

STŘEDNÍ PRŮMYSLOVÁ ŠKOLA ELEKTROTECHNIKY A INFORMATIKY,
OSTRAVA, PŘÍSPĚVKOVÁ ORGANIZACE



Vybrané techniky z Unreal Enginu

Richard Dluhoš

Vedoucí práce: Ing. Martin Němec, Ph.D.

Obor: Informatika

Školní rok: 2024/2025

Zadání maturitní práce

Jméno a příjmení žáka: Richard Dluhoš

Třída: I4C

Školní rok: 2024/2025

Obor vzdělání: 18-20-M/01 Informační technologie

Vedoucí maturitní práce: Ing. Martin Němec, Ph.D.

Téma: **3. Vybrané techniky z Unreal Enginu**

Zadání maturitní práce

Cílem práce je demonstrovat možnosti Unreal Engine 5 vytvořením 3D dobrodružné hry, ve které hráč plní úkoly, řeší hádanky a postupuje hrou prostřednictvím interakce s herním prostředím. Prostředí hry bude navrženo tak, aby zde se využilo na základě prostor naší školy. Hra bude nenásilná, vhodná pro studenty střední školy, a zaměřena na ověřování jejich znalostí na odpovídající úrovni. Jedná se o společné zadání pro dva studenty, kteří budou na hře pracovat společně. Při vytváření assetů (modely, materiály atd.) rozdělí a popíše autor pouze tu část na které pracoval. Mimo modely se ve hře bude autor této části zabývat tvorbou GUI a dialogových systémů, ovládání a chování NPC postav ve hře, tvorbou dynamických assetů a jejich animováním.

Hráč bude postupně plnit úkoly a řešit hádanky, čímž bude postupovat příběhem. Herní mechaniky a příběh budou průběžně konzultovány s vedoucím práce.

Vzhledem k tomu, že se na vývoji hry budou podílet dva studenti, v teoretické části popište konkrétní části projektu, na kterých jste pracovali.

Způsob zpracování a pokyny k obsahu a rozsahu maturitní práce:

Písemná práce bude obsahovat popis jednotlivých částí, které byly v praktické části zpracovány, včetně základní teorie. Rozsah písemné dokumentace – minimálně 15 normostran. Všechny dokumenty budou vypracovány ve verzích programů, které škola vlastní nebo ve volně šířitelných programech.

Odevzdána bude 1x elektronická verze a 1x tištěná verze vaší práce. Vytvořte archiv s názvem *Třída_Příjmení_CísloPráce.zip*, který bude obsahovat veškerou dokumentaci v elektronické formě {vedoucí práce specifikuje, co dalšího má být součástí archivu (např. zdrojové kódy, ...)} a odevzdaje ho přes aplikaci Google Classroom.

K obhajobě maturitní práce si připravte elektronickou prezentaci o minimálním rozsahu 10 snímků.

Délka obhajoby maturitní práce před zkušební maturitní komisí je 15 minut.

U prací vypracovaných týmově

- každý člen týmu má přesně konkretizovaný svůj podíl na daném úkolu
- každý člen týmu se musí podílet na práci ve všech sledovaných parametrech, míra podílu může být různá.

Počet vyhotovení maturitní práce: 1 vyhotovení

Termín zadání maturitní práce: 15. listopad 2024

Termín odevzdání maturitní práce: 31. březen 2025

Ostrava 1. listopadu 2024

Ing. Zbyněk Pospěch
ředitel školy

Prohlášení

Prohlašuji, že předložená práce je mým původním dílem, které jsem vypracoval samostatně. Veškerou literaturu a další zdroje, z nichž jsem při zpracování čerpal, v práci řádně cituji a jsou uvedeny v seznamu použité literatury a zdrojů informací.

V Ostravě dne

Richard Dluhoš

Licenční ujednání

Ve smyslu §60 autorského zákona č. 121/2000 Sb. poskytuji, Střední průmyslové škole elektrotechniky a informatiky, Ostrava, příspěvkové organizaci, Kratochvílova 1490/7, 702 00 Ostrava, bezplatně oprávnění k výkonu práva (licenci) ke školnímu dílu (maturitní práci) užít v rozsahu a způsoby uvedenými v §12 až 23 autorského zákona.

Souhlasím / Nesouhlasím se zveřejněním díla v rámci školní počítačové sítě pro potřeby studentů a zaměstnanců školy a pro potřeby výuky v souladu s §35(3) autorského zákona.

Souhlasím / Nesouhlasím s použitím práce k propagaci školy.

V Ostravě dne

Richard Dluhoš

Anotace

Tato dlouhodobá práce se zabývá vytvářením prostředí naší školy, grafických rozhraní, dialogových systémů a nehráčských postav (NPC). Cílem je vytvořit hru v dvoučlenném týmu zasazenou v prostorech naší školy, kterými bude hráč procházet a řešit hádanky a úkoly. Hra je vytvořena v herním enginu Unreal Engine 5 a obtížností by měla odpovídat znalostem základní a střední školy. Na vytvoření assetů byly použité programy *Blender* a *Adobe Substance 3D Painter*, na uživatelská rozhraní program *Aseprite* a vytvoření hry engine *Unreal Engine 5*.

Teoretická část se skládá z popisu jednotlivých programů a co pro vývoj her nabízí a v praktické části je popsán způsob použití.

Praktickým výstupem je tedy zkompilovaná hra, zdrojový kód a všechny projekty assetů.

Klíčová slova

Dialogové systémy; 3D grafika; grafické uživatelské rozhraní; Unreal Engine 5; texturování

Poděkování

Chci poděkovat všem, kteří mi poskytli cenné rady a pomoc v průběhu realizace této práce, především vedoucímu práce Ing. Martinu Němcovi Ph.D, mému kolegovi, který se podílel na druhé části výsledné práce, Stanislavu Jančovi a Adamu Gregořicovi.

Obsah

Seznam použitých symbolů a zkratek	8
1 Úvod	9
2 Teorie	10
2.1 3D grafický program <i>Blender</i>	10
2.2 Texturovací program <i>Adobe Substance 3D Painter</i>	13
2.3 2D grafický program <i>Aseprite</i>	17
2.4 Vývoj her v <i>Unreal Enginu</i>	18
3 Praktická část	20
3.1 Tvorba assetů v programu <i>Blender</i>	21
3.2 Texturování assetů v programu <i>Substance 3D Painter</i>	29
3.3 Práce v <i>Unreal Engine 5</i>	31
3.4 Tvorba grafických uživatelských rozhraní v programu <i>Aseprite</i>	43
4 Závěr	46
Seznam obrázků	50

Seznam použitých zkratok a symbolů

PC	– Personal Computer
GUI	– Graphical User Interface
NPC	– Non-Player Character
AI	– Artificial Intelligence
PBR	– Physical Based Rendering

Kapitola 1

Úvod

Vývoj her začal již koncem padesátých letech 20. století v laboratořích a na jejich zobrazování nebyly používané monitory nebo televize jako dnes, ale osciloskopy dostupné v laboratořích, které umožňovaly jen primitivní zobrazování. Mezi první hry patřily Tennis for Two, Spacewar! a Pong. Vizuálně byly velmi jednoduché kvůli nemožnosti hry dobře zobrazit a nízkému výkonu tehdejšího hardwaru, který se postupem času vyvíjel. Při spojení moderního hardwaru a herních enginů můžeme nyní tvořit velmi realistická prostředí. Jeden z těchto enginů je Unreal Engine, který je v tomto ohledu světovou špičkou. Byl vytvořen již koncem devadesátých let 20. století a dnes je díky funkcím, jež umožňuje, jedním z nejpoužívanějším. Dostatečný hardware umožňuje tvořit dobře vypadající hry i laikům, kteří si mohou assety stáhnout z různých tržišť na internetu jako Fab nebo Sketchfab. Pokud ale chce použít objekty, které na marketplacech nenašel, musí si je vytvořit sám v programech jako Blender. Celý program a všechny jeho funkce jsou od roku 2002 open-source [1], tedy všechn kód je otevřen pro veřejnost, a je zdarma, na rozdíl od populární alternativy Autodesk Maya [2].

Použitím těchto programů a dalších, jsme měli za cíl vytvořit interaktivní hru zasazenou v naší škole pro žáky základní a střední školy s gameplayem zaměřeným na luštění hádanek, řešením příkladů z oborů fyziky a matematiky a odpovídáním na jiné otázky.

Kapitola 2

Teorie

V této práci se zabývám tvorbou scén v *Unreal Engine* skládajících se převážně ze statických 3D modelů prostředí, objektů v nich umístěných a grafických uživatelských rozhraní. Proces tvorby modelů je složen z modelování, texturování a importování do projektu v *Unreal Engine*, programování ve vizuálním programovacím jazyku *Blueprint Visual Scripting*, dále jen *Blueprints* zaměřené na vizuální stránku hry, a problémy, na které jsem při práci s nimi narazil.

Teoretická část se skládá z popisů programů a pojmu nutných pro porozumění praktické části práce.

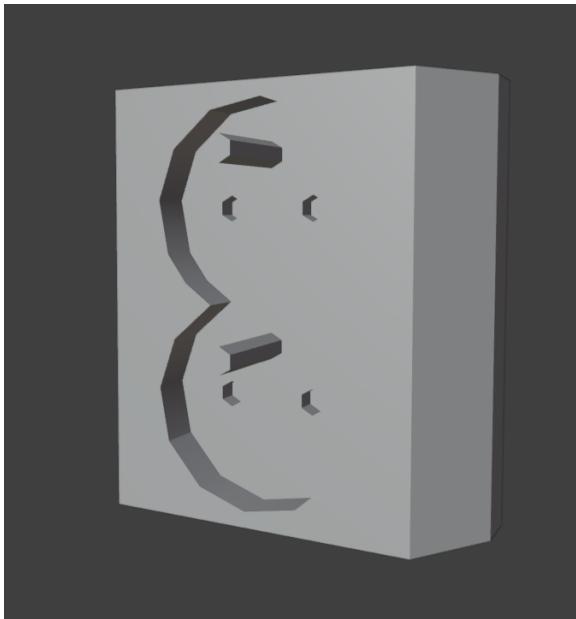
2.1 3D grafický program *Blender*

Blender je program na vytváření hlavně modelů skládající se ze sítí polygonů, neboli meshů. Mesh se skládají ze stran, hran a vrcholů. Ačkoli můžou strany mít teoreticky neomezený počet vrcholů, při exportu se strany meshu převedou na trojúhelníky. Uživatelé ale mohou pracovat i s Bézierovými křivkami, NURBS a Metaballs. Krom vytváření modelů dokáže modely rozpohybovat pomocí armatur, vytvořit modelům materiály, nebo simulovat fyziku u tekutin, měkkých a tvrdých objektů nebo látek. Především díky jeho dostupnosti (je open-source, tím pádem zdarma), ale také flexibilitě a obsažení mnohých nástrojů je velmi využíván mnohými herními vývojáři a grafiky [3].

Blender umožňuje použití mnoha technik modelování pro vytváření komplexních 3D modelů, mezi které patří:

- **Polygonové modelování** - Uživatelé si vyberou jednoduchý geometrický tvar jako krychle, koule, válec, a jiné; který poté upravují pomocí funkcí jako *Extrude*, *Rotate*, *Scale*, a další. Tento způsob se využívá zejména pro hard surface modeling, tedy modelování pevných objektů, kterým se nemění tvar jako například budovy nebo nábytek.

- **Sculpting** - Pro organic surface modeling, modelování živých, organických věcí jako lidí nebo zvířata, se používá sculpting. Funguje podobně jako sochařství a na práci mají uživatelé dostupné mnohé nástroje jako draw, smooth, grab, pull, pinch a jiné. Po vytvoření modelu tímto způsobem je ale nepoužitelný pro využití ve hře. Je nutné provést proces retopology, který vytvoří optimalizovaný mesh polygonů a zachová původní tvar objektu.
- **Geometry Nodes** - Procedurální způsob tvorby modelů, který funguje na principu vizuálního programování. Jsou podobné softwaru *Houdini*.
- **Python scripts** - Lze také vytvořit skripty a add-ony v programovacím jazyku *Python* pomocí *Blender API*. Těmito skripty lze manipulovat s Blenderem stejně jako přes GUI. Takto lze automatizovat některé repetitivní činnosti [4].



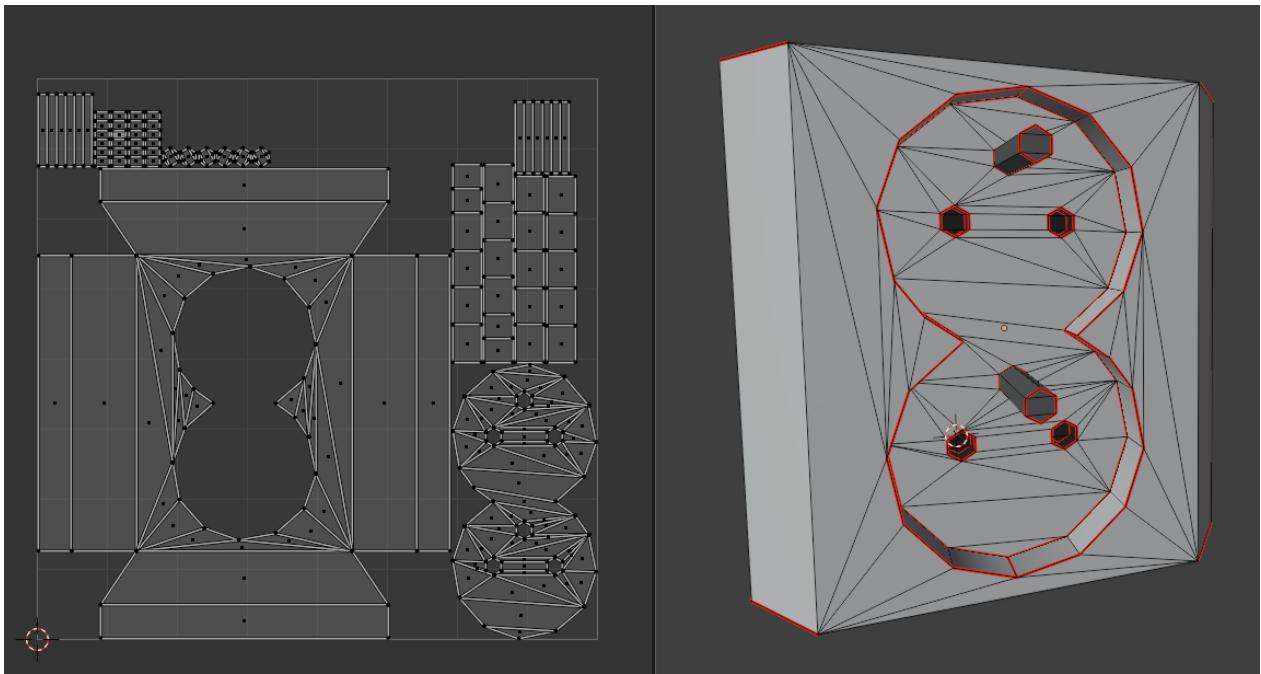
Obrázek 2.1: Model zásuvky vytvořen polygonovým modelováním



Obrázek 2.2: Základní tvar modelu jablka vytvořen sculptingem

Důležitou částí každého modelu je také UV mapa, kterou může uživatel modelu upravit bez použití externího softwaru. Je taktéž nedílnou součástí workflow vytváření modelů a to nejen pro vývoj her, ale i pro použití interně v Blenderu pro render.

Pro nastavení UV mapy, která splňuje naše požadavky, je použit proces UV Unwrapping, při kterém se většinou 3D objekt „rozbalí“ do 2D prostoru. Existují i 3D UV mapy, které ale v této práci nevyužiji. Mapování lze dělat ručně, nebo k tomu slouží metody jako Angle Based Unwrap, Smart UV Project nebo Lightmap Pack.



Obrázek 2.3: UV Mapa modelu zásuvky

Další vlastnosti 3D modelů jsou normály. Jsou to v základu kolmé vektory k polygonům. Podle úhlu mezi nimi a zdrojem světla rendering engine určuje, jak světlý nebo tmavý má polygon být. Jsou taktéž důležité pro určení, které polygony kamera „vidí“. Takto nemusí rendering engine zbytečně renderovat polygony, které nebudou ve výsledném snímku vidět. [5]

Jednou z možností animace v *Blenderu* jsou skeletal meshe. Je to technika, při které uživatel vytvoří kostru, kterou poté přidělí meshi a každé kosti nastaví jakou část modelu má ovlivňovat. Dalším způsobem animace jsou Shape Keys. Animovaný model má tvar základní a několik dalších, u kterých uživatel model mění. Tento způsob se často používá u animace tváří.

Mezi výhody *Blenderu* patří:

- **Komplexnost** - *Blender* disponuje řadou nástrojů pro modelování, texturování, animaci, renderování výsledných modelů nebo compositing.
- **Podpora** - za víc než tři dekády existence vytvořili uživatelé řadu tutoriálů nebo samotní vývojáři dokumentaci. [6]
- **Možnost rozšíření** - Pokročilí uživatelů mohou měnit zdrojový kód *Blenderu* pro své specifické potřeby, případně tvořit pluginy v Pythonu.

A nevýhodou *Blenderu* je:

- **Obtížnost** - Jako většina složitějších 3D modelovacích programů je *Blender* vcelku těžký na naučení pro nováčky.

2.2 Texturovací program *Adobe Substance 3D Painter*

Adobe Substance 3D Painter, dále jen *Substance*, je profesionální program pro texturování 3D modelů. Uživatelé s ním mohou své modely přiblížit realitě použitím zahrnutými funkcemi. Textury nebo mapy jsou nedílnou součástí každého modelu, jsou to tedy 2D obrázky aplikované na povrch 3D objektu. Používají se jak z vizuálních, tak optimalizačních důvodů.



Obrázek 2.4: List *Smart materiálů* v *Substance 3D Painteru*

Standardem napříč programy pro realisticky vypadající materiály jsou shadery typu PBR, tedy Physical Based Rendering. [7]

Některé mapy, součástí PBR materiálu jsou:

- **Diffuse (Base Color)** - Základní textura, která určuje barvu povrchu. [8]
- **Normal** - Normálová mapa určuje způsob, jak světlo interaguje s povrchem objektu. Používá se na přidání detailů bez tvoření zbytečné geometrie. Ačkoli má stejné jméno s normály, jedná se o rozdílné věci.
- **Roughness** - Roughness mapa určuje drsnost povrchu, tedy jak rozptýlené nebo ostré budou odrazy světla na povrchu modelu.
- **Metallic** - Mapa metallic určí, zda je povrch z kovu nebo ne. Ačkoli je možné nastavit hodnoty v intervalu od nuly do jedničky, ve většině případů je lepší mít tento atribut nastaven buď na nulu nebo jedničku, protože skutečné materiály budou jsou, nebo nejsou kovy.
- **Ambient occlusion** - Ambient occlusion mapa simuluje jemné stíny na místech, kde by světlo mělo dopadat méně jako rohy a hrany modelu. [9]

Existuje ještě mnoho typů map jako Height, Emission, Thickness, Curvature, World space normal, Opacity, tyto jsem ale v práci nevyužil.

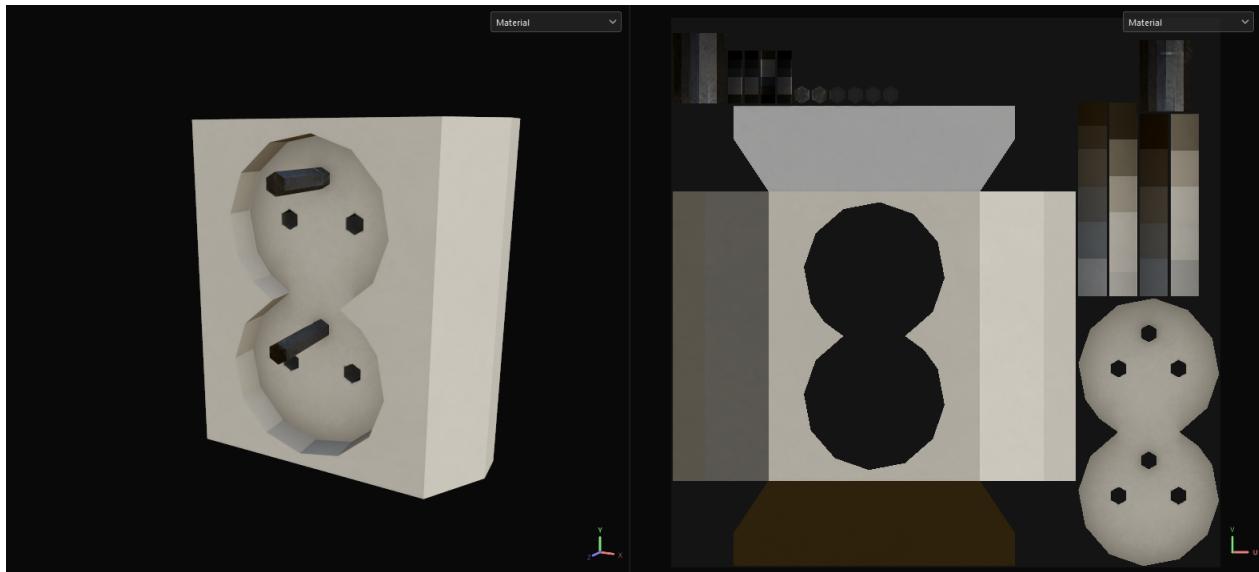
Na texturování používá *Substance* procedurální textury, které uživatel na model aplikuje a on je v reálném čase renderuje za účelem ukázky výsledku. Pro umožnění komplexnějšího setupu textur a vyšší kontrolou nad výsledkem má funkci vrstev, efektů a masek.

Mezi výhody *Substance 3D Painteru* patří:

- **Výsledky blízké realitě** - Pokročilé funkce texturování umožňují tvorbu materiálů, které se blíží realitě.
- **Flexibilita** - Snadně lze s texturami a maskami manipulovat a tím dosáhnout výsledku.
- **Podpora pro *Unreal Engine*** - Při vytváření projektu lze zvolit template pro Unreal Engine, textury tedy budou exportovány ve správném formátu.

Ale mezi jeho nevýhody naopak patří:

- **Cena** - *Substance* je v základu placený software, ale studenti a vyučující mají možnost ověřit status studenta, případně vyučujícího a mít přístup k programu na rok zdarma. Má dvě placené verze - perpetuální licence na herním online tržišti *Steam* [10] a předplatné na stránkách *Adobe*.[11]
- **Náročnost na výkon počítače** - Na bezproblémovou práci je potřeba počítač s výkonným hardwarem. [12]



Obrázek 2.5: Nateexturovaný model zásuvky

Při práci s modely, jež vyžadují použití 4K textur, mohou dosahovat projekty s oněmi modely několik desítek MB, což se týká i výsledných textur. Nekompresované obrázky s rozlišením 4096×4096

pixelů mají velikost až padesát MB. Díky použití formátů s kompresí lze výslednou velikost snížit na přijatelnějších pár MB. Toto tím pádem snižuje jak celkovou velikost hry, tak potřebnou operační paměť GPU, případně možnost mít více načtených textur.

 patro2-A708_DefaultMaterial_BaseColor.png	10/02/2025 20:29	PNG File	7,274 KB
 patro2-A708_DefaultMaterial_Normal.png	10/02/2025 20:29	PNG File	2,695 KB
 patro2-A708_DefaultMaterial_OcclusionRoughnessMetallic.png	10/02/2025 20:29	PNG File	5,145 KB

Obrázek 2.6: Velikost souborů 4K textur

Tento problém lze z části eliminovat použitím přiměřeně velkých textur pro každý model. Obecně se toto označuje jako texel density, neboli množství pixelů textury na jednotkovou plochu modelu. Použití velkých textur na malé objekty zbytečně zatěžuje počítač bez lepšího výsledku, ale naopak použitím malé textury na velkém objektu dochází ke ztrátě potřebných detailů [13].

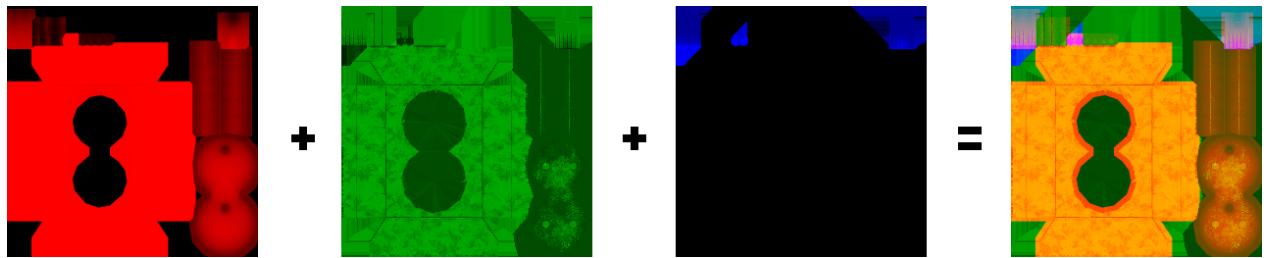
Je tedy potřeba udržovat balanc mezi dostatečně velkou texturou na zobrazení detailů, ale ne tak velkou, že by se zbytečně plýtvalo výkonem počítače. Také je potřeba vzít v úvahu, zda se na objekt bude hráč dívat zblízka, nebo z dálky. Čím blíž se hráč může k objektu přiblížit a prohlédnout si jej, tím detailnější by textura měla být.



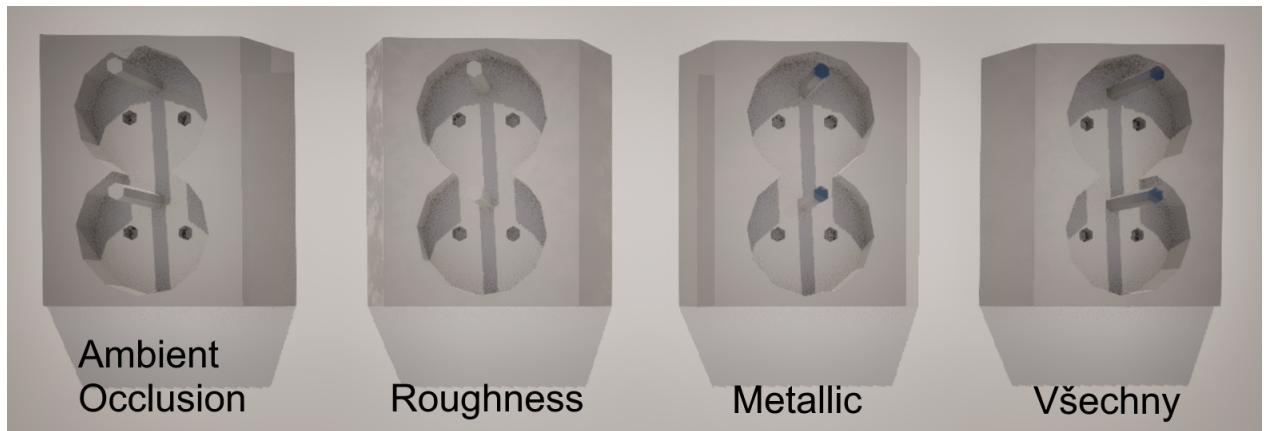
Obrázek 2.7: Porovnání Texel Density [13]

V případě, že by se tímto způsobem nanášely textury na větší objekty jako třeba prostory scén, výsledné textury by měly velikost více než 16K, takže se používají jiné způsoby texturování.

Další optimalizaci, kterou využívá nejen *Unreal Engine*, ale i jiné softwary, které pracují s 3D grafikou, je zakódování Roughness, Ambient Occlusion a Metallic map do jednoho souboru. Původně jsou tyto textury grayscale, tedy každý pixel má hodnotu v rozsahu od nuly do jedné. Této vlastnosti se využije při nastavování map do tří barevných kanálů obrázku R, G a B. Výsledkem je jeden soubor, čímž se v případě větších textur výrazně zredukuje celková velikost modelu a množství operační paměti grafické karty při vykreslování materiálů.



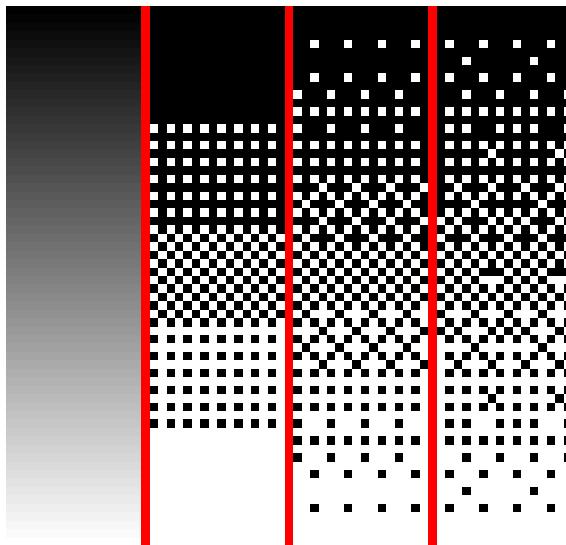
Obrázek 2.8: Spojení map Ambient Occlusion (červená), Roughness (zelená) a Metallic (modrá) do jedné



Obrázek 2.9: Assety s jednotlivými mapami a se všemi v *Unreal Engine*

2.3 2D grafický program *Aseprite*

Aseprite je open-source, 2D grafický program specializující se na pixel-art a animaci [14]. Je často používán herními vývojáři a grafiky zaměřující se na retro styl. Kromě základních funkcí, které má v podstatě každý grafický program jako vrstvy, pero, guma, přímky a křivky, disponuje *Aseprite* funkce jako například přechody s možností pro dithering, techniku stínování použitím jen dvou barev, podpora tilemap, možnost zvolení předdefinovaných palet jak s tematikou starých konzolí jako Gameboy, NES, Atari 2600, Commodore 64, tak ostatních [15]. Animátorům zase poskytuje funkci onion skin nebo možnost exportu vytvořené animace jako spritesheet [16][17].



Obrázek 2.10: Možnosti ditheringu v programu *Aseprite*

Výhody spojené s používáním *Asepritu* jsou:

- **Široká dostupnost** - Je možné jej používat na operačních systémech Windows, Linux a macOS.
- **Specializované nástroje pro pixel art** - Nástroje byly vytvořeny a jsou optimalizované specificky pro tvorbu pixel artu.
- **Barevné palety** - Tvůrci výrazně pomohou dodržet styl daného díla.

Naopak nevýhoda spojená s *Asepritem* je:

- **Cena** - Zdrojový kód je sice volný ke stažení na stránce GitHub, ale uživatel si jej musí sám zkompilovat. Freeware verze není volně dostupná. Existují sice na internetu návody na kompliaci, nezkušeného uživatele toto ale může odradit a proto jediná jeho možnost je získání zkompilované verze zakoupením na webových stránkách *Aseprite*[18], nebo na herním tržišti *Steam* [19].

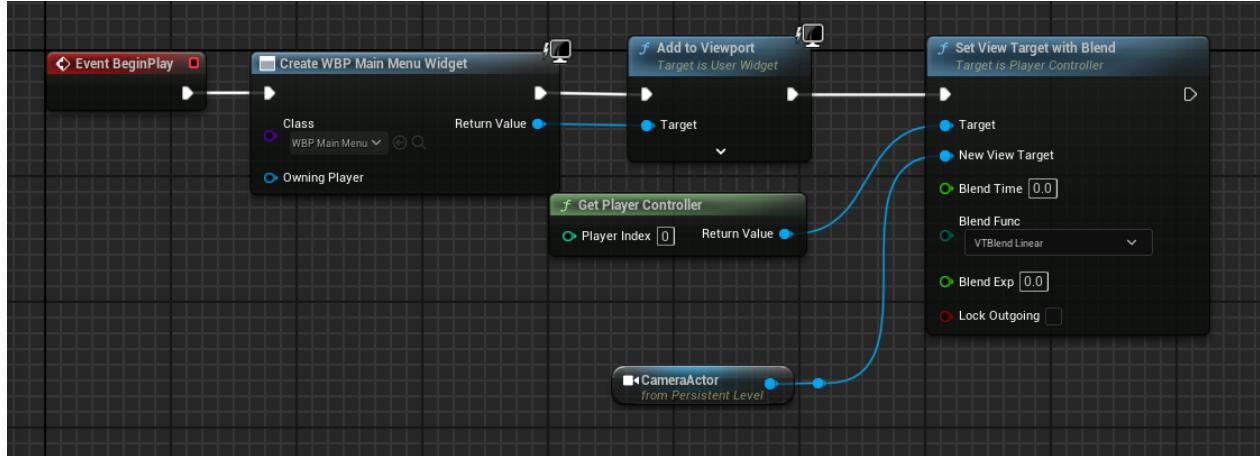
2.4 Vývoj her v *Unreal Engine*

Unreal Engine je herní engine s širokou nabídkou funkcí usnadňující vývoj her. Díky jeho flexibilitě a robustnosti je vhodný pro spoustu žánrů a typů her od malých indie projektů po velké AAA tituly. Ačkoli v něm lze vytvářet 2D hry, normou jsou hry 3D.

Unreal Engine našel také využití mimo tvorbu her, jeho starší verze číslo 4 pomáhá studiím jako ILM a Disney při tvorbě seriálu The Mandalorian. Je součástí systému StageCraft, který jej využívá pro renderování dynamického pozadí v reálném čase, které se zobrazuje na stěnu displejů kolem herců. [20]

Jednou z jeho hlavních částí je programovací jazyk *Blueprints*, který umožňuje jednodušší, ale zato výkonově náročnější, alternativu pro programování v *C++*.

- **Event-driven programming** - *Blueprints* umožňují reagovat na události, jako jsou vstupy od hráče nebo kolize, což usnadňuje implementaci interaktivních prvků.
- **Reusable components** - Vývojáři mohou vytvářet a sdílet komponenty, což zrychluje vývoj.
- **Debugger** - *Blueprints* mají zabudovaný nástroj pro ladění, který pomáhá vývojáři identifikovat sémantické chyby.



Obrázek 2.11: Příklad části *level blueprint*

Všechny části 3D grafiky používané v projektech od modelů, textur, materiálů, animací, prostředí, particle systémů, shaderů po osvětlení se označují jako **assets** [21].

Importování assetů do *Unreal Engine* je zásadním krokem v procesu vývoje. Uživatelé mohou importovat modely, textury a animace přímo z *Blenderu* nebo *Substancu*. Důležité aspekty zahrnují:

- **Optimalizace assetů** - Před importem se snažíme, aby byly modely a textury optimalizované. Lze je ale optimalizovat i přímo uvnitř *Unreal Engine*, například vytvořením více Level of Detail modelů. Tímto se zvýší plynulost hry a sníží zatížení hardwaru.
- **Organizace projektů** - Je dobré importované assety roztrídit do složek tak, abychom se ve struktuře adresářů vyznali. Importované assety nebo vytvořené skripty či třídy se doporučuje pojmenovávat podle zavedených konvencí - prefixů [22].
- **Material Editor** - *Unreal Engine* má zabudován editor materiálů, který umožňuje vytvářet komplexní materiály. Principem je podobný skriptovacímu jazyku *Blueprints*. Podobá se také Shader Editoru v programu *Blender*.

Importované modely, tedy static meshe, můžeme přidávat do „skupin“, **Blueprint actorů**. Lze do nich také přidat komponenty kolizní, zvukové, světla a další [23]. Mohou také obsahovat proměnné, funkce nebo eventy.

Po vytvoření hry je potřeba ji zkompilovat na spustitelnou verzi. *Unreal Engine* podporuje export na různé platformy (Windows, Linux, konzolí a mobilních zařízení).

Mezi výhody použití *Unreal Engine* na tvorbu her patří:

- **Dostupnost** - *Unreal Engine* je také open-source, pro všechny je zdarma k použití, po překročení hrubého příjmu jednoho milionu dolarů ze hry za poslední rok se platí buď 5% nebo fixní cena za jednoho vývojáře. [24]
- **Vysoká grafická úroveň** - *Unreal Engine* má schopnosti vytvořit skvěle vypadající hry.
- **Podpora virtuální a rozšířené reality** - Společně s hrami na klasickém počítači má *Unreal Engine* jednu z nejlepších podpor her na systémy virtuální reality a rozšířené reality.

Zato mezi nevýhody patří:

- **Náročnost na výkon počítače** - Na vývoj her v *Unreal Engine* je nutné mít velmi výkonnou počítačovou sestavu, především tedy grafickou kartu. Potřebný výkon na plynulost výsledné hry záleží hlavně na vývojáři.
- **Různá náročnost vývoje** - Začínající programátoři dokážou pomocí návodů na internetu vytvořit hru, ale i pokročilí uživatelé mohou při tvorbě komplexnějších aplikací a použitím pokročilejších technik narážet na problémy.
- **Horší kvalita dokumentace** - Pro složitější funkce prakticky neexistuje oficiální dokumentace, je tedy nutné hledat v různých zákoutích internetu.

Kapitola 3

Praktická část

Pracovali jsme ve dvoučlenném týmu a mou částí bylo zabývat se hlavně celkovým vzhledem hry od prostředí, kterým bude hráč procházet, assety, jež se v prostředí nachází a se kterými hráč interahuje, až po design grafických rozhraní, se kterými se hráč setká jako inventář nebo ovládací prvky hádanek. V pozdějším stádiu práce jsem začal programovat v systému Blueprints reaktivitu určitých prvků prostředí a NPC. Můj kolega zodpovídá tedy za tvorbu hádanek a obecně většinu programování.

Na version control našeho projektu jsme používali GitHub. S tím ale přišly ve chvíli, když jsem také začal upravovat Blueprint soubory, problémy. V případě úpravy stejného souboru více přispěvateli začne docházet k merge conflicts [25].

Git ale neumožňuje uzamykání jednotlivých souborů, aby jen jeden uživatel mohl v daný moment soubor upravovat, jako jiné version control systémy. Celkem nepraktickým řešením bylo domlouvání se, zda byl daný soubor upraven druhým. Naštěstí se tento problém vyskytoval celkem vzácně, jelikož jsme často pracovali na rozdílných částech projektu.

Na komunikaci mezi s sebou jsme používali platformu Discord.

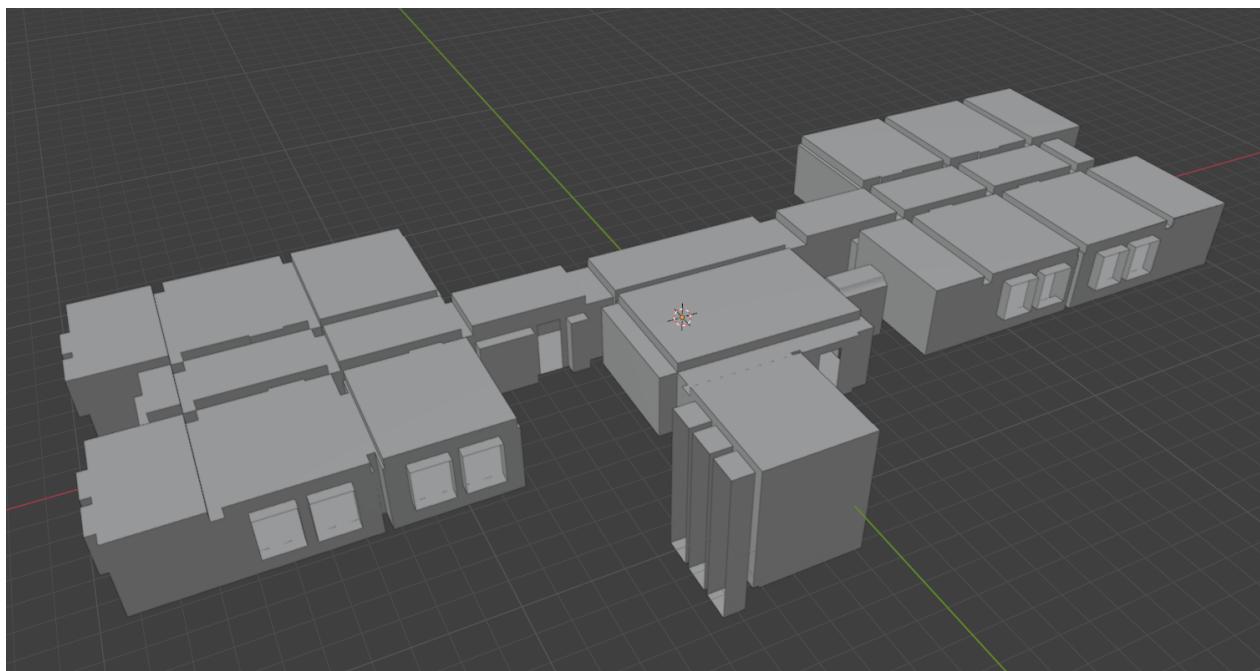
3.1 Tvorba assetů v programu *Blender*

Ačkoli jsem měl na výběr z široké škály programů jako *Blender* a *Autodesk Maya*, *Blender* byl pro mě jasnou volbou, díky v tuto chvíli několikaletým zkušenostem. Naopak s programem *Autodesk Maya* nemám dostatek zkušeností na efektivní práci.

Proces tvorby modelu se většinou skládá z několika základních kroků:

1. Získání referencí
2. Vytvoření základního meshe podle referencí
3. Optimalizace a úpravy meshe
4. Vytvoření UV mapy
5. Texturování

V této části tedy tyto kroky popíšu.



Obrázek 3.1: Model patra A800

Před samotným začátkem modelování jsem si musel vyfotit reference objektu, který chci vy-modelovat. Jelikož jsem vytvářel prostory a objekty mi dostupné, tak jsem je vyfotil a naměřil. Reference jsou důležité pro představení tvaru a materiálu při tvoření a výsledné porovnání virtuální verze se skutečnou. Pro lepší vizualizaci tvaru jsem se snažil vyfotit objekt z více úhlů.

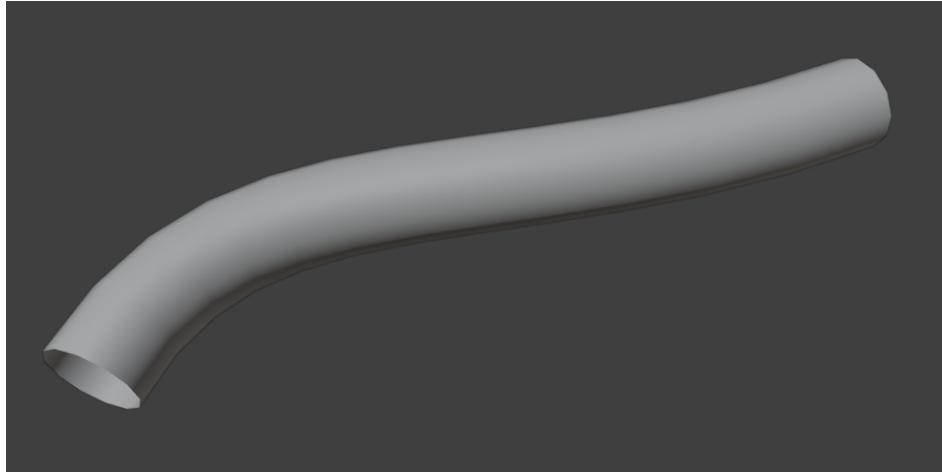


Obrázek 3.2: Příklad reference

Se mnou vyfocenými, případně jiným způsobem získanými referencemi, jsem se naučil zacházet dvěma způsoby - použitím přímo v *Blenderu* jako nárys, půdorys a bokorys; v programu *PureRef*, který umožňuje mít velké množství obrázku v jednom okně nad všemi ostatními, nebo jednoduše je mít otevřené v prohlížeči fotek *ImageGlass* na druhém monitoru. V pozdějších fázích projektu jsem používal nejvíce variantu číslo tři. Jakmile mám reference, mohu začít modelovat.

Vytvoření základního meshu se skládá z:

1. Přidání základního tělesa - *Blender* má na výběr z několika různých těles jako krychle, koule, válec, torus, křivky a mnoho dalších. Křivky také mohu převést na mesh a přidáním průměru z nich vytvořit prostorové objekty, jak je znázorněno na obrázku 3.3, čímž je mohu využít s ostatními mesh částmi.



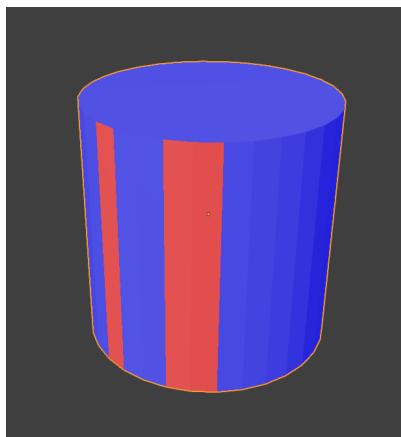
Obrázek 3.3: Prostorová křivka

2. Úprava základního tělesa - Použitím funkcí na manipulaci s meshem jako Scale, Rotate, Extrude, Bevel, Join a spousty dalších se snažím vytvořit objekt blízký tomu skutečnému.

V průběhu modelování si musím dávat pozor na pár věcí, které mají zásadní vliv na výsledný model. Jednou z takových věcí jsou správně nastavené normály. Pro správné zobrazování by měly směřovat ke hráči, takže v případě věcí směrem ven a v případě prostoru dovnitř. Na zkontovalování aktuálních normál modelu slouží funkce *Face Orientation* v kontextovém menu *Viewport Overlays*. Normály směrem ke kamere viewportu se zabarví modře a směrem od kamery červeně. Na opravení takových normálů slouží funkce:

- *Normals -> Flip*, která otočí normály zvolených stran
- *Normals -> Recalculate Outside*, která automaticky nastaví normály zvolených stran tak, aby směřovaly ven
- *Normals -> Recalculate Inside*, která automaticky nastaví normály zvolených stran tak, aby směřovaly dovnitř

Funkce *Recalculate Outside* a *Recalculate Inside* ale nejsou perfektní a občas je nutné normály otočit manuálně.

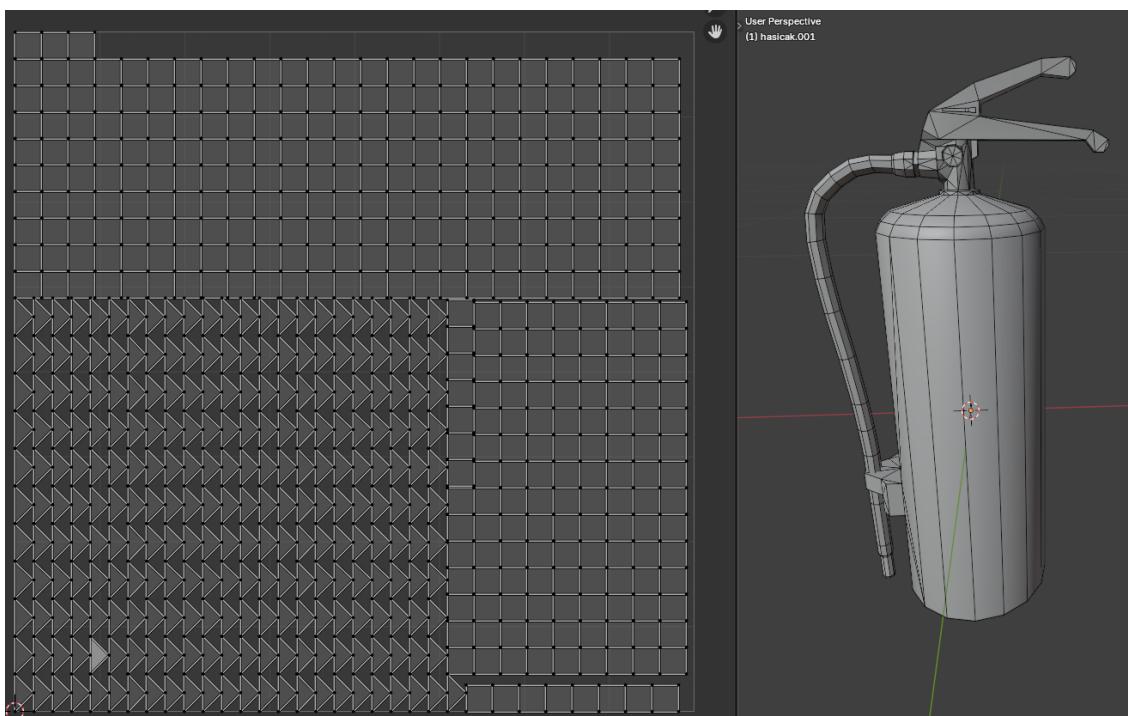


Obrázek 3.4: Ukázka špatně nastavené normály modelu v *Blenderu*



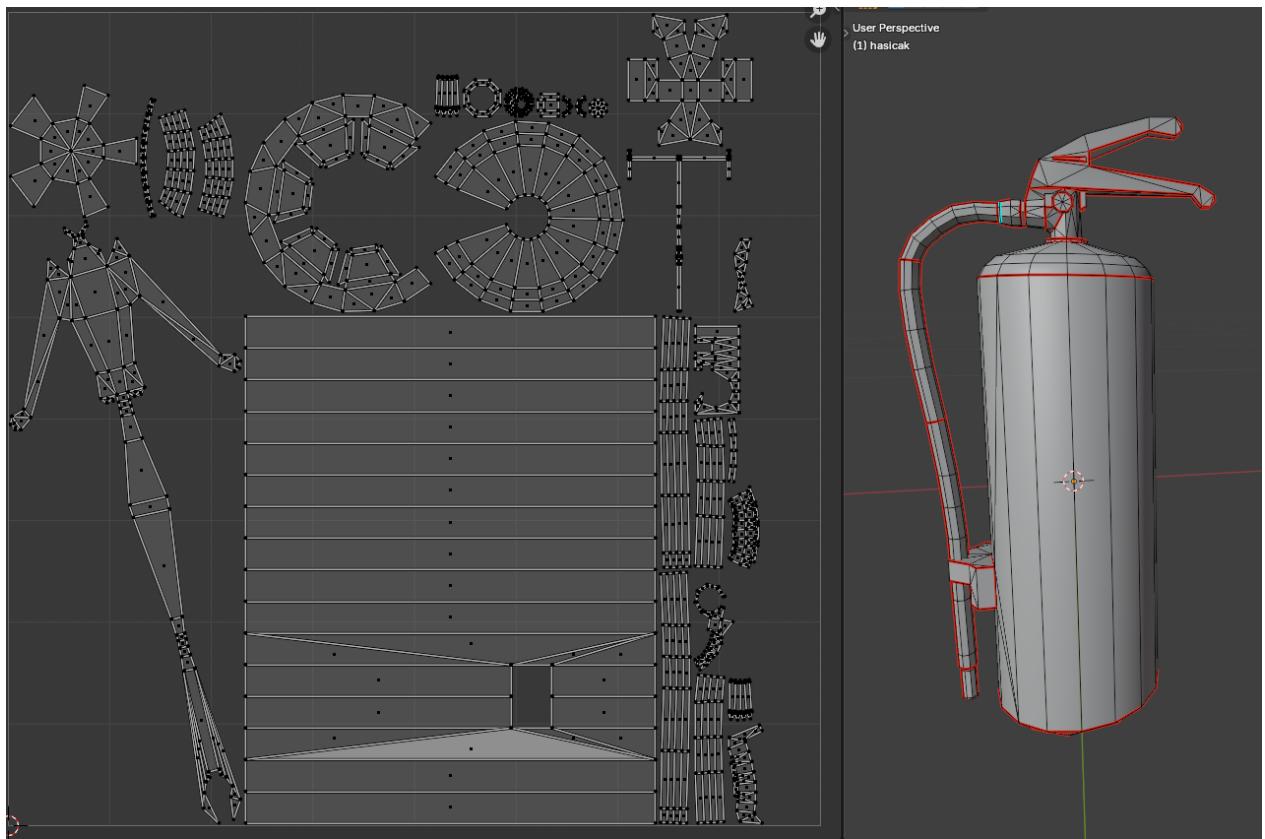
Obrázek 3.5: Důsledek špatně nastavených normálů modelu v *Substance 3D Painteru*

Po vymodelování objektu, který chci texturovat, musím vždy vytvořit UV mapu. Kdybych ale ihned použil mou preferovanou funkci *Angle Based Unwrap*, UV mapa by byla v podstatě nepoužitelná.



Obrázek 3.6: UV mapa modelu bez seams

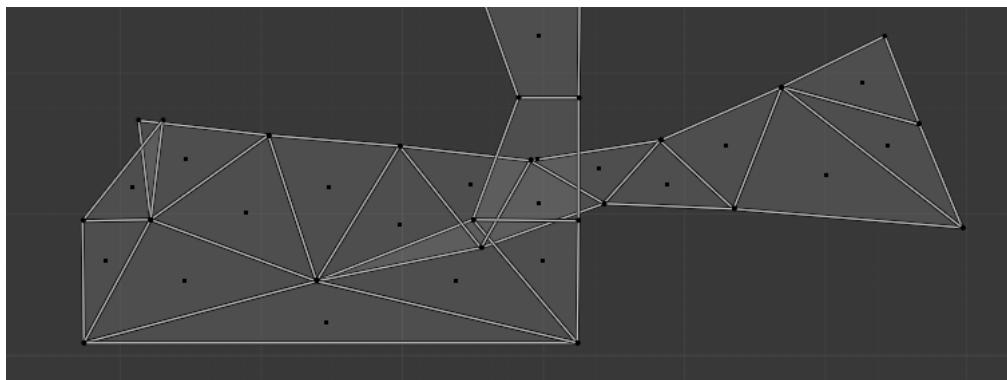
Angle Based Unwrap potřebuje na správnou funkčnost seams, tedy švy, podle kterých model rozdělí. Podle dokumentace je vhodná pro organické tvary, ale já pro ni našel využití i u pevných objektů [26].



Obrázek 3.7: UV mapa modelu s nastavenými seams

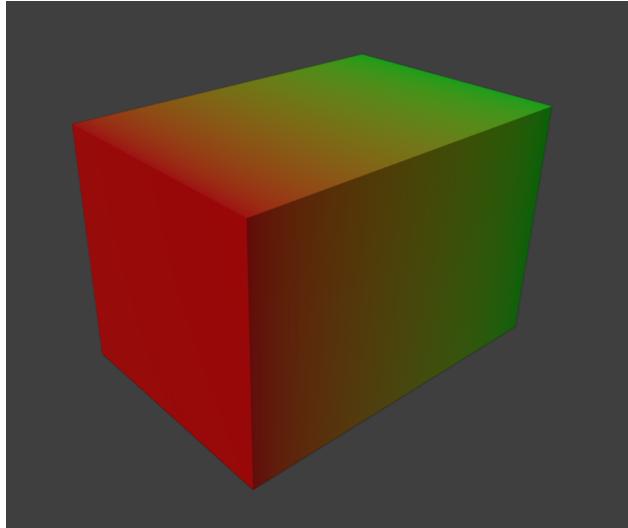
Při přidávání seams se snažím co nejvíce zachovat posloupnost stran a zároveň co nejvíce minimizovat zkreslení UV mapy.

Další věc, na kterou si musím dávat pozor při vytváření UV mapy jsou překrývající se strany, pokud to vyloženě nevyžaduju, což se mi ale zatím stalo jen jednou. V případě, že se mi strany UV mapy překrývají, musím měnit pozice seams, dokud nedosáhnu uspokojivého výsledku.



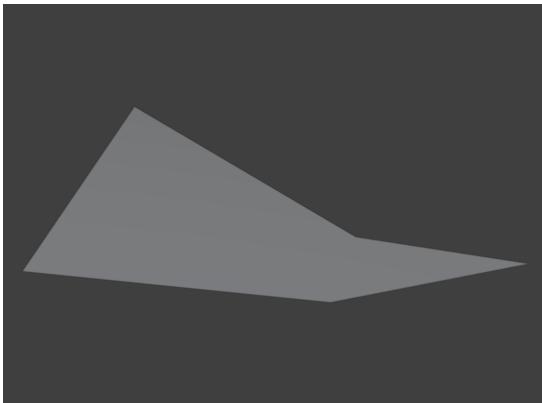
Obrázek 3.8: Překrývající se strany UV mapy

Když chci využít *Vertex Colors* modelu, věc na kterou si musím dávat pozor je jejich správné nastavení. Například když jsem již vytvořený model s nastavenými *Vertex Colors* dál upravil, může se stát, že jsou upravené části „rozpité“, tedy vytvořil se mezi nimi přechod, což není nejlepší pro použití v *Substancu*, který potřebuje barvy jednolité. Nejčastěji tento problém může vzniknout po „rozpuštění“ hran, tedy použitím funkce *Delete -> Dissolve Edges*.

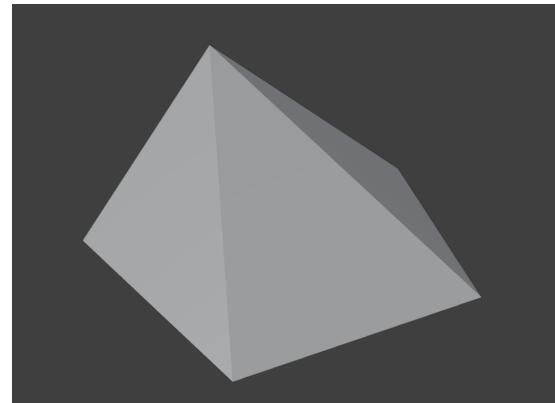


Obrázek 3.9: Přechod *Vertex Colors* modelu

Mezi části modelu, kterou musím zkонтrolovat jsou případné n-gony, neboli strany s více hranami než 4. Výsledný model by se měl skládat čistě z trojúhelníků a případně rovných čtyřúhelníků. N-gony totiž způsobují problémy pro renderovací enginy, nebo komplikují UV mapping. Nemají totiž správně „popsaný“ tvar a zvláště v případě nerovných n-gonů se špatně vykreslují. Tento problém může nastat například při vyplňování děr v modelu s vyšším počtem hran.



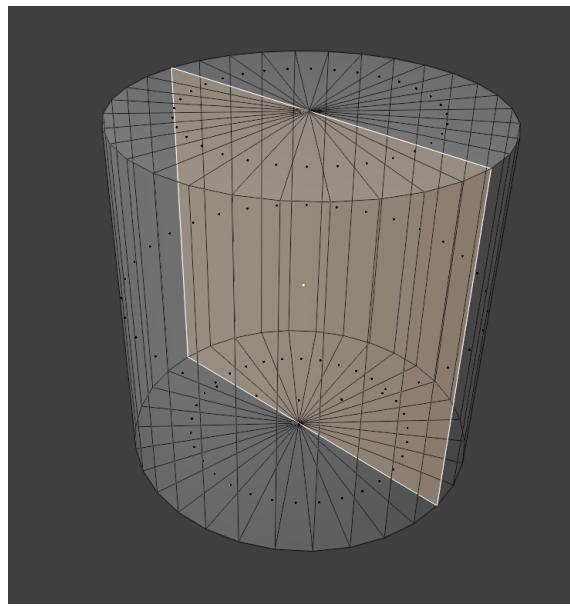
Obrázek 3.10: N-gon (pětiúhelník) s vyzdvíženým vrcholem



Obrázek 3.11: Opravený n-gon skládající se z trojúhelníků

Na výběr n-gonů v meshi existuje funkce *Select -> All by Trait -> Faces by Sides*, která zvolí strany podle námi zadaných parametrů. V tomto případě nastavím počet hran na 4 a typ na „Greater than“. Na následné opravení takových stran lze použít funkce *Face -> Triangulate faces*, která stranu rozdělí na trojúhelníky, nebo stranu se strana rozčlení postupným zvolením jednotlivých bodů a vytvořením mezi nimi hranu funkcí *Join*. Tento způsob je lepší použít v případě, že funkce *Triangulate faces* nedosáhne hezkého celkového tvaru a „toku“ vrcholů a stran.

Dalším problémovou topologií modelu jsou interior faces. Interior faces jsou strany uvnitř modelu, které nelze vidět. Problém je, že modely s těmito stranami nejsou manifold objekty, tedy objekty s geometricky správným tvarem. Tyto strany zbytečně zvyšujou komplexitu modelu, výsledný počet polygonů a zaplňují místo na UV mapě.

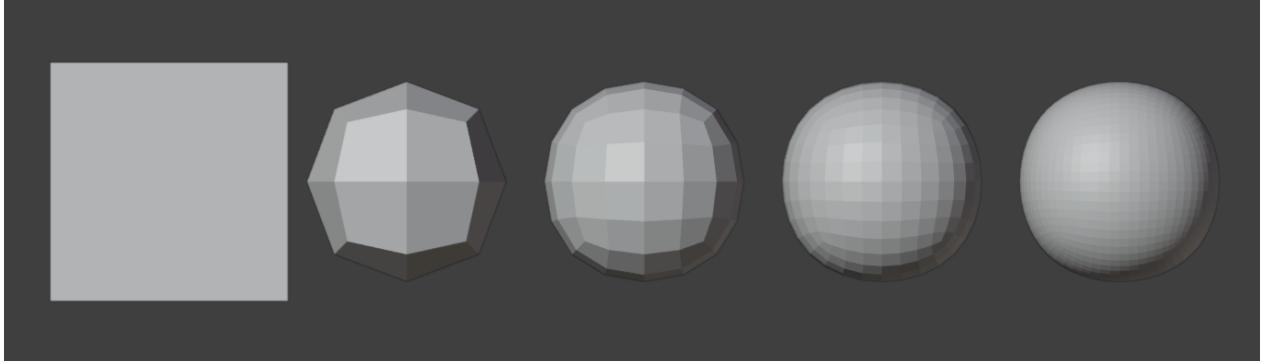


Obrázek 3.12: Strana uvnitř válce

Nejdůležitější částí optimalizace modelu je udržování rozumného počtu polygonů. Ačkoli neexistuje jakási metrika nebo konstanta, která by udávala, jak moc detailní by měl model být, tak model, který by měl být „game-ready“, tedy „připraven pro hru“, by měl mít adekvátní množství polygonů na to, jak velký je, případně jak blízko se k němu může hráč přiblížit a detailně si jej prohlédnout. Na renderování už nejsou tak striktní limity, jako na assety do her, jelikož stačí vyrenderovat jeden snímek, případně snímky animace, jen jednou. Při hraní hry by se jich mělo vyrenderovat alespoň šedesát pro plynulý gameplay.

Nejčastěji se může počet polygonů zvýšit tak, že základ celého modelu je velmi detailní, například válec nebo kruh s vysokým počtem hran; použil jsem moc segmentů při použití funkce *Bevel*, nebo počet cutů při vytváření loop cutů.

Zredukovaní počtu polygonů není jednoduché. Musím vždy přemýšlet nad tím, které hrany můžu odstranit a zároveň jak zachovám co nejlépe tvar modelu. Nejjednodušší způsob, jak snížit komplexitu modelu je ruční redukcí dřív zmíněná funkce *Dissolve Edges*. Objekty s největší komplexitou jsou většinou kulaté jako válce nebo koule kvůli zakřivení jejich povrchů. Pokud ale má objekt aplikovaný *Subdivision Surface* modifier, který celkově modelu rozdělí (subdivide-ne) existující plochy s možností zaoblení, celkem rychle mu může vzrůst počet polygonů podle úrovně rozdělení.



Obrázek 3.13: Krychle s postupně stupňující úrovní *Subdivision modifieru*

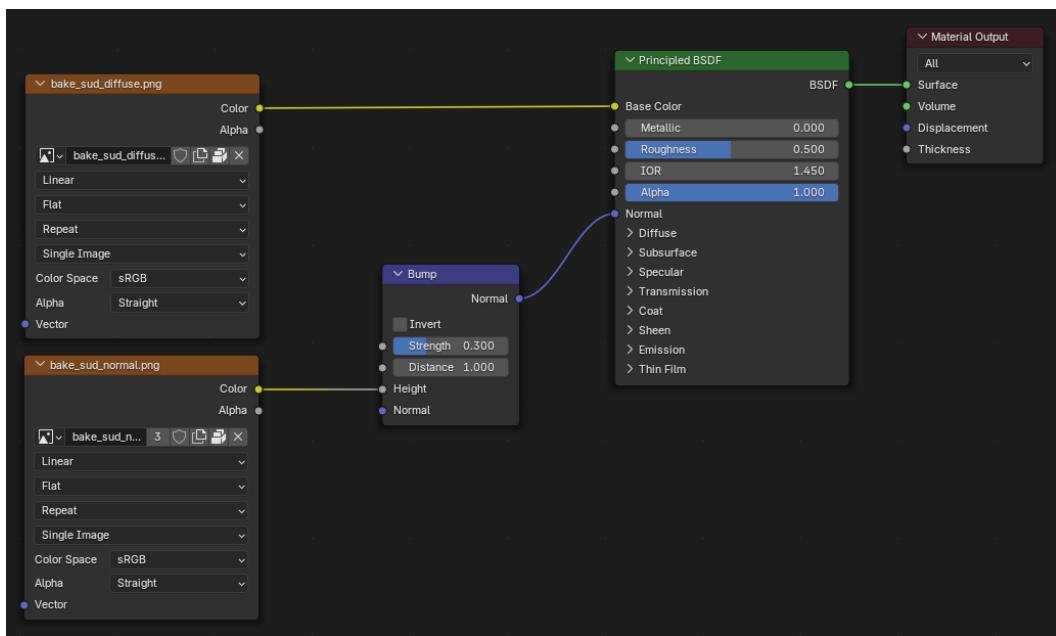
Pro rychlejší vykreslování objektů ve hře lze použít metody *Level of Detail*, kdy objekty vzdálené od kamery použijí verzi s nižším počtem polygonů a naopak pro objekty blízko kamery se použije jejich nejkomplexnější verze, čímž se celkově zrychlí renderování snímků. Toto se používá zejména u otevřených světů, kde může hráč vidět velký počet assetů najednou. Vzhledem ale k celkem malé velikosti našich map jsme tuto metodu nepoužili.

Jakmile mám model hotový, musím jej exportovat. Formát, který podporuje jak *Substance*, tak *Unreal Engine* se nazývá Filmbox, neboli FBX [27]. Modely jsem ale většinou musel exportovat vícekrát. Pro zrychlení jsem si nejdřív uložil do *Operator Presetu*, potřebné nastavení pro smoothing a zahrnutí jen zvolených objektů do výsledného souboru. Později jsem si i vytvořil klávesovou zkratku, která vytvoří dialogové okno a automaticky předvyplní požadované nastavení.

3.2 Texturování assetů v programu *Substance 3D Painter*

Na texturování vymodelovaných assetů lze použít funkce v *Blenderu*. Nejdřív je nutné získat základní texturu, kterou následně mapujeme na vytvořený model. Získat textury lze dosáhnout několika způsoby. Nejčastějším způsobem je stažení z internetu. Existuje několik stránek, které mají zdarma dostupné různé materiály podobné těm v *Substanci* v několika různých rozlišeních. Jednou z takových stránek je třeba Poly Haven [28], kterou jsem používal, než jsem se naučil používat *Substance*. Textury také lze vytvářet z fotek nebo v případě potřeby kompletně custom, tak v grafickém programu jako *Photoshop*, *Gimp* a jiné.

Po získání textur je připojím v *Shader editoru* v jednotlivých materiálech k parametrům shaderu, většinou Principled BSDF, určený pro PBR materiály. Zvolím následně jednotlivé povrchy a přiřadím je k materiálům modelu. Jakmile mám hotové materiály a přiřazené strany, manipuluju s částmi UV mapy vytvořeného modelu podle textur. Jakmile jsem spojen s vzhledem modelu, textury se musí zapéct do jednoho setu textur. Nejdřív vytvořím druhou UV mapou, na kterou je ideální funkce *Lightmap Pack*, která unwarapne síť modelu na co největší plochu, za cenu nenavazujících částí, což není v tomto případě potřeba. Textury poté zapeču a po exportnutí je můžu používat.



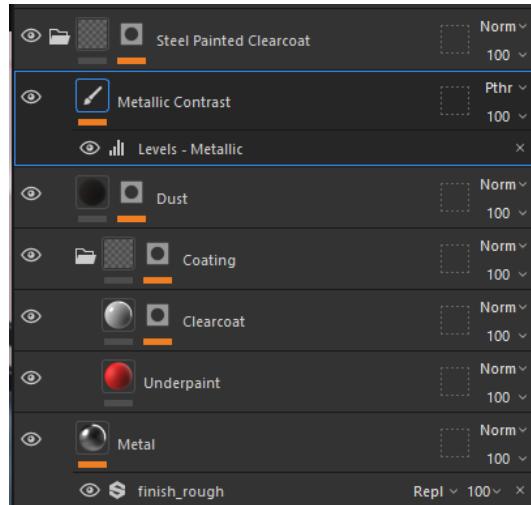
Obrázek 3.14: Jednoduchý PBR material setup s Diffuse a Normal mapou v *Blenderu*

Tímto způsobem mám ale nad výsledkem výrazně nižší kontrolu. Textury nemohu jednoduše upravovat a je to těžší, časově náročnější a složitější, než využití programu jako *Substance*.

Při vytváření projektu si musím dávat pozor na velikost výsledných textur. V případě překliku nebo změny názoru mohu ale později velikost upravit.

Samotný proces texturování se skládá ze čtyř částí:

1. **Baking** - Po importu modelu musím „zapéct“, tedy nechat *Substance* vygenerovat, generické mapy, které ovlivní jak samotný proces texturování, tak výsledné textury. Mezi snad nejdůležitější je ID mapa, podle které přiřazují textury a *Smart materiály*. Ještě před „pečením“ musím nastavit, aby združenem generace ID mapy byly dříve nastavené *Vertex Colors* místo materiálů přiřazených na modelu.
2. **Aplikace Smart materiálů a textur** - Podle reference volím především *Smart materiály* z nabídky na obrázku 2.4. V případě nenalezení odpovídajícího materiálu, který by vypadal jako ten skutečný, vybírám materiál nejbližší k tomu skutečnému a přecházím na krok třetí.
3. **Úpravy aplikovaných materiálů** - Tento krok může být volitelný. Většinou ale pro zvýšení přesnosti s materiélem skutečným musím aplikovaný materiál upravit. Mezi takové úpravy patří především zvětšení a zmenšení textury, změna barvy a úprava masek vrstev materiálů. V případě, že nemohu přímo zmenšit či zvětšit texturu, mohu upravit velikost její masky, čímž se upraví i velikost samotné textury. Krok druhý a třetí opakuji dokud celý povrch modelu není pokryt materiélem.



Obrázek 3.15: Vrstvy *Substance 3D Painter* *Smart materiálu*

4. **Export výsledných textur** - Zkratkou *Ctrl + Shift + E*, nebo kliknutím v dropdown menu *File* na *Export Textures* otevřu dialogové okno, ve kterém nastavím složku, kde se textury uloží a případně velikost výsledných textur. Taktéž můžu vybrat textury, které se exportují.

Mé použití *Substancu* je celkem povrchové. Uživatele mají možnosti poskládat si vlastní *Smart materiály*, protože to jsou v podstatě jen skupiny textur a masek. Na tvorbu komplexnějších *Smart materiálů* slouží další program, který je součástí *Adobe Substance 3D Collection* je *Substance 3D Designer* [29].

3.3 Práce v *Unreal Engine 5*

Mou hlavní náplní práce v *Unreal Enginu* byl celkový vzhled hry, který se skládá jak z použití vytvořených assetů, tvorbou pár interaktivních prvků jako dveře a výtah, tak vizuální stránkou GUI.

Import vytvořených věcí zahrnuje jen „přetažení“ souborů z prohlížeče souborů do *Content Draweru*, případně tlačítkem *Import to* v kontextovém menu *Content Draweru*. Po importu je potřeba textury sestavit do materiálu. Ten si bud vytvořím samostatně, nebo instanci již existujícího materiálu. Materiály mohou mít místo staticky nastavených textur parametry. Rozdílné instance tedy mohou mít jiné textury, ale v základu jsou identický shader. Výhodou je tedy snížený počet výměn shaderů na GPU, což může zlepšit výsledný výkon hry.

Možností jak tento proces zjednodušit je použití *Unreal Engine* pluginu *Gregstr's Importer*. Byl vytvořen specificky pro podobné potřeby na ročník 2024 praxí na VŠB-TUO zaměřených na vývoj her v *Unreal Enginu*. Použitelný je ale jen pro import jednoduchých assetů s jedním zahrnutým modelem v souboru. Při více objektech je spojí do jednoho static meshe, což není v některých případech ideální. Součástí pluginu je také Master Material, který jsem využil a upravil pro mé potřeby pro většinu objektů.

3.3.1 Tvorba prostředí

Samotné modely prostředí se v zásadě od ostatních modelů moc nelišíly. Rozdílné ale byly reference a způsob texturování.

Jakožto reference a v podstatě samotný základ modelů jsem použil půdorysy jednotlivých patrov vytvořené v programu *AutoCAD*. Měly ale pár nedostatků, první byl, že neobsahovaly výšky stropů, musel jsem si je tedy naměřit sám. Druhým nedostatkem byly chyby přímo v půdorysech. Některé přímkы nebyly spojené s ostatními, což mi dělalo menší potíže v jejich pozdějších úpravách v *Blenderu*.

Aby byly použitelné pro import do *Blenderu*, musel jsem je vyčistit od nepotřebných elementů jako kóty, popisky, a podobně, což se později ukázalo jako zbytečné, protože až při exportu jsem si všiml funkce vypnutí vrstev, neboli hladin. Po vyčištění jsem tedy projekty pater exportoval do formátu DXF a byly připravené na import do *Blenderu*.

Blender v základu podporuje import souborů jen s pár základními formáty a formát DXF není jedním z nich. Existuje ale rozšíření s podporou tohoto formátu, které jsem nainstaloval [30].

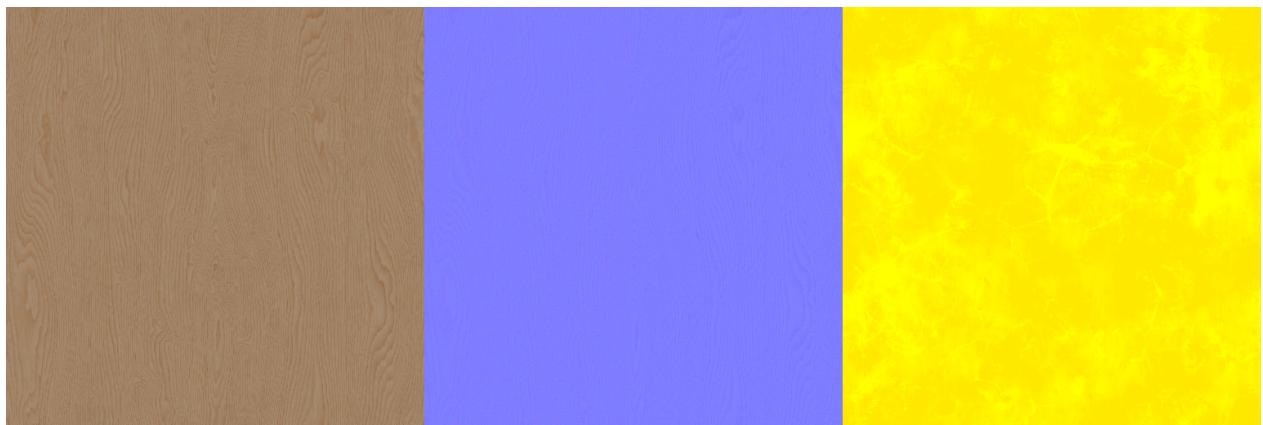
Výsledný model měl v podstatě identický postup jiné modely.

Můj první pokus o texturování nedopadl nejlépe. Použil jsem totiž stejný postup jako u menších modelů. *Substance* ale nepodporuje export textur větší než 8K, takže chodby a učebny nevypadaly ani zdaleka dobře jako ostatní assety. Výsledné prostředí tedy bylo velmi pixelované.



Obrázek 3.16: První verze textur prostředí

Po konzultaci s vedoucím práce jsem zvolil jiný přístup texturování. Zahrnoval export textur již vytvořených v *Substancu* ve velikosti 512x512 tak, aby zaplnila celou plochu, jak je znázorněno na obrázku 3.17. Textury jsem si po exportu importoval do *Unreal Engineu* a po vytvoření *Master materiálu* použil v jeho instancích. Tyto instance měly nastavené všechny místnosti.

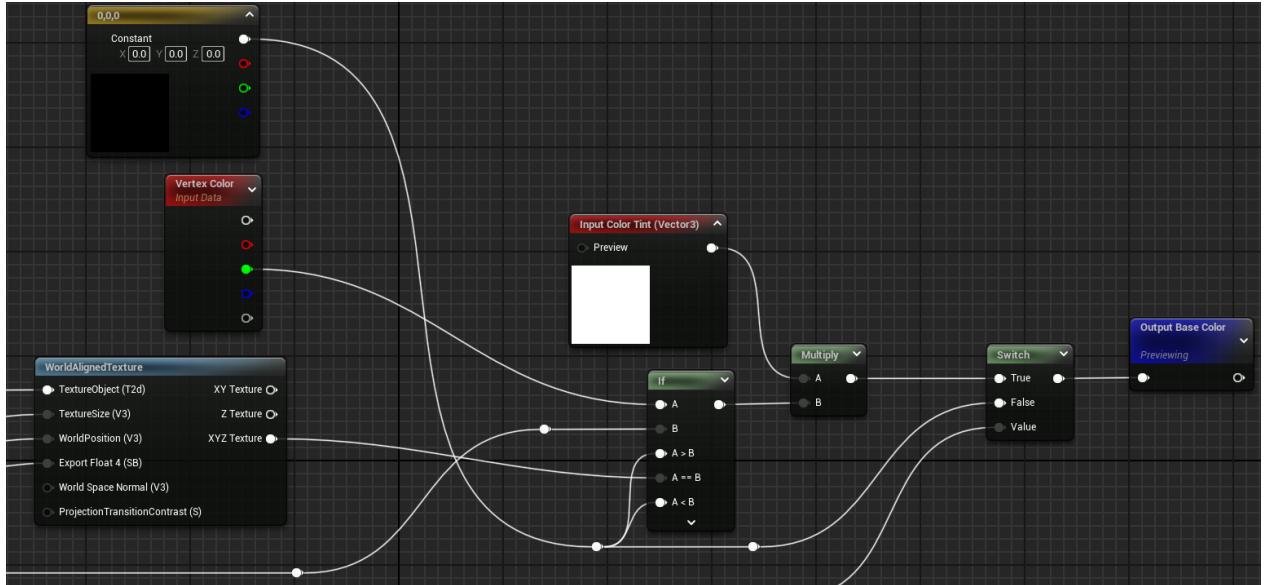


Obrázek 3.17: Textury materiálu dřeva použitý ve finální verzi prostředí

Finální princip materiálu všech objektů prostředí funguje podobně jako u modelů menších, jen je texturuju až v *Unreal Engine*. Všechny stěny mají vertex color s určitým odstínem zelené, podle kterého v *Unreal Engine* stěny od sebe rozlišuju. *Master materiál* se skládá z deseti identických částí a každá část má tyto parametry:

- **Tři texture parametry** - Base Color, Normal a spojená Ambient Occlusion, Roughness a Metallic mapa
- **Hodnota vertex color** stěny
- **Velikost textury ve světě** - Na rychlé úpravy velikosti textur bez nutnosti re-exportu.
- **Barevný odstín** - Vyskytuje se jen u dvou částí, slouží ke změně barvy; ušetří se tím paměť na disku, protože nemusím mít pro rozdílné odstíny zvláštní textury. Toto jsem využil pro obarvení části zdí a podlahy.
- **Offset pozice** - Vyskytuje se jen u jedné části, slouží k úpravě pozice textury kachliček pro lepší zarovnání s okolním prostředím.

Tyto parametry poté vedou do mnou vytvořené funkce, která textury z parametrů převede na textury zarovnané podle lokální pozice. Ze začátku jsem používal místo *Local Position* *World Position*, ale narazil jsem na problém, při kterém se textura nepohybovala spolu s pohybujícím se objektem. Samotná funkce, která se o převod stará se jmenuje *WorldAlignedTexture*.



Obrázek 3.18: Část mnou vytvořené funkce na převod textur

Po převedení textury jen porovnávám, kde má být vykreslena podle *Vertex Color* objektu. V případě neshody je výsledkem černá barva.

Totéž se opakuje pro Normal a spojenou Ambient Occlusion, Roughness a Metallic mapu. Tyto části sčítám a výsledkem je textura místonosti.

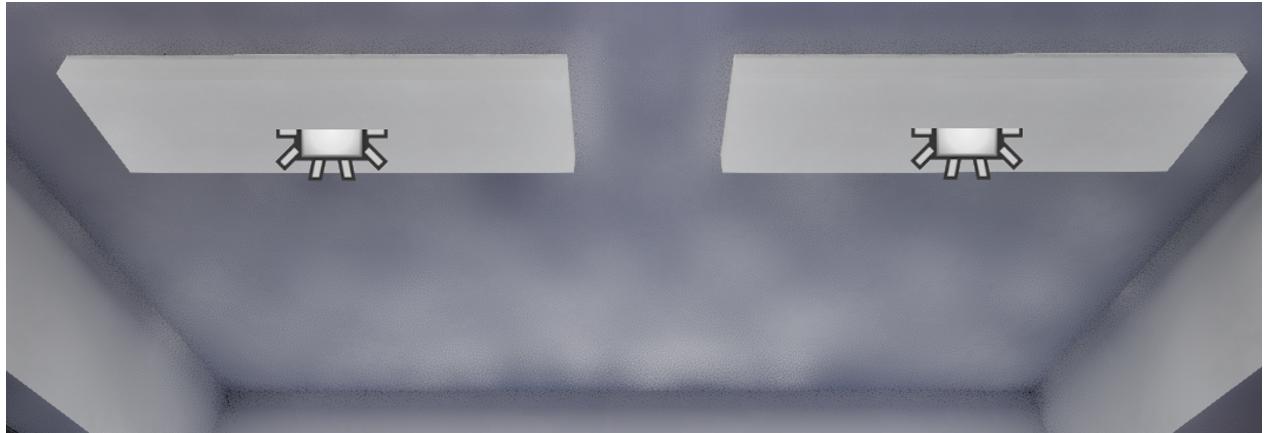


Obrázek 3.19: Chodba A700 v *Unreal Enginu*



Obrázek 3.20: Skutečná chodba A700

Další z problémů byl spojen s osvětlením. Na osvětlení prostředí používám blueprint actors s vymodelovaným světlem a Rectangle Light. Static mesh světla má upravený materiál tak, aby svítil. Potřeboval jsem, aby světla vypadaly, že svítí pomocí materiálu s emission a zároveň, aby scénu osvětlovalo hlavně Rectangle Light. První pokus o osvětlení dopadl ale opačně - scénu hlavně osvětloval emission materiál místo Rectangle Lights. Výsledek lze vidět na obrázku 3.21. Stačilo tedy upravit parametry světel a šmouhy ze stěnách zmizely. Spojené s tímto problémem a temporal anti-aliasing nastavením byly světelné glitche viditelné na obrázku 3.22 při pohybu kamerou.



Obrázek 3.21: Problém osvětlení stropu emission materiálem



Obrázek 3.22: Problém s Anti-Aliasing nastaveními

Problémy byly také s dalším nastavením projektu. Ve výchozím nastavení má projekt zapnutý Auto Exposure. Pohled hráče se tedy automaticky zesvětluje a ztmavuje, což nevypadá nejlépe.

3.3.2 Výtah

Vzhledem k zahrnutí více patří školy do naší hry bylo potřeba vytvořit způsob pohybu mezi nimi. Ačkoli byla možnost použít schody, které jsou taktéž součástí prostředí, přišlo mi lepší použít výtah, protože tímto způsobem můžeme mít prostory nebo patra rozdělené do jednotlivých levelů, čímž se šetří paměť počítače, protože nemusí mít levele načtené všechny, nebo případně nemusíme mít vytvořený systém na dynamické načítání patří pod a nad hráčem. Taktéž je můžeme jednoduše upravovat, než kdyby byly naskládané na sobě.

Bylo by tedy možné vytvořit seamless přechod mezi nimi, ale využití výtahu je v tomto ohledu jednodušší a zajímavější pro hráče, především když by je žáci neměli bez souhlasu učitele ve skutečném světě využívat.

Nejdřív popíšu funkční část výtahu a poté část vizuální.

Hráč může na chodbě výtah „privolat“ tlačítkem. Funkční je jen ten napravo od tlačítka. Pokud se hráč nenachází na patře osmé, chvíli trvá, než se výtah ocitne na patře aktuálním.

Jakmile výtah přijede a hráč do něj nastoupí, může stisknout jedno z pěti tlačítek, která korespondují s určitým patrem. Pokud ale vystoupí, dveře se za ním automaticky po chvíli zavřou.

Když hráč stiskne tlačítko, dveře se zavřou a po určité době se otevře level s daným patrem. V případě, že stiskne tlačítko s patrem, na kterém se aktuálně nachází, tak se nestane nic. Pro zachování podobnosti ke skutečné verzi je součástí výtahu tlačítko na manuální otevření dveří.

V tuto chvíli jsem řešil dva problémy. První z nich byl, že když se otevře level, tak je hráč ve výtahu zaseknut, protože se neotevřou dveře. Toto jsem zpočátku vcelku jednoduše vyřešil přidáním parametru *Options* do funkce *Open Level*. V *Game Mode Blueprintu* jsem nastavil, že pokud se hodnota option „Doors“ rovná „Open“, tak se dveře otevřou. Tuto kontrolu jsem ale později odstranil, protože to ztratilo významu, když se hráč bude pokaždé spawnovat ve výtahu.

Druhý problém spočíval v tom, že se hráč při otevření nového levelu telepotoval na místo *PlayerStartu*. Toto sice je správné chování, ale v praxi toto vytvářelo nehezký přechod mezi levele. Při rozdílném natočení a poloze než samotný *PlayerStart* to hráčem „škublo“. Toto jsem vyřešil uložením polohy hráče ve světě a rotace kamery před otevřením nového levelu do proměnných v *BP_GameInstance*. *Game Instance* je singleton blueprint, tedy jedináček, jelikož vždy existuje jen jedna instance; který se vytváří při zapnutí hry, je validní dokud se hra nevypne, umožňuje tím pádem přenos dat mezi levele a globální přístup k daným datům.

Při otevření nového levelu a tím pádem vytvořením nového hráče, se uložené hodnoty nastaví v *PlayerControlleru* hráče. Výsledkem je skoro neviditelný přechod mezi levele.

Vizuální stránka zahrnuje animaci dveří, tlačítek a displejů. Animace dveří a tlačítek jsem vytvářel přímo v *Unreal Engine*. Používal jsem na to *Timeline*. Funkcionalita *Timeline* spočívá ve vytvoření křivek, kterých průběh game timem se nastavuje do rotace a lokace objektu. V případě výtahových dveří jsem nastavoval lokaci jednotlivých částí a klasických dveří jejich rotaci celého actora.

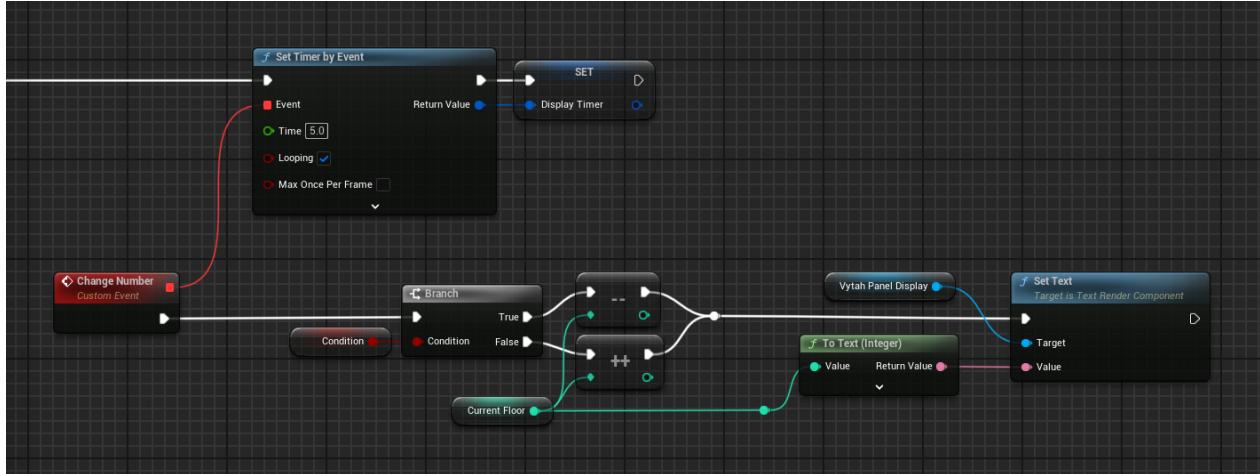
Displejů jsou dohromady dva typy. První, umístěné na každém patře, zobrazuje směr, kterým výtah zrovna jede.

Druhý se nachází uvnitř výtahu. Zobrazuje aktuální patro, na kterém se výtah nachází. Hráč tedy může vidět postup výtahu, jak se patry pohybuje.



Obrázek 3.23: Hlavní panel výtahu

Nejdřív jsem pro animaci displejů používal kombinaci cyklu *For* a *Delay*. Tento způsob ale nefungoval. Musel jsem tedy použít funkci *Set Timer by Event*, která provede kód eventu po určitém čase. Spouštění kódu eventů lze také loopovat. Pro změnu čísel a šipky jsem tedy vytvořil *Custom Event*, který lze vidět v obrázku 3.24.



Obrázek 3.24: Kód, který slouží ke zvyšování a snižování pater na displeji

Samotné ukazatele patra a směru na displejích jsou actory typu *Text Render* s fontem „LCD font“ [31].



Obrázek 3.25: Displeje, které ukazují aktuální směr pohybu výtahu

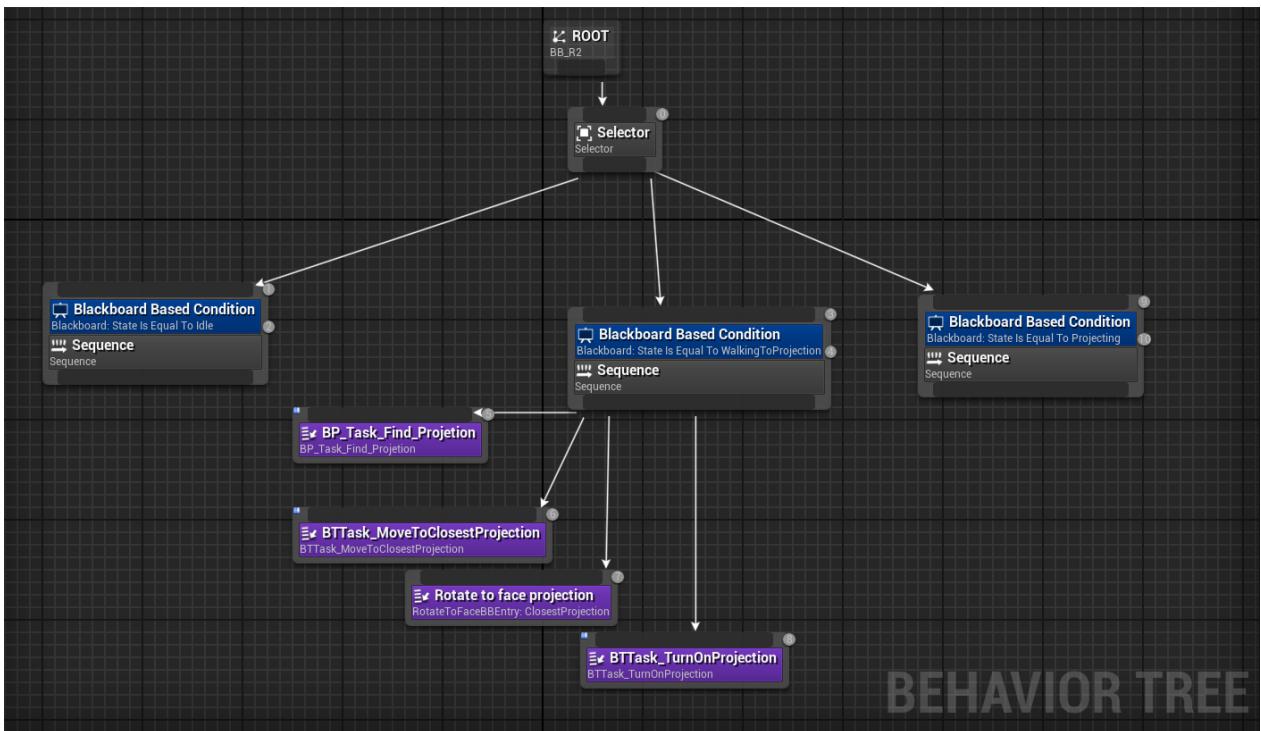
3.3.3 NPC

Základem komplexních NPC a AI v *Unreal Enginu* je *Behavior Tree*. Je to nástroj na vytváření strukturovaných a modulárních chování. Definuje, jak se postavy „rozhodují“ a vykonávají činnosti na základě podmínek a stavů [32].

Behavior Tree se skládá z:

1. **Nodes** - Základní komponenty *Behavior Tree*, každá node je reprezentací jedné akce či rozhodnutí. Existuje několik typů:

- **Composite nodes** - Řídí, jak jsou činnosti child nodes vykonány. Mohou obsahovat ostatní typy nodes a řídit podle nich činnosti. Mezi composite nodes patří například selector nebo sequence.
- **Decorator nodes** - Upravují chování child nodes přidáním podmínek nebo omezení. Mezi decorator nodes patří condition nebo repeater decorator.
- **Task nodes** - Vykonávají specifickou činnost a nemohou mít child nodes. Mezi task nodes patří třeba MoveTo a PlayAnimation, vývojáři také mohou definovat vlastní činnosti.



Obrázek 3.26: Strom chování NPC P.L.U.T.O.

2. **Blackboard** - Existuje paralelně se stromem chování. Jejím účelem je existovat jako sílené místo v paměti pro AI, odkud může zapisovat a číst informace.

Přidáním NPC jsme chtěli dosáhnout vyšší úrovně interaktivnosti s virtuálním světem, který jsme vytvářeli a vzhledem k žánru hry byla jasná volba je umístit k hádankám. Výsledkem jsou dvě NPC, které ačkoli vypadají podobně, důvody jejich existence jsou rozdílné.

3.3.4 P.L.U.T.O.

Projecting Luminous Functions To Board, neboli P.L.U.T.O., je NPC, které promítá na plochu obsah projektorem. Tato funkce je použita k promítání funkcí grafů, které následně má hráč rozpoznat a zvolit správnou možnost na displeji.

Samotná projekce je blueprint actor s widget componentem, skládající se z obrázku, a šipkou, na kterou se NPC při projekci postaví.



Obrázek 3.27: NPC P.L.U.T.O. zepředu



Obrázek 3.28: NPC P.L.U.T.O. zezadu

3.3.5 C.E.R.E.S.

Central Educational Robotic Evaluation System, neboli C.E.R.E.S., je NPC, jehož hlavní funkce je hráči pomáhat s hádankami a úkoly v případě potíží. Pro lepší interakci s hráčem disponuje měnícím se obličejem na displeji. Má několik přednastavených emocí, které se promítou při dialogu i v grafickém rozhraní.



Obrázek 3.29: NPC C.E.R.E.S. s neutrálním obličejem na displeji



Obrázek 3.30: NPC C.E.R.E.S. se šťastným obličejem na displeji



Obrázek 3.31: Příklad dialogu s NPC C.E.R.E.S.

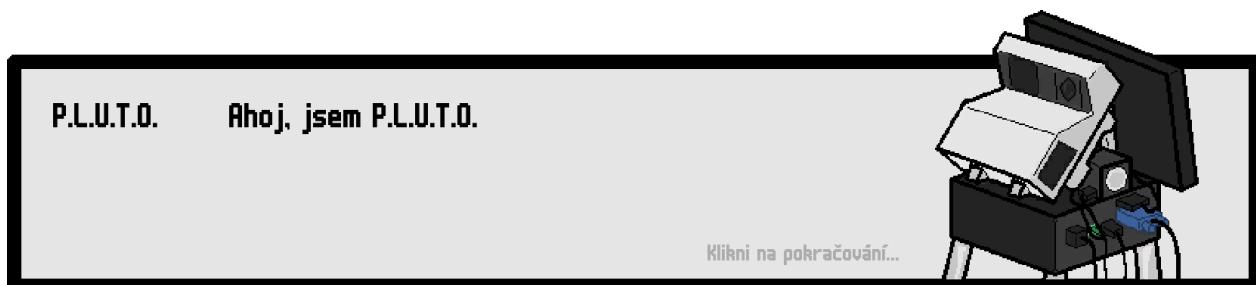
Hlavní myšlenkou za designem obou NPC byla, že se jedná o školní projekt studentů naší školy. Jsou složeny z věcí, které by mohli žáci „recyklovat“, v případě, že by byly například staré, nebo rozbité a poté opravené. Akronypy obou NPC vychází ze stejnojmenných trpasličích planet Sluneční soustavy.

3.3.6 Dialogový systém

Dialogový systém měl zpočátku sloužit jen pro jednoduché sdělení ná povědy hráči, kdyby měl s hádankami problém. Později jsem jej zkombinoval s jednoduchým systémem zpráv pro upozornění hráče, že jsou například dveře zavřené, nebo že určitou cestou nemůže projít.

Celý systém se dělí na čtyři části:

1. **Struktury** - V základní třídě pro dialog, *Dialogue_Structure*, jsou definovány všechny potřebné proměnné, především samotné hlášky; jméno řádku následujícího dialogu; data table následujícího dialogu; volba, zda má být dialog tímto řádkem ukončen a pole multiple-choice možnosti. V druhé struktuře, *OptionsDialogue_Structure*, je definována odpověď a jméno následujícího řádku dialogu.
2. **Data Tables** - Tabulky dat se strukturou řádků *Dialogue_Structure*. Jsou v nich uložené všechny hlášky, které se mohou ve hře objevit.
3. **BP_DialogueComponent** - Všechna funkcionality se nachází v této blueprint třídě, kterou obsahuje každý blueprint actor s potřebou možnosti dialogu. Vytváří widget při spuštění dialogu interakcí s blueprint actorem, zobrazuje hlášky z data table, dynamicky vytváří tlačítka na odpovědi při otázce s více odpověďmi a při ukončení dialogu widget odstraní z rodiče.
4. **WBP_Dialogue** - Widget blueprint, který zobrazuje dialogy hráči na obrazovku. V případě, že hráč mluví s NPCčkem i zobrazuje jeho ikonu.

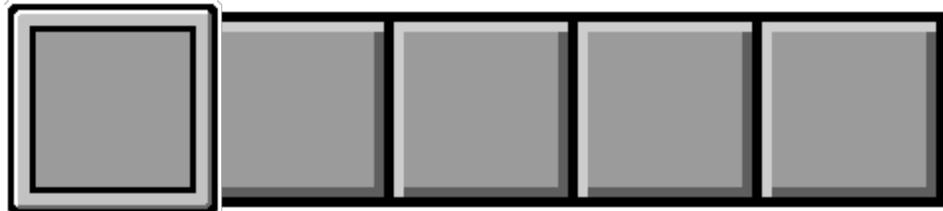


Obrázek 3.32: Finální vzhled dialogového systému

3.4 Tvorba grafických uživatelských rozhraní v programu *Aseprite*

Pro program *Aseprite* jsem se rozhodl z pocitu, že dokážu vytvořit lepší výsledek, než kdybych používal program jako *Krita* a tvořil grafiku ve větším rozlišením.

Mým záměrem bylo vytvořit především čisté a jednoduché grafické rozhraní. Usoudil bych, že výsledek tomu odpovídá.



Obrázek 3.33: Grafické rozhraní pro inventář

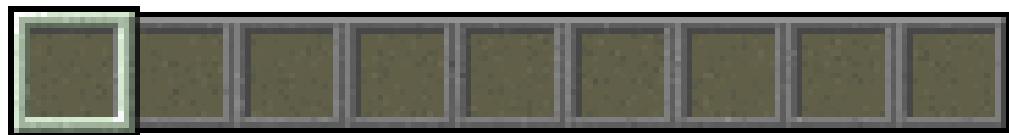
Pro přidání hloubky mají prvky stínování. Ačkoli to je vcelku jednoduchý doplněk, výsledku to přidá komplexitu.

Designy ale nejsou kompletně mé. Dialogové systémy sice u příběhových her vypadají většinou celkem podobně, u toho mého jsem z části vycházel ze hry Stardew Valley [33], jak lze vidět na obrázku 3.34.



Obrázek 3.34: Inspirace k vzhledu dialogového systému ze hry Stardew Valley [33]

Při designu inventáře jsem se celkem silně inspiroval vzhledem hotbaru ze hry Minecraft [34], viz obrázek 3.35.



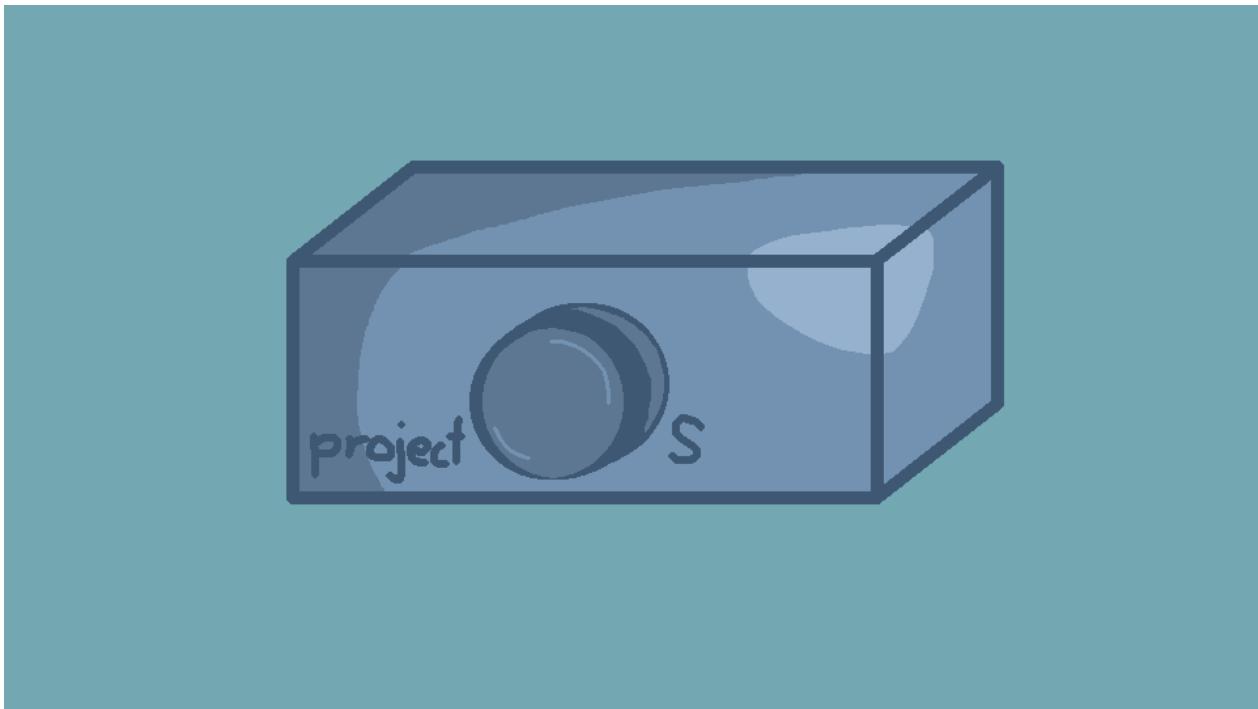
Obrázek 3.35: Inspirace k vzhledu inventáře ze hry Minecraft [34]

Tlačítka, která jsem využil u každého interaktivního rozhraní ve hře, jsem vytvořil modulární, aby šla přizpůsobit jakékoli velikosti. Výsledný design je rozdělen do devíti částí a poté složen v *Unreal Enginu*.



Obrázek 3.36: Design tlačítka

V *Aseprite* jsem také vytvářel pozadí. Ty jsou také jednoduchá, a ve světlejších barvách, aby moc nevyčnívaly a aby šly interaktivní prvky dobře vidět.



Obrázek 3.37: Design pozadí

3.4.1 Použití v *Unreal Enginu*

Pro všechny text jsem použil mnou upravenou verzi fontu „m6x11“. [35] Původní verze dostupná na itch.io má ale jeden nedostatek - postrádá českou diakritiku. Musel jsem tedy chybějící písmena dodělat v programu *FontForge* na tvorbu a úpravu fontů. Ten ale ve freeware verzi nepodporuje export fontů ve formátu OTF. Naopak zkompilovaná verze ano, musel jsem jej zkompilovat z

volně dostupného zdrojového kódu. Font jsem vybral kvůli pixelovému vzhledu, ideálně tedy zapadá do všech rozhraní.

Pro tlačítka jsem vytvořil dynamické pozadí, skládající se z *GridPane*, které obsahuje devět exportovaných částí tlačítka. Tlačítkům také lze měnit jak text, barvu textu, tak pozadí.

Ačkoli jsem výsledný vzhled rozhraní designoval já, můj kolega nejdřív vytvářel rozhraní jednoduchá jen se základními prvky pro otestování funkcí vytvořených hádanek. Jeho základním rozhraním jsem tedy upravil samotná tlačítka, jejich pozice a pozadí.



Obrázek 3.38: GUI kanón puzzlu

Kapitola 4

Závěr

V rámci maturitní práce jsem pracoval na tvorbě modelů, jejich texturování a použití ve hře; designováním grafických uživatelských rozhraní a programováním interaktivních prvků prostředí a NPC. Výsledkem je funkční hra s hádankami umístěných ve třech z pěti pater budovy A naší školy. Hra by taktéž mohla být rozšířena o další patra jiných budov, jinými druhy hádanek. Zkomplikovaná aplikace je k dispozici pro testování v příloze nebo na GitHub repozitáři. [36]

Zdroje

1. FOUNDATION, Blender. *Blender License* [online]. [B.r.]. [cit. 2025-03-16]. Dostupné z: <https://www.blender.org/about/license/>.
2. AUTODESK. *Autodesk Maya License* [online]. [B.r.]. [cit. 2025-03-17]. Dostupné z: <https://www.autodesk.com/products/maya/overview>.
3. FOUNDATION, Blender. *Blender Features* [online]. [B.r.]. [cit. 2025-03-16]. Dostupné z: <https://www.blender.org/features/>.
4. [online]. [B.r.]. [cit. 2025-03-26]. Dostupné z: <https://docs.blender.org/api/current/index.html>.
5. SCRATCHAPIXEL. *Introduction to Shading* [online]. [B.r.]. [cit. 2025-03-29]. Dostupné z: <https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/shading-normals.html>.
6. FOUNDATION, Blender. *Blender Support* [online]. [B.r.]. [cit. 2025-03-16]. Dostupné z: <https://www.blender.org/support/>.
7. WIKIPEDIA. *Physically Based Rendering Wikipedia article* [online]. [B.r.]. [cit. 2025-03-24]. Dostupné z: https://en.wikipedia.org/wiki/Physically_based_rendering.
8. GAMES, Epic. *Physically Based Materials* [online]. [B.r.]. [cit. 2025-03-17]. Dostupné z: <https://dev.epicgames.com/documentation/en-us/unreal-engine/physically-based-materials-in-unreal-engine>.
9. GAMES, Epic. *Ambient Occlusion* [online]. [B.r.]. [cit. 2025-03-17]. Dostupné z: https://dev.epicgames.com/documentation/en-us/unreal-engine/ambient-occlusion?application_version=4.27.
10. CORPORATION, Valve. *Substance 3D Perpetual License on Steam* [online]. [B.r.]. [cit. 2025-03-17]. Dostupné z: https://store.steampowered.com/app/3366290/Substance_3D_Painter_2025/.
11. ADOBE. *Substance 3D Plans* [online]. [B.r.]. [cit. 2025-03-16]. Dostupné z: <https://www.adobe.com/products/substance3d/plans.html>.

12. ADOBE. *Substance 3D Painter System Requirements* [online]. [B.r.]. [cit. 2025-03-17]. Dostupné z: <https://helpx.adobe.com/substance-3d-painter/getting-started/system-requirements.html>.
13. EBADATI, Benjamin M. *Texel Density Importance in 3D Game Asset creation* [online]. [B.r.]. [cit. 2025-03-17]. Dostupné z: <https://www.artstation.com/blogs/bendvfx/G1nB/texel-density-importance-in-3d-game-asset-creation>.
14. STUDIO, Igara. *Aseprite GitHub repository* [online]. [B.r.]. [cit. 2025-03-23]. Dostupné z: <https://github.com/aseprite/aseprite>.
15. STUDIO, Igara. *Aseprite tilemap documentation* [online]. [B.r.]. [cit. 2025-03-18]. Dostupné z: <https://www.aseprite.org/docs/tilemap/>.
16. STUDIO, Igara. *Aseprite Spritesheet documentation* [online]. [B.r.]. [cit. 2025-03-23]. Dostupné z: <https://www.aseprite.org/docs/sprite-sheet/>.
17. STUDIO, Igara. *Aseprite sprite sheets documentation* [online]. [B.r.]. [cit. 2025-03-29]. Dostupné z: <https://www.aseprite.org/docs/sprite-sheet/>.
18. STUDIO, Igara. *Aseprite website* [online]. [B.r.]. [cit. 2025-03-23]. Dostupné z: <https://www.aseprite.org/#buy>.
19. CORPORATION, Valve. *Aseprite on Steam* [online]. [B.r.]. [cit. 2025-03-23]. Dostupné z: <https://store.steampowered.com/app/431730/Aseprite/>.
20. BARNHARDT, Adam. *Loki Crew Reveals Why They Didn't Use The Mandalorian's Volume* [online]. [B.r.]. [cit. 2025-03-16]. Dostupné z: <https://comicbook.com/marvel/news/loki-crew-didnt-use-the-volume-mandalorian-industrial-light-magi/>.
21. VINEYARD, Cody. *What Are 3D Assets? Learn More About 3D Models, Alembics, VDBs, And More* [online]. [B.r.]. [cit. 2025-03-18]. Dostupné z: <https://www.actionvfx.com/blog/what-are-3d-assets-learn-more-about-3d-models-alembics-vdfs-and-more>.
22. GAMES, Epic. *Recommended Asset Naming Conventions* [online]. [B.r.]. [cit. 2025-03-29]. Dostupné z: <https://dev.epicgames.com/documentation/en-us/unreal-engine/recommended-asset-naming-conventions-in-unreal-engine-projects>.
23. GAMES, Epic. *Unreal Engine Blueprint components* [online]. [B.r.]. [cit. 2025-03-26]. Dostupné z: <https://dev.epicgames.com/documentation/en-us/unreal-engine/components-in-unreal-engine>.
24. GAMES, Epic. *Unreal Engine Licensing* [online]. [B.r.]. [cit. 2025-03-16]. Dostupné z: <https://www.unrealengine.com/en-US/license>.
25. ATLASSIAN. *Git merge conflicts article* [online]. [B.r.]. [cit. 2025-03-25]. Dostupné z: <https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts>.

26. FOUNDATION, Blender. *Blender Mapping Types* [online]. [B.r.]. [cit. 2025-03-29]. Dostupné z: <https://docs.blender.org/manual/en/latest/modeling/meshes/editing/uv.html#bpy-ops-uv-smart-project>.
27. WIKIPEDIA. *FBX file format* Wikipedia article [online]. [B.r.]. [cit. 2025-03-23]. Dostupné z: <https://en.wikipedia.org/wiki/FBX>.
28. HAVEN, Poly. *Polyhaven* [online]. [B.r.]. [cit. 2025-03-18]. Dostupné z: <https://polyhaven.com/>.
29. ADOBE. *Substance 3D Designer page* [online]. [B.r.]. [cit. 2025-03-25]. Dostupné z: <https://www.adobe.com/products/substance3d/apps/designer.html>.
30. COMMUNITY. *Import AutoCAD DXF Format Blender add-on* [online]. [B.r.]. [cit. 2025-03-24]. Dostupné z: https://extensions.blender.org/add-ons/import-autocad-dxf-format-dxf/?utm_source=blender-4.3.0.
31. BRENNAN, Fredrick. *LCD Font GitHub page* [online]. [B.r.]. [cit. 2025-03-26]. Dostupné z: <https://github.com/ctrlcctrlv/lcd-font>.
32. GAMES, Epic. *Behavior Tree Overview* [online]. [B.r.]. [cit. 2025-03-27]. Dostupné z: <https://dev.epicgames.com/documentation/en-us/unreal-engine/behavior-tree-in-unreal-engine---overview>.
33. BARONE, Eric. *Stardew Valley website* [online]. [B.r.]. [cit. 2025-03-26]. Dostupné z: <https://www.stardewvalley.net/>.
34. MOJANG. *Minecraft website* [online]. [B.r.]. [cit. 2025-03-26]. Dostupné z: <https://www.minecraft.net/>.
35. LINSSEN, Daniel. *Pixelated font Itch.io page* [online]. [B.r.]. [cit. 2025-03-26]. Dostupné z: <https://managore.itch.io/m6x11>.
36. DLUHOŠ, Richard; JANČA, Stanislav. *Repozitář výsledné aplikace* [online]. [B.r.]. [cit. 2025-03-26]. Dostupné z: <https://github.com/Rikos99/prestizniMaturitniProjekt>.

Seznam obrázků

2.1	Model zásuvky vytvořen polygonovým modelováním	11
2.2	Základní tvar modelu jablka vytvořen sculptingem	11
2.3	UV Mapa modelu zásuvky	12
2.4	List <i>Smart materiálů</i> v <i>Substance 3D Painteru</i>	13
2.5	Natexturovaný model zásuvky	14
2.6	Velikost souborů 4K textur	15
2.7	Porovnání Texel Density [13]	15
2.8	Spojení map Ambient Occlusion (červená), Roughness (zelená) a Metallic (modrá) do jedné	16
2.9	Assets s jednotlivými mapami a se všemi v <i>Unreal Engineu</i>	16
2.10	Možnosti ditheringu v programu <i>Aseprite</i>	17
2.11	Příklad části <i>level blueprint</i>	18
3.1	Model patra A800	21
3.2	Příklad reference	22
3.3	Prostorová křivka	23
3.4	Ukázka špatně nastavené normály modelu v <i>Blenderu</i>	24
3.5	Důsledek špatně nastavených normálů modelu v <i>Substance 3D Painteru</i>	24
3.6	UV mapa modelu bez seams	24
3.7	UV mapa modelu s nastavenými seams	25
3.8	Překrývající se strany UV mapy	25
3.9	Přechod <i>Vertex Colors</i> modelu	26
3.10	N-gon (pětiúhelník) s vyzdviženým vrcholem	26
3.11	Opravený n-gon skládající se z trojúhelníků	26
3.12	Strana uvnitř válce	27
3.13	Krychle s postupně stupňující úrovní <i>Subdivision modifieru</i>	28
3.14	Jednoduchý PBR material setup s Diffuse a Normal mapou v <i>Blenderu</i>	29
3.15	Vrstvy <i>Substance 3D Painter Smart materiálu</i>	30
3.16	První verze textur prostředí	32

3.17	Textury materiálu dřeva použitý ve finální verzi prostředí	32
3.18	Část mnou vytvořené funkce na převod textur	33
3.19	Chodba A700 v <i>Unreal Enginu</i>	34
3.20	Skutečná chodba A700	34
3.21	Problém osvětlení stropu emission materiélem	35
3.22	Problém s Anti-Aliasing nastaveními	35
3.23	Hlavní panel výtahu	37
3.24	Kód, který slouží ke zvyšování a snižování pater na displeji	38
3.25	Displeje, které ukazují aktuální směr pohybu výtahu	38
3.26	Strom chování NPC P.L.U.T.O.	39
3.27	NPC P.L.U.T.O. zepředu	40
3.28	NPC P.L.U.T.O. ze zadu	40
3.29	NPC C.E.R.E.S. s neutrálním obličejem na displeji	41
3.30	NPC C.E.R.E.S. se šťastným obličejem na displeji	41
3.31	Příklad dialogu s NPC C.E.R.E.S.	41
3.32	Finální vzhled dialogového systému	42
3.33	Grafické rozhraní pro inventář	43
3.34	Inspirace k vzhledu dialogového systému ze hry Stardew Valley [33]	43
3.35	Inspirace k vzhledu inventáře ze hry Minecraft [34]	43
3.36	Design tlačítka	44
3.37	Design pozadí	44
3.38	GUI kanón puzzlu	45

Použitý software

- Unreal Engine 5.4. a 5.5
- Blender
- Adobe Substance 3D Painter
- Aseprite
- AutoCAD
- FontForge
- PureRef
- ImageGlass
- Overleaf
- Git

Seznam příloh

- Zkompilovaná aplikace
- Zdrojové soubory aplikace