VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

VYUŽITÍ HERNÍHO ENGINU UNITY

USING THE UNITY GAME ENGINE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Martin Dundálek
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. Jan Roupec, Ph.D
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Martin Dundálek**
Studijní program:	Strojírenství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**doc. Ing. Jan Roupec, Ph.D.**
Akademický rok:	2023/24

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

Využití herního enginu Unity

STRUČNÁ CHARAKTERISTIKA PROBLEMATIKY ÚKOLU:

Při tvorbě počítačových her se s výhodou využívají různé knihovny a herní enginy. Jedním z nich je Unity. Předpokládaným výstupem bude jednoduchá počítačová hra s využitím tohoto enginu.

CÍLE BAKALÁŘSKÉ PRÁCE:

Hlavním výstupem práce bude realizovaná počítačová hra v jazyce C# využívající herní engine Unity. V textové části práci budou obecně popsány vybrané herní enginy. Dále bude popsána zvolená technologie a samozřejmě i hra samotná.

SEZNAM DOPORUČENÉ LITERATURY:

HOLAN, T.: Unity. CZ.NIC, 2021, ISBN 978-80-88168-57-7.

VIRIUS, M.: Programování v C# od základů k profesionálnímu použití. Grada, 2020, ISBN 978-80-271-1216-6.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2023/24

V Brně, dne

L. S.

prof. Ing. Radomil Matoušek, Ph.D.ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D.děkan fakulty

ABSTRAKT

V této bakalářské práci bude rozebrán koncept herního enginu, bude popsána stručná historie herních enginů a čtenář bude seznámen s herními enginy využívanými společnostmi vyvíjejícími videohry, které nejsou pro veřejnost dostupné („in-house“ enginy). V druhé části práce bude psáno programové řešení mechanik hry, představení a stručný popis 3D modelovacího programu Blender, který autor využil pro tvorbu modelů do hry, představení a stručný popis prostředí samotného enginu Unity a závěrem se bude zabývat autorem vytvořenou hrou Stanice Heaven Square.

ABSTRACT

In this bachelor's thesis, the concept of a game engine will be discussed, a brief history of game engines will be described and the reader will be introduced to game engines used by companies developing video games that are not available to the public ("in-house" engines). In the second part of the work, a program solution to the mechanics of the game will be written, an introduction and a brief description of the 3D modeling program Blender, which the author used to create models for the game, an introduction and a brief description of the environment of the Unity engine itself, and the conclusion will deal with the Heaven Square Station game created by the author.

KLÍČOVÁ SLOVA

Herní engine, Herní engine Unity

KEYWORDS

Game engine, Unity game engine

ÚSTAV AUTOMATIZACE A INFORMATIKY

2024

BIBLIOGRAFICKÁ CITACE

DUNDÁLEK, Martin. *Využití herního enginu Unity*. Brno, 2024. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/154169. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Jan Roupec.

PODĚKOVÁNÍ

V prvé řadě bych chtěl poděkovat svému vedoucímu práce, panu docentovi Roupcovi, za jeho rady a vedení ve vypracování této práce. Dále bych chtěl poděkovat panu inženýrovi Šoustkovi za tipy k formátování kódové části bakalářské práce. Jako poslednímu bych chtěl poděkovat svému bratrovi za to, že mi při praktické části této práce dělal druhý pár očí a byl schopen mě upozornit na chyby, které ovlivňovaly funkčnost mého kódu.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.
Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestně právních důsledků.

V Brně dne 20. 5. 2024 ……

Martin Dundálek

OBSAH

1 ÚVOD

Vytvoření prvních videoher vedlo v minulém století k novému druhu digitální zábavy, který se v dnešní době rozvinul v multimiliardový průmysl. Hry se vyskytují v různých podobách a variacích, ale jádro toho, co je považováno za videohru se v průběhu let tolik nezměnilo.

Videohra se vyznačuje tím, že hráč aktivně interaguje se světem na obrazovce. Pomocí stisknutí tlačítek se může bavit s lidmi, jít se podívat, jestli se za vodopádem neschovává jeskyně nebo třeba za kolik místní obchodník prodává léčivé elixíry. Videohra tak vyžaduje aktivní účast, aby se příběh odvíjel (v dnešní době jsou také stále běžnější interaktivní filmy. Ty ale v porovnání s videohrou vyžadují mnohem menší zapojení hráče do příběhu. Také se objevují hry, které od hráče vyžadují převážně navigování postav a občas stisknutí tlačítka. Stále se ale jedná podle definice o videohry).

Stejně jako v každém průmyslu, i ve videoherním průmyslu šlo o kvalitu a rychlost inovace. Technologie, která byla jeden rok průlomová mohla být další rok běžná nebo dokonce už zastaralá. Programování her bylo v té době navíc brzděno nízkým výkonem a cenou (na dnešní poměry) tehdejšího hardwaru. Proto vytvoření a využití prvního herního enginu (který zjednodušil mnohé aspekty vývoje a v mnoha případech také snížil zátěž na hardware) byla revoluce, která odstartovala novou etapu v tvorbě videoher.

Herní engine je programové vývojové prostředí (lze nalézt také pod chybným označením „game framework“ (herní framework) sloužící k optimalizaci a zjednodušení vývoje videoher v různých programovacích jazycích. Samotný engine neobsahuje žádný kód videohry. Pouze je využit jako prostředí, ve kterém je kód zkompilován společně s texturami a dalšími soubory do výsledného produktu.

Moderní herní enginy obsahují funkce jako 2D/3D grafický engine sloužící pro vykreslování grafiky nebo fyzikální engine sloužící k simulaci reálných fyzikálních vlivů. Herní enginy také slouží jako platforma, kterou je možné rozšířit pomocí různých přídavných modulů. Mezi další funkce herních enginů patří možnost importovat soubory z různých grafických a 3D modelovacích programů (např. Blender) a digitální obchod (Marketplace) umožňující využít/zakoupit modely (nebo textury na modely) od jiných uživatelů.

V současnosti používané enginy mají široké možnosti využití. Herní průmysl se ale neustále vyvíjí a existuje velké množství žánrů a kombinací žánrů videoher. Každý engine má silné a slabé stránky což je činí více či méně vhodné (ale ne nepoužitelné) pro určité žánry videoher. Při vývoji videohry je také nutné vzít v potaz jazyk, se kterým herní engine pracuje a jeho vliv na výkon videohry.

2 HISTORIE

Koncept herního enginu se od svého vzniku neustále vyvíjí. Historicky se tak objevují dva podobné a občas chybně zaměnitelné koncepty. *Herní engine* a „*herní framework*“. Rozdíl v těchto dvou konceptech je v jejich struktuře a taktéž možnostech využití:

- **Herní engine** umožňuje vývoj videohry v plném rozsahu. Buď přímo přímo v samotném enginu nebo s využitím rozšiřujících modulů pro herní engine (tyto moduly bývají obvykle již vytvořené a je možné je zakoupit nebo bezplatně přidat do verze enginu, kterou uživatel používá). Herní engine umožňuje propojení modelů předmětů a postav s kódem v programovacím jazyce (tvorbu samotných herních modelů enginy zpravidla neumožňují, je nutné použít rozšiřující modul nebo externí program). Stejně tak je možné pomocí integrovaných nebo dodatečně stažených funkcí vytvořit zvuky a hudbu [1].

- **Herní framework** je sbírka knihoven a nástrojů sloužících pro vývoj některých aspektů videoher. Cílem herních frameworků je pouze poskytnout základy, které mohou být na rozdíl od herního enginu snadno modifikovány a uzpůsobeny potřebám konkrétního vývojáře/videohry. Po dostatečném rozšíření se z herního frameworku stává herní engine. Příklady herních frameworků jsou např. Phaser, LibGDX [1].

V další části této kapitoly budou více dopodrobna rozebrány určité produkty tak jak byly postupně vyvinuty. Ať už jde o nový přístup k v té době technický nemožné věci nebo vymyšlení konceptu, který značně snížil výpočetní zátěž tehdejších počítačových komponentů.

SCUMM (Script Creation Utility for Maniac Mansion)

Historicky první program s funkcemi herního enginu (jedná se o herní framework, v době své první verze neměl dostatečné množství funkcí, aby byl klasifikován jako herní engine) s označením SCUMM byl vyvinut v roce 1987 Ronem Gilbertem, zaměstnancem

Obr. 1: Snímek ze hry Maniac Mansion, první hry využívající koncept herního frameworku [5]

firmy LucasArts. Jeho původní využití bylo jako pomocný nástroj při vývoji hry Maniac Mansion [2].

SCUMM byl určen na převedení příkazů od uživatele na tokeny o velikosti bytů. Ty byly interpretovány programem, který hráči zobrazoval hru (tedy ne se stejnou funkcí jako moderní herní enginy, ale jako mezikrok). Díky této funkci SCUMM umožňoval tvorbu lokací, předmětů a dialogů bez nutnosti zapisovat do zdrojového kódu. Využití SCUMM umožnilo multitasking a rapid prototyping při vývoji her [3].

Herní framework SCUMM byl v průběhu let upravován a byly do něj přidávány funkce (pozdější verze SCUMM již splňují definici herního enginu). Došlo například ke změně z textového ovládání v dolní části obrazovky na interaktivní kurzor. Také byly přidány nástroje pro podporu animovaných sekvencí a dabingu [4].

Doom Engine/idTech1

Vyvinut v roce 1993 společností id Software (konkrétně Johnem Carmackem), idTech1 byl ve své době považován za revoluci. Byl napsán s využitím jazyků C a Assembler a je považován za první skutečný herní engine [4].

Obr. 2: Doom, první videohra využívají herní engine [4]

Tento engine byl jako první schopen vykreslit takzvané „pseudo 3D“. Toho bylo docíleno pomocí algoritmu ray casting (předchůdce moderního ray tracingu). Toho je možné si všimnout ve hrách vytvořených v tomto enginu. Před příchodem výkonnějšího hardwaru se jednalo o jediný používaný způsob, jak vykreslit 3D grafiku. (zajímavost: vykreslování grafiky bylo odděleno od zbytku enginu a optimalizované pro co největší rychlost) [6;7;8].

Engine také umožňoval měnit intenzitu světla a jako první umožňoval hráčům se pohybovat i jinak než po rovné ploše (po schodech, mezi patry). Engine je snadno modifikovatelný a portovatelný (převeditelný na jiné platformy, než pro které byl původně vyvinut) [4].

VoxelSpace

Tento engine byl vyvinut v roce 1994 společností NovaLogic. Společnost NovaLogic se zaměřovala na tvorbu vojenských simulátorů a potřebovala engine, který by byl schopen vykreslit velké otevřené plochy [4].

Obr. 3: Comanche: Maximum Overkill byla první hra na enginu, který dokázal vytvořit velké otevřené prostory[4]

VoxelSpace byl unikátní tím, že grafiku vytvářel z 3D voxelů a ne z 2D pixelů, jak bylo v té době zvykem. Engine tak byl schopen vykreslit na svoji dobu grafiku, která vypadala lépe než v případě jiných vojenských simulátorů. Další unikátní schopností enginu bylo vykreslení „nekonečné“ krajiny (krajina se od určitého bodu prostě opakovala) [4;9].

Build Engine

V roce 1995 vytvořen Kenem Silvermanem, Build Engine byl pokus o konkurování Doom Enginu. Engine ale nedokázal konkurovat populárnějšímu Doom enginu a ve své době poháněl pouze několik her [4].

Obr. 4: Duke Nukem 3D poháněný Build enginem byl první hrou umožňující (i když pouze částečnou) destrukci prostředí [4]

Build engine má proti Doom enginu několik aspektů, které vylepšují hráčovu imerzi (vcítění, pocit že je hráč opravdu ve hře) ve hře. Jednalo se o možnosti dynamické

interakce s prostředím. Hráč mohl rozbíjet zdi, potápět se pod vodou nebo se například pohybovat po šikmých plochách (to Doom engine neumožňoval) [4].

Quake Engine/idTech 2

Tento engine vytvořený v roce 1996 Johnem Carmackem a Michaelem Abrashem pod záštitou idSoftware byl poprvé použit ve hře Quake a jedná se o evoluci staršího Doom engine. Jedná se o zcela nový engine, ne pouze o vylepšenou verzi staršího enginu. Stejně jako jeho předchůdce je napsán v jazycích C a Assembler [4;10].

Obr. 5: Quake na Quake enginu byl první hrou, ve které byl schopen engine vykreslit skutečné 3D [4]

Quake engine byl první herní engine, který dokázal vykreslit skutečné 3D prostředí pomocí polygonů. Dále obsahuje dynamické nasvětlení a přímo v enginu zabudované modderské nástroje. Vysoké náročnosti renderování 3D prostředí na tehdejších zařízeních bylo docíleno speciálním algoritmem, který renderoval pouze části hráčem skutečně viditelné v daný moment. Tato procedura ale trvala velice dlouho a při chybách v tvorbě mapy mohla vést k nutnosti celou mapu začít tvořit znovu [4].

Unreal Engine

Unreal engine byl poprvé použit ve hře Unreal v roce 1998 a byl napsán Timem Sweeneyem v jazyce C++. Engine původně využíval zcela softwarový rendering, což mělo za následek vysokou zátěž procesoru. Postupem času byl engine upraven, aby mohl využít dedikovaných grafických karet [9;11].

Obr. 6: Unreal byl první hrou na Unreal enginu, jednom z nejpopulárnějších herních enginů současnosti [14]

Unreal engine už ve své první verzi měl detektor kolizí, barevné nasvícení, volumetrickou mlhu (simulace světla procházejícího mlhou) a omezenou formu filtrování textur (metoda post-processingu určená k vylepšení barvy textur). Také obsahoval editor levelů s možností editace levelů v reálném čase a jako první přišel s 16-bitovým barevným schématem [12;13].

Unity

Tento engine byl vyvinut společností Unity Technologies a byl vydán v roce 2005 jako herní engine pro platformu Mac OS X. Hlavní programovací jazyk je C#, části, které vyžadují kompilaci využívají jazyk C++ [15].

Obr. 7: GooBall, hra na Unity enginu původně určeném pro Mac OS [17]

Engine je považován za přívětivý pro začínající vývojáře, umožňuje tvorbu jak v 3D tak 2D prostoru a své využití našel i jiných průmyslech než je ten herní (filmový průmysl, architektura, filmový průmysl) [16].

Godot

Godot byl vyvíjen v letech 2001–2014 argentinskými programátory Juanem Linietskym a Arielem Manzurem a byl využíván několika společnostmi v Latinské Americe. V roce 2014 byl uvolněn k použití široké veřejnosti pod MIT License a dnes se jedná o open source engine vyvíjený původními tvůrci s přispěním komunity [18;19;20].

Obr. 8: Dome Keeper, hra na Godot enginu

Godot je napsán v jazyce C++ a v samotném prostředí enginu lze využít jazyky C# (pouze v .NET verzi), C++ a jazyk používaný samotným enginem GDScript (GDScript má syntaxi podobnou Pythonu). Engine také umožňuje kompresi textur a díky tomu je dostupný na široké škále platforem a na ještě větší množství může být portován (převeden) [21;22].

Engine lze využít jak k vývoji 3D, tak 2D aplikací. Toho je v případě Godotu dosaženo tím, že 3D a 2D grafické enginy jsou odděleny. Mohou tak pracovat nezávisle na sobě a také na stejném zobrazovacím zařízení pro vytvoření mixu 2D a 3D grafiky. Engine obsahuje také pokročilý grafický editor umožňující animovat téměř jakoukoliv proměnnou na herní entitě [23;24].

2.1 Historie verzí Unity

Herní engine Unity (a tvorba her v něm) je cílem této bakalářské práce. Proto bude věnována větší pozornost jak samotnému enginu, tak práci v něm a jeho vývoji. Následují jednotlivé verze Unity a popis jejich funkcí:

- **Unity 2.0** - Unity verze 2.0 byla vydána v roce 2007 a přidala odhadem padesát nových funkcí. Mezi ně patří např. vylepšený engine pro vykreslování povrchů, přehrávání videí a vykreslování dynamických stínů v reálném čase. Verze 2.0 byla v roce 2008 rozšířena o podporu mobilních telefonů Iphone, což vedlo k vysoké popularitě tohoto enginu při tvorbě her na těchto zařízeních [25;26].

- **Unity 3.0** - Vydána v roce 2010, verze 3.0 přinesla vylepšené grafické možnosti pro počítače a konzole. Verze 3.0 obsahuje podporu Androidu. Další integrované

funkce zahrnují Illuminate Labs' Beast Lightmap (nástroj pro práci se světlem), nativní renderování fontu a audio filtry [27].

- **Unity 4.0** - Verze 4.0 vyšla v roce 2012 a přidala do enginu podporu knihoven DirectX 11, produktů od firmy Adobe a systém animací jménem Mecanim. V roce 2013 byla integrována podpora softwarového vývoje pro sociální síť Facebook. Tyto nástroje umožnovaly sdílení specifických částí ze hry přímo na sociální síť. V roce 2016 Unity s Facebookem vytvořili vlastní herní platformu [28;29;30].

- **Unity 5.0** - Ve verzi 5.0 vydané v roce 2015 byla integrována podpora WebGL, která umožňuje nahrát vytvořené hry do prohlížeče, aniž by uživatel musel použít dodatečné rozšíření. Mezi další vylepšení oproti předchozím verzím patří globální nasvícení ve skutečném čase, Unity Cloud, novou verzi audio systému a integraci enginu pro simulování fyziky předmětů Nvidia PhysX 3.3. Verze 5.6 upravila nasvícení a částicové efekty, vylepšila celkový výkon enginu a přidala podporu pro několik herních platforem a API Vulkan. Tato verze také jako první podporuje 4K video, a je tak vhodná pro využití v kombinaci s brýlemi pro virtuální realitu [31;32;33].

- **Unity 2017 až současnost –** V roce 2016 došlo ke změně jmenovací konvence. Nové verze byly pojmenovávány podle let, kdy byly vydány z důvodu vydávání častějších aktualizací. Novinky ve verzi 2017 byly převážně diagnostického charakteru zaměřeného na pokročilejší uživatele. Pozdější verze 2017.2 integrovala funkce využitelné mimo vyvíjení videoher. Jedná se o systém pro importování animací Timeline a systém chytré kamery Cinemachine. Verze také integrovala podporu importování modelů ze softwarů Autodesk 3DS Maya a Autodesk 3DS Max pro zjednodušení procesu tvorby modelů. Verze 2018 přidala prvky machine learning (učení se na základě hráčova průchodu hrou) a předpřipravené šablony pro začínající vývojáře. Ve verzi 2020 byl přidán systém MARS (Mixed and Augmented Reality Studio), který poskytuje dodatečné funkce pro aplikace určené pro rozšířenou realitu (AR). V říjnu 2023 bylo oznámeno, že další verze enginu se vrátí k původní pojmenovávací konvenci. Unity 6 má být zaměřeno na přidání nástrojů generativního AI (AI schopné generovat obrázky, text a další typy dat) pojmenované Unity Muse a Unity Sentis [34;35;36;37;38;39].

2.2 In-house herní enginy

In-house herní enginy jsou enginy speciálně vytvořené a využívané společnostmi vyvíjejícími videohry. Jejich struktura a zdrojový kód podléhají firemnímu tajemství a pro běžného uživatele nebo začínajícího vývojáře nepracujícího ve větší firmě nejsou volně dostupné. Je tak možné pouze soudit jejich silné/slabé stránky.

Frostbite

Frostbite engine je využívám exkluzivně firmou Electronic Arts (EA) a dalšími firmami, které EA vlastní. Engine byl původně vyvinut exkluzivně pro sérii videoher Battlefield, postupem času byl využit i ve hrách mimo sérii Battlefield (např. Mass Effect Andromeda nebo série FIFA).

Hlavním důvodem, proč je engine používán, je jeho schopnost vykreslit destrukci prostředí s určitou mírou realističnosti (tato funkce je využita primárně ve zmíněné sérii Battlefield). Frostbite engine je mnohými vývojáři kritizován za jeho přílišnou komplexitu a špatnou využitelnost pro RPG (role-playing game) a střílečky z pohledu třetí osoby [40;41;42].

Obr. 9: Videohra Battlefiled 4 využívající Frostbite Engine 3

Creation Engine

Tento engine exkluzivně využívaný firmou Bethesda Games Studios je postavený na dříve používaném enginu Gamebryo od stejné firmy. Creation engine je určen pro RPG hry s velkým otevřeným světem [43;44;45].

Obr. 10: Běžící na Creation enginu

Hlavní výhodou tohoto enginu jsou malé změny v kódu při přechodech na nové verze, díky čemuž je poměrně nenáročné vytvářet pro hry na tomto enginu herní módy (úprava herních mechanik, přidání nového obsahu, obvykle vytvářené komunitou okolo dané hry).

Clausewitz Engine

Tento engine využívaný exkluzivně společností Paradox Interactive je určen pro strategie s velkým rozsahem. Umožňuje zobrazení celé herní mapy najednou (např. zobrazení celé mapy světa v historické strategii Hearts of Iron IV). V posledních verzích enginu byly přidány nástroje pro lepší tvorbu herních modelů a vylepšeno renderování grafiky [46].

Obr. 11: Hearts of Iron IV, strategie poháněna Clausewitz enginem, kde hráči mohou pohybovat jednotkami v rámci většiny kontinentů na planetě

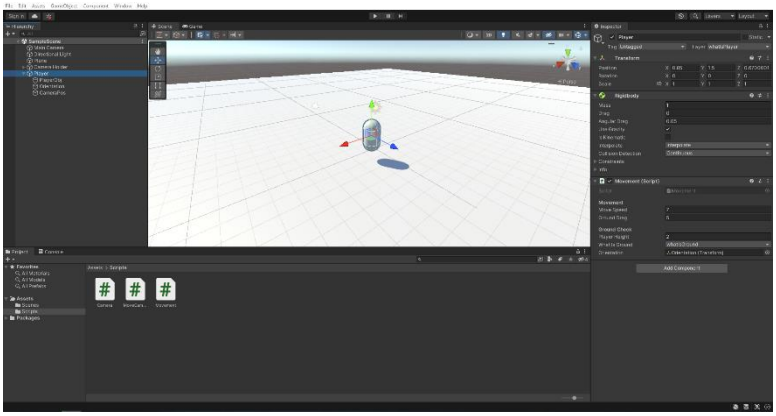
Engine je náročný na výpočetní hardware osobních počítačů, speciálně když se v pozdních fázích strategií na tomto enginu pohybuje po herní ploše velké množství jednotek v reálném čase.

3 ROZBOR VYUŽITÝCH PROSTŘEDKŮ A PROGRAMŮ

V této části budou představeny programy využité v této práci. Tyto programy zahrnují: **Herní engine Unity** (cíl této práce) a **Blender** (3D modelovací program).

Unity

Herní engine Unity využit v této práci je verze 2021.3.11f1 (o jednotlivých verzích enginu Unity bylo podrobně pojednáno v kapitole **Historie verzí Unity**) s modulem ProBuilder. V základní verzi Unity jsou možnosti tvoření 3D objektů omezené. ProBuilder je v rámci práce využit pro tvorbu jednoduchých ploch a objektů sloužících k ověření funkčnosti skriptů vytvořených uživatelem. Veškeré modely využité v závěrečném produktu jsou vytvořeny autorem práce v programu Blender.



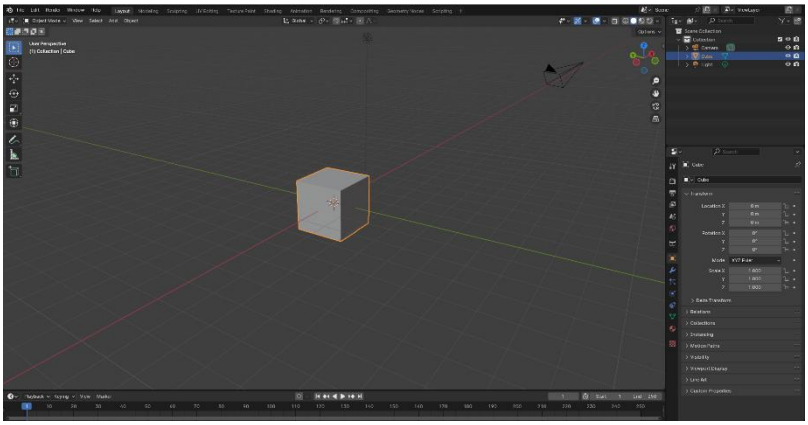
Obr. 12: Základní okno Unity. Vlevo panel ukazující objekty nacházející se v projektu, dole seznam složek a použitých skriptů

Engine Unity je možné propojit s externím programovacím prostředím (v této práci je engine propojen s Visual Studio 2022), nebo je možné využít integrované prostředí enginu. V případě využití externího programovacího prostředí je nutné ověřit a zajistit kompatabilitu enginu s daným prostředím (instalace patřičných knihoven).

Unity využívá podporuje kombinaci jazyků C#/Assembler nebo Java. Všechny skripty použité v této práci jsou napsány ve Visual Studio 2022 za využití jazyka C#.

Blender

Blender je open-source modelovací software umožňující tvorbu 3D modelů, vytváření animací, rendování a motion capture (zachycení pohybu pomocí senzorů pro jeho aplikaci v jiném médiu). Blender využívá pro renderování knihovnu OpenGL, který umožňuje hardwarovou akceleraci při renderu a snadný přenos na různé platformy [47].



Obr. 13: Deformace krychle jako základního tělesa stojí na počátku většiny hranatých 3D modelů

Základním prvkem modelování v Blenderu je deformace několika základních těles (krychle, koule, aj.). Přesnější modelování využívá dvou prvků: **Faces** („tváře“) a **Nodes** („uzly“):

- **Faces** jsou velmi malé plošky obdélníkového tvaru, které slouží jako základní prvek modelu v Blenderu. Za pomocí Faces se v Blenderu vytváří například zaoblené plochy (Faces se zmenšují do nekonečně malého tvaru do bodu, kdy je lidské oko nerozezná, a oblouk složený z 2D obdelníků se jeví jako zaoblený).

- **Nodes** jsou místa (body), ve kterých se stýkají rohy čtyř přilehlých Faces. S Nody je možné pohybovat (vytáhnout je směrem ven z tělesa, vnořit je směrem do tělesa) a tak vytvářet složitější tvary z původního hranatého tělesa. Každý Node je spojen se sousedním, a vytváří tak linie naznačující, kde se bude těleso deformovat při manipulaci s Nody.

Součástí každého projektu je také kamera a zdroj světla (kamera a nasvícení jsou využity při renderu scén). Blender taktéž obsahuje vlastní modul určený pro vytváření textur pro aplikaci na vytvořené modely. Dále obsahuje několik integrovaných renderovacích enginů, které se liší kvalitou finálního výstupu, schopností zpracovat nasvícení a náročností na hardware.

4 STANICE HEAVEN SQUARE

Stanice Heaven Square je adventura odehrávající se v opuštěné stanici metra. Hráč se na této stanici ocitá nešťastnou shodou náhod a jeho úkolem je dostat se ven.

Stanice metra právě prochází opravou, aby mohla být znovu používána. Proto jsou všude rozmístěny světla, je k dispozici funkční generátor a dveře jsou zamčené. V době, kdy se do stanice dostává hráč je stanice prázdná. Jestli z důvodu že pracovníkům skončila směna nebo se zde stalo něco jiného už musí zjistit sám hráč.

V neznámém prostředí má k dispozici pouze světlo ze svého mobilního telefonu. Informace, které jej dovedou pryč ze stanice musí zjistit z poznámek rozmístěných po herním světě.

Autor byl při tvorbě lokace inspirován hororovými filmy (jmenovitě Metro z roku 2004 a Death Line z roku 1972) a fenoménem liminálního prostoru (opuštěná místa působící děsivě).

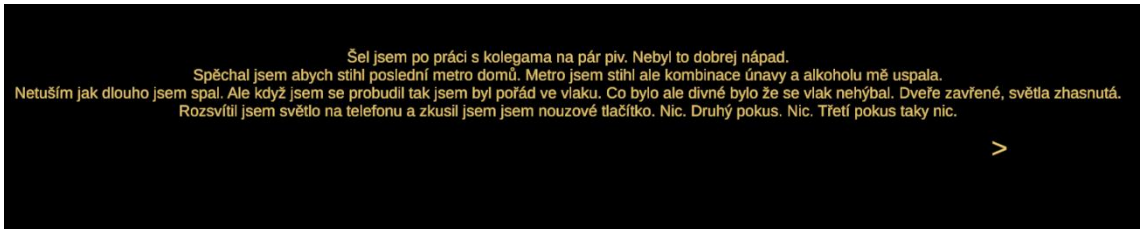
4.1 Předmluva

Že se nemáme bát tmy nás učí od dětství. Rodiče nás učili že ve tmě se neskrývá žádné monstrum, žádný duch připravený na nás zaútočit jen co se zhasne světlo.

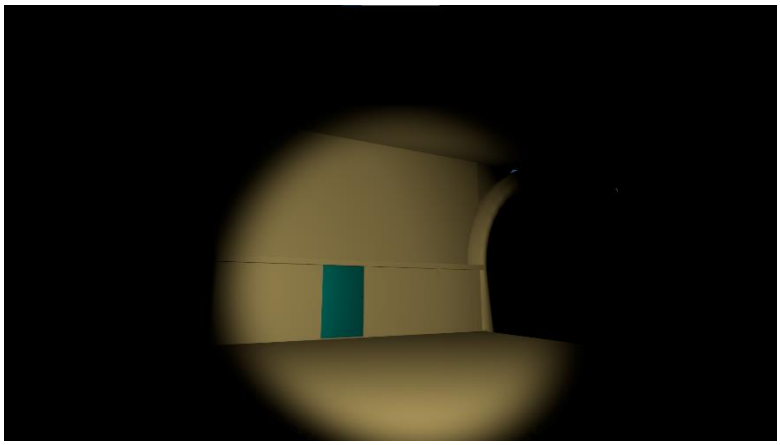
Realita ale není hezké místo. Monstrum nás sice ve tmě nečeká, ale bát se tmy je přirozené. Člověk není uzpůsoben pro život ve tmě a bez světla nebo dlouhého pobytu v temnotě nevidí ve tmě dále než na pár metrů. V takovém případě stačí, aby se člověku zdálo že zahlédl pohyb mimo světlo baterky nebo svit ohně a už může nastoupit panika. Jistě, vše se dá racionálně vysvětlit. Ale nebezpečí číhá všude.

Pamatuji si, když jsem na táboře šel Stezku odvahy. Věděl jsem, že to není nic jiného než součást testu. Do momentu, než jsem zaslechl prasknutí větve možná jen pár metrů ode mě (když jsme ještě předtím byli upozorněni že se v lese vyskytují divočáci) tak jsem sprintoval pryč obdivuhodnou rychlostí.

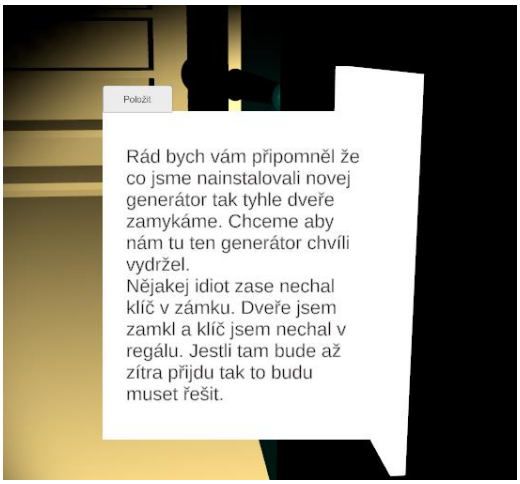
Přesně na tento typ strachu cílí i hra Stanice Heaven Square. Na nejistotu, jestli zpoza rohu nevykukuje něco, co by v prázdné stanici metra nemělo být. Na nejistotu, jestli se něco neschovává ve tmě mimo světlo z mobilního telefonu. Bát se strašidel je sice směšné, ale na to že člověk se přirozeně bojí nebezpečí už se zapomíná. A je jedno, jestli je to duch nebo divoké zvíře. Ve tmě nebezpečí může číhat. I když ho tam nečekáme nebo si namlouváme že tam není.



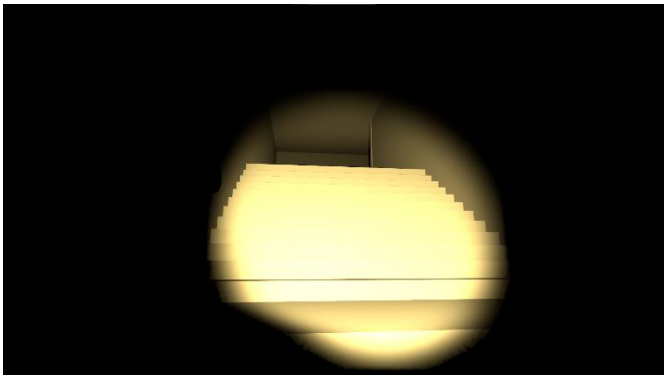
Obr. 14: Informace pro hráče, co předcházelo nezáviděníhodné situace, ve které se hlavní hrdina ocitl



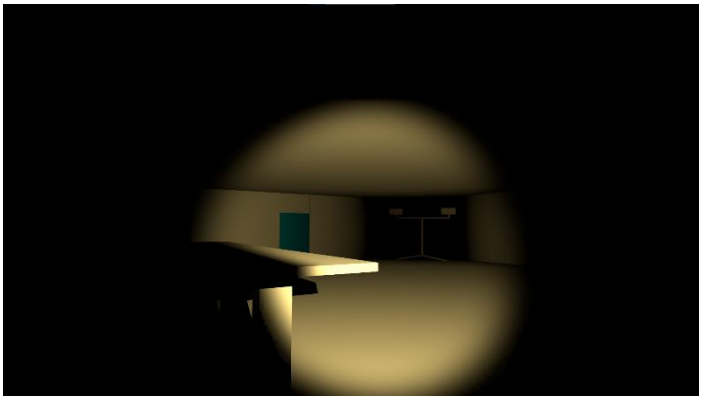
Obr. 15: Hráč nezná rozložení stanice, ve které se nachází. Jedinou možností je tak přirozeně zjistit které dveře se dají otevřít a kudy tak přirozeně postupovat dále



Obr. 16: Poznámky ve světě mohou hráči napovědět kde hledat předměty, jaký zadat kód nebo jej mohou informovat o událostech, které se ke stanici metra vztahují



Obr. 17: Schody vedoucí ven. V tomto bodě má již hráč většinu překážek za sebou

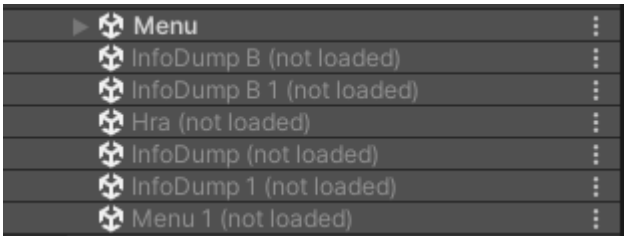


Obr. 18: Poslední zamčené dveře, které musí hráč překonat

4.2 Řešení v Unity

V následující pasáži budou rozebrány aspekty hry, které vyžadují dodatečné úpravy/modifikace přímo v prostředí herního enginu Unity.

Scény



Obr. 19: Scény seřazeny do zamýšleného pořadí

Hra byla v rámci Unity rozdělena po scénách. Scény fungují jako oddělené bloky, do který je možné nezávisle na další blocích vkládat objekty a zvuky. *Menu* funguje jako úvodní scéna, hráč zde může spustit samotné demo, nebo hru ukončit. Scény označené jako *InfoDump* slouží pro uvedení hráče do děje prostřednictvím informací o světě a událostech předcházejících/následujících hernímu segmentu. Veškeré interaktivní části hry se odehrávají ve scéně *Hra*.

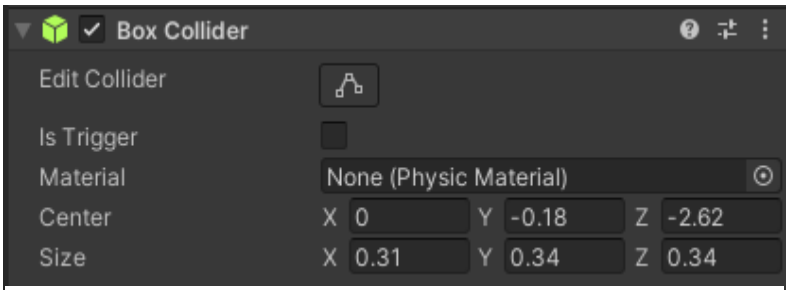
Obr. 20: Úvodní menu hry

Práce s modely

V rámci práce byly naimportovány modely vytvořené autorem v programu Blender. Tyto modely bylo nutné pro potřeby Unity převést přes funkci Unpack („Rozbalit“) do stavu, kdy se model rozložil na stejné soubory jako v Blenderu.

Box Collidery

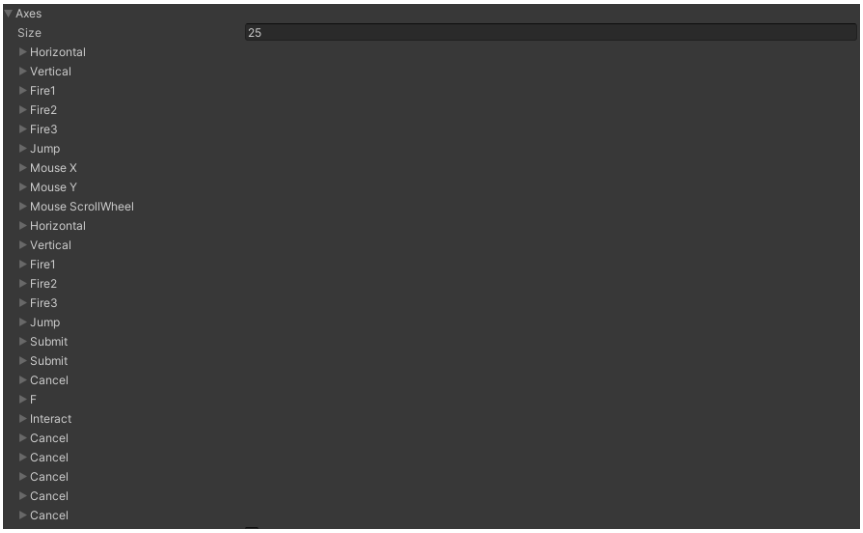
Dále bylo nutné každému objektu přiřadit Box Collider. Box Collider vytváří fyzické plochy objektu (bez Box Collideru by se hráč propadl podlahou nebo prošel zdí) a je nutné jej manuálně přiřadit všem objektům vytvořeným mimo Unity. Box Collider může obklopovat celý objekt (pokud je přiřazen k objektu, který slouží jako Parent dalším objektům) nebo jednotlivé dílčí objekty. Box Collidery jsou v této práci také využity jako oblast, která se bere jako blízkost hráče k objektu (přesněji pokud se jich hráč dotkne).



Obr. 21: Box Collider přiřazený k objektu

Interakce s předměty

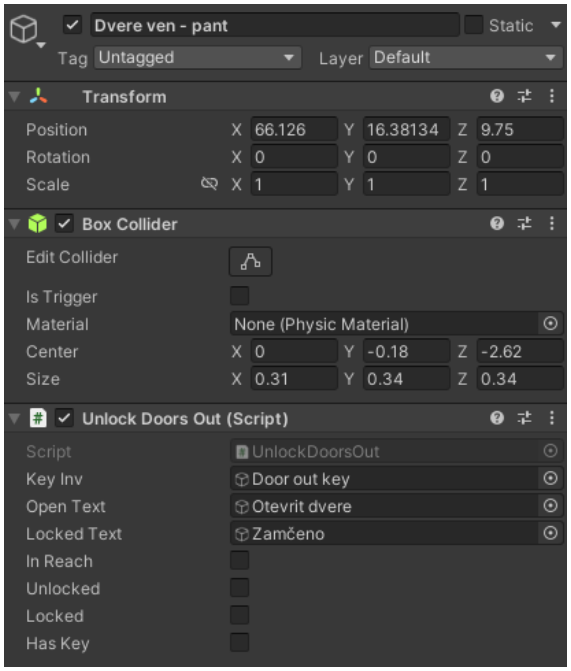
Veškeré akce/hodnoty (například natočení kamery na osách X/Y) musí být nadefinovány v rámci projektu v Input Manageru. V Input Manageru jsou definovány všechny akce, které je možné v rámci projektu provést.



Obr. 22: Input Manager výsledné hry

Dveře

Hráč může ve hře otevírat dveře. Některé je možné otevřít pouhým stisknutím klávesy, jiné vyžadují alternativní přístup (nalezení klíče nebo nalezení kódu, který zadat do číselné klávesnice). Aby mohl hráč otevírat dveře tak jsou nutné dvě věci: Animace pro otevření dveří a script (program který popisuje chování objektu v určitých situacích), který animaci spustí. Animaci je možné vytvořit přímo v Unity, script je nutné vytvořit v programovacím prostředí.



Obr. 23: Parent objekt sloužící pro otvírání dveří

Dveře musí mít „pant“, okolo kterého se budou otáčet (v případě, že by tam tento „pant“ nebyl se při rotaci budou otáčet celé dveře). Toho bylo dosaženo umístěním prázdného objektu do místa kde by v realitě mohly být panty. Aby animace fungovala správně je nutné samotné dveře (fyzický model) dát v hierarchii jako podsložku „pantu“.

Dále je k objektu přiřazen script (*UnlockDoorOut*). Ve scriptu jsou definovány objekty, které poté uživatel jednoduše přetáhne do políček k tomu určených. Použití proměnných typu public také umožňuje dílčí otestování funkčnosti vynuceným spuštěním/změnou stavu některých proměnných (autor tuto funkci využil při testování správného zobrazování textů).

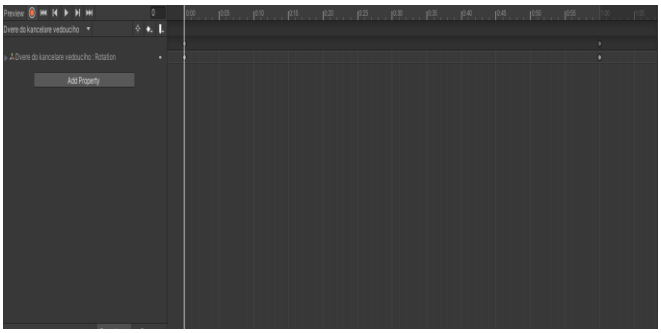
Reach tool (dosah interakce)

V rámci hry musí být vymezena vzdálenost, do které může hráč interagovat s objekty. Autor tohoto docílil pomocí neviditelného objektu („paže“), která trčí z hráčského

modelu dopředu a v případě, že se dotkne objektu pro interakci je objekt považován za dost blízký hráči.

Animace

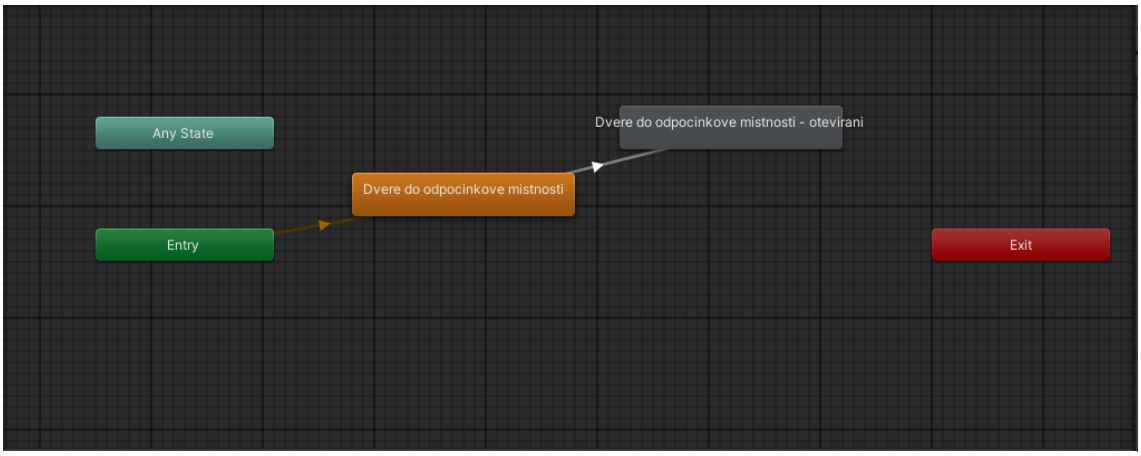
Aby mohly být v enginu použity animace je nutné naanimovat pohyb samotného objektu a vytvořit ovladač, který bude animace řídit (řešit přechody z jednoho stavu do druhého).



Obr. 24: Animace pro otevírání dveří

Nástroj pro animace funguje na bázi keyframů („klíčových snímků“). Unity standartně má obsaženy tři složky v keyframech: Rotaci, Posun a Měřítko. V této práci je využita pouze Rotace (efekt otevírání dveří byl dosažen rotací). Keyframy se v animaci používají pro označení hraničních stavů (kde končí jeden pohyb a začíná druhý).

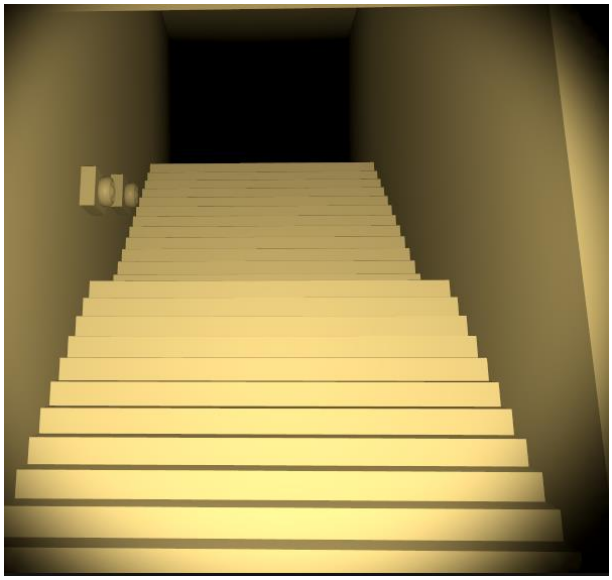
Ovladač ovládající animace funguje na bázi blokového schématu. V přechodech mezi jednotlivými stavy jsou podmínky umožňující přesun do dalšího bloku. Každý z bloků má k sobě přiřazenou animaci, která byla pro daný objekt vytvořena pomocí prostředí pro tvorbu animací.



Obr. 25: Schéma ovládající animace otevírání dveří

Schody

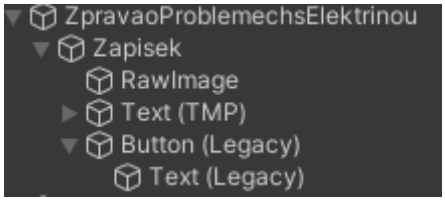
Ke konci hry musí hrát vyjít po schodech do dalšího patra, kde se nachází dveře ven. Zvolený fyzikální model (RigidBody) nedisponuje funkcí, která by umožnila jednoduše simulovat chůzi po schodech. Tento problém autor vyřešil použitím neviditelných plošin nakloněných tak, aby co nejpřesněji kopírovaly stoupání schodů. Hráč tak ve skutečnosti nestoupá po schodech, ale po plošinách v úhlovém rozmezí 30-40 stupňů.



Obr. 26: Schody pokryté neviditelnými platformami

Poznámky

Poznámky, které si může hráč přečíst, jsou vytvořeny kombinací scriptu a několika objektů Unity. V Unity je vytvořen obrázek sloužící jako pozadí, samotný text a tlačítko sloužící k položení poznámky. Script poté obstarává otevření této kombinace objektů a funkčnost tlačítka.



Obr. 27: Struktura poznámky ve hře

4.3 Řešení v Blenderu

Všechny modely použité v této práci byly autorem vytvořeny v 3D modelovacím programu Blender. Níže budou uvedeny příklady postupu tvorby použitých modelů.

Cedule

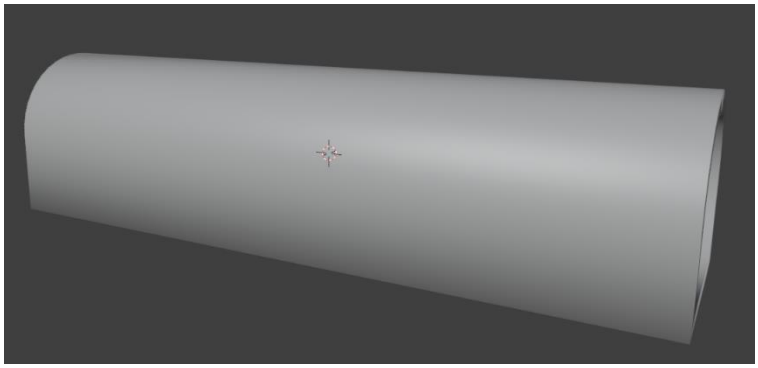
Cedule, která je k vidění na úvodní a finální obrazovce hry byla vytvořena zploštěním základního tělesa (kostky) na nejmenší povolenou tloušťku a poté nastavením dalších rozměrů na požadované hodnoty. K zaoblení byla využita funkce *Úkos* (Bevel přímo v Blenderu). Na celou ceduli byla aplikována textura zelené barvy s metalickým efektem.

Obr. 28: Cedule, která je k vidění v menu a na závěrečné obrazovce hry

Text byl přidán samostatně a napozicován do výsledné podoby. Využitý font je jeden z fontů dostupných v Blenderu. Na text byla aplikována bílá textura s lehce metalickým vzhledem.

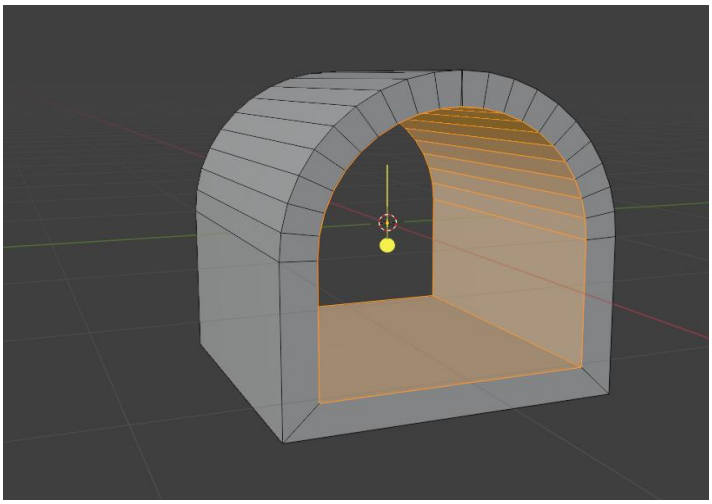
Bílé lemování na okrajích bylo vytvořeno zmenšením krychlí na požadované rozměry a jejich následným umístěním na kraje cedule.

Tunel



Obr. 29: Model tunelu v podobě, v jaké je využit ve hře

Model začal vytvořením krychle. Krychle byla zvětšena do požadované velikosti. Poté pomocí funkce Úkos a nastavení množství segmentů tvořících každou plochu. Poté pomocí funkce Inset (Vložení) a funkce Bridge Loops („Spojit smyčky“) byl vytvořen výběr, který vytváří vnitřek tunelu.



Obr. 30: Tunel před aplikací finálních rozměrů

4.4 Programové řešení

V rámci programového řešení bylo nutné propojit fyzické objekty s kódem běžícím na pozadí. Proměnné definované jako public mohou být ručně aktivovány nebo změněny přímo v prostředí Unity. Níže jsou uvedeny řešení některých klíčových funkcí pro samotnou hru.

Kamera

Pohyb kamery je ve hře (hlavně ve spojení s baterkou kvůli nízkému osvitu v lokaci) je důležitý kvůli správnému směřování hráčské postavy a kvůli interakci s předměty.

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

public class Camera : MonoBehaviour
{
  public float senX;
  public float senY;
  public bool canrotate;

  public Transform orientation;

  float xRotation;
  float yRotation;
```

Výpis.1: Script **Kamera** a proměnné definované v rámci tohoto scriptu

Script odpovídající za pohyb hráče sestává z proměnných *senX* (citlivost v ose X), *senY* (citlivost v ose Y) a proměnné typu bool *canrotate*. Do proměnných *senX* a *sensY* jsou ukládány hodnoty pohybu myši od uživatele v osách X a Y. *canrotate* je funkce, která vypíná otáčení kamery v určitých sekvencích.

Proměnné typu float *yRotation* a *xRotation* slouží jako mezistupeň mezi hráčem a samotnou hrou. Výsledná rotace kamery ve hře probíhá na základě hodnot z těchto dvou funkcí.

```
private void Start()
{
 canrotate = true;
 Cursor.lockState = CursorLockMode.Locked;
 Cursor.visible = false;
}
```

Výpis.2: Funkce *Start* deaktivuje kurzor a umožní rotaci kamery

Funkce *Start* v kódu slouží pro nastavení funkcí a stavů, které mohou hned na začátku nastat (kamera se může otáčet, kursor myši je zamčen v okně a není vidět).

```
private void Update()
{
  if(canrotate == true)
    {float mouseX = Input.GetAxisRaw("Mouse X") * Time.deltaTime * senX;
     float mouseY = Input.GetAxisRaw("Mouse Y") * Time.deltaTime * senY;

     yRotation += mouseX;
     xRotation -= mouseY;

     xRotation = Mathf.Clamp(xRotation, -90f, 90f);

     transform.rotation = Quaternion.Euler(xRotation, yRotation, 0);
     orientation.rotation = Quaternion.Euler(0, yRotation, 0);
     }
  }
}
```

Výpis.3: Funkce *Update* definovaná ve scriptu **Kamera**

Funkce *Update* se obnovuje každý snímek a jsou v ní vloženy funkce, u kterých záleží na jejich stavu a jejich stav se může ze sekundy na sekundu změnit. V tomto případě funkce neustále kontroluje, jestli se kamera může otáčet/došlo k rotaci kamery a přes zadané funkce provádí transformaci vstupu od uživatele a aplikuje jej ve hře.

Číselná klávesnice (Keypad)

Script popisuje chování číselné klávesnice („Keypadu“) sloužící k otevření dveří. Hráč se ve hře potká s číselnými klávesnice celkem dvakrát. Poprvé při snaze nastartovat generátor, podruhé při vstupu do jedné z kanceláří.

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


public class Keypad : MonoBehaviour
{
  public GameObject player;
  public GameObject keypad;
  public GameObject Animate;

  public Animator Ani;

  public TMPro.TextMeshProUGUI texts;
  public string answer = "12345";

  public bool animate;
  private int CharNumber = 0;

  private Movement movement;
  private Camera cam;
```

Výpis.4: Script **Keypad** a proměnné definované v rámci tohoto scriptu

V rámci scriptu byly definovány objekty *player* (hráčský objekt), *keypad* (fyzický objekt sloužící jako číselná klávesnice), *Animator* (objekt, který bude animován) a speciální objekt Animator *Ani* (objekt typu animátor slouží k řízení animací aplikovaných na daný objekt). Pro účely zamezení potenciálního přeplnění proměnné je definována proměnná *CharNumber*. Dále jsou zde objekty typu private *Movement* a *Camera* (tyto objekty budou uzamčeny během hráčovi interakce s číselnou klávesnicí, aby se nemohl hráč otáčet nebo odejít od modelu a mohl zadávat číselný kód).

```
void Start()
{
 movement = GameObject.Find("Player").GetComponent<Movement>();
 cam = GameObject.Find("PlayerCam").GetComponent<Camera>();
 keypad.SetActive(false);
}
```

Výpis.5: Funkce *Start* ve scriptu **Keypad**

Na začátku se hráč může volně pohybovat a rozhrání číselné klávesnice není vidět (účelem příkazu *FindObject* je najít objekt spojený s danou funkcí (např. funkcí *Movement*), aby tyto funkce mohly být později vypnuty/zapnuty).

```
public void Number(int Number)
{
 texts.text += Number.ToString();
 CharNumber++;
 ClearDisp();
}
```

Výpis.6: Funkce *Number* převádí stisknutí tlačítka na výstup na displeji a číslo ve stringu

Účelem funkce *Number* je převést tlačítka, která hráč stiskne, do proměnné typu string. Dále je zvyšována hodnota proměnné *CharNumber*. Hodnota této funkce je kontrolována ve funkci *ClearDisp* (účel této funkce bude vysvětlen samostatně).

```
public void Execute()
{
  if(texts.text == answer)
    {
     texts.text = "Odemceno";
    }
  else
    {
     texts.text = "Chyba";
    }
}
```

Výpis.7: Funkce *Execute* potvrdí zadaný kód a ověří správnost odpovědi

Funkce *Execute* (ve hře je propojena s tlačítkem *Potvrdit*) posoudí, jestli číselný kód zadaný hráčem odpovídá kódu, který je nastaven jako řešení. Pokud kód odpovídá, na displeji se objeví „Odemčeno“. V případě jakékoliv nesrovnalosti se objeví „Chyba“.

```
public void Exit()
{
 keypad.SetActive(false);
 movement.canmove = true;
 cam.canrotate = true;
 Cursor.visible = false;
 Clear();
}
```

Výpis.8: Funkce *Exit* vypne rozhraní číselné klávesnice a umožní pohyb

Funkce *Exit* (ve hře spojená s tlačítkem „Odejít“) deaktivuje rozhraní číselné klávesnice, povolí pohyb hráčovy postavy a pohyb kamery. Také opět zneviditelní kurzor.

```
public void Clear()
{
 texts.text = "";
 CharNumber = 0;
```

Výpis.9: Funkce *Clear* vymaže textové pole (např. při chybném zadání)

Funkce *Clear* (ve hře tlačítko „Vymazat“) vymaže text zadaný uživatelem (zobrazený v textovém poli ve vrchní části klávesnice).

```
void Update()
{
  if(texts.text == "Odemceno" && animate)
   {
    Ani.SetBool("Open", true);
   }
  if(keypad.activeInHierarchy)
   {
    movement.canmove = false;
    cam.canrotate = false;
    Cursor.visible = true;
    Cursor.lockState = CursorLockMode.None;
   }
}
```

Výpis.10: Funkce *Update* otevře dveře a spustí animaci dveří při správném kódu

Ve funkci *Update* je v podmínce *if* kontrola, jestli hráč správně zadal kód. Pokud ano, spustí se přidružená animace (dveře se otevřou). Druhá podmínka kontroluje, jestli hráč spustil rozhraní číselné klávesnice (jestli je objekt *Keypad* aktivní v hierarchii scény

v herním enginu) a pokud je aktivní, tak deaktivuje pro hráče možnost se pohybovat, a otáčet kamerou. Dále odemkne možnost s kurzorem normálně ohybovat a zviditelní jej.

```
public void ClearDisp()
 {
  if (CharNumber > 4)
   {
    Clear();
    CharNumber = 0;
   }
 }
}
```

Výpis.11: Funkce *ClearDisp* vymaže textové pole při pokusu o přeplnění

Na závěr scriptu byla vytvořena funkce *ClearDisp* určená k omezení množství zadaných vstupů do textového pole ve funkci *Number*. Funkce *ClearDisp* se aktivuje v případě, že se hráč pokusí zadat více než je počet číslic samotného kódu (podmínka *if* se aktivuje v případě, že se hráč pokusí zadat pátou číslici nebo zadávat číslice, pokud je na displeji text). V ten moment funkce vymaže text v textovém poli.

Displej generátoru

Podobný kód jako výše byl použit i pro displej, pomocí kterého musí hráč obnovit elektřinu na stanici. Rozdílné funkce byly v obrázku červeně zvýrazněny.

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

public class KeypadGenerator : MonoBehaviour
{
  public GameObject player;
  public GameObject Keypad;
  public GameObject invPower;

  public bool isPowered;
  public TMPro.TextMeshProUGUI texts;
  public string answer = "12345";
  public int CharNumber = 0;

  private Movement movement;
  private Camera cam;

  void Start()
  {
   movement = GameObject.Find("Player").GetComponent<Movement>();
   cam = GameObject.Find("PlayerCam").GetComponent<Camera>();
   Keypad.SetActive(false);
   isPowered = false;
   invPower.SetActive(false);
  }

  public void Number(int Number)
  {
   texts.text += Number.ToString();
   CharNumber++;
   ClearDisp();
  }
```

Výpis.12: Script **Displej generátoru** je podobný jako ten pro **Číselnou klávesnici**. Liší se pouze v podtržených částech

Ve funkci *Start* byly přidány řádky kódu znemožňující hráči aktivovat číselnou klávesnici u dveří kanceláře (nejede elektřina). Hráč musí tedy prvně nastartovat generátor.

```
public void Execute()
{
  if(texts.text == answer)
    {
     texts.text = "Spoustim generator";
     invPower.SetActive(true);
    }
  else
    {
     texts.text = "Chyba";
    }
}
public void Clear()
{
  texts.text = "";
  CharNumber = 0;
}
public void Exit()
{
  Keypad.SetActive(false);
  movement.canmove = true;
  cam.canrotate = true;
  Cursor.visible = false;
  Clear();
}
```

Výpis.13: Script **Displej generátoru** je podobný jako ten pro **Číselnou klávesnici**. Liší se pouze v podtržených částech

```
 void Update()
 {
   if(invPower.activeInHierarchy)
     {
      isPowered = true;
     }
    if(Keypad.activeInHierarchy)
     {
      movement.canmove = false;
      cam.canrotate = false;
      Cursor.visible = true;
      Cursor.lockState =   CursorLockMode.None;
     }
 }
 public void ClearDisp()
 {
   if (CharNumber > 4)
     {
      Clear();
      CharNumber = 0;
     }
 }
}
```

Výpis.14: Script **Displej generátoru** je podobný jako ten pro **Číselnou klávesnici**. Liší se pouze v podtržených částech

Proměnné *isPowered* a *invPower* začínají na hodnotě *False*. Poté co hráč nastartuje generátor se jejich hodnoty změní na *True* a hráč může zadat číselný kód na druhé číselné klávesnici. Funkce *ClearDisp* je zde identická jako v případě scriptu pro číselnou klávesnici.

Pohyb

Script **Movement** obsahuje vzorce, pomocí kterých je počítám pohyb hráčské postavy.

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

public class Movement : MonoBehaviour
{
  public bool canmove;
  [Header("Movement")]
  public float moveSpeed;
  public float groundDrag;

  [Header("Ground Check")]
  public float playerHeight;
  public LayerMask whatIsGround;
  bool grounded;

  public Transform orientation;

  float horizontalInput;
  float verticalInput;

  Vector3 moveDirection;

  Rigidbody rb;
```

Výpis.15: Obecné proměnné a proměnné sloužící specificky k úpravě pohybu

Ve scriptu je definována proměnná typu bool *canmove* (pokud je proměnná *False* hráč se nemůže pohybovat). Pro samotný pohyb jsou definovány proměnné typu float *moveSpeed* (rychlost s jakou se hráč může pohybovat) a *groundDrag* („tření“ když se postava pohybuje po zemi. Slouží k omezení pocitu „plavání“ při pohybu).

Pro ověření, jestli se hráč nachází na zemi jsou definovány proměnné *playerHeight* (z jaké výšky je vysílán paprsek, který ověřuje, jestli se hráč nachází na zemi), vrstvu *whatIsGround* (vrstva je aplikována na všechny povrchy, po kterých se hráč pohybuje) a proměnnou typu bool grounded (proměnná na hodnotě *True* říká, že je hráč na zemi a má se aplikovat tření). Funkce *Orientation* je definována kvůli sjednocení pohybu a kamery.

Dále jsou definovány proměnné typu float *horizontalInput* (pro natočení kamery v ose X) a *verticalInput* (pro natočení v ose Y). Funkce *moveDirection* je definována jako typ Vector3 (umožňuje pohyb ve více osách zároveň). Na zároveň je definován fyzikální model *RigidBody*.

```
private void Start()
{
 canmove = true;
 rb = GetComponent<Rigidbody>();
 rb.freezeRotation = true;
}
```

Výpis.16: Hráč se na začátku může pohybovat. Dále je uzamčena rotace modelu

Při spuštění hry se hráč může pohybovat, je uzamčena rotace modelu je provedeno propojení scriptu s fyzickým RigidBody modelem.

```
private void FixedUpdate()
{
 if(canmove == true)
  {
   rb.AddForce(Vector3.down * 10f, ForceMode.Force);
   MovePlayer();
  }
}
```

Výpis.17: Pokud se hráč může pohybovat, podmínka se aktivuje a předá hodnoty funkci *MovePlayer*

Ve funkci *FixedUpdate* probíhá aplikace síly k vektoru pohybu. Zatímco *Update* se aktualizuje každý snímek, *FixedUpdate* se aktualizuje stanovený počet snímků a vztahuje se k fyzice.

```
private void MovePlayer()
{
 moveDirection = orientation.forward * verticalInput + orientation.right
 *  horizontalInput;
 rb.AddForce(moveDirection.normalized * moveSpeed * 10f, ForceMode.Force);
}
```

Výpis.18: Ve směru, kterým se hráč chce pohybovat je na model hráče aplikována síla

Ve funkci *MovePlayer* je popsán vztah, podle kterého je vypočítána síla, která bude aplikována na model hráče.

```
private void MyInput()
{
 horizontalInput = Input.GetAxisRaw("Horizontal");
 verticalInput = Input.GetAxisRaw("Vertical");
}
```

Výpis.19: Funkce pro zaznamenání pohybu myši a přenesení do hry

MyInput funkce zaznamenává pohyb myši v osách X a Y a ukládá hodnoty do funkcí (pohyb v X do funkce *horizontalInput*, pohyb v Y do funkce *verticalInput*).

```
private void SpeedControl()
{
 Vector3 flatVel = new Vector3(rb.velocity.x, 0f, rb.velocity.z);
 if(flatVel.magnitude > moveSpeed)
  {
   Vector3 limitedVel = flatVel.normalized * moveSpeed;
   rb.velocity = new Vector3(limitedVel.x, rb.velocity.y, limitedVel.z);
  }
  }
}
```

Výpis.20: Funkce *SpeedControl* omezuje maximální rychlost, kterou se hráč pohybuje

SpeedControl funkce omezuje maximální rychlost, kterou se hráč může pohybovat (fyzikální engine zabudovaný v Unity nedodržuje uživatelem zadanou rychlost. Skutečná hodnota je o něco vyšší). Pokud je zaznamenána vyšší rychlost, než je zadána, spustí se podmínka *if*, která rychlost omezí.

```
private void Update()
{
 grounded = Physics.Raycast(transform.position, Vector3.down,
 playerHeight * 0.7f + 0.5f, whatIsGround);
 MyInput();
 SpeedControl();
 if (grounded)
  {
   rb.drag = groundDrag;
  }
}
```

Výpis.21: Funkce *Update* kontroluje, jestli se hráč pohybuje po zemi

Ve funkci *Update* je popsána formule, pomocí které se metodou raycasting kontroluje, jestli je hráč na ploše, která je definována jako země (proměnná *grounded*). Dále jsou spuštěny funkce pro pohyb kamery (*MyInput*) a omezovače rychlosti (*SpeedControl*). Na závěr je zařazena podmínka *if* která aplikuje tření pokud se hráč nachází na zemi.

Rozhrání číselné klávesnice

Script **OpenKeypad** kontroluje, jestli jsou splněny podmínky pro zobrazení rozhrání číselné klávesnice. Hráč musí obnovit energii na stanici a poté může na číselné klávesnici zadat kód pro postup.

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

public class OpenKeypad : MonoBehaviour
{
  public GameObject keypad;
  public GameObject keypadText;
  public GameObject invPower;
  public GameObject notPoweredText;

  public bool inReach;

  void Start()
  {
   inReach = false;
```

Výpis.22: Objekty, proměnná a funkce *Start* scriptu **OpenKeypad**

V rámci scriptu byly definovány objekty *keypad* (objekt se kterým bude hráč interakovat), *keypadText* (text, který se zobrazí, když je hráč poblíž klávesnice), *invPower* (objekt který se aktivuje, pokud hráč aktivoval generátor) a *notPoweredText* (text který se zobrazí, pokud se hráč pokusí zadat kód před tím než je obnovena dodávka elektřiny). Dále je definována proměnná bool *inReach* (kontrola, jestli je hráč v dosahu modelu a může s ním interagovat). Tato funkce je na začátku nastavena na hodnotu *False*.

```
void OnTriggerEnter(Collider other)
{
 if (other.gameObject.tag == "Reach")
  {
   inReach = true;
   keypadText.SetActive(true);
  }
}
private void OnTriggerExit(Collider other)
  {
   if (other.gameObject.tag == "Reach")
    {
     inReach = false;
     keypadText.SetActive(false);
     notPoweredText.SetActive(false);
    }
  }
```

Výpis.23: Zobrazení textu pomocí funkcí *OnTriggerEnter* a *OnTriggerExit*

Funkce *OnTriggerEnter* a *OnTriggerExit* popisují chování, když hráč vstoupí do okolí fyzického modelu číselné klávesnice (*OnTriggerEnter*) a když jej opustí (*OnTriggerExit*). V případě vstupu do blízkosti klávesnice se objeví možnost s ní interagovat (text a tlačítko pro interakci), v případě opuštění text zmizí.

```
    void Update()
    {
      if(invPower.activeInHierarchy && Input.GetButtonDown("Interact") && inReach)
       {
        keypad.SetActive(true);
        keypadText.SetActive(false);
       }
      else if (Input.GetButtonDown("Interact") && inReach)
       {
        keypadText.SetActive(false);
        notPoweredText.SetActive(true);
       }
    }
}
```

Výpis.24: Funkce *Update* kontroluje, jestli hráč aktivoval generátor

Ve funkci *Update* jsou dvě kontrolní *if* podmínky. První se aktivuje v případě, že hráč obnovil dodávku elektřiny a pokouší se zadat kód. V tomto případě se aktivuje rozhrání samotné klávesnice. Druhá podmínka se spustí v případě, že hráč dosud dodávku elektřiny neobnovil. V takovém případě se objeví text, že číselná klávesnice není pod proudem a kód není možné zadat.

Sebrání klíče

Script **PickupKey** je ve hře využita dvakrát pro sebrání klíčů. Když hráč sebere daný klíč funkce odstraní objekt klíče ze světa a aktivuje se v inventáři hráče.

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

public class PickupKey : MonoBehaviour
{
  public GameObject key;
  public GameObject inv;
  public GameObject pickupText;

  public bool inReach;
  void Start()
  {
   inReach = false;
   pickupText.SetActive(false);
   inv.SetActive(false);
  }
```

Výpis.25: Proměnné, objekty a funkce *Start* scriptu ***PickupKey***

V rámci funkce jsou definovány objekty *key* (fyzický model klíče v prostředí), *inv* (objekt v inventáři hráče. Ten se aktivuje, když hráč klíč sebere) a *pickupText* (text který se objeví, když je hráč v okolí klíče). Dále je definována proměnná typu bool *inReach* (fungování popsáno ve scriptu **OpenKeypad**).

Při startu hry není hráč v okolí žádného klíče a žádný klíč u sebe nemá. Proto jsou ve funkci *Start* objekty v dosahu (funkce *inReach*, *inv* a *pickupText*) nastaveny na stav *False*.

```
private void OnTriggerEnter(Collider other)
{
 if(other.gameObject.tag == "Reach")
  {
   inReach = true;
   pickupText.SetActive(true);
  }
}
private void OnTriggerExit(Collider other)
{
 if(other.gameObject.tag == "Reach")
  {
   inReach = false;
   pickupText.SetActive(false);
  }
}
```

Výpis.26: Zobrazení textu pomocí funkcí *OnTriggerEnter* a *OnTriggerExit*

Funkce *OnTriggerEnter* a *OnTriggerExit* popisují chování totožně jako v případě scriptu **OpenKeypad**.

```
 void Update()
 {
  if(inReach && Input.GetButtonDown("Interact"))
   {
    key.SetActive(false);
    inv.SetActive(true);
    pickupText.SetActive(false);
   }
 }
}
```

Výpis.27: Funkce *Update* přidá klíč do inventáře, pokud hráč v jeho okolí stiskl tlačítko pro interakci

Pokud je hráč v okolí klíče a stiskne tlačítko pro interakci, aktivuje se podmínka *if* která přidá hráči objekt do inventáře, deaktivuje fyzický model klíče ve světě a deaktivuje text.

Čtení poznámek

Ve hře může hráč najít poznámky. Na nich mohou být kódy nutné pro postup dále nebo informace dokreslující svět hry. Script **ReadNotes** zprostředkovává interakci hráče s těmito poznámka a umožňuje mu poznámky číst.

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

public class ReadNotes : MonoBehaviour
{
  public GameObject player;
  public GameObject noteUI;
  public GameObject pickUpText;

  public bool inReach;

  private Movement movement;
  private Camera cam;
  void Start()
  {
   movement = GameObject.Find("Player").GetComponent<Movement>();
   cam = GameObject.Find("PlayerCam").GetComponent<Camera>();
   noteUI.SetActive(false);
   pickUpText.SetActive(false);
   inReach = false;
  }
```

Výpis.28: Funkce *Start* a proměnné potřebné pro script **ReadNotes**

Pro tento script byly definovány objekty *player* (vysvětlen a popsán dříve), *NoteUI* (samotná poznámka s textem), *pickupText* (text, který se zobrazí, když hráč namíří kameru na poznámku) a *inReach* (funkce popsána ve scriptu **OpenKeypad**).

Při startu hry není hráč v okolí žádné poznámky a může se pohybovat. Proto jsou ve funkci *Start* objekty v dosahu (funkce *inReach*) a funkce spojené se zobrazením poznámky (*pickupText* a *noteUI*) nastaveny na stav *False*.

```
void OnTriggerEnter(Collider other)
{
  if(other.gameObject.tag == "Reach")
   {
    inReach = true;
    pickUpText.SetActive(true);
   }
}
void OnTriggerExit(Collider other)
{
  if(other.gameObject.tag == "Reach")
   {
    inReach = false;
    pickUpText.SetActive(false);
   }
```

Výpis.29: Funkce *OnTriggerEnter* a *OnTriggerExit* popisující chování, když hráč vstoupí do/opustí okolí objektu

Funkce *OnTriggerEnter* a *OnTriggerExit* popisují chování totožně jako v případě scriptu **OpenKeypad**.

```
void Update()
{
 if(Input.GetButtonDown("Interact") && inReach)
  {
   noteUI.SetActive(true);
   movement.canmove = false;
   cam.canrotate = false;
   Cursor.visible = true;
   Cursor.lockState = CursorLockMode.None;
  }
}
```

Výpis.30: Funkce *Update* kontroluje, jestli hráč v okolí poznámky stiskl tlačítko pro interakci

Pokud je hráč v dosahu a stiskne tlačítko pro interakci spustí se *if* podmínka ve funkci *Update*. Ta otevře menu, ve kterém si hráč může poznámku přečíst. Zároveň znemožní hráči pohyb kamerou a modelem a zviditelní kurzor.

```
public void ExitButton()
{
 noteUI.SetActive(false);
 movement.canmove = true;
 cam.canrotate = true;
 Cursor.visible = false;
}
```

Výpis.31: Funkce popisující fungování tlačítka Položit když si hráč čte poznámky

Funkce *ExitButton* (ve hře spojená s tlačítkem „Položit“) vypne rozhrání poznámky, odemkne pohyb kamery a hráče a zneviditelní kurzor.

Otevření dveří ven

Script **UnlockDoorsOut** popisuje, jak fungují dveře na konci hry. Po nalezení klíče a odemčení těchto dveří hráč úspěšně dokončil hru.

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

public class UnlockDoorsOut : MonoBehaviour
{
  public GameObject KeyInv;
  public GameObject openText;
  public GameObject lockedText;

  public bool inReach;
  public bool unlocked;
  public bool locked;
  public bool hasKey;

  void Start()
  {
    inReach = false;
    hasKey = false;
    unlocked = false;
    locked = true;
  }
```

Výpis.32: Proměnné a funkce *Start* scriptu **OpenDoorsOut**

Ve funkci jsou definovány objekty *KeyInv* (objekt v inventáři hráče, který se aktivuje, pokud hráč sebral příslušný klíč), *openText* (text který se objeví, pokud je hráč v blízkosti dveří) a *lockedText* (text, který se zobrazí, pokud se hráč pokusí otevřít dveře bez příslušného klíče). Dále jsou definovány proměnné typu bool *inReach* (funkce popsána ve scriptu **OpenKeypad**), *unlocked* (jestli může hráč dveře otevřít), *locked* (platí v případě, že hráč nemá klíč ke konkrétním dveřím) a *lockedText* (text, který se zobrazí, pokud se hráč pokusí otevřít dveře bez odpovídajícího klíče).

Ve funkci *Start* jsou nastaveny výchozí hodnoty (hráč není poblíž dveří, nemá u sebe žádné klíče a dveře není možné otevřít).

```
void OnTriggerEnter(Collider other)
{
 if(other.gameObject.tag == "Reach")
  {
   inReach = true;
   openText.SetActive(true);
  }

}
 void OnTriggerExit(Collider other)
 {
  if (other.gameObject.tag == "Reach")
   {
    inReach = false;
    openText.SetActive(false);
    lockedText.SetActive(false);
   }
  }
```

Výpis.33: Funkce *OnTriggerEnter* a *OnTriggerExit* popisující chování, když hráč vstoupí do/opustí okolí objektu

Funkce *OnTriggerEnter* a *OnTriggerExit* popisují chování totožně jako v případě scriptu **OpenKeypad**.

```
    void Update()
    {
     if(KeyInv.activeInHierarchy)
      {
       locked = false;
       hasKey = true;
      }
     else
      {
       hasKey = false;
      }
     if (hasKey && inReach && Input.GetButtonDown("Interact"))
      {
       unlocked = true;
       Cursor.visible = true;
       SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
      }
     if (locked == true && inReach && Input.GetButtonDown("Interact"))
      {
       openText.SetActive(false);
       lockedText.SetActive(true);
      }
     if (hasKey && inReach && unlocked)
      {
       openText.SetActive(false);
      }
    }
   }
```

Výpis.34: Podmínky *if* ve scriptu **OpenDoorsOut** popisující různé možnosti, které mohou nastat

Ve funkci *Update* jsou pomocí *if* podmínek řešeny stavy, které mohou nastat. První podmínka if změny stavy proměnných *locked* a *hasKey* pokud byl klíč hráčem sebrán. Druhá podmínka *if* se aktivuje, pokud hráč má klíč a spustil interakci s dveřmi. Stav proměnné *unlocked* se změní na *True*, zviditelní se kurzor a hra se přesune o scénu dále (hráč dokončil demo, následují stránky s příběhovým textem). Třetí podmínka *if* se spustí, pokud se hráč pokusí dveře otevřít i v případě, že nenašel klíč. Text popisující interakci s dveřmi a objeví se *lockedText* (text říkající hráči, že dveře jsou zamčeny a nemá správný klíč).

5 ZÁVĚR

Napříč prací byly rozebrány postupy, které autor využil pro realizaci jednoduché hry. Ve hře je možné interagovat s předměty, svět je dotvořen pomocí poznámek a informací o tom co hratelné pasáži předcházelo, a to co následovalo po ní.

Výslednou hru je možno vylepšit v několika oblastech: v grafice (přidáním textur a dalších objektů pro zvýšení realistického vzhledu), rozšířením herní smyčky (nové lokace, které může hráč navštívit, nové hádanky, které může hráč vyřešit), po zvukové stránce (přidáním zvuků hráčova dechu, hráčových kroků, otevírání dveří, pípání klávesnice aj.), po příběhové stránce (více informací o světě ve kterém se hráč nachází, namluvené nahrávky) a po stránce technické (např. přidáním dalších obtížností nebo přidáním monstra, které bude hráče pronásledovat, reagovat na jeho akce a bylo by řízeno jednoduchou umělou inteligencí).

Pro účely tvorby hororové hry či logické adventury z pohledu první osoby je v tuto chvíli tato práce platformou s funkčními mechanismy, které umožňují jak vylepšení, tak další rozvinutí.

SEZNAM POUŽITÉ LITERATURY

[1] BHATTI, Kaushik. *How do you learn and master Blender new game engine or a game framework quickly and efficiently?* Online. 2023, 28.12.2023. Dostupné z: https://www.linkedin.com/advice/0/how-do-you-learn-master-new-game-engine-framework-quickly. [cit. 2024-05-18].

[2] *Classic Studio Postmortem: Lucasfilm Games*. Online. 2014. Dostupné z: Gamespot, https://www.youtube.com/watch?v=HDvEFbh6I2g. [cit. 2024-05-21].

[3] BEVAN, Mike. *The SCUMM Diary: Stories behind one of the greatest game engines ever made*. Online. In: Game Devoloper. 2000, s. 1. Dostupné z: https://web.archive.org/web/20160323193531/http://www.gamasutra.com/view/feature/196009/the_scumm_diary_stories_behind_.php. [cit. 2024-04-07].

[4] SRP, Honza. *Ohlédněte se s námi za nejpopulárnějšími enginy herní historie*. Online. In: IDnes.cz. 1998, s. 1. Dostupné z: https://www.idnes.cz/hry/magazin/popularni-enginy-minulosti.A151014_141712_bw-klasika_srp. [cit. 2024-04-07]

[5] *Maniac Mansion*. Online. In: Classic Gaming. 2016. Dostupné z: https://classicgaming.cc/pc/maniac-mansion/. [cit. 2024-05-21].

[6] ANNAND, Gavin. *The History of the id Tech Engine*. Online. In: SUPERJUMP. 2018. Dostupné z: https://www.superjumpmagazine.com/the-history-of-the-id-tech-engine/. [cit. 2024-04-07].

[7] *Lode's Computer Graphics Tutorial Raycasting*. Online. Lode's Computer Graphics Tutorial. 2004, s. 1. Dostupné z: https://lodev.org/cgtutor/raycasting.html. [cit. 2024-04-07].

[8] SCHUYTEMA, Paul C. The Lighter Side of Doom. Online. *Computer Gaming World*. 1994, roč. 1994, č. 121, s. 140-142. Dostupné z: https://www.cgwmuseum.org/galleries/issues/cgw_121.pdf. [cit. 2024-05-21].

[9] KEPPERT, Standa. *Jakou cestu urazil vývoj her 2: herní engines*. Online. In: SYSTEUM. 2011, s. 1. Dostupné z: https://www.systeum.cz/cs/blog/jakou-cestu-urazil-vyvoj-her-2-herni-engines. [cit. 2024-04-08].

[10] GITHUB. *Id-Software / Quake*. Online. In: GitHub. 2011. Dostupné z: https://github.com/id-Software/Quake. [cit. 2024-04-08].

[11] *GPU Gems*. Online. In: NVIDIA.DEVOLOPER. 2001, s. 1. Dostupné z: https://web.archive.org/web/20171003225316/https://developer.nvidia.com/gpugems/GPUGems2/gpugems2_frontmatter.html. [cit. 2024-04-08].

[12] LILLY, Paul. *Doom to Dunia: A Visual History of 3D Game Engines*. Online. In: Tom´s HARDWARE. 1998, s. 1. Dostupné z: https://web.archive.org/web/20090724065520/http://www.maximumpc.com/article/features/3d_game_engines?page=0%2C3. [cit. 2024-04-08].

[13] KEIGHLEY, Geoffrey. *Blinded By Reality: The True Story Behind the Creation of Unreal*. Online. In: Gamespot. 1996, s. 1. Dostupné z: https://web.archive.org/web/20010519154729/http://www.gamespot.com/features/makeunreal/. [cit. 2024-04-08].

[14] WALES, Matt. *Epic Games' classic first-person shooter Unreal is free right now*. Online. In: Eurogamer. 2018. Dostupné z: https://www.eurogamer.net/you-can-get-the-original-unreal-for-free-right-now. [cit. 2024-04-08].

[15] AXON, Samuel. *Unity at 10: For better—or worse—game development has never been easier*. Online. In: Ars Technica. 1999, s. 1. Dostupné z: https://web.archive.org/web/20181005025906/https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/. [cit. 2024-04-08].

[16] UNITY TECHNOLOGIES. *Government & Aerospace*. Online. UNITY TECHNOLOGIES. Unity. 2021. Dostupné z: https://web.archive.org/web/20210913125838/https://unity.com/solutions/government-aerospace. [cit. 2024-05-21].

[17] *Macintosh Repository: Gooball*. Online. In: Macintosh Repository. 2020. Dostupné z: https://www.macintoshrepository.org/26983-gooball. [cit. 2024-04-08].

[18] CODENIX. *About Codenix*. Online. CODENIX. Codenix. 2014. Dostupné z: https://web.archive.org/web/20140625091420/http://www.codenix.com/. [cit. 2024-05-21].

[19] LINIETSKY, Juan [@reduzio]. *Tweet*. Online. 2018. Dostupné z: Twitter.com, https://twitter.com/reduzio/status/998185981101658113. [cit. 2024-05-21].

[20] LINIETSKY, Juan. *Godot history in images!*. Online. GodotEngine. 2011, s. 1. Dostupné z: https://godotengine.org/article/godot-history-images/. [cit. 2024-04-08].

[21] LINIETSKY, Juan. *Godot 2.0: Talking with the Creator*. Online. In: 80.lv. 2014, s. 1. Dostupné z: https://80.lv/articles/godot2-interview/. [cit. 2024-04-08].

[22] HILL, Paul. *Godot Engine arrives on Epic Games Store making it easier to download*. Online. In: Neowin. 2001, s. 1. Dostupné z: https://www.neowin.net/news/godot-engine-arrives-on-epic-games-store-making-it-easier-to-download/. [cit. 2024-04-08].

[23] *Specialized 2D workflow for games and apps*. Online. GodotEngine. 2011. Dostupné z: https://godotengine.org/features/#features_2d. [cit. 2024-04-08].

[24] *Simple yet powerful 3D engine*. Online. GodotEngine. 2011. Dostupné z: https://godotengine.org/features/#features_3d. [cit. 2024-04-08].

[25] COHEN, Peter. *Unity 2.0 game engine now available*. Online. In: Macworld. 1997. Dostupné z: https://web.archive.org/web/20190320210247/https://www.macworld.com/article/1060484/unity.html. [cit. 2024-04-17].

[26] BRODKIN, Jon. *How Unity3D Became a Game-Development Beast*. Online. In: Dice. 1998. Dostupné z: https://web.archive.org/web/20181019160750/https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/. [cit. 2024-04-17].

[27] GIRARD, Dave. *Unity 3 brings very expensive dev tools at a very low price*. Online. In: Ars Technica. 1999. Dostupné z: https://web.archive.org/web/20140224085322/http://arstechnica.com/information-technology/2010/09/unity-3-brings-very-expensive-dev-tools-at-a-very-low-price. [cit. 2024-04-17].

[28] COHEN, David. *How Facebook Integrated With The Unity Game Engine*. Online. In: Adweek. 1996. Dostupné z: https://web.archive.org/web/20181212211302/https://www.adweek.com/digital/unity-sdk-out-of-beta/. [cit. 2024-04-17].

[29] COHEN, David. *Facebook Developing New PC Gaming Platform; Teams Up With Unity Technologies*. Online. In: Adweek. 1996. Dostupné z: https://web.archive.org/web/20181211184313/https://www.adweek.com/digital/facebook-developing-pc-gaming-platform-unity-technologies/. [cit. 2024-04-17].

[30] TACH, Dave. *Unity 4.0 available for download today with DX 11 support and Linux preview*. Online. In: Polygon. 1996. Dostupné z: https://web.archive.org/web/20190320210245/https://www.polygon.com/2012/11/14/3645122/unity-4-0-available-download. [cit. 2024-04-17].

[31] GRUBB, Jeff. *Unity 5.6 launches with support for Vulkan graphics, Nintendo Switch, and more*. Online. In: VentureBeat. 2006. Dostupné z: https://web.archive.org/web/20190320210243/https://venturebeat.com/2017/03/31/unity-5-6-launches-with-support-for-vulkan-graphics-nintendo-switch-and-more/. [cit. 2024-04-17].

[32] ORLAND, Kyle. *How new graphics effects can make Unity Engine games look less generic*. Online. In: Ars Technica. 1999. Dostupné z: https://web.archive.org/web/20190320210242/https://arstechnica.com/gaming/2016/03/how-new-graphics-effects-can-make-unity-engine-games-look-less-generic/. [cit. 2024-04-17].

[33] KUMPARAK, Greg. *Unity 5 Announced With Better Lighting, Better Audio, And "Early" Support For Plugin-Free Browser Games*. Online. In: Techcrunch. 2005. Dostupné z: https://web.archive.org/web/20190219220925/https://techcrunch.com/2014/03/18/unity-5-announced-with-early-support-for-plugin-free-browser-games/. [cit. 2024-04-17].

[34] *Unity Technologies – The World's Leading Game Engine*. Online. In: Nanalyze. 2003. Dostupné z: https://web.archive.org/web/20190212101854/https://www.nanalyze.com/2017/10/unity-technologies-leading-game-engine/. [cit. 2024-04-17].

[35] MCALOON, Alissa. *Unity 2017.2 brings Autodesk integration into the fold*. Online. In: Game Devoloper. 2000. Dostupné z: https://web.archive.org/web/20190320210245/https://www.gamasutra.com/view/news/307050/Unity_20172_brings_Autodesk_integration_into_the_fold.php. [cit. 2024-04-17].

[36] BATCHELOR, James. *Unity 2018 detailed in GDC keynote*. Online. In: Games Industry. 2002. Dostupné z: https://web.archive.org/web/20190320210246/https://www.gamesindustry.biz/articles/2018-03-20-unity-2018-detailed-in-gdc-keynote. [cit. 2024-04-17].

[37] SPRIGG, Sam. *Unity MARS Augmented and Mixed Reality authoring studio now available*. Online. In: Auganix. 2017. Dostupné z: https://web.archive.org/web/20200926190900/https://www.auganix.org/unity-mars-augmented-and-mixed-reality-authoring-studio-now-available/. [cit. 2024-04-17].

[38] FRANCIS, Bryant. *The next version of Unity will be called Unity 6*. Online. In: Game Devoloper. 2000. Dostupné z: https://web.archive.org/web/20231116095642/https://www.gamedeveloper.com/business/the-next-version-of-unity-will-be-called-unity-6. [cit. 2024-04-17].

[39] BATCHELOR, James. *Unity dropping major updates in favour of date-based model*. Online. In: Games Industry. 2002. Dostupné z: https://web.archive.org/web/20170319022436/http://www.gamesindustry.biz/articles/2016-12-14-unity-dropping-major-updates-in-favour-of-date-based-model. [cit. 2024-04-17].

[40] *Frostbite 3 'Battlefield 4' Demo; Engine Will Remain Exclusive to EA Games*. Online. In: Game Rant. 2002. Dostupné z: https://web.archive.org/web/20211019013133/https://gamerant.com/frostbite-3-engine-exclusive-ea-battlefield-4-tech-demo/. [cit. 2024-04-17].

[41] SEPPALA, Timothy J. *From Battlefield to Mass Effect: How one engine is shaping the future of EA Games*. Online. In: Engadget. 2004. Dostupné z: https://web.archive.org/web/20211019010815/https://www.engadget.com/2013-11-19-electronic-arts-frostbite-battlefield-mass-effect.html. [cit. 2024-04-17].

[42] WILLIAMS, Mike. *How the Frostbite Engine Became a Nightmare for EA in General, and BioWare in Particular*. Online. In: VG247. 2008. Dostupné z: https://web.archive.org/web/20201124004844/https://www.usgamer.net/articles/ea-frostbite-engine-history-bioware-ea-sports. [cit. 2024-04-17].

[43] THIRLWELL, Edwin Evans. *Game developers shed light on why Starfield didn't use Unreal Engine*. Online. In: RockPaperShotgun. 2007. Dostupné z: https://www.rockpapershotgun.com/game-developers-shed-light-on-why-starfield-didnt-use-unreal-engine-5. [cit. 2024-04-17].

[44] *The Graphics Technology of Fallout 4*. Online. In: BETHESDA. Bethesda.net. 2015. Dostupné z: https://bethesda.net/en/article/2Y37xeRPeUW0EgkgaKW8oA/the-graphics-technology-of-fallout-4. [cit. 2024-05-19].

[45] SMITH, Dave. *An inside look at the technology powering 'Fallout 4,' one of the biggest games of the year*. Online. In: Business Insider. 1998. Dostupné z: https://www.businessinsider.com/fallout-4-graphics-technology-2015-11. [cit. 2024-04-17].

[46] BROWN, Fraser. *The engine behind Paradox Development Studio's future games*. Online. In: VentureBeat. 2006. Dostupné z: https://venturebeat.com/pc-gaming/the-engine-behind-paradox-development-studios-future-games/. [cit. 2024-04-17].

[47] BLENDER FOUNDATION. *Blender.org*. Online. BLENDER FOUNDATION. Blender. 2002. Dostupné z: https://www.blender.org/. [cit. 2024-04-17].

SEZNAM ZKRATEK, SYMBOLŮ, OBRÁZKŮ A TABULEK

SEZNAM PŘÍLOH

Stanice Heaven Square.zip

PŘÍLOHY