension script windows
/// </summary>
public class WindowPopupBase : EditorWindow
{
  public static string[] options;
  protected static string[] optionNames;
  public int index = 0;
  protected static FileInfo[] objectsPlacementsInfo;
  protected static string positionsFolderPath;
  protected static Dictionary<string, string> objectsPaths;

  /// <summary>
  /// Generates the window
  /// and initializes the options
  /// </summary>
  public static void GenerateWindow()
  {
    EditorWindow window = GetWindow(typeof(WindowPopupBase));
    window.Show();

    GenerateOptions();
  }
```

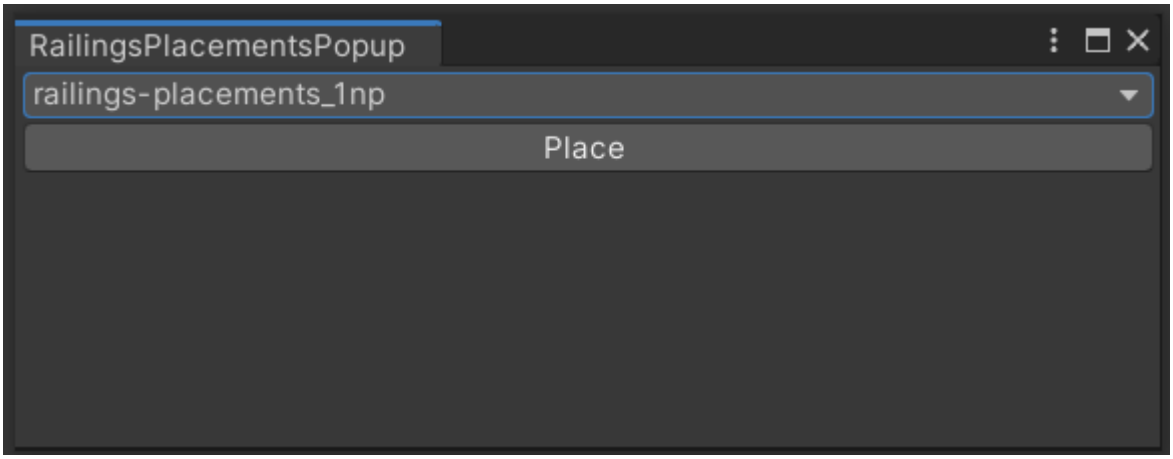
```
  /// <summary>
  /// Generates the popup options
  /// </summary>
  private static void GenerateOptions()
  {
    options = new string[objectsPlacementsInfo.Length];
    optionNames = new string[objectsPlacementsInfo.Length];

    for (int i = 0; i < objectsPlacementsInfo.Length; i++)
    {
      options[i] = objectsPlacementsInfo[i].Name;
      optionNames[i] = objectsPlacementsInfo[i].Name.Replace(".json", "");
    }
  }

  void OnGUI()
  {
    index = EditorGUILayout.Popup(index, optionNames);
    if (GUILayout.Button("Place"))
    {
      SceneBuilder.PlaceAssets(positionsFolderPath + options[index], objectsPaths);
    }
  }
}
```

Zdrojový kód 2: Třída WindowPopupBase. Kód vytvořen podle [21, s. EditorGUILayout.Popup] a upraven autorem.

Vytvořené možnosti po jejich výběru zobrazí nové okno obsahující rozbalovací nabídku pro výběr podlaží a tlačítko s nápisem Place, kterým se vykoná operace rozmístění prefabů. Tyto prvky jsou znázorněny na obrázcích níže.



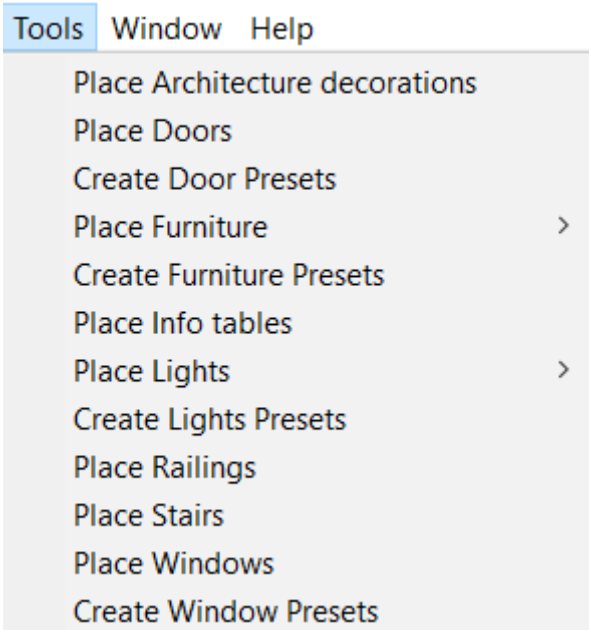
Obrázek 18: Hlavní lišta nástrojů s přidanou položkou menu. Zdroj: Autor



Obrázek 19: Ukázka vyskakovacího okna pro umístění zábradlí. Zdroj: Autor

Tento způsob se projevil jako značně efektivní a snadno rozšiřitelný. Pomocí pár kliknutí bylo tímto způsobem rozmístěno všechno zábradlí na všech patrech společně se všemi schody. Zároveň však bylo zajištěno stejně rychlé aktualizování případných změn těchto pozic.

Jak se později ukázalo, tyto skripty jsou využitelné také pro rychlé sestavení předepsaných skupin modelů oken, nábytku, světel a dveří, jejichž definice byly též již dříve uloženy do souborů ve formátu JSON.



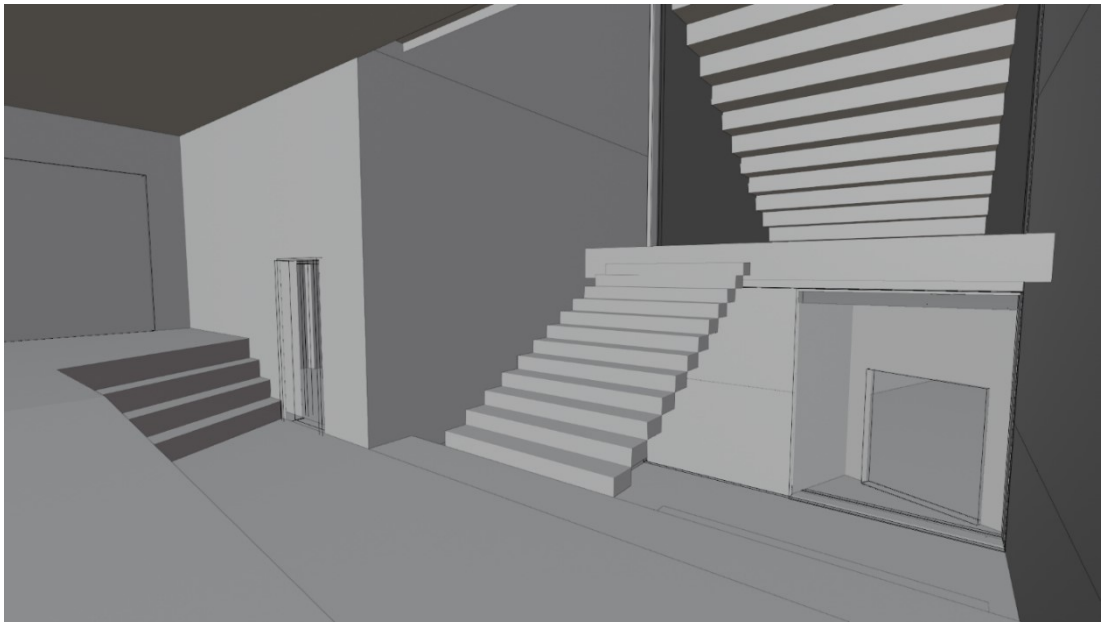
Obrázek 20: Konečné vytvořené menu v editoru. Zdroj: Autor

Z takto vytvořených skupin objektů byly následně vytvořeny prefaby stejným způsobem jako v případě objektů, které je tvoří. Jejich rozmístění bylo společně s informačními tabulemi následně provedeno identickým způsobem jako u schodů a zábradlí.

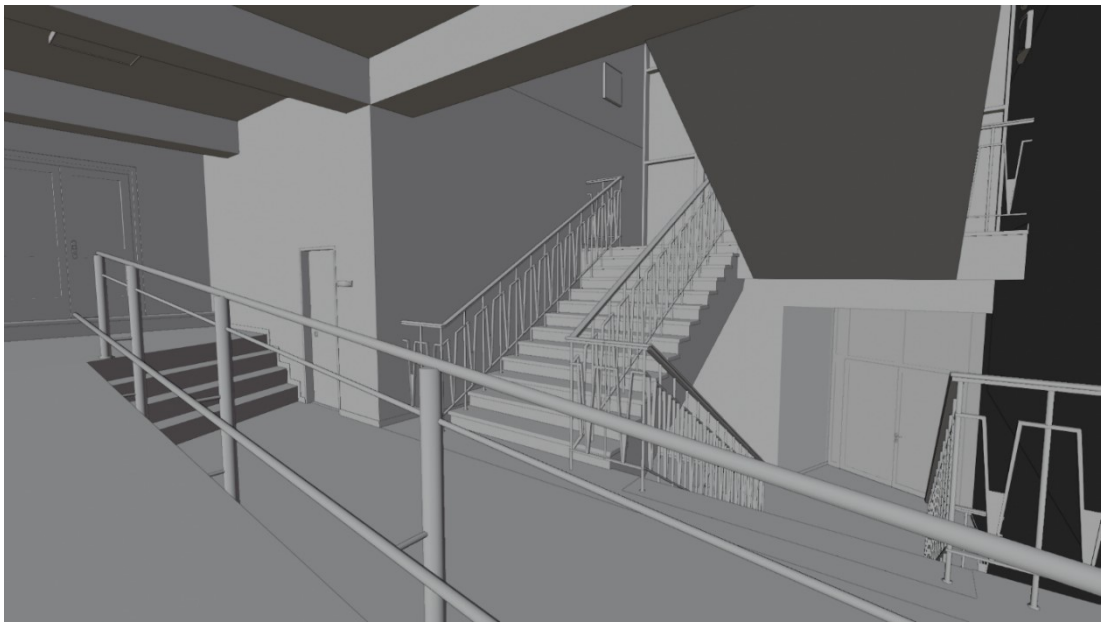
Po sestavení modelu budovy a rozmístění jejího zařízení byly do scény přidány modely pro zem, po které se uživatel může pohybovat v exteriéru budovy. Dále byly rozmístěny primitivní vytvořené modely budov a pomocí výše zmíněných kolizních komponent vytvořeny neviditelné bariéry pro ohraničení přístupné oblasti. Pro dotvoření vizuálního rázu scény bylo poté do scény umístěno několik stromů z Unity Asset Store ([27]).

Pro dosažení lepšího výkonu za běhu aplikace byl ve scéně využit occlusion culling. Jak je uvedeno v dokumentaci, při použití occlusion culling nejsou na objektech plně v pohledu zakrytých za jinými objekty prováděny výpočty pro vykreslováni. Jinými slovy jsou z vykreslování vynechány objekty, které není potřeba vykreslovat. Nejlépe funguje v malých, jasně oddělených oblastech jako jsou například místnosti. O scéně jsou zapečena data, která jsou poté načtena do paměti a podle nich se rozhoduje, zda příslušná kamera dané objekty vidí. Pro lepší kontrolu nad tímto procesem lze použít tzv. Occlusion Areas [12, s. Occlusion culling].

Komponentou typu Occlusion Area lze označit oblasti, ve kterých se kamera bude za běhu nejspíše nacházet. Při zapékání dat jsou v těchto oblastech prováděny přesnější výpočty [12, s. Occlusion Areas]. Objekty s těmito komponentami tudíž byly rozmístěny do jednotlivých místností. Pro lepší výsledky byly následně na okna a dveře přidány komponenty typu Occlusion Portal. V dokumentaci je uvedeno, že komponenta Occlusion Portal může být otevřená a nezakrývat ostatní objekty, či zavřená a ostatní objekty zakrývat. Lze je použít na objekty, které mohou být otevřeny a zavřeny a dle stavu tohoto objektu lze upravovat stav této komponenty [12, s. Occlusion Portals]. Na obrázcích níže je vizuálně shrnut celý proces tvorby výsledného modelu a samotné scény.



Obrázek 21: Část hrubého konceptu modelu. Zdroj: Autor



Obrázek 22: Část dokončeného modelu bez materiálů. Zdroj: Autor

Obrázek 23: Část sestavené scény v Unity. Zdroj: Autor

V tuto chvíli již byl sestaven základ celé scény. Stále však nebylo možné se v ní pohybovat, zobrazovat informace pomocí HUD, ani vyvolávat na dveřích a informačních tabulích příslušné akce reagující na vstup uživatele. Implementace těchto systémů jsou popsány v následujících částech.

7.3 Implementace ovládání a informačního systému

Pro reprezentaci avataru hráče byl vytvořen prázdný objekt s komponentou Character Controller. Jak uvádí Hocking, s komponentou Character Controller se může objekt chovat jako postava [19, s. 33]. K tomuto objektu byla následně přiřazena kamera jako potomek. Pro snazší definici možné vzdálenosti pro interakce byly této kameře jako potomci přidány dva prázdné objekty určující počáteční a koncové souřadnice pro kontrolu kolizí s interaktivními objekty.

Nejprve byla implementována logika umožňující otáčet s kamerou a tím se rozhlížet ve scéně. Zároveň tato logika zabraňuje otáčet s kamerou ve vertikálním směru nad stanovenou mez pro snazší ovládání. Následující kód ukazuje konkrétní finální implementaci této funkcionality.

```
using UnityEngine;

/// <summary>
/// A script for controlling the camera
/// </summary>
public class MouseLook : MonoBehaviour
{
  public enum RotationAxes
  {
    MouseXAndY = 0,
    MouseX = 1,
    MouseY = 2
  }

  public RotationAxes axes = RotationAxes.MouseXAndY;

  public float verticalSensitivity = 9.0f;
  public float horizontalSensitivity = 9.0f;

  public float verticalMinimum = -45.0f;
  public float verticalMaximum = 45.0f;

  private float _rotationX = 0;

  private bool isInMenu = false;

  void Awake()
  {
    Messenger.AddListener(UIEvent.MENU_OPENED, OnMenuOpen);
    Messenger.AddListener(UIEvent.MENU_CLOSED, OnMenuClose);
  }

  void OnDestroy()
  {
    Messenger.RemoveListener(UIEvent.MENU_OPENED, OnMenuOpen);
    Messenger.RemoveListener(UIEvent.MENU_CLOSED, OnMenuClose);
  }

  void Update()
  {
    if (!isInMenu) // prevent camera rotation while paused
    {
      if (axes == RotationAxes.MouseX)
      {
        transform.Rotate(0, Input.GetAxis("Mouse X") * horizontalSensitivity, 0);
      }
```

```
      else if (axes == RotationAxes.MouseY)
      {
        _rotationX -= Input.GetAxis("Mouse Y") * verticalSensitivity;
        _rotationX = Mathf.Clamp(_rotationX, verticalMinimum, verticalMaximum);

        float rotationY = transform.localEulerAngles.y;

        transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0);
      }
      else
      {
        _rotationX -= Input.GetAxis("Mouse Y") * verticalSensitivity;
        _rotationX = Mathf.Clamp(_rotationX, verticalMinimum, verticalMaximum);

        float delta = Input.GetAxis("Mouse X") * horizontalSensitivity;
        float rotationY = transform.localEulerAngles.y + delta;

        transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0);
      }
    }

  }

  private void OnMenuOpen()
  {
    isInMenu = true;
  }

  private void OnMenuClose()
  {
    isInMenu = false;
  }
}
```

Zdrojový kód 3: Celý skript MouseLook. Kód převzat z: [19, s. 43] a upraven autorem.

Pro ovládání hráčova avataru v prostoru byl následně vytvořen skript FPSInput, který implementuje požadavky stanovené v kapitole 6.2. Finální implementace této funkcionality je uvedena v části kódu níže.

```
using UnityEngine;

/// <summary>
/// A script for character movement
/// </summary>
[RequireComponent(typeof(CharacterController))]
[AddComponentMenu("Control Script/FPS Input")]
public class FPSInput : MonoBehaviour
{
  public float speed = 6.0f;
  public float gravity = -9.8f;
  private CharacterController _characterController;
  private bool isInMenu = false;

  void Awake()
  {
    Messenger.AddListener(UIEvent.MENU_OPENED, OnMenuOpen);
    Messenger.AddListener(UIEvent.MENU_CLOSED, OnMenuClose);
  }

  void OnDestroy()
  {
    Messenger.RemoveListener(UIEvent.MENU_OPENED, OnMenuOpen);
```

```
    Messenger.RemoveListener(UIEvent.MENU_CLOSED, OnMenuClose);
  }

  void Start()
  {
    _characterController = GetComponent<CharacterController>();
  }

  void Update()
  {
    if (!isInMenu) // prevent movement while paused
    {
      float deltaX = Input.GetAxis("Horizontal") * speed;
      float deltaZ = Input.GetAxis("Vertical") * speed;
      Vector3 movement = new Vector3(deltaX, 0, deltaZ);
      movement = Vector3.ClampMagnitude(movement, speed);

      movement.y = gravity;

      movement *= Time.deltaTime;
      movement = transform.TransformDirection(movement);
      _characterController.Move(movement);
    }
  }

  private void OnMenuOpen()
  {
    isInMenu = true;
  }

  private void OnMenuClose()
  {
    isInMenu = false;
  }
}
```

Zdrojový kód 4: Celý skript FPSInput. Kód převzat z: [19, s. 48–49] a upraven autorem.

Po implementaci základního ovládání přišlo na řadu implementovat zbylé požadavky na interakci ve scéně. Objekty dveří a informačních tabulí musí být schopné reagovat na příkazy uživatele a tento požadavek bylo možné přidáním daných scriptů do prefabů naplnit bez nutnosti opětovného rozmístění těchto objektů pro implementaci těchto funkcí.Před samotnou implementací těchto funkcí byl vytvořen skript pro hráčův avatar, který zkoumá možnosti interakcí metodou Physics.OverlapCapsule s pomocí výše zmíněných pomocných objektů připojených ke kameře. Tento skript při detekci kolize umožní provést požadovanou akci stiskem tlačítka. Pro filtrování interaktivních objektů byly použity vrstvy. Jak je uvedeno v dokumentaci, vrstvami lze možnosti interakcí daných objektů [12, s. Layers]. Z dokumentace API dále vyplývá, že pomocí vrstev lze určit které kolizní objekty mají být vytvářenou kapslí ignorovány [21, s. Physics.OverlapCapsule]. Konečná implementace tohoto skriptu je uvedena v kódu níže.

```
using UnityEngine;

/// <summary>
/// A script to allow
/// the player to interact
```

```
/// with objects
/// </summary>
public class DeviceOperator : MonoBehaviour
{
  public float radius = 0.5f;
  [SerializeField] private LayerMask doorLayerMask;
  [SerializeField] private LayerMask infoTableNameLayerMask;
  [SerializeField] private LayerMask infoTableTimetableLayerMask;
  [SerializeField] private GameObject collisionCapsuleStart;
  [SerializeField] private GameObject collisionCapsuleEnd;
  private bool isInMenu = false;

  void Awake()
  {
    Messenger.AddListener(UIEvent.MENU_OPENED, OnMenuOpen);
    Messenger.AddListener(UIEvent.MENU_CLOSED, OnMenuClose);
  }

  void OnDestroy()
  {
    Messenger.RemoveListener(UIEvent.MENU_OPENED, OnMenuOpen);
    Messenger.RemoveListener(UIEvent.MENU_CLOSED, OnMenuClose);
  }

  void Update()
  {
    if (!isInMenu) // prevent interactionw while in menu
    {
      if (Input.GetKeyUp(KeyCode.P)) // opening the main menu
      {
        Messenger.Broadcast(UIEvent.MAIN_MENU_OPENED);
        Messenger.Broadcast(UIEvent.MENU_OPENED);
      }

      HandleGeneralObjectOperations();
      HandleTimetableOperations();
    }
  }

  /// <summary>
  /// Handles the interactions with
  /// doors and name info tables
  /// </summary>
  private void HandleGeneralObjectOperations()
  {
    Collider[] hitColliderDoors =
Physics.OverlapCapsule(collisionCapsuleStart.transform.position,
collisionCapsuleEnd.transform.position, radius, doorLayerMask);

    Collider[] hitColliderInfoTablesNames =
Physics.OverlapCapsule(collisionCapsuleStart.transform.position,
collisionCapsuleEnd.transform.position, radius, infoTableNameLayerMask);

    if (hitColliderDoors.Length > 0) // can interact with a door
    {
      Messenger<string>.Broadcast(UIEvent.DOOR_APPROACHED, "Použít dveře");

      if (Input.GetKeyDown(KeyCode.E))
      {
        hitColliderDoors[0].SendMessage("Operate", SendMessageOptions.DontRequireReceiver);
      }
    }
    else if (hitColliderInfoTablesNames.Length > 0) // can interact with a name info table
    {
      Messenger<string>.Broadcast(UIEvent.INFO_TABLE_NAME_APPROACHED, "Zobrazit informace o
místnosti");
```

```
      if (Input.GetKeyDown(KeyCode.E))
      {
        hitColliderInfoTablesNames[0].SendMessage("Operate",
SendMessageOptions.DontRequireReceiver);
      }
    }
    else if (hitColliderDoors.Length <= 0)
    {
      Messenger.Broadcast(UIEvent.DOOR_LEAVE);
    }
    else if (hitColliderInfoTablesNames.Length <= 0)
    {
      Messenger.Broadcast(UIEvent.INFO_TABLE_NAME_LEAVE);
    }
  }

  /// <summary>
  /// Handles the interactions with
  /// timetable info tables
  /// </summary>
  private void HandleTimetableOperations()
  {
    Collider[] hitColliderInfoTablesTimetables =
Physics.OverlapCapsule(collisionCapsuleStart.transform.position,
collisionCapsuleEnd.transform.position, radius, infoTableTimetableLayerMask);

    if (hitColliderInfoTablesTimetables.Length > 0) // can interact with a timetable info
table
    {
      Messenger<string>.Broadcast(UIEvent.INFO_TABLE_TIMETABLE_APPROACHED, "Zobrazit rozvrh
místnosti");

      if (Input.GetKeyDown(KeyCode.Q))
      {
        hitColliderInfoTablesTimetables[0].SendMessage("Operate",
SendMessageOptions.DontRequireReceiver);
      }

    }
    else if (hitColliderInfoTablesTimetables.Length <= 0)
    {
      Messenger.Broadcast(UIEvent.INFO_TABLE_TIMETABLE_LEAVE);
    }
  }

  private void OnMenuOpen()
  {
    isInMenu = true;
  }

  private void OnMenuClose()
  {
    isInMenu = false;
  }
}
```

Zdrojový kód 5: Celý skript DeviceOperator. Kód vytvořen podle [19, s. 200] a upraven autorem.

Pro funkci otevírání dveří byly vytvořeny animace otevření a zavření dveří, které jsou aktivovány spouštěčem (trigger), který je vyvolán v přiřazeném skriptu. Pro vybrané kliky byly též

vytvořeny animace pomocí stejného způsobu. K tvorbě těchto animací byl využit systém Mecanim. Jak uvádí Hocking, Mecanim je sofistikovaný systém pro správu animací modelů v Unity, který nahrazuje starší animační systém [19, s. 189].

Animovatelný model má komponentu Animator, kterému lze do slotu Controller vložit konkrétní animator controller. Animator controller je strom uzlů, ve kterém lze nastavit animační stavy a přechody mezi nimi. Lze jej editovat v okně Animator, které lze otevřít kliknutím na možnost Window > Animator [19, s. 192]. Uzel, ve kterém tato síť začíná předtím, než hra provede změny se označuje jako výchozí uzel a má oranžovou barvu. Uzel lze nastavit jako výchozí zvolením možnosti Set As Layer Default State v nabídce vyvolané pravým tlačítkem myši. Když se na uzel klikne pravým tlačítkem myši, zobrazí se kontextová nabídka, ve které je možnost Make Transition. Zvolením této možnosti lze následně táhnout šipku a kliknout na uzel, ke kterému má být propojena a tím vytvořit přechod mezi těmito stavy. Nastavení těchto přechodů lze v záložce Inspector zobrazit kliknutím na příslušné čáry. Přechody jsou závislé na kontrolních hodnotách, které lze vytvářet stiskem tlačítka + v panelu Parameters. V nastavení přechodu je lze přiřadit pod Conditions v sekci Settings [19, s. 193–194]. V kódu následně stačí získat referenci na kopmonentu Animator, které lze poté nastavovat hodnoty [19, s. 196].

Vzhledem ke skutečnosti, že ne všechny dveře mají poskytovat možnost je otevřít byl do skriptu na jejich ovládání přidán atribut, který určuje, zda jsou dané dveře zamčené a ve výchozím nastavení je nastaven na hodnotu značící zamčení. Pro místnosti, do kterých lze vstoupit následně stačilo pouze tyto dveře odemknout. Dále byly přidány atributy pro komponenty Animator příslušných klik, aby mohly být jejich animace při pokusu o otevření přehrány.

```
using System.Collections;
using UnityEngine;

/// <summary>
/// A script to react to the user input
/// for interaction with doors
/// </summary>
[RequireComponent(typeof(Animator))]
public class DoorOpenDevice : MonoBehaviour
{
  [SerializeField] private bool isLocked = true;
  [SerializeField] private Animator _childAnimationHolder;
  [SerializeField] private Animator _secondChildAnimationHolder;
  [SerializeField] private OcclusionPortal _occlusionPortal;
  private Animator _animationHolder;
  private bool _isPortalClosing = false;
  private bool _cancelPortalClose = false;
  private float _closeAfterSeconds = 1f;

  private void Start()
  {
    _animationHolder = GetComponent<Animator>();
  }
```

```
  /// <summary>
  /// Executes the desired action
  /// </summary>
  public void Operate()
  {
    AnimateDoorHandles();

    if (!isLocked)
    {
      _animationHolder.SetTrigger("TriggerOpenState");

      if (_occlusionPortal != null)
      {
        if (!_occlusionPortal.open)
        {
          _occlusionPortal.open = true;
        }
        else if (!_isPortalClosing) // don't change the portal state while not done
        {
          _isPortalClosing = true;
          StartCoroutine(ClosePortalAfterSeconds(_closeAfterSeconds));
        }
        else
        {
          _cancelPortalClose = true; // to handle rapid opening and closing
        }
      }
    }
    else
    {
      Messenger<string>.Broadcast(UIEvent.DOOR_LOCKED_OPEN, "Zamčeno");
    }
  }

  /// <summary>
  /// A Coroutine to hide portal closing
  /// </summary>
  /// <param name="seconds">how many seconds to delay for</param>
  /// <returns>IEnumerator as usual for coroutines</returns>
  private IEnumerator ClosePortalAfterSeconds(float seconds)
  {
    yield return new WaitForSeconds(seconds);
    if (!_cancelPortalClose)
    {
      _occlusionPortal.open = false;
    }
    _isPortalClosing = false;
  }

  /// <summary>
  /// Plays the animations of door handles if there are any
  /// </summary>
  private void AnimateDoorHandles()
  {
    if (_childAnimationHolder != null)
      _childAnimationHolder?.SetTrigger("TriggerOpenState");
    if (_secondChildAnimationHolder != null)
      _secondChildAnimationHolder?.SetTrigger("TriggerOpenState");
  }
}
```

Zdrojový kód 6: Celý skript DoorOpenDevice. Kód vytvořen podle [19, s. 199] a upraven autorem.

Data zobrazovaná po interakci s informačními tabulemi byla v obou případech získána z veřejně přístupných částí API STAG web services ([28], [29]) ve formátu XML pro snadnou práci s těmito daty. Jak uvádí Hocking, funkce pro parsování XML je zahrnuta přímo ve frameworku Mono, který je vestavěn do Unity [19, s. 238]. Tato data však při konečném sestavení nebylo možné získávat přímo v aplikaci pomocí dotazů (requests). Jak je uvedeno v dokumentaci, ve WebGL je pro WWW dotazy používán prohlížeč, jelikož třída UnityWebRequest je implementována v JavaScriptu třídou XMLHttpRequest. Přístup ke zdrojům v jiné doméně tudíž musí daný vzdálený server schválit pomocí CORS, jinak konzole v prohlížeči zobrazí chybovou hlášku a přístup k těmto datům nebude WebGL umožněn [12, s. WebGL Networking]. Veškerá potřebná data tudíž byla uložena do souborů v projektu. Pro snazší práci s těmito daty v samotné aplikaci došlo v případě rozvrhových akcí k ručnímu sloučení těchto dat do jednoho souboru.

Pro správu základních informací o místnostech byla vytvořena třída RoomInfo, která ve svých instancích umožňuje udržovat informace o jméně místnosti, kapacitě, typu místnosti a pracovišti. Tuto třídu v konečné implementaci využívaly i místnosti, o kterých nebyla k dispozici žádná data v API. V těchto případech bylo doplněno pouze jméno dle zjevného účelu či informací z plánů.

Instance třídy RoomTimeTable uchovávají informace o roku, semestru. Pro každý den v týdnu drží navíc seznam instancí třídy RoomTimetableAction. V každém z těchto seznamů jsou tyto instance seřazeny chronologicky dle času konání. Ve třídě RoomTimetableAction jsou evidovány atributy pro název akce, oddělení, předmět, typ rozvrhové akce, den, číselné ohraničení trvání akce a časové ohraničení trvání akce.

Po spuštění aplikace jsou instance tříd RoomInfo a RoomTimetable vytvořené na základě dat z uložených XML souborů. Tyto instance jsou následně načteny do slovníku ve třídě RoomInfoSystem pro snadný přístup k těmto datům pomocí jména příslušné místnosti. Informace o místnosti jsou takto zpřístupňovány metodou GetRoomInfo přijímající v parametru jméno místnosti. Rozvrh místnosti lze získat podobným způsobem zavoláním metody GetRoomTimetable. Skript RoomInfoSystem byl následně připojen na prázdný GameObject ve scéně, aby jeho metody mohly být dle potřeby volány při interakci s informačními tabulemi.

Jak se ukázalo, systém pro udržování informací o patře a místnosti, ve které se uživatel nachází bylo možné implementovat pomocí dvojic spouštěčů umístěných na pozice všech dveří a schodišť. Objekty s těmito spouštěči v sobě drží příslušné informace o jméně místnosti či patra a při jejich aktivaci obeznámí HUD, že se změnila současná místnost či patro.

7.4 Implementace HUD prvků

Jak uvádí Hocking, o událostech ve scéně lze UI informovat přes broadcast messenger system, ve kterém se mohou skripty registrovat pro naslouchání události, které jiný kód vysílá. Narozdíl od vestavěného systému událostí v C# nevyžaduje tento systém, aby kód znal původce zpráv. Tento systém není v Unity vestavěn. Proto je použit systém z Unity Community Wiki ([30]), který Hocking ve své knize pro tyto účely uvádí [19, s. 164–165]. Konkrétně je použit kód sekce Messenger.cs, který bylo nutné do projektu vložit jako nový skript. Pro lepší správu událostí byla vytvořena třída UIEvent uvedená níže.

```
/// <summary>
/// A helper class to manage
/// the events for the message system
/// </summary>
public static class UIEvent
{
  public const string DOOR_APPROACHED = "DOOR_APPROACHED";
  public const string DOOR_LEAVE = "DOOR_LEAVE";
  public const string DOOR_LOCKED_OPEN = "DOOR_LOCKED_OPEN";
  public const string ROOM_CHANGED = "ROOM_CHANGED";
  public const string FLOOR_CHANGED = "FLOOR_CHANGED";
  public const string INFO_TABLE_NAME_APPROACHED = "INFO_TABLE_NAME_APPROACHED";
  public const string INFO_TABLE_NAME_LEAVE = "INFO_TABLE_NAME_LEAVE";
  public const string INFO_TABLE_TIMETABLE_APPROACHED = "INFO_TABLE_TIMETABLE_APPROACHED";
  public const string INFO_TABLE_TIMETABLE_LEAVE = "INFO_TABLE_TIMETABLE_LEAVE";
  public const string INFO_TABLE_NAME_ACTIVATED = "INFO_TABLE_NAME_ACTIVATED";
  public const string INFO_TABLE_NAME_DEACTIVATED = "INFO_TABLE_NAME_DEACTIVATED";
  public const string INFO_TABLE_TIMETABLE_ACTIVATED = "INFO_TABLE_TIMETABLE_ACTIVATED";
  public const string INFO_TABLE_TIMETABLE_DEACTIVATED =
"INFO_TABLE_TIMETABLE_DEACTIVATED";
  public const string MENU_OPENED = "MENU_OPENED";
  public const string MENU_CLOSED = "MENU_CLOSED";
  public const string MAIN_MENU_OPENED = "MAIN_MENU_OPENED";
  public const string CONTROLS_MENU_OPENED = "CONTROLS_MENU_OPENED";
  public const string FONT_LICENCE_MENU_OPENED = "FONT_LICENCE_MENU_OPENED";
  public const string SUB_MENU_CLOSED = "SUB_MENU_CLOSED";
}
```

Zdrojový kód 7: Třída UIEvent. Kód vytvořen podle [19, s. 165] a upraven autorem.

Po integraci systému předávání zpráv byl implementován skript pro ovládání prvků UI reagující na tyto události. Pro příslušné prvky UI byly následně vytvořeny skripty pro jejich správu, jejichž metody jsou následně v hlavním skriptu při daných událostech volány. Tímto způsobem je dosaženo snazší správy jednotlivých funkcionalit.

Jak bylo nastíněno v kapitole 4.5, veškeré prvky UI jsou umístěny jako potomci objektu typu Canvas. Pro lepší organizaci byly v rámci této hierarchie prvky pro okna informačních tabulí a HUD umístěny do samostatných hierarchií. Na základě jednotného grafického stylu zmíněného v kapitole 6.3 byl pro většinu textu v UI použit font Gill Sans ([31]), ze kterého byly dle postupu popsaného v kapitole 4.5 vytvořeny assety pro normální a tučnou obdobu textu.

Zatímco tvorba většiny HUD prvků spočívala pouze v jejich umístění v objektu typu Canvas a připojení skriptů, v případě minimapy bylo nutné dosáhnout následování pozice hráče a určit grafickou podobu zobrazovaných informací. Jak uvádí Polidario, v případě většiny minimap jsou použity dvě kamery, přičemž ta druhá je v ortografickém režimu a bývá nad hráčem namířena směrem k němu. Do místa minimapy lze pak umístit objekt typu Raw Image, který může přijímat vykreslovací textury (render textures). Tento typ textur je za běhu vytvářen a aktualizován podle výstupu kamery, které je tato textura přiřazena v atributu Target Texture [32]. Tato kamera pro minimapu byla následně připojena k objektu hráče kvůli aktualizace její pozice.

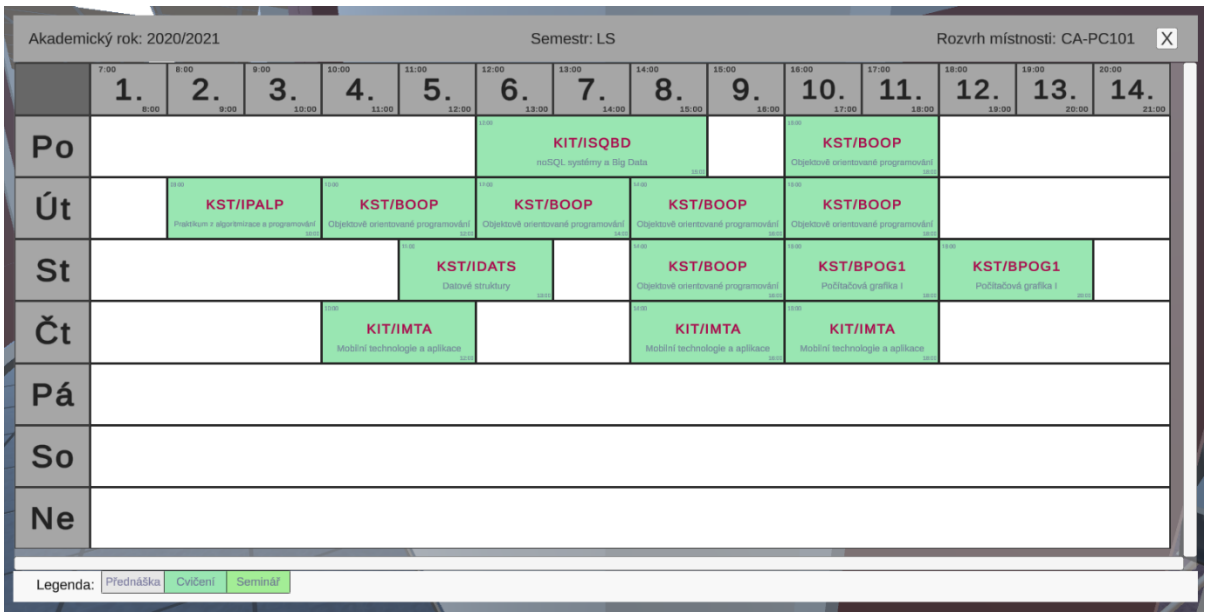
Pro dosažení stylizovaného vzhledu minimapy byly v Blenderu pomocí vykreslovacího režimu Wireframe vytvořeny zjednodušené pohledy shora pro jednotlivá patra, které byly následně vyčištěny a obarveny dle barev fakulty. Tyto textury byly následně aplikovány pomocí materiálů na čtyřúhelníky umístěných ve výšce odpovídajících pater. Pro popisky názvů přístupných místností byly následně ve scéně rozmístěny objekty typu TextMesh Pro a k avataru hráče připojen objekt sféry pro reprezentaci jeho pozice na minimapě. Veškerým těmto objektům pro minimapu byla poté nastavena nová vrstva MinimapInfo, která byla příslušné kameře nastavena jako jediná viditelná. Výsledný vzhled HUD je znázorněn na následujícím obrázku.

Obrázek 24: Ukázka implementovaného HUD. Zdroj: Autor

Jak bylo zmíněno v kapitole 6.3 grafická reprezentace tabulek s informacemi o místnostech byla silně založena na pořízených fotografiích. I zde je použit výše popsaný font a pro barvu záhlaví a zápatí byla zvolena barva této fakulty. Zavřít tento prvek je po otevření možné příslušným tlačítkem. Výsledný vzhled je znázorněn na obrázku níže.

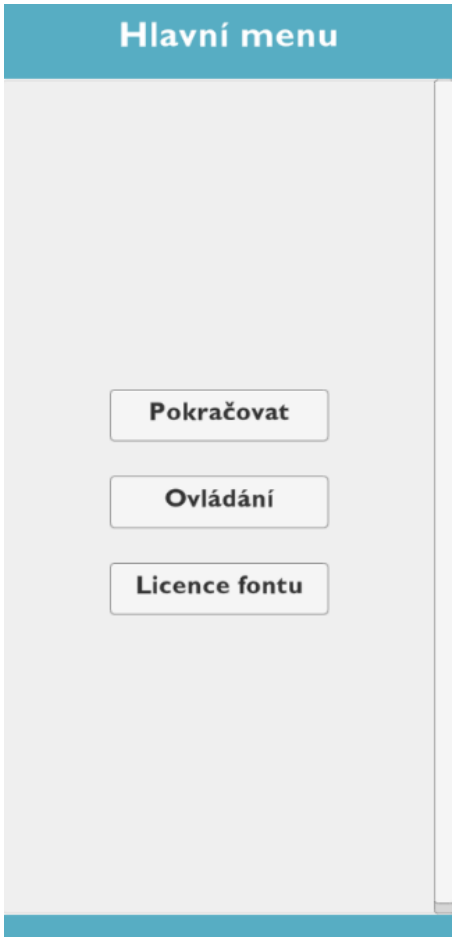
Obrázek 25: Ukázka UI informační tabule. Zdroj: Autor

Jelikož se informace o rozvrhu značně mezi místnostmi liší, bylo nutné sestavit dostatečně flexibilní grafickou reprezentaci, kterou je možné dynamicky naplnit prvky. Za tímto účelem byl kromě horizontálních a vertikálních skupin vytvořen také prefab reprezentující buňku pro rozvrhovou akci. Tato buňka je poté podle rozvrhu dané místnosti do této struktury instancována a dle typu rozvrhové akce a délky je upravována její barva a délka. Tímto způsobem je možné věrně graficky reprezentovat rozvrhové akce dynamicky dle vstupních dat. I v tomto případě lze tento prvek zavřít příslušným tlačítkem. Výsledná podoba je uvedena na obrázku níže.



Obrázek 26: Ukázka UI tabule s rozvrhem. Zdroj: Autor

Nakonec bylo implementováno hlavní menu, které uživateli umožňuje blíže se seznámit s ovládáním této aplikace. Pro snazší prvotní přístup k tomuto menu bylo nastaveno, že se zobrazí ihned při startu aplikace a po jeho zavření jej bude možné opět vyvolat stiskem klávesy P během pohybu ve scéně. Výsledný vzhled tohoto menu je popsán na obrázku níže.



Obrázek 27: Hlavní menu. Zdroj: Autor

7.5 Sestavení aplikace na platformu WebGL

V tuto chvíli je po funkční stránce aplikace hotová a zbývá provést její sestavení. Jak uvádí Hocking, pro otevření okna Build Settings je potřeba vybrat možnost Build Settings pod položkou File v hlavní liště. Po jeho otevření se zobrazí seznam všech platforem, na které je možné projekt sestavit. U aktivní platformy je ikonka Unity. [19, s. 308]

Jak bylo avizováno v předešlých kapitolách, sestavení tohoto projektu proběhne na platformu WebGL kvůli možnosti spuštění ve webovém prohlížeči. Jak uvádí Hocking, v minulosti nebyla v prohlížečích vestavěná podpora 3D grafiky a projekty sestavené pro web musely být

spouštěny ve speciálním pluginu. Během posledních let se však WebGL ujalo jako standard 3D grafiky pro web. Podpora WebGL byla přidána v Unity 5 a dnes se jedná o jediný způsob tvorby sestavených projektů pro web, jelikož plugin byl o pár verzí později odstraněn. [19, s. 314]

Po sestavení projektu jej je možné spustit v prohlížeči bez nutnosti přístupu k editoru Unity. Jak uvádí Hocking, výstupem sestavení je složka obsahující soubor index.html a podsložky se všemi kódy a assety [19, s. 315]. Je pravděpodobné, že prohlížeč neumožní spuštění sestaveného projektu na platformu WebGL přímo z uvedeného souboru. Řešením tohoto problému je umístit je na vzdálený server, či jej spustit na lokálním serveru.

Obrázek 28: Sestavený projekt v prohlížeči. Zdroj: Autor

ZÁVĚR

Cílem této práce bylo vytvořit aplikaci umožňující interaktivní prohlídku modelu Fakulty elektrotechniky a informatiky Univerzity Pardubice a zobrazovat na obrazovce dodatečné informace o jednotlivých místnostech, která velice dobře poslouží v orientaci studentům prvních ročníků. Lze konstatovat, že tento cíl byl splněn a díky zvolené platformě WebGL může být také umožněna prohlídka ostatním uživatelům pomocí webového prohlížeče. Volba této platformy však zároveň přinesla komplikace v podobě určitých omezení v oblasti implementace logiky informačního systému.

Vlivem těchto omezení jsou zobrazované informace o rozvrhových akcích místností pevně uloženy. Informace by však bylo možné získávat dynamicky z API STAG web services pomocí proxy serveru, který by měl pro tuto aplikaci řádně nakonfigurované hlavičky pro splnění CORS policy prohlížeče.

Dále je možné aplikaci dále rozšířit o další učebny, které zde byly vynechány. Přidání těchto místností a jejich vybavení by vzhledem k vytvořené struktuře a pomocným skriptům nemělo představovat mnoho obtíží, jelikož struktura projektu se projevila jako přehledá a flexibilní. Vytvořené skripty se též projevily jako značná výhoda, jelikož mnoho časově náročných úkonů s nimi bylo možné provést velmi rychle.

Aplikace zvolené pro tvorbu této práce se též projevily jako dobrá volba. Díky volně přístupným dokumentacím pro Blender a herní engine Unity bylo možné vcelku rychle nacházet řešení na případné problémy při vývoji aplikace. Jako možnou nevýhodu volby těchto aplikací lze však zmínit znatelnou absenci publikací vydaných v českém jazyce pro relevantní verze těchto aplikací. Další nevýhodou této kombinace je skutečnost, že obě aplikace používají jiný souřadnicový systém a neshodují se v tom, která osa slouží pro vertikální směr. Tento nedostatek však byl značně eliminován při importu objektů mezi aplikacemi.

POUŽITÁ LITERATURA

[1] WARD, Jeff. What is a Game Engine? *GameCareerGuide* [online]. 29. 04. 2008 [cit. 18. 02. 2021]. Dostupné z: http://www.gamecareerguide.com/features/529/what_is_a_game_.php

[2] CHARBONNEAU, Miko. Choosing the Perfect Game Engine. *Gamasutra* [online]. 22. 02. 2013 [cit. 16. 09. 2020]. Dostupné z: https://www.gamasutra.com/blogs/Miko-Charbonneau/20130222/187185/Choosing_the_Perfect_Game_Engine.php

[3] Epic Games. Unreal Engine | Features. *Unreal Engine* [online]. ©2004–2021 [cit. 02. 04. 2021]. Dostupné z: http://www.unrealengine.com/features

[4] Epic Games. HTML5 Game Development. *Unreal Engine Documentation* [online]. ©2004–2021 [cit. 02. 04. 2021]. Dostupné z: https://docs.unrealengine.com/en-US/SharingAndReleasing/HTML5/index.html

[5] Epic Games. Unreal Engine | Frequently Asked Questions. *Unreal Engine* [online]. ©2004–2021 [cit. 22. 02. 2021]. Dostupné z: https://www.unrealengine.com/en-US/faq

[6] Crytek. Script Usage - Technical Documentation - Documentation. *CRYENGINE V Manual* [online]. 21. 06. 2013 [cit. 03. 04. 2021]. Dostupné z: https://docs.cryengine.com/display/SDKDOC4/Script+Usage

[7] Crytek. CRYENGINE Programming - CRYENGINE Programming - Documentation. *CRYENGINE V Manual* [online]. 27. 05. 2020 [cit. 03. 04. 2021]. Dostupné z: https://docs.cryengine.com/display/CEPROG/CRYENGINE+Programming

[8] Crytek. CRYENGINE | Features: Sandbox. *CRYENGINE* [online]. ©2021 [cit. 22. 02. 2021]. Dostupné z: https://www.cryengine.com/features/view/sandbox

[9] Crytek. CRYENGINE | Support: Licensing. *CRYENGINE* [online]. ©2021 [cit. 22. 02. 2021]. Dostupné z: https://www.cryengine.com/support/view/licensing

[10] Crytek. CRYENGINE | Support: General. *CRYENGINE* [online]. ©2021 [cit. 03. 04. 2021]. Dostupné z: https://www.cryengine.com/support/view/general

[11] Unity Technologies. 2D 3D Game Creator & Editor | Augmented / Virtual Reality Software | Game Engine | Unity. *Unity* [online]. ©2021 [cit. 20. 02. 2021]. Dostupné z: https://unity.com/products/unity-platform

[12] Unity Technologies. Unity - Manual: Unity User Manual (2019.4 LTS). *Unity3D* [online]. 25. 04. 2021 [cit. 26. 04. 2021]. Dostupné z: https://docs.unity3d.com/2019.4/Documentation/Manual/index.html

[13] Unity Technologies. Powerful 2D, 3D, VR, & AR software for cross-platform development of games and mobile apps. *Unity Store* [online]. ©2021 [cit. 26. 02. 2021]. Dostupné z: https://store.unity.com/

[14] Unity Technologies. Multiplatform | Unity. *Unity* [online]. ©2021 [cit. 03. 04. 2021]. Dostupné z: https://unity.com/features/multiplatform

[15] VAUGHAN, William. *Digital modeling*. Berkeley, CA: New Riders, 2012. ISBN 978-0321700896.

[16] Creative Bloq Staff, JARRAT Steve. The best 3D modelling software in 2021. *Creative Bloq* [online]. 21. 01. 2021 [cit. 07. 04. 2021]. Dostupné z: https://www.creativebloq.com/features/best-3d-modelling-software

[17] Blender Foundation. Blender 2.83 Reference Manual — Blender Manual. *Blender Manual* [online]. 25. 02. 2021 [cit. 26. 04. 2021]. Dostupné z: https://docs.blender.org/manual/en/2.83/index.html

[18] Blender Foundation. Blender 2.83.0 Python API Documentation — Blender Python API. *Blender Python API* [online]. 25. 02. 2021 [cit. 30. 04. 2021]. Dostupné z: https://docs.blender.org/api/blender2.8/index.html

[19] HOCKING, Joseph. *Unity in action: multiplatform game development in C#. Second edition*. Shelter Island, NY: Manning Publications, 2018. ISBN 978-1617294969.

[20] Unity Technologies. Editor Scripting. *Unity Learn* [online]. 15. 03. 2019 [cit. 27. 04. 2021]. Dostupné z: https://learn.unity.com/tutorial/editor-scripting

[21] Unity Technologies. Unity - Scripting API. *Unity3D* [online]. 30. 04. 2021 [cit. 30. 04. 2021]. Dostupné z: https://docs.unity3d.com/2019.4/Documentation/ScriptReference/index.html

[22] Unity Technologies. Unity UI: Unity User Interface | Unity UI | 1.0.0. *Unity3D* [online]. 06. 10. 2020 [cit. 08.05.2021]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/index.html

[23] Univerzita Pardubice. Den otevřených dveří. *UPCE* [online]. ©2021 [cit. 08.05.2021]. Dostupné z: https://dod.fei.upce.cz/

[24] NIGRETTI, Alessia. Making the most of TextMesh Pro in Unity 2018 - Unity Technologies Blog. *Unity Technologies Blog* [online]. 16. 10. 2018 [cit. 01. 05. 2021]. Dostupné z: https://blogs.unity3d.com/2018/10/16/making-the-most-of-textmesh-pro-in-unity-2018/

[25] Unity Technologies. Project Organization. *Unity Learn* [online]. 28. 09. 2020 [cit. 07. 04. 2021]. Dostupné z: https://learn.unity.com/tutorial/project-organization-2019-3

[26] Univerzita Pardubice. Manuál jednotného vizuálního stylu. *UPCE* [online] [cit. 02. 05. 2021]. Dostupné z: https://www.upce.cz/sites/default/binary_www_old/deska/dokumenty/jvs/manual.pdf

[27] Laxer. Mobile Tree Package | 3D Trees | Unity Asset Store. *Unity Asset Store* [online]. 06. 02. 2016 [cit. 08.05.2021]. Dostupné z: https://assetstore.unity.com/packages/3d/vegetation/trees/mobile-tree-package-18866

[28] Univerzita Pardubice. IS/STAG Web Services. *UPCE* [online]. [cit. 02. 05. 2021]. Dostupné z: https://stag-ws.upce.cz/ws/web?pp_locale=en&selectedTyp=REST&pp_reqType=render&pp_page=serviceList&addr=%2Fservices%2Frest2%2Fmistnost

[29] Univerzita Pardubice. IS/STAG Web Services. *UPCE* [online]. [cit. 02. 05. 2021]. Dostupné z: https://stag-ws.upce.cz/ws/web?pp_locale=en&selectedTyp=REST&pp_reqType=render&pp_page=serviceList&addr=%2Fservices%2Frest2%2Frozvrhy

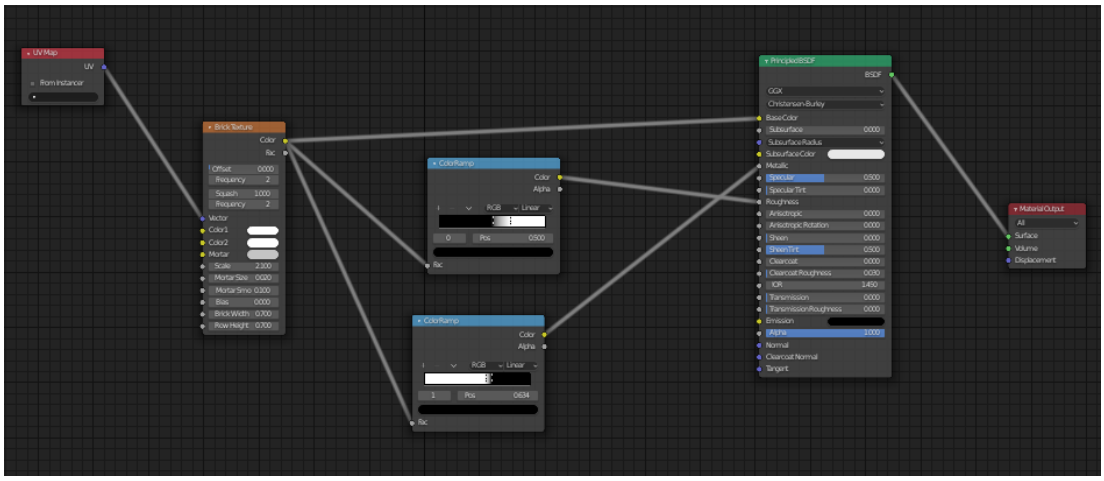
[30] WOLFFELT, Magnus, HYDE Rod, IACCARIANO, Julie. CSharpMessenger Extended - Unify Community Wiki. *Unity Community Wiki* [online]. 06. 09. 2018 [cit. 02. 05. 2021]. Dostupné z: http://wiki.unity3d.com/index.php/CSharpMessenger_Extended

[31] BONISLAWSKY, Brian J. GIll Sans free font. *Free Best Fonts* [online]. ©2011 [cit. 03. 05. 2021]. Dostupné z: https://www.freebestfonts.com/gill-sans-font

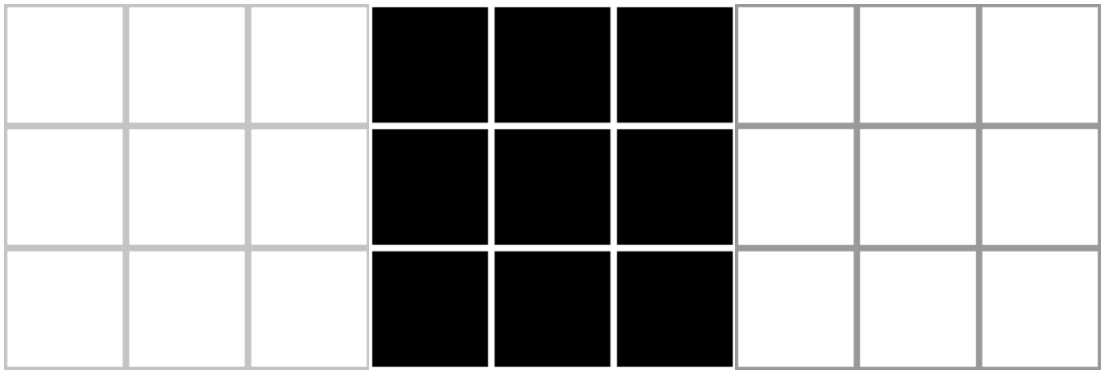
[32] POLIDARIO, Bernard. Unity Tutorials: How to make a MINIMAP in 10 Minutes. *WeeklyHow* [online]. 17. 6. 2020 [cit. 03. 05. 2021]. Dostupné z: https://weekly-how.com/unity-minimap-in-10-minutes/

SEZNAM PŘÍLOH

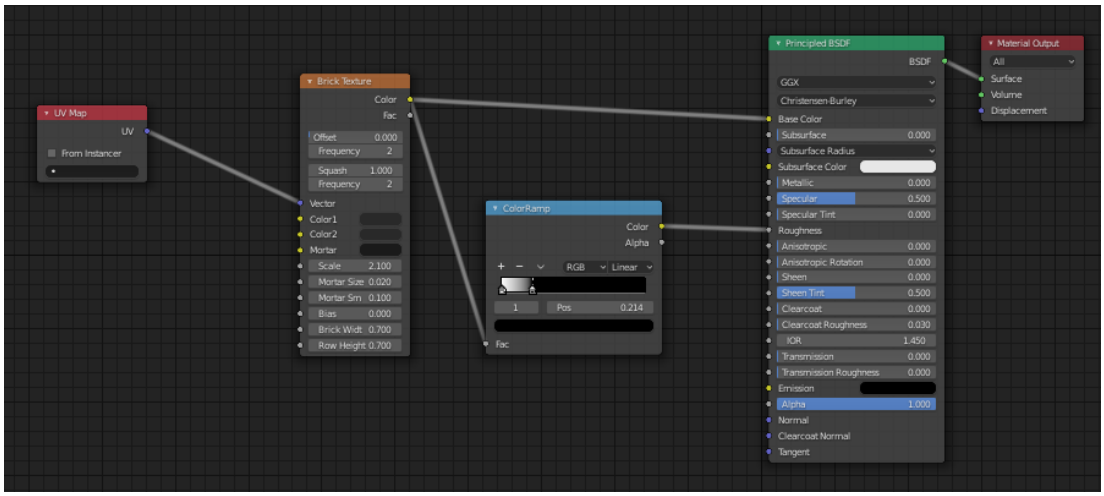
PŘÍLOHA A – OBRÁZKY



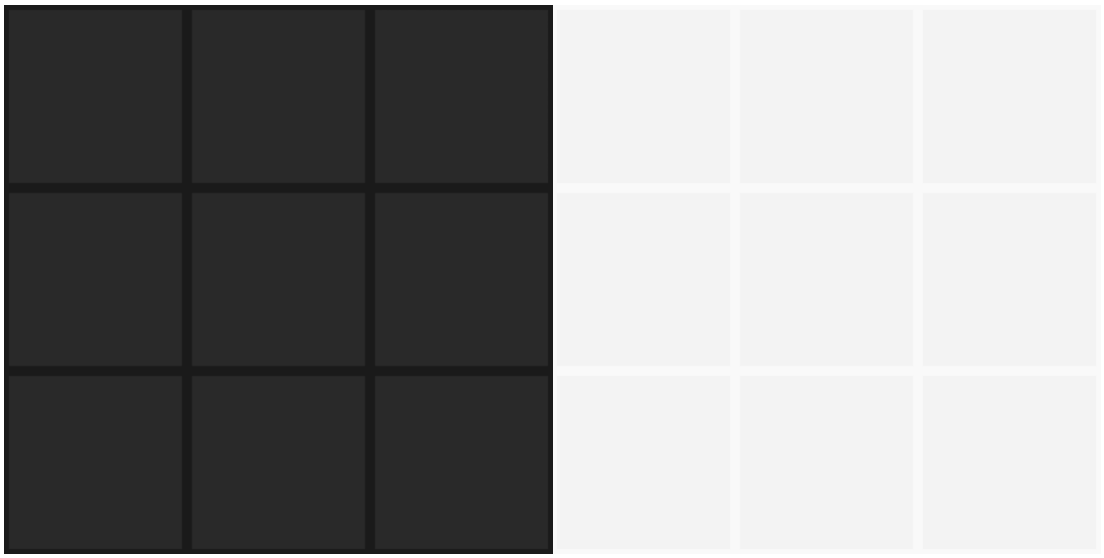
Obrázek stromu uzlů materiálu fei_ceiling_tiles.



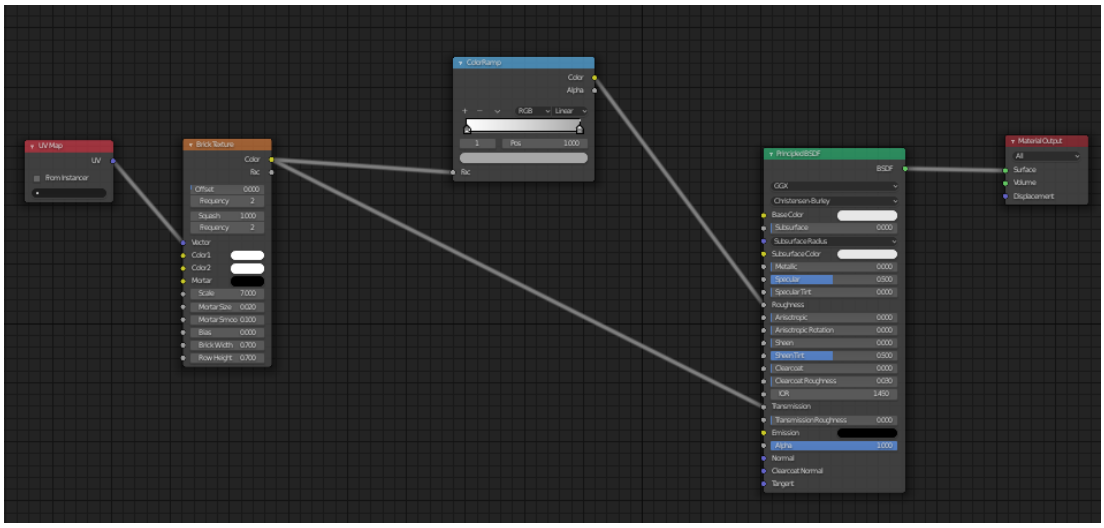
Obrázek mapy albedo, mapy kovovosti a mapy drsnosti materiálu fei_ceiling_tiles.



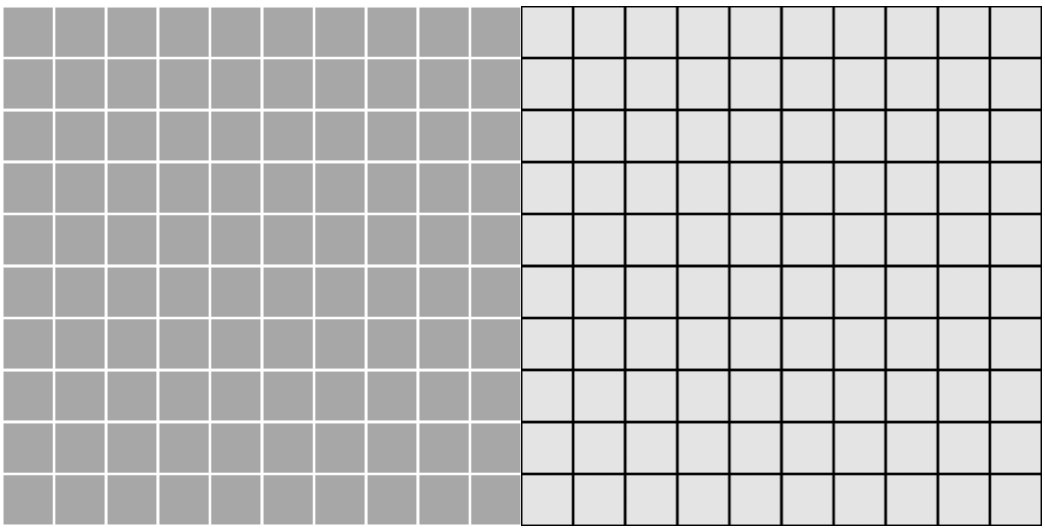
Obrázek stromu uzlů materiálu fei_ceiling_tiles_black.



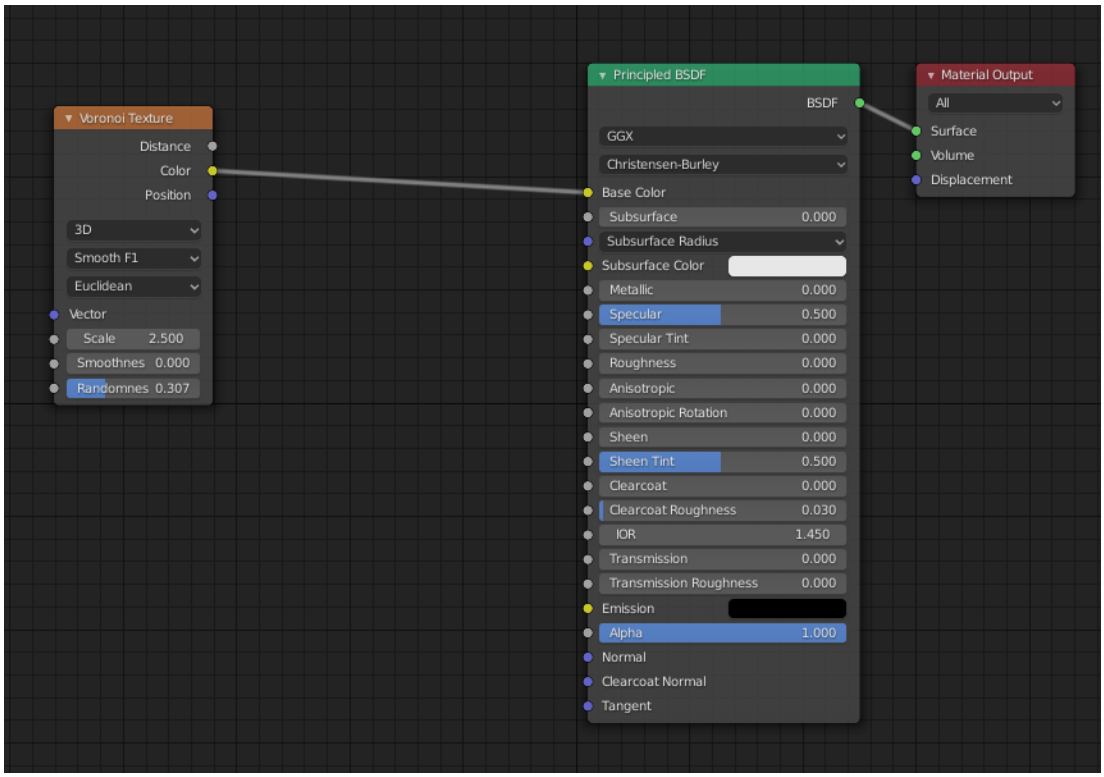
Obrázek mapy albedo a mapy drsnosti materiálu fei_ceiling_tiles_black.



Obrázek stromu uzlů materiálu fei_glass-bricks.

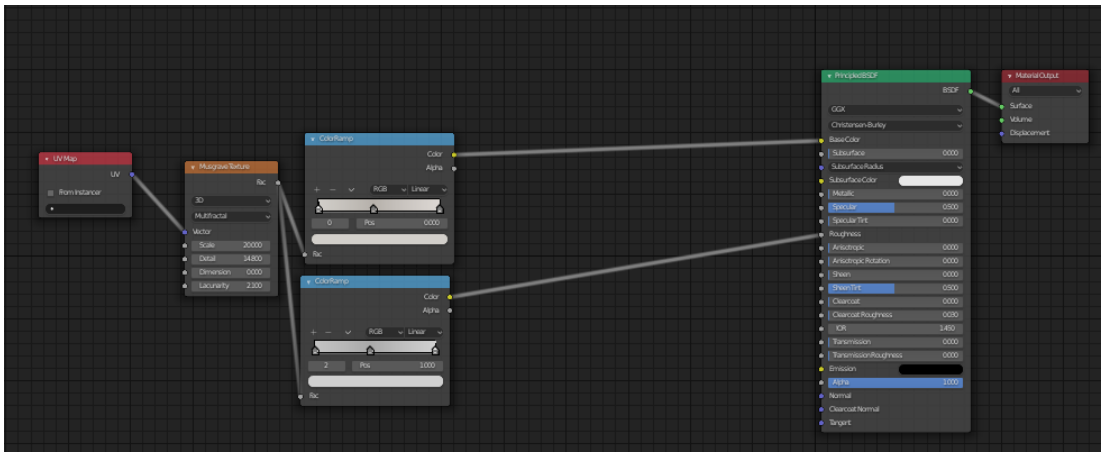


Obrázek mapy drsnosti a mapy transmission materiálu fei_glass_bricks.

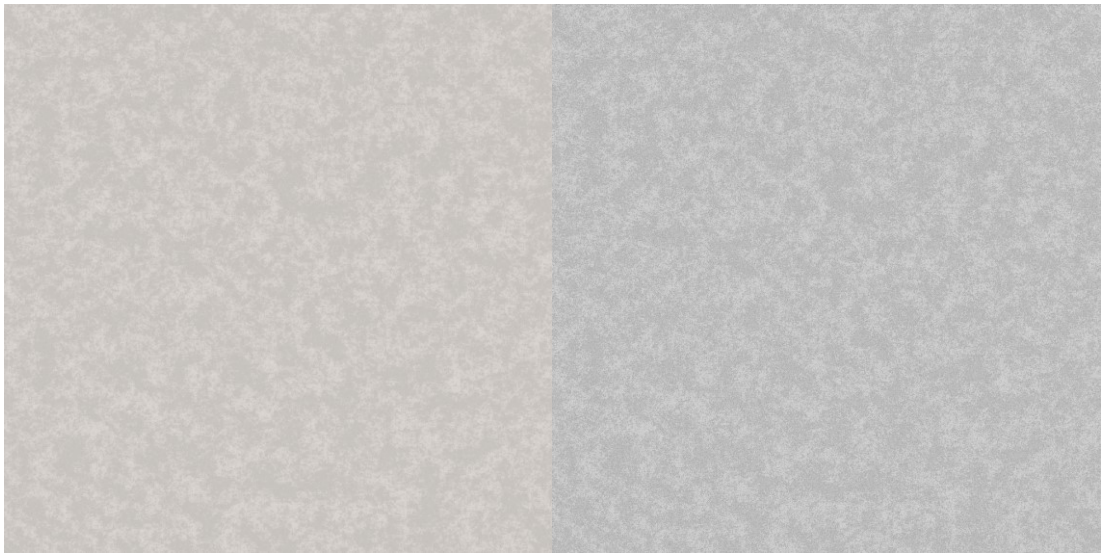


Obrázek stromu uzlů materiálu fei_glass_ornaments.

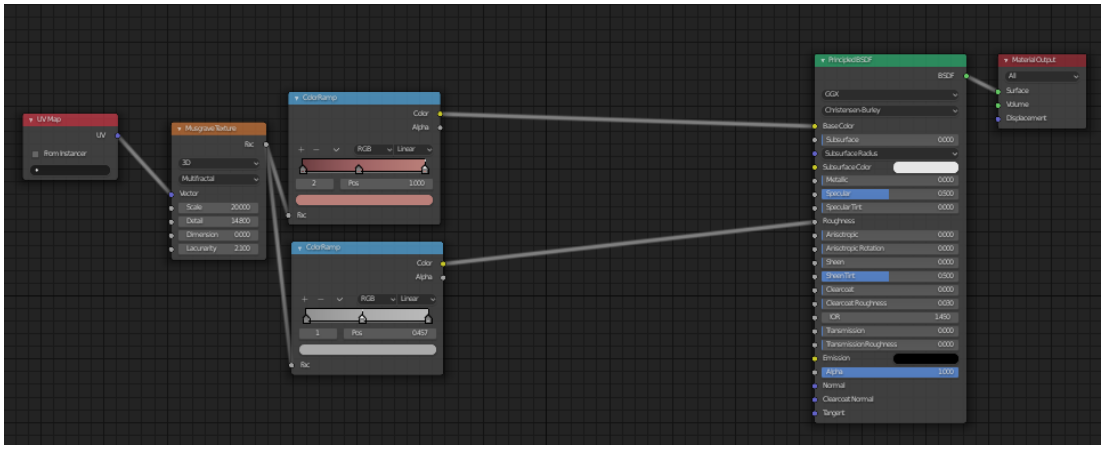
Obrázek mapy albedo materiálu fei_glass_ornaments.



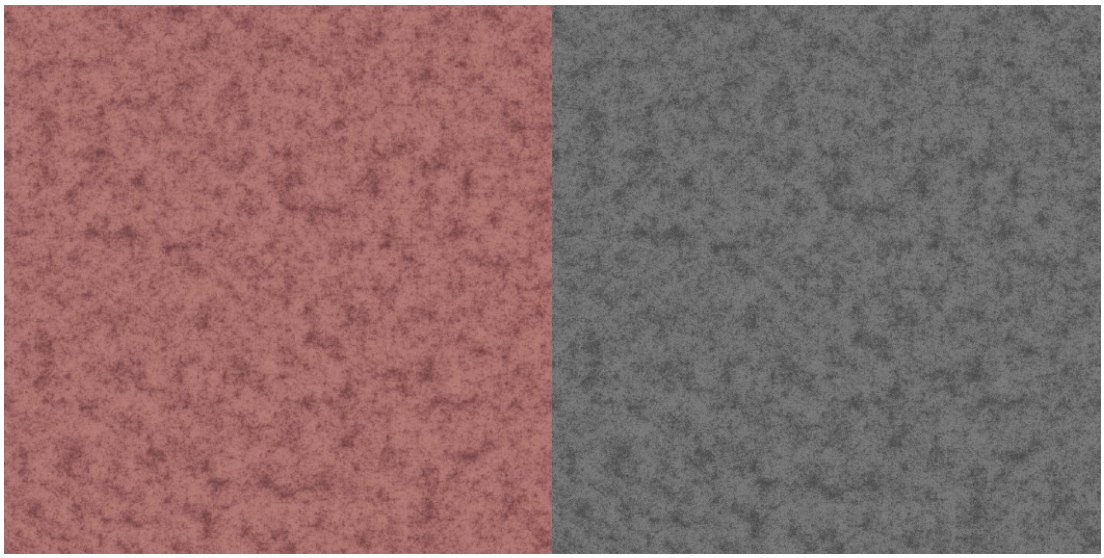
Obrázek stromu uzlů materiálu fei_pvc_grey-pattern.



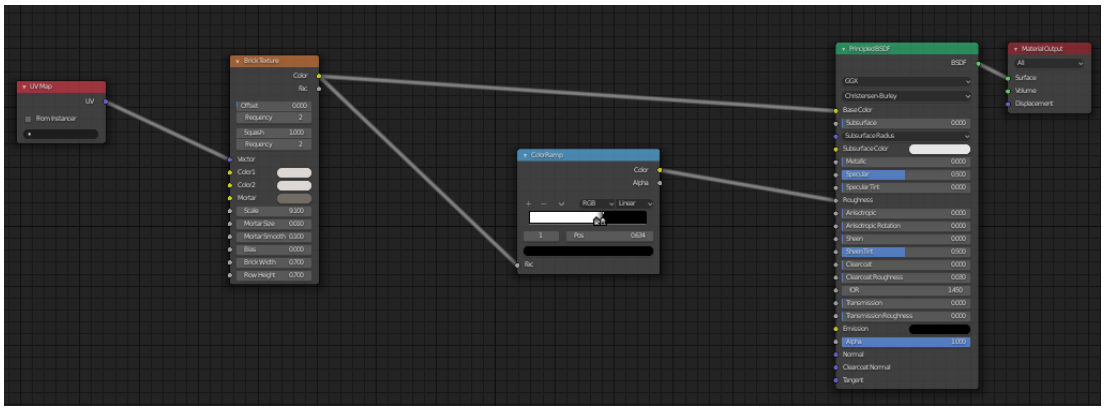
Obrázek mapy albedo a mapy drsnosti materiálu fei_pvc_grey-pattern.



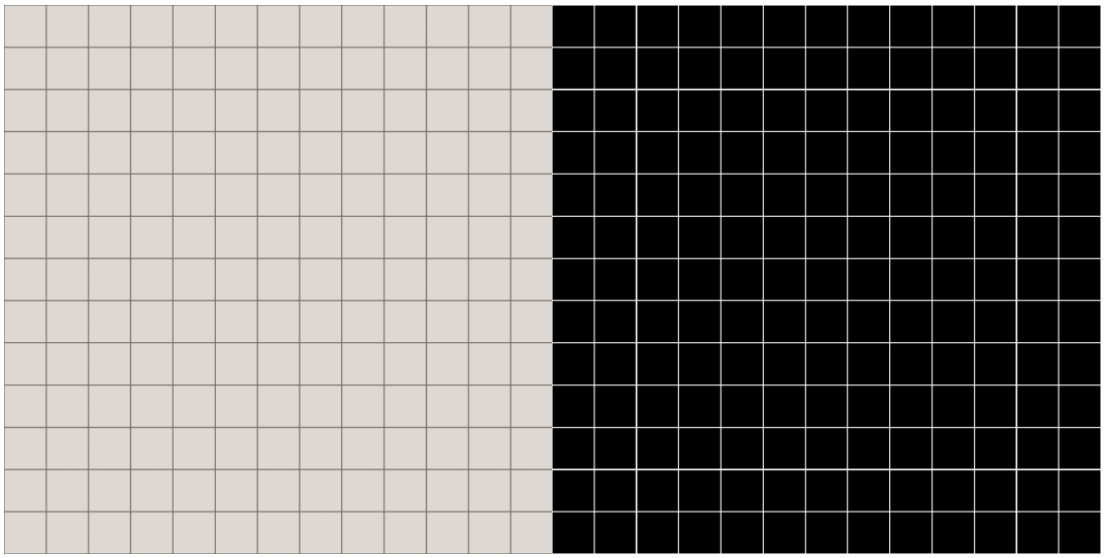
Obrázek stromu uzlů materiálu fei_pvc_red-pattern.



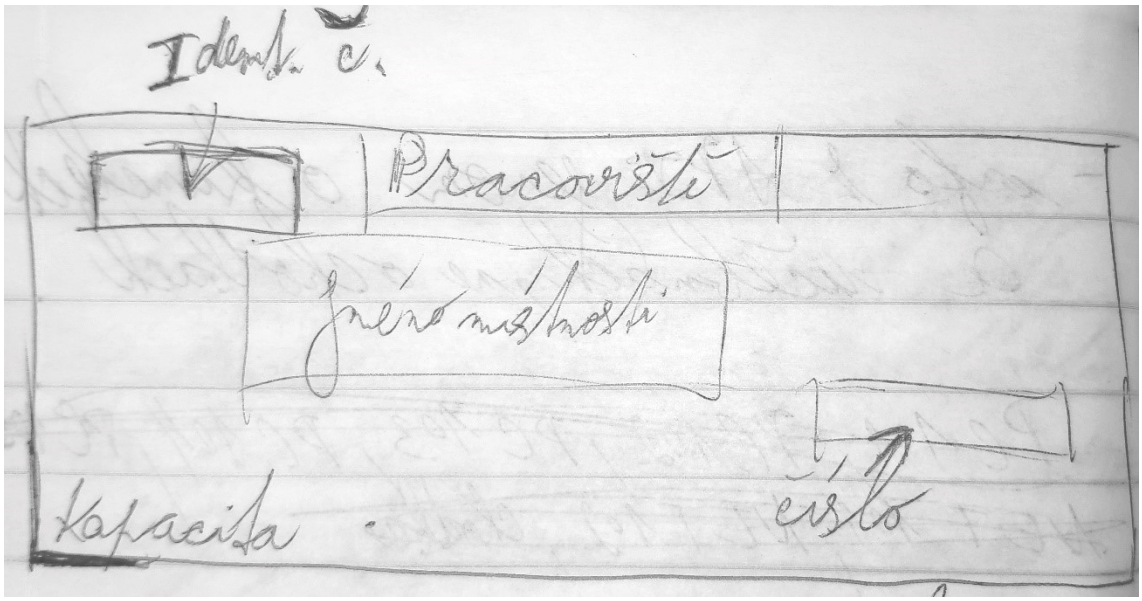
Obrázek mapy albedo a mapy drsnosti materiálu fei_pvc_red-pattern.



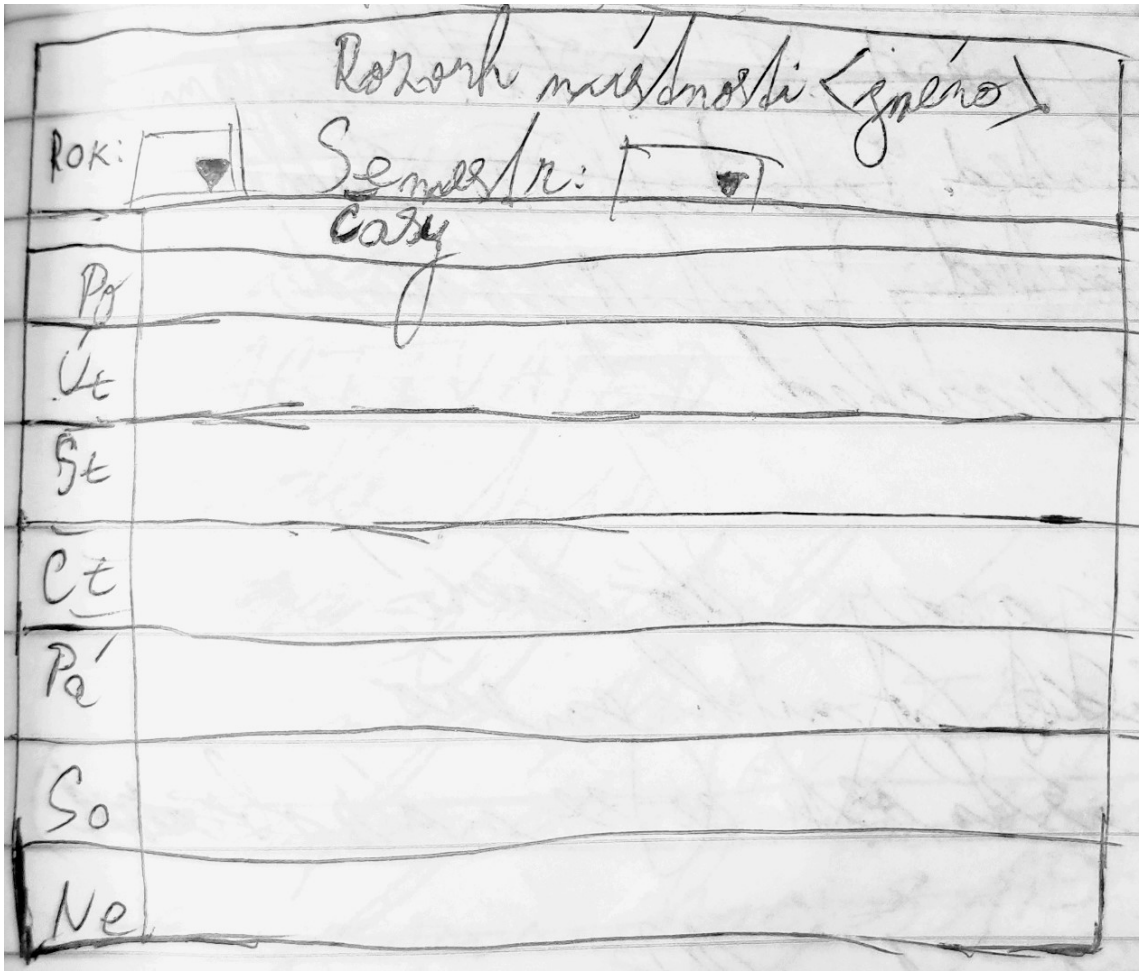
Obrázek stromu uzlů materiálu fei_wall_tiles.



Obrázek mapy albedo a mapy drsnosti materiálu fei_wall_tiles.

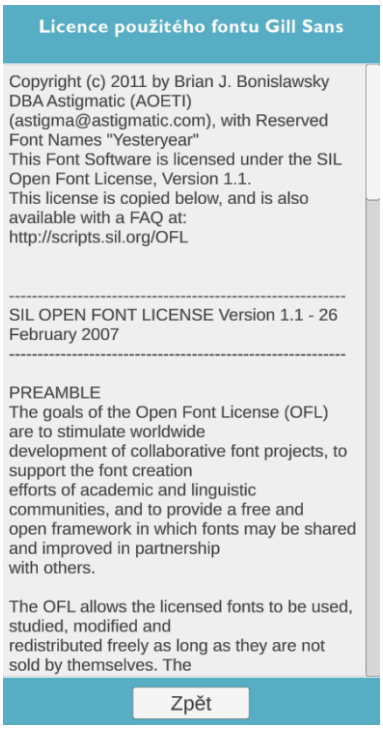


Obrázek konceptu tabulky pro zobrazení dodatečných informací o místnosti.



Obrázek konceptu tabulky pro zobrazení rozvrhu o místnosti.

Obrázek menu ovládání.



Obrázek menu zobrazující licenci použitého fontu Gill Sans.

PŘÍLOHA B – ZDROJOVÉ KÓDY

```
import bpy
import os

root_folder = "//fei_models_exports"

"""
Exports objects in the specified collection.
"""
def export_collection_objects(object_path, collection_name):
  for obj in bpy.data.collections[collection_name].all_objects:
    obj.select_set(True)

  bpy.ops.export_scene.fbx(filepath=object_path, check_existing=True,
  filter_glob='*.fbx', use_selection=True, use_active_collection=False,
  global_scale=1.0, apply_unit_scale=True,
  apply_scale_options='FBX_SCALE_NONE', bake_space_transform=True,
  object_types={'ARMATURE', 'CAMERA', 'EMPTY', 'LIGHT', 'MESH', 'OTHER'},
  use_mesh_modifiers=True, use_mesh_modifiers_render=True,
  mesh_smooth_type='OFF', use_subsurf=False, use_mesh_edges=False,
  use_tspace=False, use_custom_props=False, add_leaf_bones=True,
  primary_bone_axis='Y', secondary_bone_axis='X',
  use_armature_deform_only=False, armature_nodetype='NULL', bake_anim=True,
  bake_anim_use_all_bones=True, bake_anim_use_nla_strips=True,
  bake_anim_use_all_actions=True, bake_anim_force_startend_keying=True,
  bake_anim_step=1.0, bake_anim_simplify_factor=1.0, path_mode='AUTO',
  embed_textures=False, batch_mode='OFF', use_batch_own_dir=True,
  use_metadata=True, axis_forward='-Z', axis_up='Y')

  for obj in bpy.data.collections[collection_name].all_objects:
    obj.select_set(False)


"""
Creates the specified folder hierarchy.
"""
def create_folder_structure(folder_path):
    names_list = folder_path.split("\\")
    full_path = bpy.path.abspath(f"{root_folder}")
    if not os.path.isdir(full_path):
        os.mkdir(full_path) # create the root folder if missing

    for i in range(len(names_list)):
        full_path += f"\{names_list[i]}"
        if not os.path.isdir(full_path):
            os.mkdir(full_path) # create the target folder if missing
```

```
"""
Exports the content in the collection
along with the content of all of its
child collections if there are any.
This function should be called
to begin the whole process of exporting.
"""
def export_collection_hierarchy(collection_name, excluded_collections = []):
    col_root_path = collection_name
    current_collection = bpy.data.collections[collection_name]
    for child_collection in current_collection.children:
        if not child_collection.hide_viewport and not child_collection.name in ex-
cluded_collections:
            export_collection_hierarchy_recursive(col_root_path, child_collection, ex-
cluded_collections)
    if(len(current_collection.objects) > 0): # contains objects directly
        folder_path = bpy.path.abspath(f"{root_folder}\{collection_name}")
        full_path = bpy.path.abspath(f"{folder_path}\{collection_name}.fbx")
        create_folder_structure(collection_name)
        export_collection_objects(full_path, collection_name)

"""
Exports the content of the collection
and all of its child collections recursively.
"""
def export_collection_hierarchy_recursive(input_path, collection, excluded_collecti-
ons = []):
    current_path = f"{input_path}\{collection.name}"
    for child_collection in collection.children:
        if not child_collection.hide_viewport and not child_collection.name in ex-
cluded_collections:
            export_collection_hierarchy_recursive(current_path, child_collection, ex-
cluded_collections)
    if(len(collection.objects) > 0): # contains objects directly
        folder_path = bpy.path.abspath(f"{root_folder}\{input_path}")
        full_path = bpy.path.abspath(f"{folder_path}\{collection.name}.fbx")
        create_folder_structure(input_path)
        export_collection_objects(full_path, collection.name)


export_collection_hierarchy("fei_windows", ["fei_window_presets"])
export_collection_hierarchy("fei_doors", ["fei_door_presets"])
export_collection_hierarchy("fei_furniture")
export_collection_hierarchy("fei_lights")
export_collection_hierarchy("fei_railings")
export_collection_hierarchy("fei_arch-decors")
export_collection_hierarchy("fei_Cb")
export_collection_hierarchy("fei_backdrops")
export_collection_hierarchy("fei_stairs")
```

```
export_collection_hierarchy("fei_info_tables")
export_collection_hierarchy("fei_Ca")
export_collection_hierarchy("fei_ground")
```

Zdrojový kód skriptu models_exporter.

```
import bpy
import os
import json
import math

root_folder = "//fei_positions_exports"

"""
Gets the names of the specified
collection's children
"""
def get_child_collection_names(collection_name):
    children_names = []
    for collection in bpy.data.collections[collection_name].children:
        children_names.append(collection.name)

    return children_names


"""
Exports names of the collections being instantiated
along with the corresponding transforms altered
for Unity coordinate system
"""
def get_data_from_collection(collection_name):
    input_data = bpy.data.collections[collection_name].objects
    output_data = { "name": collection_name, "pos_list": [] } # structure must match in Uni
ty for parsing
    for obj in input_data:
        name = obj.instance_collection.name if obj.instance_type != "NONE" else "pivot"
        output_data["pos_list"].append({
            "name": name,
            "position": [(-1 * obj.location[0]), obj.location[2],
            (-1 * obj.location[1])], # y-up and -z-
forward, so the z and y need to be switched and local x and local y need to be negated
            "rotation": [math.degrees(obj.rotation_euler[0]),
            math.degrees(obj.rotation_euler[2]) * (-1),
            math.degrees(obj.rotation_euler[1])], # -z-
forward, so the local z rotation needs to be negated
            "scale": [obj.scale[0], obj.scale[2], obj.scale[1]] # switch y and z
        })
    return output_data
```

```
"""
Creates the specified folder hierarchy.
"""
def create_folder_structure(folder_path):
    names_list = folder_path.split("\\")
    full_path = bpy.path.abspath(f"{root_folder}")
    if not os.path.isdir(full_path):
        os.mkdir(full_path) # create the root folder if missing

    for i in range(len(names_list)):
        full_path += f"\{names_list[i]}"
        if not os.path.isdir(full_path):
            os.mkdir(full_path) # create the target folder if missing


"""
Saves the results into a JSON file
"""
def save_collection_data(collection_name, file_path):
    with open(file_path, "w") as j:
        output_data = get_data_from_collection(collection_name)
        j.write(json.dumps(output_data, indent=4))

    j.close()


"""
Goes over the specified list of collection names
and fetches the transform and instantiation data
and then saves them
"""
def save_multiple_collections_data(collection_names, folder_path):
    for collection_name in collection_names:
        create_folder_structure(folder_path)
        save_collection_data(collection_name, bpy.path.abspath(f"{root_folder}\{folder_path
}\{collection_name}.json"))

stairs_placements = get_child_collection_names("stairs-placements")
door_placements = get_child_collection_names("door-placements")
window_placements = get_child_collection_names("window-placements")
info_tables_placements = get_child_collection_names("info_tables-placements")
railings_placements = get_child_collection_names("railings-placements")
furniture_placements_1_np = get_child_collection_names("furniture-placements_1np")
furniture_placements_2_np = get_child_collection_names("furniture-placements_2np")
furniture_placements_3_np = get_child_collection_names("furniture-placements_3np")
furniture_placements_4_np = get_child_collection_names("furniture-placements_4np")
furniture_placements_h1 = ["furniture-placements_h1"]
furniture_placements_tunnel = ["furniture-placements_tunnel"]
light_placements_1_np = get_child_collection_names("light-placements_1_np")
```

```
light_placements_2_np = get_child_collection_names("light-placements_2_np")
light_placements_3_np = get_child_collection_names("light-placements_3_np")
light_placements_4_np = get_child_collection_names("light-placements_4_np")
light_placements_h1 = ["light-placements_h1"]
light_placements_tunnel = ["light-placements_tunnel"]
furniture_presets = get_child_collection_names("fei_furniture_presets")
door_presets = get_child_collection_names("fei_door_presets")
window_presets = get_child_collection_names("fei_window_presets")
light_presets = get_child_collection_names("fei_lights_presets")
arch_decors_placements = ["arch-decors-placements"]

save_multiple_collections_data(light_placements_1_np, "light_placements" + "\\" + "light_pl
acements_1_np")
save_multiple_collections_data(light_placements_2_np, "light_placements" + "\\" + "light_pl
acements_2_np")
save_multiple_collections_data(light_placements_3_np, "light_placements" + "\\" + "light_pl
acements_3_np")
save_multiple_collections_data(light_placements_4_np, "light_placements" + "\\" + "light_pl
acements_4_np")
save_multiple_collections_data(light_placements_h1, "light_placements" + "\\" + "light_plac
ements_h1")
save_multiple_collections_data(light_placements_tunnel, "light_placements" + "\\" + "light_
placements_tunnel")
save_multiple_collections_data(furniture_placements_1_np, "furniture_placements" + "\\" + "
furniture_placements_1_np")
save_multiple_collections_data(furniture_placements_2_np, "furniture_placements" + "\\" + "
furniture_placements_2_np")
save_multiple_collections_data(furniture_placements_3_np, "furniture_placements" + "\\" + "
furniture_placements_3_np")
save_multiple_collections_data(furniture_placements_4_np, "furniture_placements" + "\\" + "
furniture_placements_4_np")
save_multiple_collections_data(furniture_placements_h1, "furniture_placements" + "\\" + "fu
rniture_placements_h1")
save_multiple_collections_data(furniture_placements_tunnel, "furniture_placements" + "\\" +
 "furniture_placements_tunnel")
save_multiple_collections_data(stairs_placements, "stairs_placements")
save_multiple_collections_data(door_placements, "door_placements")
save_multiple_collections_data(window_placements, "window_placements")
save_multiple_collections_data(railings_placements, "railings_placements")
save_multiple_collections_data(info_tables_placements, "info-tables_placements")
save_multiple_collections_data(furniture_presets, "furniture_presets")
save_multiple_collections_data(door_presets, "door_presets")
save_multiple_collections_data(window_presets, "window_presets")
save_multiple_collections_data(light_presets, "light_presets")
save_multiple_collections_data(arch_decors_placements, "arch_decors_placements")
```

Zdrojový kód skriptu positions_exporter.

```
using UnityEditor;
```

```
/// <summary>
/// Extension script to create an editor window
/// for placing architecture decorations
/// </summary>
public class ArchDecorsPlacementsPopup : WindowPopupBase
{
  static ArchDecorsPlacementsPopup()
  {
    objectsPlacementsInfo = PathsHolder.archDecorsPlacementsInfo;
    positionsFolderPath = PathsHolder.archDecorsPlacementsFolderPath;
    objectsPaths = PathsHolder.archDecorsPaths;
  }

  [MenuItem("Tools/Place Architecture decorations")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy ArchDecorsPlacementsPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing doors
/// </summary>
public class DoorPlacementsPopup : WindowPopupBase
{
  static DoorPlacementsPopup()
  {
    objectsPlacementsInfo = PathsHolder.doorPlacementsInfo;
    positionsFolderPath = PathsHolder.doorPlacementsFolderPath;
    objectsPaths = PathsHolder.doorsPaths;
  }

  [MenuItem("Tools/Place Doors")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy DoorPlacementsPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for creating door presets
/// </summary>
public class DoorPresetsPopup : WindowPopupBase
{
  static DoorPresetsPopup()
  {
    objectsPlacementsInfo = PathsHolder.doorPresetsInfo;
    positionsFolderPath = PathsHolder.doorPresetsFolderPath;
    objectsPaths = PathsHolder.doorsPaths;
  }

  [MenuItem("Tools/Create Door Presets")]
  static void Init()
  {
    GenerateWindow();
  }
```

```
}
```

Zdrojový kód třídy DoorPresetsPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing furniture on the floor 1np
/// </summary>
public class FurniturePlacements1npPopup : WindowPopupBase
{
  static FurniturePlacements1npPopup()
  {
    objectsPlacementsInfo = PathsHolder.furniturePlacements1npInfo;
    positionsFolderPath = PathsHolder.furniturePlacements1npFolderPath;
    objectsPaths = PathsHolder.furniturePaths;
  }

  [MenuItem("Tools/Place Furniture/1NP")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy FurniturePlacements1npPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing furniture on the floor 2np
/// </summary>
public class FurniturePlacements2npPopup : WindowPopupBase
{
  static FurniturePlacements2npPopup()
  {
    objectsPlacementsInfo = PathsHolder.furniturePlacements2npInfo;
    positionsFolderPath = PathsHolder.furniturePlacements2npFolderPath;
    objectsPaths = PathsHolder.furniturePaths;
  }

  [MenuItem("Tools/Place Furniture/2NP")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy FurniturePlacements2npPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing furniture on the floor 3np
/// </summary>
public class FurniturePlacements3npPopup : WindowPopupBase
{
  static FurniturePlacements3npPopup()
  {
    objectsPlacementsInfo = PathsHolder.furniturePlacements3npInfo;
    positionsFolderPath = PathsHolder.furniturePlacements3npFolderPath;
    objectsPaths = PathsHolder.furniturePaths;
  }
```

```
  [MenuItem("Tools/Place Furniture/3NP")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy FurniturePlacements3npPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing furniture on the floor 4np
/// </summary>
public class FurniturePlacements4npPopup : WindowPopupBase
{
  static FurniturePlacements4npPopup()
  {
    objectsPlacementsInfo = PathsHolder.furniturePlacements4npInfo;
    positionsFolderPath = PathsHolder.furniturePlacements4npFolderPath;
    objectsPaths = PathsHolder.furniturePaths;
  }

  [MenuItem("Tools/Place Furniture/4NP")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy FurniturePlacements4npPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing furniture in the room H1
/// </summary>
public class FurniturePlacementsH1Popup : WindowPopupBase
{
  static FurniturePlacementsH1Popup()
  {
    objectsPlacementsInfo = PathsHolder.furniturePlacementsH1Info;
    positionsFolderPath = PathsHolder.furniturePlacementsH1FolderPath;
    objectsPaths = PathsHolder.furniturePaths;
  }

  [MenuItem("Tools/Place Furniture/H1")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy FurniturePlacementsH1Popup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing furniture in the tunnel
/// </summary>
public class FurniturePlacementsTunnelPopup : WindowPopupBase
{
```

```
  static FurniturePlacementsTunnelPopup()
  {
    objectsPlacementsInfo = PathsHolder.furniturePlacementsTunnelInfo;
    positionsFolderPath = PathsHolder.furniturePlacementsTunnelFolderPath;
    objectsPaths = PathsHolder.furniturePaths;
  }

  [MenuItem("Tools/Place Furniture/Tunnel")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy FurniturePlacementsTunnelPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for creating furniture presets
/// </summary>
public class FurniturePresetsPopup : WindowPopupBase
{
  static FurniturePresetsPopup()
  {
    objectsPlacementsInfo = PathsHolder.furniturePresetsInfo;
    positionsFolderPath = PathsHolder.furniturePresetsFolderPath;
    objectsPaths = PathsHolder.furniturePaths;
  }

  [MenuItem("Tools/Create Furniture Presets")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy FurniturePresetsPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing info tables
/// </summary>
public class InfoTablesPlacementsPopup : WindowPopupBase
{
  static InfoTablesPlacementsPopup()
  {
    objectsPlacementsInfo = PathsHolder.infoTablesPlacementsInfo;
    positionsFolderPath = PathsHolder.infoTablesPlacementFolderPath;
    objectsPaths = PathsHolder.infoTablesPaths;
  }

  [MenuItem("Tools/Place Info tables")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy InfoTablesPlacementsPopup.

```
using UnityEditor;
```

```
/// <summary>
/// Extension script to create an editor window
/// for placing lights on the floor 1np
/// </summary>
public class LightPlacements1npPopup : WindowPopupBase
{
  static LightPlacements1npPopup()
  {
    objectsPlacementsInfo = PathsHolder.lightPlacements1npInfo;
    positionsFolderPath = PathsHolder.lightPlacements1npFolderPath;
    objectsPaths = PathsHolder.lightPaths;
  }

  [MenuItem("Tools/Place Lights/1np")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy LightPlacements1npPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing lights on the floor 2np
/// </summary>
public class LightPlacements2npPopup : WindowPopupBase
{
  static LightPlacements2npPopup()
  {
    objectsPlacementsInfo = PathsHolder.lightPlacements2npInfo;
    positionsFolderPath = PathsHolder.lightPlacements2npFolderPath;
    objectsPaths = PathsHolder.lightPaths;
  }

  [MenuItem("Tools/Place Lights/2np")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy LightPlacements2npPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing lights on the floor 3np
/// </summary>
public class LightPlacements3npPopup : WindowPopupBase
{
  static LightPlacements3npPopup()
  {
    objectsPlacementsInfo = PathsHolder.lightPlacements3npInfo;
    positionsFolderPath = PathsHolder.lightPlacements3npFolderPath;
    objectsPaths = PathsHolder.lightPaths;
  }

  [MenuItem("Tools/Place Lights/3np")]
  static void Init()
  {
    GenerateWindow();
  }
```

```
}
```

Zdrojový kód třídy LightPlacements3npPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing lights on the floor 4np
/// </summary>
public class LightPlacements4npPopup : WindowPopupBase
{
  static LightPlacements4npPopup()
  {
    objectsPlacementsInfo = PathsHolder.lightPlacements4npInfo;
    positionsFolderPath = PathsHolder.lightPlacements4npFolderPath;
    objectsPaths = PathsHolder.lightPaths;
  }

  [MenuItem("Tools/Place Lights/4np")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy LightPlacements4npPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing lights in the room H1
/// </summary>
public class LightPlacementsH1Popup : WindowPopupBase
{
  static LightPlacementsH1Popup()
  {
    objectsPlacementsInfo = PathsHolder.lightPlacementsh1Info;
    positionsFolderPath = PathsHolder.lightPlacementsh1FolderPath;
    objectsPaths = PathsHolder.lightPaths;
  }

  [MenuItem("Tools/Place Lights/H1")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy LightPlacementsH1Popup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing lights in the tunnel
/// </summary>
public class LightPlacementsTunnelPopup : WindowPopupBase
{
  static LightPlacementsTunnelPopup()
  {
    objectsPlacementsInfo = PathsHolder.lightPlacementstunnelInfo;
    positionsFolderPath = PathsHolder.lightPlacementstunnelFolderPath;
    objectsPaths = PathsHolder.lightPaths;
  }
```

```
  [MenuItem("Tools/Place Lights/Tunnel")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy LightPlacementsTunnelPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for creating light presets
/// </summary>
public class LightPresetsPopup : WindowPopupBase
{
  static LightPresetsPopup()
  {
    objectsPlacementsInfo = PathsHolder.lightPresetsInfo;
    positionsFolderPath = PathsHolder.lightPresetsFolderPath;
    objectsPaths = PathsHolder.lightPaths;
  }

  [MenuItem("Tools/Create Lights Presets")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy LightPresetsPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing railings
/// </summary>
public class RailingsPlacementsPopup : WindowPopupBase
{

  static RailingsPlacementsPopup()
  {
    objectsPlacementsInfo = PathsHolder.railingsPlacementsInfo;
    positionsFolderPath = PathsHolder.railingsPlacementFolderPath;
    objectsPaths = PathsHolder.railingsPaths;
  }

  [MenuItem("Tools/Place Railings")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy RailingsPlacementsPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing stairs
/// </summary>
public class StairsPlacementsPopup : WindowPopupBase
```

```
{
  static StairsPlacementsPopup()
  {
    objectsPlacementsInfo = PathsHolder.stairsPlacementsInfo;
    positionsFolderPath = PathsHolder.stairsPlacementFolderPath;
    objectsPaths = PathsHolder.stairsPaths;
  }

  [MenuItem("Tools/Place Stairs")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy StairsPlacementsPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing windows
/// </summary>
public class WindowPlacementsPopup : WindowPopupBase
{
  static WindowPlacementsPopup()
  {
    objectsPlacementsInfo = PathsHolder.windowPlacementsInfo;
    positionsFolderPath = PathsHolder.windowPlacementsFolderPath;
    objectsPaths = PathsHolder.windowsPaths;
  }

  [MenuItem("Tools/Place Windows")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy WindowPlacementsPopup.

```
using UnityEditor;

/// <summary>
/// Extension script to create an editor window
/// for placing windows
/// </summary>
public class WindowPlacementsPopup : WindowPopupBase
{
  static WindowPlacementsPopup()
  {
    objectsPlacementsInfo = PathsHolder.windowPlacementsInfo;
    positionsFolderPath = PathsHolder.windowPlacementsFolderPath;
    objectsPaths = PathsHolder.windowsPaths;
  }

  [MenuItem("Tools/Place Windows")]
  static void Init()
  {
    GenerateWindow();
  }
}
```

Zdrojový kód třídy WindowPresetsPopup.

```
using System.Collections.Generic;
```

```
using System.IO;
using System.Text;
using UnityEngine;

/// <summary>
/// A helper script to get the necessary paths
/// for the editor scripts placing objects in the scene
/// </summary>
public class PathsHolder : MonoBehaviour
{

  public static readonly string infoTablesPlacementFolderPath =
"Assets/Editor/serialized_positions/info-tables_placements/";
  private static DirectoryInfo infoTablesPlacemenstDirInfo = new
DirectoryInfo(infoTablesPlacementFolderPath);
  public static readonly FileInfo[] infoTablesPlacementsInfo =
infoTablesPlacemenstDirInfo.GetFiles("*.json");

  public static readonly string archDecorsPlacementsFolderPath =
"Assets/Editor/serialized_positions/arch_decors_placements/";
  private static DirectoryInfo archDecorsPlacemenstDirInfo = new
DirectoryInfo(archDecorsPlacementsFolderPath);
  public static readonly FileInfo[] archDecorsPlacementsInfo =
archDecorsPlacemenstDirInfo.GetFiles("*.json");

  public static readonly string stairsPlacementFolderPath =
"Assets/Editor/serialized_positions/stairs_placements/";
  private static DirectoryInfo stairsPlacemenstDirInfo = new
DirectoryInfo(stairsPlacementFolderPath);
  public static readonly FileInfo[] stairsPlacementsInfo =
stairsPlacemenstDirInfo.GetFiles("*.json");

  public static readonly string railingsPlacementFolderPath =
"Assets/Editor/serialized_positions/railings_placements/";
  private static DirectoryInfo railingsPlacemenstDirInfo = new
DirectoryInfo(railingsPlacementFolderPath);
  public static readonly FileInfo[] railingsPlacementsInfo =
railingsPlacemenstDirInfo.GetFiles("*.json");

  public static readonly string doorPresetsFolderPath =
"Assets/Editor/serialized_positions/door_presets/";
  private static DirectoryInfo doorPresetsDirInfo = new
DirectoryInfo(doorPresetsFolderPath);
  public static readonly FileInfo[] doorPresetsInfo =
doorPresetsDirInfo.GetFiles("*.json");

  public static readonly string doorPlacementsFolderPath =
"Assets/Editor/serialized_positions/door_placements/";
  private static DirectoryInfo doorPlacementsDirInfo = new
DirectoryInfo(doorPlacementsFolderPath);
  public static readonly FileInfo[] doorPlacementsInfo =
doorPlacementsDirInfo.GetFiles("*.json");

  public static readonly string windowPresetsFolderPath =
"Assets/Editor/serialized_positions/window_presets/";
  private static DirectoryInfo windowPresetsDirInfo = new
DirectoryInfo(windowPresetsFolderPath);
  public static readonly FileInfo[] windowPresetsInfo =
windowPresetsDirInfo.GetFiles("*.json");

  public static readonly string windowPlacementsFolderPath =
"Assets/Editor/serialized_positions/window_placements/";
  private static DirectoryInfo windowPlacementsDirInfo = new
DirectoryInfo(windowPlacementsFolderPath);
  public static readonly FileInfo[] windowPlacementsInfo =
windowPlacementsDirInfo.GetFiles("*.json");
```

```
  public static readonly string lightPresetsFolderPath =
"Assets/Editor/serialized_positions/light_presets/";
  private static DirectoryInfo lightPresetsDirInfo = new
DirectoryInfo(lightPresetsFolderPath);
  public static readonly FileInfo[] lightPresetsInfo =
lightPresetsDirInfo.GetFiles("*.json");

  public static readonly string lightPlacements1npFolderPath =
"Assets/Editor/serialized_positions/light_placements/light_placements_1_np/";
  private static DirectoryInfo lightPlacements1npDirInfo = new
DirectoryInfo(lightPlacements1npFolderPath);
  public static readonly FileInfo[] lightPlacements1npInfo =
lightPlacements1npDirInfo.GetFiles("*.json");

  public static readonly string lightPlacements2npFolderPath =
"Assets/Editor/serialized_positions/light_placements/light_placements_2_np/";
  private static DirectoryInfo lightPlacements2npDirInfo = new
DirectoryInfo(lightPlacements2npFolderPath);
  public static readonly FileInfo[] lightPlacements2npInfo =
lightPlacements2npDirInfo.GetFiles("*.json");

  public static readonly string lightPlacements3npFolderPath =
"Assets/Editor/serialized_positions/light_placements/light_placements_3_np/";
  private static DirectoryInfo lightPlacements3npDirInfo = new
DirectoryInfo(lightPlacements3npFolderPath);
  public static readonly FileInfo[] lightPlacements3npInfo =
lightPlacements3npDirInfo.GetFiles("*.json");

  public static readonly string lightPlacements4npFolderPath =
"Assets/Editor/serialized_positions/light_placements/light_placements_4_np/";
  private static DirectoryInfo lightPlacements4npDirInfo = new
DirectoryInfo(lightPlacements4npFolderPath);
  public static readonly FileInfo[] lightPlacements4npInfo =
lightPlacements4npDirInfo.GetFiles("*.json");

  public static readonly string lightPlacementsh1FolderPath =
"Assets/Editor/serialized_positions/light_placements/light_placements_h1/";
  private static DirectoryInfo lightPlacementsh1DirInfo = new
DirectoryInfo(lightPlacementsh1FolderPath);
  public static readonly FileInfo[] lightPlacementsh1Info =
lightPlacementsh1DirInfo.GetFiles("*.json");

  public static readonly string lightPlacementstunnelFolderPath =
"Assets/Editor/serialized_positions/light_placements/light_placements_tunnel/";
  private static DirectoryInfo lightPlacementstunnelDirInfo = new
DirectoryInfo(lightPlacementstunnelFolderPath);
  public static readonly FileInfo[] lightPlacementstunnelInfo =
lightPlacementstunnelDirInfo.GetFiles("*.json");

  public static readonly string furniturePresetsFolderPath =
"Assets/Editor/serialized_positions/furniture_presets/";
  private static DirectoryInfo furniturePresetsDirInfo = new
DirectoryInfo(furniturePresetsFolderPath);
  public static readonly FileInfo[] furniturePresetsInfo =
furniturePresetsDirInfo.GetFiles("*.json");

  public static readonly string furniturePlacements1npFolderPath =
"Assets/Editor/serialized_positions/furniture_placements/furniture_placements_1_np/";
  private static DirectoryInfo furniturePlacements1npDirInfo = new
DirectoryInfo(furniturePlacements1npFolderPath);
  public static readonly FileInfo[] furniturePlacements1npInfo =
furniturePlacements1npDirInfo.GetFiles("*.json");

  public static readonly string furniturePlacements2npFolderPath =
"Assets/Editor/serialized_positions/furniture_placements/furniture_placements_2_np/";
```

```
  private static DirectoryInfo furniturePlacements2npDirInfo = new
DirectoryInfo(furniturePlacements2npFolderPath);
  public static readonly FileInfo[] furniturePlacements2npInfo =
furniturePlacements2npDirInfo.GetFiles("*.json");

  public static readonly string furniturePlacements3npFolderPath =
"Assets/Editor/serialized_positions/furniture_placements/furniture_placements_3_np/";
  private static DirectoryInfo furniturePlacements3npDirInfo = new
DirectoryInfo(furniturePlacements3npFolderPath);
  public static readonly FileInfo[] furniturePlacements3npInfo =
furniturePlacements3npDirInfo.GetFiles("*.json");

  public static readonly string furniturePlacements4npFolderPath =
"Assets/Editor/serialized_positions/furniture_placements/furniture_placements_4_np/";
  private static DirectoryInfo furniturePlacements4npDirInfo = new
DirectoryInfo(furniturePlacements4npFolderPath);
  public static readonly FileInfo[] furniturePlacements4npInfo =
furniturePlacements4npDirInfo.GetFiles("*.json");

  public static readonly string furniturePlacementsH1FolderPath =
"Assets/Editor/serialized_positions/furniture_placements/furniture_placements_h1/";
  private static DirectoryInfo furniturePlacementsH1DirInfo = new
DirectoryInfo(furniturePlacementsH1FolderPath);
  public static readonly FileInfo[] furniturePlacementsH1Info =
furniturePlacementsH1DirInfo.GetFiles("*.json");


  public static readonly string furniturePlacementsTunnelFolderPath =
"Assets/Editor/serialized_positions/furniture_placements/furniture_placements_tunnel/";
  private static DirectoryInfo furniturePlacementsTunnelDirInfo = new
DirectoryInfo(furniturePlacementsTunnelFolderPath);
  public static readonly FileInfo[] furniturePlacementsTunnelInfo =
furniturePlacementsTunnelDirInfo.GetFiles("*.json");

  public static readonly string stairsPrefabsFolderPath =
"Assets/Models/fei_stairs/prefabs/";
  public static readonly string railingsPrefabsFolderPath =
"Assets/Models/fei_railings/prefabs/";

  public static readonly string doorPresetsPrefabsFolderPath =
"Assets/Models/fei_doors/fei_door_presets/";
  public static readonly string doorThresholdsPrefabsFolderPath =
"Assets/Models/fei_doors/fei_door_thresholds/prefabs/";
  public static readonly string doorFramesPrefabsFolderPath =
"Assets/Models/fei_doors/fei_door_frames/prefabs/";
  public static readonly string doorBoardsPrefabsFolderPath =
"Assets/Models/fei_doors/fei_door_boards/prefabs/";
  public static readonly string doorHandlesPrefabsFolderPath =
"Assets/Models/fei_doors/fei_door_handles/prefabs/";
  public static readonly string doorWholeModelsPrefabsFolderPath =
"Assets/Models/fei_doors/fei_door_whole-models/prefabs/";

  public static readonly string furniturePrefabsFolderPath =
"Assets/Models/fei_furniture/prefabs/";
  public static readonly string furniturePresetsPrefabsFolderPath =
"Assets/Models/fei_furniture/presets/";

  public static readonly string lightPrefabsFolderPath =
"Assets/Models/fei_lights/prefabs/";
  public static readonly string lightPresetsPrefabsFolderPath =
"Assets/Models/fei_lights/presets/";

  public static readonly string windowBoardsPrefabsFolderPath =
"Assets/Models/fei_windows/fei_window_boards/prefabs/";
  public static readonly string windowFramesPrefabsFolderPath =
"Assets/Models/fei_windows/fei_window_frames/prefabs/";
```

```
  public static readonly string windowHandlesPrefabsFolderPath =
"Assets/Models/fei_windows/fei_window_handles/prefabs/";
  public static readonly string windowWindowsillsPrefabsFolderPath =
"Assets/Models/fei_windows/fei_window_windowsills/prefabs/";
  public static readonly string windowWholeModelsPrefabsFolderPath =
"Assets/Models/fei_windows/fei_window_whole-models/prefabs/";
  public static readonly string windowsPresetsPrefabsFolderPath =
"Assets/Models/fei_windows/fei_window_presets/";

  public static readonly string infoTablesPrefabsFolderPath =
"Assets/Models/fei_info_tables/prefabs/";
  public static readonly string archDecorsPrefabsFolderPath = "Assets/Models/fei_arch-
decors/prefabs/";

  public static readonly Dictionary<string, string> stairsPaths = new Dictionary<string,
string>
    {
        { "fei_Ca_stairs_middle_7-steps", stairsPrefabsFolderPath +
"fei_Ca_stairs_middle_7-steps Variant.prefab" },
        {"fei_Ca_stairs_middle_12-steps", stairsPrefabsFolderPath +
"fei_Ca_stairs_middle_12-steps Variant.prefab" },
        {"fei_Ca_stairs_front_13-steps", stairsPrefabsFolderPath + "fei_Ca_stairs_front_13-
steps Variant.prefab" },
        {"fei_stairs_front_6-steps", stairsPrefabsFolderPath + "fei_stairs_front_6-steps
Variant.prefab" },
        {"fei_Ca_stairs_back_12-steps", stairsPrefabsFolderPath + "fei_Ca_stairs_back_12-
steps Variant.prefab" },
        {"fei_stairs_back_7-steps", stairsPrefabsFolderPath + "fei_stairs_back_7-steps
Variant.prefab" },
        {"fei_stairs_back_11-steps", stairsPrefabsFolderPath + "fei_stairs_back_11-steps
Variant.prefab" },
        {"fei_Ca_stairs_back_9-steps", stairsPrefabsFolderPath + "fei_Ca_stairs_back_9-
steps Variant.prefab" },
        {"fei_Ca_stairs_h2_bottom", stairsPrefabsFolderPath + "fei_Ca_stairs_h2_bottom
Variant.prefab" },
        {"fei_Ca_stairs_h2_regular", stairsPrefabsFolderPath + "fei_Ca_stairs_h2_regular
Variant.prefab" },
        {"fei_Ca_stairs_h2_top", stairsPrefabsFolderPath + "fei_Ca_stairs_h2_top
Variant.prefab" }
    };

  public static readonly Dictionary<string, string> railingsPaths = new Dictionary<string,
string>
    {
        {"fei_railings_03002", railingsPrefabsFolderPath + "fei_railings_03002
Variant.prefab" },
        {"fei_railings_back_ramp", railingsPrefabsFolderPath + "fei_railings_back_ramp
Variant.prefab" },
        {"fei_railings_h2", railingsPrefabsFolderPath + "fei_railings_h2 Variant.prefab" },
        {"fei_railings_stairs_back_bottom", railingsPrefabsFolderPath +
"fei_railings_stairs_back_bottom Variant.prefab" },
        {"fei_railings_stairs_back_regular", railingsPrefabsFolderPath +
"fei_railings_stairs_back_regular Variant.prefab" },
        {"fei_railings_stairs_back_top", railingsPrefabsFolderPath +
"fei_railings_stairs_back_top Variant.prefab" },
        {"fei_railings_stairs_front_13-steps_bottom_l", railingsPrefabsFolderPath +
"fei_railings_stairs_front_13-steps_bottom_l Variant.prefab" },
        {"fei_railings_stairs_front_13-steps_bottom_p", railingsPrefabsFolderPath +
"fei_railings_stairs_front_13-steps_bottom_p Variant.prefab" },
        {"fei_railings_stairs_front_13-steps_regular_l", railingsPrefabsFolderPath +
"fei_railings_stairs_front_13-steps_regular_l Variant.prefab" },
        {"fei_railings_stairs_front_13-steps_regular_p", railingsPrefabsFolderPath +
"fei_railings_stairs_front_13-steps_regular_p Variant.prefab" },
        {"fei_railings_stairs_front_straight", railingsPrefabsFolderPath +
"fei_railings_stairs_front_straight Variant.prefab" },
```

```
        {"fei_railings_stairs_middle_bottom", railingsPrefabsFolderPath +
"fei_railings_stairs_middle_bottom Variant.prefab" },
        {"fei_railings_stairs_middle_regular", railingsPrefabsFolderPath +
"fei_railings_stairs_middle_regular Variant.prefab" },
        {"fei_railings_stairs_middle_top", railingsPrefabsFolderPath +
"fei_railings_stairs_middle_top Variant.prefab" }
    };


  public static readonly Dictionary<string, string> doorsPaths = GeneratePaths(
      new string[] { doorThresholdsPrefabsFolderPath, doorBoardsPrefabsFolderPath,
            doorFramesPrefabsFolderPath, doorHandlesPrefabsFolderPath,
          doorWholeModelsPrefabsFolderPath, doorPresetsPrefabsFolderPath });

  public static readonly Dictionary<string, string> furniturePaths = GeneratePaths(
    new string[] { furniturePrefabsFolderPath, furniturePresetsPrefabsFolderPath });

  public static readonly Dictionary<string, string> lightPaths = GeneratePaths(
    new string[] { lightPrefabsFolderPath, lightPresetsPrefabsFolderPath });

  public static readonly Dictionary<string, string> windowsPaths = GeneratePaths(
    new string[] { windowBoardsPrefabsFolderPath, windowFramesPrefabsFolderPath,
    windowWindowsillsPrefabsFolderPath, windowWholeModelsPrefabsFolderPath,
        windowHandlesPrefabsFolderPath, windowsPresetsPrefabsFolderPath});

  public static readonly Dictionary<string, string> infoTablesPaths = GeneratePaths(
    new string[] { infoTablesPrefabsFolderPath });

  public static readonly Dictionary<string, string> archDecorsPaths = GeneratePaths(
    new string[] { archDecorsPrefabsFolderPath, windowWindowsillsPrefabsFolderPath });

  /// <summary>
  /// Goes over the specified folders and gets
  /// the paths to all the files with a .prefab extension.
  /// This information is then saved into a dictionary
  /// where the key is the name of the file without the .prefab
  /// extension and witout anything past the first space character.
  /// This can be used only on folders and files without
  /// whitespaces in their names.
  /// </summary>
  /// <param name="folderPaths">string array of paths to go over</param>
  /// <returns>Dictionary of names of the prefabs and their paths</returns>
  public static Dictionary<string, string> GeneratePaths(string[] folderPaths)
  {
    Dictionary<string, string> filePaths = new Dictionary<string, string>();
    foreach (string currentPath in folderPaths)
    {
      DirectoryInfo dirInfo = new DirectoryInfo(currentPath);
      FileInfo[] prefabsInfo = dirInfo.GetFiles("*.prefab");

      foreach (var item in prefabsInfo)
      {
        string keyName = item.Name.Split(' ')[0].Replace(".prefab", "");
        filePaths.Add(keyName, currentPath + item.Name);
      }
    }

    return filePaths;
  }

  /// <summary>
  /// Builds the string representation of dictionary
  /// values for the paths to prefabs.
  /// It's puprpose is to automate manual process of specifiing
  /// the paths when tha paths generated automatically
  /// need to be slightly modified to work properly.
```

```
  /// </summary>
  /// <param name="folderPath">path of the folder to get prefabs from</param>
  /// <param name="folderVarName">name of the variable holding the path of the
folder</param>
  /// <returns>string representing the elements in a dictionary</returns>
  public static string GetFilesPaths(string folderPath, string folderVarName)
  {
    DirectoryInfo dirInfo = new DirectoryInfo(folderPath);
    FileInfo[] prefabsInfo = dirInfo.GetFiles("*.prefab");
    StringBuilder builder = new StringBuilder();
    foreach (var item in prefabsInfo)
    {
      string keyName = item.Name.Split(' ')[0];
      builder.Append("{\"" + keyName + "\", " + folderVarName + " + \"" + item.Name +
"\"},\n");
    }

    return builder.ToString();
  }
}
```

Zdrojový kód skriptu PathsHolder.

```
using System.Collections.Generic;
using UnityEditor;
using UnityEngine;

/// <summary>
/// A script for placing the assets
/// in the scene based on the serialized
/// JSON data
/// </summary>
public class SceneBuilder : MonoBehaviour
{
  /// <summary>
  /// Places assets into the scene
  /// based on the data from
  /// the specified JSON file
  /// </summary>
  /// <param name="jsonPath">path to the JSON file</param>
  /// <param name="keyValuePairs">dictionary with collection names mapped to asset
paths</param>
  public static void PlaceAssets(string jsonPath,
Dictionary<string, string> keyValuePairs)
  {
    PosListData dataJson = FetchJSONData(jsonPath);

    GameObject parent = new GameObject(dataJson.name);
    List<GameObject> spawnedObjects = new List<GameObject>();

    foreach (PosData currPos in dataJson.pos_list)
    {
      Vector3 pos = new Vector3(currPos.position[0], currPos.position[1],
currPos.position[2]);
      Vector3 rot = new Vector3(currPos.rotation[0], currPos.rotation[1],
currPos.rotation[2]);
      Vector3 scale = new Vector3(currPos.scale[0], currPos.scale[1], currPos.scale[2]);

      if (currPos.name == "pivot")
      {
        parent.transform.position = pos;
        parent.transform.rotation = Quaternion.Euler(rot);
        parent.transform.localScale = scale;
      }
      else if (keyValuePairs.ContainsKey(currPos.name))
```

```
      {
        UnityEngine.Object loadedPrefab =
          AssetDatabase.LoadAssetAtPath(keyValuePairs[currPos.name],
typeof(UnityEngine.Object));

        GameObject spawnedObject =
(GameObject)PrefabUtility.InstantiatePrefab(loadedPrefab);
        spawnedObject.transform.position = pos;
        spawnedObject.transform.rotation = Quaternion.Euler(rot);
        spawnedObject.transform.localScale = scale;

        spawnedObjects.Add(spawnedObject);
      }
      else
      {
        Debug.LogWarning($"Key of name {currPos.name} not found!");
      }
    }

    foreach (GameObject item in spawnedObjects)
    {
      item.transform.SetParent(parent.transform);
    }
  }

  /// <summary>
  /// Deserializes the specified
  /// JSON positions data
  /// </summary>
  /// <param name="path">path to JSON file</param>
  /// <returns>an instance of PosLitData class</returns>
  public static PosListData FetchJSONData(string path)
  {
    TextAsset jsonDataSource = (TextAsset)AssetDatabase.LoadAssetAtPath(path,
typeof(TextAsset));
    PosListData dataJson = JsonUtility.FromJson<PosListData>(jsonDataSource.ToString());

    return dataJson;
  }
}
```

Zdrojový kód třídy SceneBuilder.

```
/// <summary>
/// A class representing the transformation
/// data in the serialized JSON files
/// </summary>
[System.Serializable]
public class PosData
{
  public string name;
  public float[] position;
  public float[] rotation;
  public float[] scale;

}
```

Zdrojový kód třídy PosData.

```
using System.Collections.Generic;

/// <summary>
/// A class representing the whole
/// serialized JSON object with the list
/// of transforms
/// </summary>
```

```
[System.Serializable]
public class PosListData
{
  public string name;
  public List<PosData> pos_list;
}
```

Zdrojový kód třídy PosListData.

```
using UnityEngine;

/// <summary>
/// A script to handle floor names
/// and entering the given floor
/// </summary>
public class FloorTrigger : MonoBehaviour
{
  [SerializeField] string floorName;


  private void OnTriggerEnter(Collider other)
  {
    Messenger<string>.Broadcast(UIEvent.FLOOR_CHANGED, floorName);
  }
}
```

Zdrojový kód skriptu FloorTrigger.

```
using UnityEngine;

/// <summary>
/// A script to handle room names
/// and entering the given room
/// </summary>
public class RoomTriggerIDHolder : MonoBehaviour
{
  [SerializeField] string roomName;


  private void OnTriggerEnter(Collider other)
  {
    Messenger<string>.Broadcast(UIEvent.ROOM_CHANGED, roomName);
  }
}
```

Zdrojový kód skriptu RoomTriggerIDHolder.

```
using UnityEngine;

/// <summary>
/// A script to handle name info tables
/// </summary>
public class RoomInfoTableNameHolder : MonoBehaviour
{
  [SerializeField] private string roomName;

  public void Operate()
  {
    Messenger.Broadcast(UIEvent.MENU_OPENED);
    Messenger<string>.Broadcast(UIEvent.INFO_TABLE_NAME_ACTIVATED, roomName);
  }
}
```

Zdrojový kód skriptu RoomInfoTableNameHolder.

```
using UnityEngine;

/// <summary>
/// A script to handle timetable info tables
/// </summary>
public class RoomInfoTableTimetableHolder : MonoBehaviour
{
  [SerializeField] private string roomName;

  public void Operate()
  {
    Messenger.Broadcast(UIEvent.MENU_OPENED);
    Messenger<string>.Broadcast(UIEvent.INFO_TABLE_TIMETABLE_ACTIVATED, roomName);
  }
}
```

Zdrojový kód skriptu RoomInfoTableTimetableHolder.

```
using System;
using System.Collections.Generic;
using System.Xml;
using UnityEngine;

/// <summary>
/// A class representing the basic
/// information about a room
/// </summary>
public class RoomInfo
{
  public string roomName;
  public string capacity;
  public string roomType;
  public string workplace;

  public RoomInfo(string roomName, string capacity, string roomType, string workplace)
  {
    this.roomName = roomName;
    this.capacity = capacity;
    this.roomType = roomType;
    this.workplace = workplace;
  }
}

/// <summary>
/// A class representing the timetable
/// of a room
/// </summary>
public class RoomTimetable
{
  private int year;
  private string semester;
  public List<RoomTimetableAction> mondayActions;
  public List<RoomTimetableAction> tuesdayActions;
  public List<RoomTimetableAction> wednesdayActions;
  public List<RoomTimetableAction> thursdayActions;
  public List<RoomTimetableAction> fridayActions;
  public List<RoomTimetableAction> saturdayActions;
  public List<RoomTimetableAction> sundayActions;
  public int Year { get => year; set => year = value; }
  public string Semester { get => semester; set => semester = value; }

  public RoomTimetable()
  {
    mondayActions = new List<RoomTimetableAction>();
    tuesdayActions = new List<RoomTimetableAction>();
    wednesdayActions = new List<RoomTimetableAction>();
```

```
    thursdayActions = new List<RoomTimetableAction>();
    fridayActions = new List<RoomTimetableAction>();
    saturdayActions = new List<RoomTimetableAction>();
    sundayActions = new List<RoomTimetableAction>();
  }

  /// <summary>
  /// Assigns the given timetable action
  /// based on the day of the week it belongs to
  /// </summary>
  /// <param name="action">timetable action to be assigned</param>
  public void AddTimetableActionToList(RoomTimetableAction action)
  {
    if (action.day == "Po")
    {
      mondayActions.Add(action);
    }
    else if (action.day == "Út")
    {
      tuesdayActions.Add(action);
    }
    else if (action.day == "St")
    {
      wednesdayActions.Add(action);
    }
    else if (action.day == "Čt")
    {
      thursdayActions.Add(action);
    }
    else if (action.day == "Pá")
    {
      fridayActions.Add(action);
    }
    else if (action.day == "So")
    {
      saturdayActions.Add(action);
    }
    else if (action.day == "Ne")
    {
      sundayActions.Add(action);
    }
  }

  /// <summary>
  /// Sorts the timetable actions by their beginning hours
  /// </summary>
  public void SortListsByTime()
  {
    mondayActions.Sort((x, y) => x.realHourFrom.CompareTo(y.realHourFrom));
    tuesdayActions.Sort((x, y) => x.realHourFrom.CompareTo(y.realHourFrom));
    wednesdayActions.Sort((x, y) => x.realHourFrom.CompareTo(y.realHourFrom));
    thursdayActions.Sort((x, y) => x.realHourFrom.CompareTo(y.realHourFrom));
    fridayActions.Sort((x, y) => x.realHourFrom.CompareTo(y.realHourFrom));
    saturdayActions.Sort((x, y) => x.realHourFrom.CompareTo(y.realHourFrom));
    sundayActions.Sort((x, y) => x.realHourFrom.CompareTo(y.realHourFrom));
  }
}

/// <summary>
/// A class to represent the timetable action
/// </summary>
public class RoomTimetableAction
{
  public string actionName;
  public string department;
  public string subject;
```

```
  public string actionType;
  public string day;
  public int hourFrom;
  public int hourTo;
  public DateTime realHourFrom;
  public DateTime realHourTo;

  public RoomTimetableAction(string actionName, string department, string subject,
    string actionType, string day, int hourFrom, int hourTo, DateTime realHourFrom,
DateTime realHourTo)
  {
    this.actionName = actionName;
    this.department = department;
    this.subject = subject;
    this.actionType = actionType;
    this.day = day;
    this.hourFrom = hourFrom;
    this.hourTo = hourTo;
    this.realHourFrom = realHourFrom;
    this.realHourTo = realHourTo;
  }
}

/// <summary>
/// A script to handle management
/// of information about the rooms
/// </summary>
public class RoomInfoSystem : MonoBehaviour
{
  private Dictionary<string, RoomInfo> roomIdNamePairs;
  private Dictionary<string, RoomTimetable> roomIdTimetablePairs;
  [SerializeField] private TextAsset roomsXML;
  [SerializeField] private TextAsset timetablesAll;
  private string defaultSemester;
  private int defaultYear;

  void Awake()
  {
    defaultYear = 2020;
    defaultSemester = "LS";
  }

  void Start()
  {
    roomIdNamePairs = LoadRoomInfo(roomsXML.ToString());
    roomIdTimetablePairs = LoadRoomsTimetables(timetablesAll.ToString());
  }


  /// <summary>
  /// Returns the info about the
  /// room based on the provided name.
  /// If no data are present for such
  /// name, a generic output is created.
  /// </summary>
  /// <param name="roomID">name of the room</param>
  /// <returns>an instance of RoomInfo class</returns>
  public RoomInfo GetRoomInfo(string roomID)
  {
    if (roomIdNamePairs.ContainsKey(roomID))
    {
      return roomIdNamePairs[roomID];
    }
    else
    {
      return new RoomInfo(roomID, "", "", "Univerzita Pardubice");
```

```
    }
  }

  /// <summary>
  /// Loads the information about rooms
  /// from the provided string of XML data
  /// into a dictionary for later use
  /// </summary>
  /// <param name="input">string with XML data</param>
  /// <returns>a dictionary with deserialized data</returns>
  private Dictionary<string, RoomInfo> LoadRoomInfo(string input)
  {
    Dictionary<string, RoomInfo> keyValuePairs = new Dictionary<string, RoomInfo>();

    XmlDocument doc = new XmlDocument();
    doc.LoadXml(input);

    XmlNodeList roomsList = doc.GetElementsByTagName("mistnostInfo");

    for (int i = 0; i < roomsList.Count; i++)
    {
      string roomName = roomsList[i].ChildNodes[1].InnerText;
// room number is the second node
      string workplace = roomsList[i].ChildNodes[3].InnerText;
// workplace is the fourth node
      string roomType = roomsList[i].ChildNodes[5].InnerText;
// capacity is the sixth node
      string capacity = roomsList[i].ChildNodes[6].InnerText;
// capacity is the seventh node



      keyValuePairs.Add(roomName, new RoomInfo(roomName, capacity, roomType, workplace));
    }

    return keyValuePairs;
  }

  /// <summary>
  /// Loads the timetables of the rooms
  /// based on provided string of XML data
  /// </summary>
  /// <param name="input">string with XML data</param>
  /// <returns>a dictionary with timetables for later use</returns>
  public Dictionary<string, RoomTimetable> LoadRoomsTimetables(string input)
  {
    Dictionary<string, RoomTimetable> keyValuePairs = new Dictionary<string,
RoomTimetable>();

    XmlDocument doc = new XmlDocument();
    doc.LoadXml(input);

    foreach (XmlNode timetable in doc.DocumentElement.SelectNodes("mistnostRozvrh"))
    {
      keyValuePairs.Add(timetable.Attributes["cislo"].Value,
GetTimetableFromXml(timetable.ChildNodes));
    }

    return keyValuePairs;
  }

  /// <summary>
  /// Returns the timetable
  /// based on the name of the room
  /// </summary>
  /// <param name="mistnost">name of the room</param>
```

```
  /// <returns>an instance of RoomTimetable</returns>
  public RoomTimetable GetRoomTimetable(string mistnost)
  {
    if (roomIdTimetablePairs.ContainsKey(mistnost))
    {
      return roomIdTimetablePairs[mistnost];
    }
    else
    {
      return new RoomTimetable();
    }
  }

  /// <summary>
  /// Parses the timetables of the given rooms
  /// from the XML nodes
  /// </summary>
  /// <param name="roomsList">XML data with the rooms' timetables</param>
  /// <returns>an instance of RoomTimetable class</returns>
  public RoomTimetable GetTimetableFromXml(XmlNodeList roomsList)
  {

    RoomTimetable timetable = new RoomTimetable();

    for (int i = 0; i < roomsList.Count; i++)
    {
      string name = roomsList[i].SelectSingleNode("nazev").InnerText;
      string department = roomsList[i].SelectSingleNode("katedra").InnerText;
      string subject = roomsList[i].SelectSingleNode("predmet").InnerText;
      string actionType = roomsList[i].SelectSingleNode("typAkceZkr").InnerText;
      string day = roomsList[i].SelectSingleNode("denZkr").InnerText;
      int hourFrom;
      if (!int.TryParse(roomsList[i].SelectSingleNode("hodinaOd")?.InnerText, out
hourFrom))
        hourFrom = 0;
      int hourTo;
      if (!int.TryParse(roomsList[i].SelectSingleNode("hodinaDo")?.InnerText, out hourTo))
        hourTo = 0;

      DateTime realHourFrom =
DateTime.Parse(roomsList[i].SelectSingleNode("hodinaSkutOd")?.InnerText);
      DateTime realHourTo =
DateTime.Parse(roomsList[i].SelectSingleNode("hodinaSkutDo")?.InnerText);

      RoomTimetableAction currAction = new RoomTimetableAction(
        name, department, subject, actionType, day, hourFrom, hourTo, realHourFrom,
realHourTo);

      timetable.AddTimetableActionToList(currAction);
    }

    timetable.SortListsByTime();

    timetable.Semester = defaultSemester;
    timetable.Year = defaultYear;

    return timetable;
  }
}
```

Zdrojový kód skriptu RoomInfoSystem.

```
using TMPro;
using UnityEngine;

/// <summary>
```

```
/// A script to control the action status prompt
/// </summary>
public class ActionStatusPrompt : MonoBehaviour
{
  [SerializeField] TextMeshProUGUI messagePrompt;

  /// <summary>
  /// Enables the prompt GameObject and updates its message
  /// </summary>
  /// <param name="message"></param>
  public void ShowPrompt(string message)
  {
    messagePrompt.text = message;
    gameObject.SetActive(true);
  }

  /// <summary>
  /// Hides the prompt GameObject
  /// </summary>
  public void HidePrompt()
  {
    gameObject.SetActive(false);
  }
}
```

Zdrojový kód skriptu ActionStatusPrompt.

```
using TMPro;
using UnityEngine;

/// <summary>
/// A script to control the main interaction
/// button prompt
/// </summary>
public class DoorButtonPrompt : MonoBehaviour
{
  [SerializeField] TextMeshProUGUI messagePrompt;
  public void ShowPrompt(string message)
  {
    messagePrompt.text = message;
    gameObject.SetActive(true);
  }

  public void HidePrompt()
  {
    gameObject.SetActive(false);
  }
}
```

Zdrojový kód skriptu DoorButtonPrompt.

```
using UnityEngine;

/// <summary>
/// A script to control the HUD
/// </summary>
public class HUDController : MonoBehaviour
{
  public void Show()
  {
    gameObject.SetActive(true);
  }

  public void Hide()
  {
    gameObject.SetActive(false);
```

```
  }
}
```

Zdrojový kód skriptu HUDController.

```
using UnityEngine;

/// <summary>
/// A script to control the main menu
/// </summary>
public class MainMenuPopup : MonoBehaviour, IMenu
{
  public void Show()
  {
    gameObject.SetActive(true);
  }

  public void Hide()
  {
    gameObject.SetActive(false);
  }

  public void OnContinueButtonClick()
  {
    Messenger.Broadcast(UIEvent.MENU_CLOSED);
  }

  public void OnControlsButtonClick()
  {
    Messenger.Broadcast(UIEvent.CONTROLS_MENU_OPENED);
  }

  public void OnFontLicenceButtonClick()
  {
    Messenger.Broadcast(UIEvent.FONT_LICENCE_MENU_OPENED);
  }
}
```

Zdrojový kód skriptu MainMenuPopup.

```
using TMPro;
using UnityEngine;

/// <summary>
/// A class to handle the logic for the minimap
/// </summary>
public class MinimapController : MonoBehaviour
{
  [SerializeField] private TextMeshProUGUI locationLabel;
  [SerializeField] private TextMeshProUGUI floorLabel;

  /// <summary>
  /// Updates the name of the current room
  /// displayed in the first label under the minimap
  /// </summary>
  /// <param name="locationName">name of the location to display</param>
  public void UpdateLocationLabel(string locationName)
  {
    locationLabel.text = locationName;
  }

  /// <summary>
  /// Updates the name of the current floor
  /// displayed in the second label under the minimap
  /// </summary>
  /// <param name="floorName">name of the floor to display</param>
```

```
  public void UpdateFloorLabel(string floorName)
  {
    floorLabel.text = floorName;
  }
}
```

Zdrojový kód skriptu MinimapController.

```
using TMPro;
using UnityEngine;

/// <summary>
/// A script to control the room info screen
/// </summary>
public class RoomInfoPopup : MonoBehaviour, IMenu
{
  [SerializeField] TextMeshProUGUI capacityLabel;
  [SerializeField] TextMeshProUGUI capacityDescriptionLabel;
  [SerializeField] TextMeshProUGUI typeLabel;
  [SerializeField] TextMeshProUGUI workplaceLabel;
  [SerializeField] TextMeshProUGUI nameLabel;

  /// <summary>
  /// Shows the room info screen
  /// and allows to set its values
  /// </summary>
  /// <param name="capacity">capacity of th room</param>
  /// <param name="roomName">name of the room</param>
  /// <param name="roomType">type of the room</param>
  /// <param name="workplace">workplace the room belongs to</param>
  public void Show(string capacity, string roomName, string roomType, string workplace)
  {
    capacityDescriptionLabel.gameObject.SetActive(!string.IsNullOrEmpty(capacity));
    capacityLabel.text = capacity;
    nameLabel.text = roomName;
    typeLabel.text = roomType;
    workplaceLabel.text = workplace;
    gameObject.SetActive(true);
  }

  public void Show()
  {
    gameObject.SetActive(true);
  }

  public void Hide()
  {
    gameObject.SetActive(false);
  }

  public void OnCloseButtonClick()
  {
    Messenger.Broadcast(UIEvent.INFO_TABLE_NAME_DEACTIVATED);
    Messenger.Broadcast(UIEvent.MENU_CLOSED);
  }
}
```

Zdrojový kód skriptu RoomInfoPopup.

```
using System;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

/// <summary>
/// A script to control the room timetable screen
```

```
/// </summary>
public class RoomTimetablePopup : MonoBehaviour, IMenu
{
  [SerializeField] private Transform mondayRow;
  [SerializeField] private Transform tuesdayRow;
  [SerializeField] private Transform wednesdayRow;
  [SerializeField] private Transform thursdayRow;
  [SerializeField] private Transform fridayRow;
  [SerializeField] private Transform saturdayRow;
  [SerializeField] private Transform sundayRow;
  [SerializeField] private GameObject tableCell;
  [SerializeField] private TextMeshProUGUI roomNameLabel;
  [SerializeField] private TextMeshProUGUI yearLabel;
  [SerializeField] private TextMeshProUGUI semesterLabel;
  private float baseCellWidth = 100f;
  private DateTime lastWorkingHour = DateTime.Parse("21:00");

  /// <summary>
  /// Fills the table based on the content
  /// corresponding to the name of the room
  /// </summary>
  /// <param name="timetable">timetable to display</param>
  /// <param name="roomName">name of the room</param>
  public void Populate(RoomTimetable timetable, string roomName)
  {
    roomNameLabel.text = "CA-" + roomName;
    yearLabel.text = timetable.Year + "/" + (timetable.Year + 1);
    semesterLabel.text = timetable.Semester;
    PopulateRow(mondayRow, timetable.mondayActions);
    PopulateRow(tuesdayRow, timetable.tuesdayActions);
    PopulateRow(wednesdayRow, timetable.wednesdayActions);
    PopulateRow(thursdayRow, timetable.thursdayActions);
    PopulateRow(fridayRow, timetable.fridayActions);
    PopulateRow(saturdayRow, timetable.saturdayActions);
    PopulateRow(sundayRow, timetable.sundayActions);
  }

  /// <summary>
  /// Fills the specified row of the table
  /// based on the provided timetable actions
  /// </summary>
  /// <param name="row">row to put the actions to</param>
  /// <param name="actions">timetable actions to display</param>
  private void PopulateRow(Transform row, List<RoomTimetableAction> actions)
  {
    DateTime lastHour = DateTime.Parse("7:00"); // start from the first action

    foreach (RoomTimetableAction currAction in actions)
    {
      if (currAction.realHourTo == lastHour) // quick fix for multiple actions at the same
time
      {
        continue;
      }

      if ((float)(currAction.realHourFrom - lastHour).TotalHours > 0.0f)
      {
        GameObject paddingCell = Instantiate(tableCell);
        float increaseBy = (float)(currAction.realHourFrom - lastHour).TotalHours;
        paddingCell.GetComponent<TimetableCellController>().UpdateWidth(baseCellWidth *
increaseBy);
        lastHour = currAction.realHourFrom;
        paddingCell.transform.SetParent(row);
      }

      GameObject currCell = Instantiate(tableCell);
```

```
currCell.GetComponent<TimetableCellController>().UpdateBackgroundColor(GetColorByActionType
(currAction.actionType));
      currCell.GetComponent<TimetableCellController>().UpdateLabels(
        currAction.department + "/" + currAction.subject, currAction.actionName,
currAction.realHourFrom.ToString("HH:mm"),
        currAction.realHourTo.ToString("HH:mm"));
      float increaseCellBy = (float)(currAction.realHourTo -
currAction.realHourFrom).TotalHours;
      currCell.GetComponent<TimetableCellController>().UpdateWidth(baseCellWidth *
increaseCellBy);
      currCell.transform.SetParent(row);

      lastHour = currAction.realHourTo;
    }

    if (lastHour <= lastWorkingHour)
    {
      GameObject paddingCell = Instantiate(tableCell);
      float increaseBy = (float)(lastWorkingHour - lastHour).TotalHours;
      paddingCell.GetComponent<TimetableCellController>().UpdateWidth(baseCellWidth *
increaseBy);
      paddingCell.transform.SetParent(row);
    }
  }

  /// <summary>
  /// Changes the color of the timetable action cell
  /// to correctly represent the type of the action
  /// </summary>
  /// <param name="actionType">string with the type of the action</param>
  /// <returns>a Color</returns>
  private Color GetColorByActionType(string actionType)
  {
    if (actionType == "Cv")
      return new Color(0.6f, 0.9f, 0.7f);
    else if (actionType == "Př")
      return new Color(0.9f, 0.9f, 0.9f);
    else if (actionType == "Se")
      return new Color(0.64f, 0.93f, 0.59f);

    return new Color(0.5f, 0.5f, 0.5f);
  }

  public void Show()
  {
    gameObject.SetActive(true);
  }

  public void Hide()
  {
    gameObject.SetActive(false);

  }

  /// <summary>
  /// Depopulates the table.
  /// Should be called before
  /// trying to populate
  /// the table again.
  /// </summary>
  public void ClearRows()
  {
    ClearRow(mondayRow);
    ClearRow(tuesdayRow);
    ClearRow(wednesdayRow);
```

```
    ClearRow(thursdayRow);
    ClearRow(fridayRow);
    ClearRow(saturdayRow);
    ClearRow(sundayRow);
  }

  public void OnCloseButtonClick()
  {
    Messenger.Broadcast(UIEvent.INFO_TABLE_TIMETABLE_DEACTIVATED);
    Messenger.Broadcast(UIEvent.MENU_CLOSED);
  }

  /// <summary>
  /// Clears the specific row of the table
  /// </summary>
  /// <param name="clearedRow">row ot the table to clear</param>
  private void ClearRow(Transform clearedRow)
  {
    int counter = 0;
    foreach (Transform child in clearedRow)
    {
      if (counter > 0)
        Destroy(child.gameObject);

      counter++;
    }
  }
}
```

Zdrojový kód skriptu RoomTimetablePopup.

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

/// <summary>
/// A script to control the timetable cell
/// </summary>
public class TimetableCellController : MonoBehaviour
{
  [SerializeField] private TextMeshProUGUI mainLabel;
  [SerializeField] private TextMeshProUGUI nameLabel;
  [SerializeField] private TextMeshProUGUI timeFromLabel;
  [SerializeField] private TextMeshProUGUI timeToLabel;
  [SerializeField] private RawImage backgroundImage;
  [SerializeField] private RawImage border;
  [SerializeField] private RectTransform rectTransform;

  /// <summary>
  /// Updates the label values in this cell
  /// </summary>
  /// <param name="mainText">acronym of the action</param>
  /// <param name="nameText">full name of the action</param>
  /// <param name="timeFrom">real starting hour</param>
  /// <param name="timeTo">real ending hour</param>
  public void UpdateLabels(string mainText, string nameText, string timeFrom, string
timeTo)
  {
    mainLabel.text = mainText;
    nameLabel.text = nameText;
    timeFromLabel.text = timeFrom;
    timeToLabel.text = timeTo;
  }

  /// <summary>
  /// Updates the background color
```

```
  /// to the specified value
  /// </summary>
  /// <param name="newColor">color to change to</param>
  public void UpdateBackgroundColor(Color newColor)
  {
    backgroundImage.color = newColor;
  }

  /// <summary>
  /// Updates the border color
  /// to the specified value
  /// </summary>
  /// <param name="newColor">color to change to</param>
  public void UpdateBorderColor(Color newColor)
  {
    border.color = newColor;
  }

  /// <summary>
  /// Changes the width of the cell
  /// to the specified value
  /// </summary>
  /// <param name="width">new width of the cell</param>
  public void UpdateWidth(float width)
  {
    rectTransform.sizeDelta = new Vector2(width, rectTransform.sizeDelta.y);
  }
}
```

Zdrojový kód skriptu TimetableCellController.

```
using UnityEngine;

/// <summary>
/// A script to control the controls submenu
/// </summary>
public class ControlsMenuPopup : MonoBehaviour, ISubMenu
{
  public void Show()
  {
    gameObject.SetActive(true);
  }

  public void Hide()
  {
    gameObject.SetActive(false);
  }

  public void OnBackMenuClick()
  {
    Messenger.Broadcast(UIEvent.SUB_MENU_CLOSED);
  }
}
```

Zdrojový kód skriptu ControlsMenuPopup.

```
using UnityEngine;

/// <summary>
/// A script to control the font licence submenu
/// </summary>
public class FontLicenceMenuPopup : MonoBehaviour, ISubMenu
{
  public void Show()
  {
    gameObject.SetActive(true);
```

```
  }

  public void Hide()
  {
    gameObject.SetActive(false);
  }

  public void OnBackButtonClick()
  {
    Messenger.Broadcast(UIEvent.SUB_MENU_CLOSED);
  }
}
```

Zdrojový kód skriptu FontLicenceMenuPopup.

```
/// <summary>
/// An interface for better
/// menu management
/// </summary>
public interface IMenu
{
  /// <summary>
  /// Activates the GameObject
  /// so it is visible
  /// </summary>
  void Show();

  /// <summary>
  /// Deactivates the GameObject
  /// so it is not visible
  /// </summary>
  void Hide();
}
```

Zdrojový kód rozhraní IMenu.

```
/// <summary>
/// An interface for better
/// submenu management
/// </summary>
public interface ISubMenu
{
  /// <summary>
  /// Activates the GameObject
  /// so it is visible
  /// </summary>
  void Show();

  /// <summary>
  /// Deactivates the GameObject
  /// so it is not visible
  /// </summary>
  void Hide();
}
```

Zdrojový kód rozhraní ISubMenu.

```
using System.Collections;
using UnityEngine;
using UnityEngine.UI;

/// <summary>
/// A script to centrally manage
/// all of the UI
/// </summary>
public class UIController : MonoBehaviour
```

```
{
  [SerializeField] private DoorButtonPrompt doorButtonPrompt;
  [SerializeField] private DoorButtonPrompt timetableButtonPrompt;
  [SerializeField] private ActionStatusPrompt actionStatusPrompt;
  [SerializeField] private MinimapController minimapController;
  [SerializeField] private RawImage statusPrompt;
  [SerializeField] private HUDController hudHolder;
  [SerializeField] private RoomInfoSystem roomInfoSystem;
  [SerializeField] private RoomInfoPopup roomInfoPopup;
  [SerializeField] private RoomTimetablePopup roomTimetablePopup;
  [SerializeField] private MainMenuPopup mainMenuPopup;
  [SerializeField] private ControlsMenuPopup controlsMenuPopup;
  [SerializeField] private FontLicenceMenuPopup fontLicenceMenuPopup;

  private IMenu openedMenu;
  private ISubMenu openedSubmenu;

  void Awake()
  {
    Messenger<string>.AddListener(UIEvent.DOOR_APPROACHED, OnDoorApproached);
    Messenger<string>.AddListener(UIEvent.INFO_TABLE_NAME_APPROACHED,
OnInfoTableNameApproached);
    Messenger<string>.AddListener(UIEvent.INFO_TABLE_TIMETABLE_APPROACHED,
OnInfoTableTimetableApproached);
    Messenger<string>.AddListener(UIEvent.DOOR_LOCKED_OPEN, OnLockedDoorOpen);
    Messenger<string>.AddListener(UIEvent.ROOM_CHANGED, OnRoomEntered);
    Messenger<string>.AddListener(UIEvent.FLOOR_CHANGED, OnFloorChanged);
    Messenger<string>.AddListener(UIEvent.INFO_TABLE_NAME_ACTIVATED,
OnInfoTableNameActivated);
    Messenger<string>.AddListener(UIEvent.INFO_TABLE_TIMETABLE_ACTIVATED,
OnInfoTableTimetableActivated);
    Messenger.AddListener(UIEvent.INFO_TABLE_NAME_DEACTIVATED, OnInfoTableNameDeActivated);
    Messenger.AddListener(UIEvent.INFO_TABLE_TIMETABLE_DEACTIVATED,
OnInfoTableTimetableDeActivated);
    Messenger.AddListener(UIEvent.MENU_OPENED, OnMenuOpened);
    Messenger.AddListener(UIEvent.MENU_CLOSED, OnMenuClosed);
    Messenger.AddListener(UIEvent.MAIN_MENU_OPENED, OnMainMenuOpened);
    Messenger.AddListener(UIEvent.DOOR_LEAVE, OnDoorLeave);
    Messenger.AddListener(UIEvent.INFO_TABLE_NAME_LEAVE, OnInfoTableNameLeave);
    Messenger.AddListener(UIEvent.INFO_TABLE_TIMETABLE_LEAVE, OnInfoTableTimetableLeave);
    Messenger.AddListener(UIEvent.FONT_LICENCE_MENU_OPENED, OnFontLicenceMenuOpened);
    Messenger.AddListener(UIEvent.CONTROLS_MENU_OPENED, OnControlsMenuOpened);
    Messenger.AddListener(UIEvent.SUB_MENU_CLOSED, OnSubMenuClosed);
  }

  void OnDestroy()
  {
    Messenger<string>.RemoveListener(UIEvent.DOOR_APPROACHED, OnDoorApproached);
    Messenger<string>.RemoveListener(UIEvent.INFO_TABLE_NAME_APPROACHED,
OnInfoTableNameApproached);
    Messenger<string>.RemoveListener(UIEvent.INFO_TABLE_TIMETABLE_APPROACHED,
OnInfoTableTimetableApproached);
    Messenger<string>.RemoveListener(UIEvent.DOOR_LOCKED_OPEN, OnLockedDoorOpen);
    Messenger<string>.RemoveListener(UIEvent.ROOM_CHANGED, OnRoomEntered);
    Messenger<string>.RemoveListener(UIEvent.FLOOR_CHANGED, OnFloorChanged);
    Messenger<string>.RemoveListener(UIEvent.INFO_TABLE_NAME_ACTIVATED,
OnInfoTableNameActivated);
    Messenger.RemoveListener(UIEvent.INFO_TABLE_NAME_DEACTIVATED,
OnInfoTableNameDeActivated);
    Messenger.RemoveListener(UIEvent.DOOR_LEAVE, OnDoorLeave);
    Messenger.RemoveListener(UIEvent.INFO_TABLE_NAME_LEAVE, OnInfoTableNameLeave);
    Messenger.RemoveListener(UIEvent.INFO_TABLE_TIMETABLE_LEAVE,
OnInfoTableTimetableLeave);
    Messenger.RemoveListener(UIEvent.MENU_OPENED, OnMenuOpened);
    Messenger.RemoveListener(UIEvent.MENU_CLOSED, OnMenuClosed);
    Messenger.RemoveListener(UIEvent.MAIN_MENU_OPENED, OnMainMenuOpened);
```

```
    Messenger.RemoveListener(UIEvent.FONT_LICENCE_MENU_OPENED, OnFontLicenceMenuOpened);
    Messenger.RemoveListener(UIEvent.CONTROLS_MENU_OPENED, OnControlsMenuOpened);
    Messenger.RemoveListener(UIEvent.SUB_MENU_CLOSED, OnSubMenuClosed);
  }

  void Start()
  {
    doorButtonPrompt.HidePrompt();
    timetableButtonPrompt.HidePrompt();
    actionStatusPrompt.HidePrompt();
    roomInfoPopup.Hide();
    roomTimetablePopup.Hide();
    controlsMenuPopup.Hide();
    fontLicenceMenuPopup.Hide();
    Messenger.Broadcast(UIEvent.MENU_OPENED);
    Messenger.Broadcast(UIEvent.MAIN_MENU_OPENED); // open the menu at start
  }

  private void OnMenuOpened()
  {
    hudHolder.Hide();
    UnityEngine.Cursor.lockState = CursorLockMode.None;
  }

  private void OnMenuClosed()
  {
    hudHolder.Show();
    openedMenu.Hide();
    UnityEngine.Cursor.lockState = CursorLockMode.Locked;
  }

  private void OnMainMenuOpened()
  {
    mainMenuPopup.Show();
    openedMenu = mainMenuPopup;
  }

  public void OnFontLicenceMenuOpened()
  {
    fontLicenceMenuPopup.Show();
    openedSubmenu = fontLicenceMenuPopup;
  }

  public void OnControlsMenuOpened()
  {
    controlsMenuPopup.Show();
    openedSubmenu = controlsMenuPopup;
  }

  public void OnSubMenuClosed()
  {
    openedSubmenu.Hide();
  }

  /// <summary>
  /// Handles the event of interacting
  /// with the name info table
  /// </summary>
  /// <param name="roomID">name of the room</param>
  private void OnInfoTableNameActivated(string roomID)
  {
    RoomInfo roomInfo = roomInfoSystem.GetRoomInfo(roomID);
    roomInfoPopup.Show(roomInfo.capacity, roomInfo.roomName, roomInfo.roomType,
roomInfo.workplace);
    openedMenu = roomInfoPopup;
  }
```

```
private void OnInfoTableNameDeActivated()
{
  roomInfoPopup.Hide();
}

/// <summary>
/// Handles the event of interacting
/// with the timetable info table
/// </summary>
/// <param name="roomID">name of the room</param>
private void OnInfoTableTimetableActivated(string roomID)
{
  roomTimetablePopup.Populate(roomInfoSystem.GetRoomTimetable(roomID), roomID);
  roomTimetablePopup.Show();
  openedMenu = roomTimetablePopup;
}

/// <summary>
/// Handles the event of closing
/// th timetable info table menu
/// </summary>
private void OnInfoTableTimetableDeActivated()
{
  roomTimetablePopup.Hide();
  roomTimetablePopup.ClearRows();
}

/// <summary>
/// Handles the event of entering a new room
/// </summary>
/// <param name="roomID">name of the room</param>
private void OnRoomEntered(string roomID)
{
  minimapController.UpdateLocationLabel(roomID);
}

/// <summary>
/// Handles the event of entering a new floor
/// </summary>
/// <param name="floorName">name of the floor</param>
private void OnFloorChanged(string floorName)
{
  minimapController.UpdateFloorLabel(floorName);
}

/// <summary>
/// Handles the event of approaching a door
/// </summary>
/// <param name="message">message to display</param>
private void OnDoorApproached(string message)
{
  doorButtonPrompt.ShowPrompt(message);
}

/// <summary>
/// Handles the event of approaching a name info table
/// </summary>
/// <param name="message">message to display</param>
private void OnInfoTableNameApproached(string message)
{
  doorButtonPrompt.ShowPrompt(message);
}

/// <summary>
/// Handles the event of approaching a timetable info table
```

```
    /// </summary>
    /// <param name="message">message to display</param>
    private void OnInfoTableTimetableApproached(string message)
    {
      timetableButtonPrompt.ShowPrompt(message);
    }

    /// <summary>
    /// Handles the event of interacting with a locked door
    /// </summary>
    /// <param name="message">message to display</param>
    private void OnLockedDoorOpen(string message)
    {
      actionStatusPrompt.ShowPrompt(message);
      StartCoroutine(HideActionStatusPromptAfterSeconds(3));
    }

    private void OnDoorLeave()
    {
      doorButtonPrompt.HidePrompt();
    }

    private void OnInfoTableNameLeave()
    {
      doorButtonPrompt.HidePrompt();
    }

    private void OnInfoTableTimetableLeave()
    {
      timetableButtonPrompt.HidePrompt();
    }

    /// <summary>
    /// A coroutine to slightly delay
    /// the hiding of action status prompt
    /// </summary>
    /// <param name="seconds">how many seconds to delay for</param>
    /// <returns>IEnumerator as usual for coroutines</returns>
    private IEnumerator HideActionStatusPromptAfterSeconds(float seconds)
    {
      yield return new WaitForSeconds(seconds);
      actionStatusPrompt.HidePrompt();
    }

}
```

Zdrojový kód skriptu UIController (některé komentáře byly vynechány).