Středoškolská technika 2022

Setkání a prezentace prací středoškolských studentů na ČVUT

Tvorba počítačových her

Jakub Podzimek

První soukromé jazykové gymnázium

Brandlova 875, Hradec Králové

Poděkování

Chtěl bych poděkovat Mgr. Michaele Toman za odborné vedení při tvorbě Tvořivé

Klávesnice, za její rady, podněty a komentáře.

Také děkuji Karlu Podzimku za trpělivou pomoc a podporu při jazykových úpravách práce.

Děkuji spolužáku Jakubu Zimmerovi za vzájemné konzultace při zpracování praktické části a beta testování vytvořené hry.

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně, a že jsem v seznamu použité literatury uvedl všechny prameny, ze kterých jsem vycházel.

.....

V Hradci Králové dne 20. ledna 2022 Jakub Podzimek

Anotace

PODZIMEK, Jakub. *Tvorba počítačových her*. Hradec Králové, 2022. Tvořivá Klávesnice. První soukromé jazykové gymnázium. Vedoucí práce Mgr. Michaela Toman.

Zaměřením práce je ukázat, jak vytvořit počítačovou hru v Unity. Teoretická část popisuje výhody a nevýhody vybraných herních a vývojářských enginů a několika programovacích jazyků. V praktické části byla uskutečněna a popsána tvorba hry v Unity v *Low polygon* grafice v jazyce C#. Součástí popisu je, jak při tvorbě takové hry postupovat a jak užité skripty fungují. Cílem práce bylo vytvořit souhrn herních a vývojářských enginů a v jednom z nich vytvořit hru.

Klíčová slova: engine, hra, editor, skript, postup, programovací jazyk, Unity

Anotation

PODZIMEK, Jakub. *Tvorba počítačových her*. Hradec Králové, 2022. Tvořivá Klávesnice. První soukromé jazykové gymnázium. Vedoucí práce Mgr. Michaela Toman.

The focus of the work is to show how to create a computer game in Unity. The theoretical part describes the advantages and disadvantages of selected game and development engines and several programming languages. In the practical part, the creation of the game in Unity in *Low polygon* graphics in C # was realized and described. Part of the description is how to create such a game and how the used scripts work. The aim of the work was to create a summary of game and development engines and to create a game in one of them.

Key words: engine, hra, editor, skript, postup, programovací jazyk, Unity

Obsah

Úvod ... 9

Teoretická část práce ... 10

1 Herní Enginy ... 10

1.1 Unity ... 10

1.2 Unreal Engine ... 12

1.3 CryEngine ... 14

1.4 RAGE (Rockstar Advanced Game Engine) ... 15

1.5 Anvil ... 16

1.6 4A Engine ... 16

1.7 Source a Source 2 ... 17

2 Programovací a skriptovací jazyky ... 19

2.1 C ... 20

2.2 C++ ... 20

2.3 C# ... 21

2.4 Java ... 21

2.5 JavaScript ... 22

2.6 Lua ... 23

2.7 Python ... 24

2.8 Scratch ... 24

3 Vývojářské Enginy ... 26

3.1 Blender ... 26

3.2 Microsoft Visual Studio ... 27

Praktická část práce ... 29

4 Metodika ... 29

5 Tvorba hry v herním enginu Unity ... 29

5.1 Vytvoření a pojmenování souboru 29
5.2 Začátek samotné tvorby 30
5.3 Vytvoření modelu hráče a pohyb postavy 32
5.4 Pohled Hráče 34
5.5 Prostředí 35
5.6 Začáteční menu a úvod do hry 37
5.7 Pauza 43
5.8 Minihra – Bludiště 45
5.9 Minihra – Létání 47
5.10 Minihra – Hledání 51
Diskuse 53
Závěr 54

Seznam Obrázků

Obrázek 1 Hra v Unity [Autor] 11
Obrázek 2 Ukázka hry z Unreal Engine 5 [36] 14
Obrázek 3 Logo CryEngine [37] 14
Obrázek 4 Metro 2033: Last Light [38] 17
Obrázek 5 Scratch: Hello World! [Autor] 25
Obrázek 6 Ukázka animace v Blenderu [39] 27
Obrázek 7 Tvorba nového projektu v Unity [Autor] 30
Obrázek 8 Vytváření nových scén [Autor] 31
Obrázek 9 Vytváření terénu [Autor] 31
Obrázek 10 Výběr úpravy terénu [Autor] 35
Obrázek 11 Vytváření nové barvy terénu [Autor] 36
Obrázek 12 Farma ptačí pohled [Autor] 37
Obrázek 13 Vytváření Panelu [Autor] 37
Obrázek 14 Změna typu obrázku [Autor] 38
Obrázek 15 Vkládání obrázku na panel [Autor] 39
Obrázek 16 Vytváření tlačítka [Autor] 39

Obrázek 17 Text tlačítka [Autor] 40
Obrázek 18 Přidání funkce tlačítku [Autor] 42

Úvod

Tvorba počítačových her je v dnešní době samostatným, plně hodnotným, průmyslovým odvětvím, kterému se věnuje nemalé množství specialistů zaměřených na počítačové technologie. Je otázkou, zda by mohl člověk, který se nevěnuje ve své profesi tvorbě počítačových her, vytvořit kvalitní počítačovou hru, která by oslovila odbornou i neodbornou veřejnost. Na otázku se snažím odpovědět v této práci. Nikde jsem nenalezl porovnání jednotlivých herních a vývojářských enginů a programátorských jazyků pro zmíněnou neodbornou veřejnost a začínající tvůrce.

Teoretická část se věnuje rozdělení enginů, jejich specifikacím, výhodám a nevýhodám s ohledem na využití pro začínající tvůrce nebo již pokročilejší programátory. Pro doplnění znalostí jsou zmíněny informace o historii jednotlivých aplikačních rámců; uvedeny jsou i příklady vybraných her, které byly v těchto enginech vytvořeny za léta jejich existence, a jaké možnosti tyto softwarové frameworky nabízí.

V praktické části, je využit jeden z popsaných enginů s názvem Unity. V něm byla vytvořena jednoduchá hra, na které jsou demonstrovány základní techniky programování her.

Hra byla vytvořena v *Low-poly* grafice, která se vyznačuje vzhledem podobným animovaným filmům, jako je třeba Vaiana od společnosti Disney. Tento styl byl použit z důvodu jeho jednoduchosti renderování a manipulace pro nezkušené tvůrce. Do vytvořené hry bylo zařazeno několik jednoduchých úkolů, které musí pro dokončení hry hráč splnit, a na kterých byly ukázány i jiné techniky využívané při tvorbě her.

Cílem práce bylo vytvořit souhrn informací o jednotlivých herních a vývojářských enginech a programovacích jazycích, tvorba hry, na které byly popsány techniky a mechaniky používané při tvorbě her.

Teoretická část práce

1 Herní Enginy

Herní engine je softwarový framework (přeloženo jako aplikační rámec) primárně určený pro vývoj videoher. Zahrnuje příslušné knihovny a podpůrné programy. Herní enginy obvykle nabízí sadu nástrojů a funkcí pro vývoj her. [1]

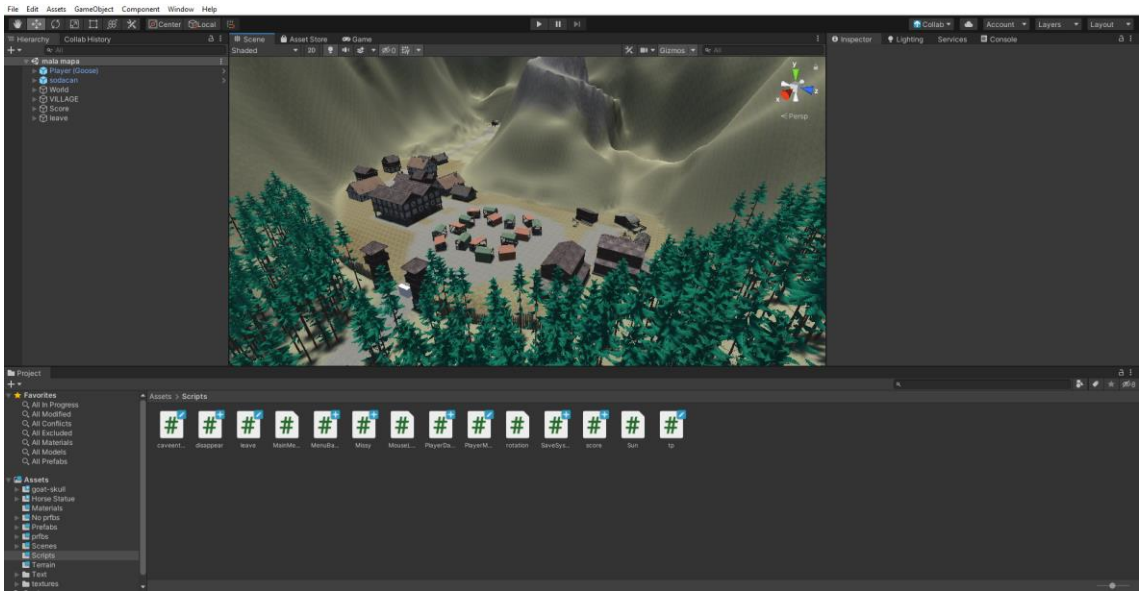
Tyto enginy jsou vývojáři her využívány k tvorbě her pro herní konzole a další typy počítačů. Základní funkce, které obvykle poskytuje herní engine, můžou zahrnovat vykreslovací modul pro 2D nebo 3D grafiku, fyzikální engine nebo detekci kolizí, zvuk, skriptování, animace, umělou inteligenci, vytváření sítí, streamování, správu paměti, vlákna, podporu lokalizace objektů, grafických scén a podporu tvorby videa. [1]

Nyní budou popsány jednotlivé herní enginy, historie jejich vývoje, jejich rozdíly, funkce, výhody a nevýhody. [1]

1.1 Unity

Unity je multiplatformní herní engine vyvinutý společností Unity Technologies, který byl poprvé oznámen a vydán v červnu 2005 na celosvětové konferenci Apple Inc. jako herní engine pro macOS X. Unity je od té doby postupně rozšiřováno, aby podporovalo různé platformy pro počítače, mobilní telefony, herní konzole a virtuální realitu. Je obzvláště populární pro vývoj mobilních her pro iOS a Android a byly v něm vytvořeny také hry jako například *Pokémon Go, Monument Valley, Call of Duty: Mobile, Beat Saber* a *Cuphead* a další. Je považováno za snadno použitelné pro začínající vývojáře a nezkušené tvůrce. Unity lze použít k tvorbě trojrozměrných (3D) a dvourozměrných (2D) her nebo i interaktivních simulací. Tento engine našel využití v mnoha průmyslech než jenom v herním. Začal být používán v průmyslech jako jsou například filmový a automobilový nebo v architektuře, strojírenství a stavebnictví. [1][2][3]

Unity bylo vydáno již ve zmíněném roce 2005 za účelem přiblížení tvorby her i malým, nezkušeným vývojářům mimo velká herní studia. V první verzi bylo možno vyvíjet hry pouze pro macOS, ale o rok později, tedy v roce 2006, byla ve verzi 1.1 přidána podpora pro Windows a webové prohlížeče. Další verze, 2.0, z roku 2007, přinesla novou podporu Windows, která zahrnovala i Direct X, což vedlo k asi třicetiprocentnímu zrychlení na všech systémech Windows. Mimo to přinesla tato aktualizace asi dalších 50 funkcí, mezi nimiž byl i optimalizovaný 3D engine s vylepšenými možnostmi pro tvorbu terénu, pokročilé nastavení světel, dynamické stíny a podporu pro tvorbu her pro více hráčů. Plná podpora vývoje her pomocí 2D Sprite přišla až v roce 2013 ve verzi 4.3. [1][2][3]



Obrázek 1 Hra v Unity [Autor]

V dnešní verzi Unity můžeme nalézt množství různých komponentů. Jedním z nich je vykreslovací engine, který zajišťuje *renderování*, neboli vykreslování reálně vypadajících objektů na základě dat vložených do Unity. Mezi taková data může patřit takzvaná *normal map*, česky normálová mapa, která je používána pro navození iluze osvětlení, konkávnost a konvexnost plochy v jinak 2D textuře. [1][2][3]

Fyzikální engine, slouží k simulaci fyzikálních zákonů, jako jsou třeba gravitace, působení sil na objekty, nebo dokonce i pohyb jako takový. Aby objekty mohly používat fyzikální engine, musí k sobě mít připojen komponent *rigidbody,* neboli tuhé těleso, díky kterému jsou všechny fyzikální zákony uplatňovány. Zároveň také musí objekty obsahovat *collidery,* takzvané hranice, které kontrolují kontakt objektu s jinými tělesy. Následná reakce na podnět je naprogramována uživatelem editoru. [1][2][3],

Další složkou je možnost skriptování, psaní programů. Pomocí skriptů jsou vytvářeny veškeré herní mechaniky. Díky skriptům je možné například přidávat, odebírat a měnit různé objekty scény, kontrolovat pravidla hry, uchovávat hodnoty, řídit nárazy nebo číst vstupy od uživatele jako jsou stisknuté klávesy, pohyb myší nebo použití jakéhokoliv z tlačítek myši. Unity pro své skriptování používá jazyk C# od společnosti Microsoft. [1][2][3]

Důležitou součástí Unity je zvukový engine, jehož základní funkcí je načíst, dekomprimovat a přehrát zvukovou stopu. Unity ale nabízí i pokročilejší funkce jako integrace více různých zvukových zdrojů pomocí vlastního zvukového mixeru. Mezi zajímavé funkce patří také produkce Dopplerova jevu, ozvěn nebo změna výšek tónů. [1][2][3]

1.2 Unreal Engine

Unreal engine je herní engine vytvořený firmou Epic Games. Jeho první verze, která vyšla v roce 1998, byla použita pro hru s názvem *Unreal*. Původně byl tento engine určen hlavně pro FPS hry, což je zkratka pro anglické *first-person shooter*, česky „střílečky z pohledu první osoby“; tímto typem hry je i *Unreal*. Hra v době vzniku překvapila hlavně svým grafickým zpracováním, ale také vylepšením již známých herních prvků, jako třeba zobrazování oblohy. Od doby vydání hry *Unreal* byl Unreal engine několikrát vylepšen a doplněn. Do dnešní doby byl Unreal engine využit k tvorbě desítek populárních herních titulů. Nejedná se jen o zmiňované FPS hry. Zahrnuje i typ RPG hry, neboli takzvané *role-playing game*, česky „hra na hrdiny“. RPG hra může být hrána jedním nebo více hráči. Hra pro více hráčů, je označována MMORPG, zkratka je odvozena z anglického názvu *massively multiplayer online role-playing game*. Unreal engine se uplatnil i v žánru her známých jako „adventura“, kde jedna nebo více hlavních postav postupuje dějem příběhu a během své cesty řeší rozličné úkoly. Příkladem her vytvořených v Unreal enginu jsou: *Batman: Arkham Asylum*, *Brothers in Arms – Hell's Highway*, *BioShock Infinite* a *TERA: The Exiled Realm of Arborea*. [4] [5] [6] [7] [8]

Jádro Unreal enginu je napsáno v jazyce C++, který bude popsán později. Ostatní části enginu jsou psány ve speciálním kódu s názvem UnrealScript. Proto není nezbytné při vytváření různých herních módů zasahovat hluboko do jádra, ale stačí pouze měnit tyto kódy. Engine podporuje mnoho uživatelských platforem, jako například Windows, Linux

mac OS a mac OS X na počítačích. Nejnovější verze podporují také platformy herních konzolí, jakými jsou třeba Dreamcast, Xbox, Xbox 360, PlayStation 2, 3 i 4. [4] [5] [6] [7] [8]

Unreal Engine také podporuje šest *renderovacích* technologií. Vysoké množství podporovaných *renderovacích* technologií bylo nutné v době vzniku, protože v 90. letech docházelo k masivnímu rozšíření používání grafických akcelerátorů, jejichž ovladače mezi sebou nebyly kompatibilní. Současná verze enginu podporuje barevné, až dvaatřiceti bitové, dynamické osvětlení, kde je možno barvy míchat, vrstvit nebo tvořit ostré i rozostřené stíny. Je možno v něm vytvořit více než 20 různých světelných efektů, mezi které patří paprskové, válcové, kuželové, světlometné nebo ambientní druhy osvětlení. V enginu je možné nasimulovat efekty záře nebo mlhy, umožňuje vytvořit zrcadlový efekt včetně několika násobného odrazu zrcadel umístěných naproti sobě. Lze v něm také nasimulovat odrážení světla od objektů, nebo takzvaný *warpový efekt*, neboli portálový efekt. Dále engine umožňuje zobrazovat zakulacený tvar povrchu těles bez nutnosti využití velkého počtu mnohoúhelníků, zobrazovat textury s funkcemi imitujícími vypouklý tvar jinak rovného povrchu, což bývá označováno jako *Emboss bump mapping*. [4] [5] [6] [7] [8]

Unreal engine zahrnuje také negrafické komponenty, například plně digitalizovaný zvukový systém s podporou formátů mp3, CD audia, a zvukových stop vytvořených pomocí hudebního programu *Tracker*. Použití *Trackeru* umožňuje nejen plynulý přechod mezi hudebními stopami, podle děje, který se ve hře právě odehrává, ale také značně zmenšuje velikost zvukových souborů a tím snižuje celkovou velikost hry. K dokreslení trojrozměrných efektů zvuku využívá tento engine při softwarovém *renderingu* stereo-fázový posun a Dopplerův jev. [4] [5] [6] [7] [8]

Obrázek 2 Ukázka hry z Unreal Engine 5 [36]

1.3 CryEngine

CryEngine byl vytvořen německým studiem Crytek. První hra vytvořená ve verzi 1 tohoto enginu se jmenuje *Far Cry*. Byla vytvořena jako technologické demo pro společnost NVIDIA zabývající se tvorbou grafických procesorů. Vzhledem k vysoké kvalitě demo verze následně byla vytvořena na základě tohoto konceptu celá hra. Vlastnická práva hry 30. března 2006 zakoupila společnost Ubisoft a tím také získala trvalou licenci k používání *Far Cry* edice CryEnginu. Po zavedení do provozu grafických karet podporujících *pixel* a *vertex shadery* verze 3.0, byla na trh firmou Crytek uvedena verze 1.2 tohoto enginu, která těchto funkcí využívala. Mimo to využívala také funkci *polybump*, která vykresluje textury plastičtěji. [9] [10] [11] [12] [13]

Obrázek 3 Logo CryEngine [37]

Verze CryEngine 2 byla použita pro vytvoření titulu *Crysis*. První licence zakoupené pro CryEngine 2 náleží francouzské společnosti IMAGTP, která se specializuje na

architektonické a územní plánování. Licence si tato společnost zakoupila, protože chtěla svým zákazníkům umožnit vidět vzhled budov v 3D projekci, měnit úhly pohledu na budovy a prozkoumat i vnitřní struktury před započetím stavebních prací. Dne 17. září 2007 si jako první vysokoškolská instituce pořídila licenci z výukových důvodů škola Ringling College of Art & Design v USA. [9] [10] [11] [12] [13]

V roce 2009 Crytek představil na *Game Developers Conference* následníka, kterým je CryEngine 3. Nový CryEngine 3 byl vyvíjen, aby podporoval platformy Windows, PlayStation 3, Xbox 360 a Wii U. Mimo to na PC podporoval DirectX9, 10 a 11. Nové technologie byly zařazeny i do vládních projektů a například roce 2011 australské ozbrojené síly zařadily virtuální tréning na vrtulníkové výsadkové lodi vyrobené právě v CryEngine 3 do výcviku armády. Od roku 2011 je možno si zdarma stáhnout CryEngine 3 SDK a vytvářet v něm nekomerční hry. Zároveň v tomto roce Crytek oznámil přesunutí původního *Crysis* na konzole a v neposlední řadě byl vydán na Xbox Live a PlayStation Network. [9] [10] [11] [12] [13]

Dnes je nejnovější verzí CryEngine V. Byla vydána v roce 2016, přináší podporu rozhraní DirectX12, možnost vývoje pro virtuální realitu a možnost skriptování v jazyce C#. Verze CryEngine V funguje na principu „Pay what you want“, kdy může kdokoliv podpořit vývojáře enginu libovolnou finanční částkou, ale platba není podmínkou pro využití enginu. [9] [10] [11] [12] [13]

1.4 RAGE (Rockstar Advanced Game Engine)

Zkratka RAGE, která je názvem tohoto enginu, pochází z anglického Rockstar Advanced Game Engine. Tento engine byl a stále je neveřejný neboli *closed-source engine*. Vlastník, v tomto případě společnost Rockstar Games, požaduje zakoupení licence, aby ji člověk mohl využívat, upravovat a sdílet obsah nebo rozšíření tohoto enginu. RAGE byl vytvořen odvětvím RAGE Technology Group americké společnosti Rockstar Games. [14] [15]

První hra vytvořená v tomto enginu se jmenuje *Rockstar Games Presents Table Tennis*, která byla vydána v roce 2006. Tato hra je realistická simulace stolního tenisu, kde se hráč snaží, aby protivník minul míček. Tento simulátor byl vydán pouze na konzole Xbox 360 a Nintendo Wii. RAGE byl od té doby použit na tvorbu vyspělých *open world* her, neboli her „s otevřenými světy“, kde se může hráč volně pohybovat. Příkladem

takovýchto her jsou série GTA, neboli *Grand Theft Auto*, nebo *Red Dead Redemption*. [14] [15]

1.5 Anvil

Anvil je engine vytvořený kanadskou společností Ubisoft Montreal. Do roku 2009 byl tento engine znám pod jménem Scimitar. Lze zde vytvářet hry pro platformy jako jsou Microsoft Windows, Stadia, PlayStation 3, 4, 5 a Vita, Wii U, Nintendo Switch, a Xbox 360, One, Series X a Series S. [16] [17]

K modelování prostředí využívá Anvil software *3ds Max* a pro postavy *ZBrush*. Aby animace Anvilu vypadaly co nejrealističtěji, je zde využíván také software *HumanIK* od společnosti *Autodesk*. Při tvorbě hry *Assassin's Creed II* byl Anvil vylepšen a byly do něj přidány funkce jako denní cyklus, „vylepšené" osvětlení, které využívá nových světelných efektů a odrazů. Byl zde vylepšen navigační systém UI, to znamená umělé inteligence, a *NPC*, neboli nehratelných postav. [16] [17]

Anvil byl také použit při tvorbě krátkých filmů například *Assassin's Creed: Lineage* jehož tvůrci jsou společnosti Hybride Technologies a Ubisoft Digital Arts. V Anvilu bylo vytvořeno prostředí, a poté do něj vloženi předem natočení živí herci. [16] [17]

Po mnoha vylepšeních můžeme v dnešní podobě Anvilu najít funkce jako globální osvětlení, odrazy světel, mlhu nebo měnící se počasí. Byla přidána funkce PBR, což je zkratka pro *Physically Based Rendering*, která dovoluje objektům a jejich povrchům reagovat na světlo, šerosvit i stíny reálněji než bez ní. [16] [17]

1.6 4A Engine

4A Engine je grafický middleware engine, což znamená software, který poskytuje aplikacím služby, které operační systém nemá. Například usnadňuje vývojářům vývoj komunikace vstupů a výstupů. Základy tohoto enginu byly vytvořeny v ukrajinské společnosti GSC Game World. Programátoři firmy se snažili o zlepšení funkce enginu a nápravu chyb. Engine nesynchronizoval řádky kódu tak, aby se daly příkazy dělat po sobě v požadovaném pořadí. Proto poté, když spustili zkoušku hry na Xbox 360, naměřili asi 3000 různých operací běžících najednou během 30 milisekund, což zabralo 100 % kapacity procesoru a dalších komponentů hardwaru, a bylo velmi obtížné v něm kódovat

při tvorbě hry *S.T.A.L.K.E.R.: Тінь Чорнобиля*, česky stín Černobylu, ale vlastníci společnosti nic měnit nechtěli. Z některých členů této společnosti následně vznikla společnost nová, 4A Games, a to v roce 2006, která pracovala na dalším vývoji enginu. Nejvíce se na tvorbě tohoto enginu podíleli programátoři Oleksandr Shyshkovtsov a Александр Максимчук, Alexandr Maximčuk v přepisu. Dokončili upravenou hru *S.T.A.L.K.E.R.: Тінь Чорнобиля*, a následně byly v enginu vytvořeny hry ze série *Metro 2033*, zpracované podle knižní předlohy. Současnou verzi enginu lze spustit na PC, Xbox 360 nebo PlayStation 3, hry se dají hrát i na jiných platformách. [18] [19]

Obrázek 4 Metro 2033: Last Light [38]

1.7 Source a Source 2

Oba tyto enginy byly vytvořeny společností Valve. V Source byla vytvořena hra *Half Life: Source* v roce 2004. Jako jeden z mála nemá Source žádné „meziverze“, je průběžně aktualizován. Mezi významnější updaty patří ten z roku 2006, kdy při vydání *Half-Life 2: Episode One* kdy byly vylepšeny HDR *rendering* a technologie světla a stínů. V roce 2007 proběhl update, kdy bylo v Source vylepšeno kódování a byla přidána funkce rozdělení obrazovky, když hraje na jednom zařízení více hráčů, což funguje díky multiprocesorové podpoře. Byl vydán u balíčku her nazvaným *The Orange Box*. [20] [21]

V roce 2015 byl pak Source nahrazen enginem Source 2 a do něj byly všechny hry ze Source předány tak, aby bylo vše kompatibilní. První hra vytvořená v tomto novém enginu byla *Dota 2* náležící do žánru MOBA, anglicky *multiplayer online battle arena*, neboli bojová aréna pro více hráčů. Tento engine byl oznámen již na *Game Developers Conference* v roce 2014. Majitel společnosti Valve poté upřednostnil vývoj vlastních her před vydáním nekompletního enginu pro veřejnost, a proto byl vydán až v roce 2015. Výhodou tohoto enginu je možnost tvorby her pro platformy iOS, Android, a virtuální realitu. V enginu byly například vytvořeny hry *Artifact*, nebo *Half-Life: Alyx* nebo již zmiňovaná *Dota 2*. [20] [21]

2 Programovací a skriptovací jazyky

Programovací jazyky slouží pro zápis algoritmů na počítačovém zařízení a vytváří strojový kód, díky kterému mohou počítače provádět zadané operace. Programovacím jazykem programátor řeší jednotlivé problémy a podle sepsaného kódu poté po řádcích počítač postupuje. Je to soubor pravidel pro zápis algoritmů. Zápisu algoritmu ve specifickém programovacím jazyce se říká program. Každý programovací jazyk má určitá pravidla, která se musí dodržovat při psaní algoritmů, aby mohl program správně fungovat. Programovacích jazyků je přes 2 tisíce, ale pouze několik málo z nich je široce používáno. Programovací jazyky se dělí podle míry abstrakce na vyšší programovací jazyky a nižší programovací jazyky. Krom tohoto specifika se jazyky ještě dělí podle jejich orientace a vzorců, které jazyk podporuje, a to například na objektově orientované, strukturované, funkční nebo deklarativní. [22] [23] [24] [25] [26]

Jazykům, pracujícím v algoritmech s objekty, se říká jazyky objektově orientované. V objektovém programovaní je vysoce výkonný kód přiřazen k datům. Tento typ je jeden z nejvíce srozumitelných programátorských jazyků. Mezi nejznámější patří Python, C#, C++ a Java. [22] [23] [24] [25] [26]

Strukturované jazyky byly vyvinuty za účelem dosažení vyšší srozumitelnosti při jejich využívání, vyšší kvality a kratší doby zpracování kódů počítačem, což je zaručeno skládáním programu z menších struktur. [22] [23] [24] [25] [26]

Do funkčních jazyků se řadí takové jazyky, které řeší a vyjadřují úkoly pomocí matematických rovnic a funkcí. Patří mezi ně F#, Wolfram Language nebo Scheme. [22] [23] [24] [25] [26]

Deklarativní jazyky se nezabývají řešením úkolů, ale spíše je strukturují podle jejich typu. Tyto jazyky nejsou moc známé a nejsou univerzální. Mezi tyto jazyky se řadí například HTML nebo SQL. [22] [23] [24] [25] [26]

U každého programovacího jazyka popsaného níže v práci budou uváděny nejjednodušší možné cesty zápisu, jak vypsat do konzole frázi „Hello World!“, podle které se dá poznat typ a úroveň vývoje jazyka. [22] [23] [24] [25] [26]

2.1 C

Jazyk C je vyšší programovací jazyk. Zápis lze provést ve formě vyššího typu jazyka, který je uživatelsky více intuitivní pro programátora, ale je možné zápis provést i ve formě nižšího typu jazyka, kdy kód je méně srozumitelný a přehledný pro tvůrce, ale výsledný program je efektivnější a rychlejší. [22]

C je imperativní jazyk, řeší jednotlivé výpočty podle určité posloupnosti a určuje přesný postup, jak úlohu řešit. Je to, ale i jazyk strukturovaný, což znamená, že je vyvinut tak, aby byl co nejvíce srozumitelný pro uživatele, ale zároveň co nejrychleji zpracovatelný do implementovaných algoritmů. Tento jazyk také podporuje rekurzi, což je technika, při níž je určitá funkce nebo akce znovu volána dříve, než je ukončeno její minulé volání. [22]

Jazyk C je používán k programování aplikačních softwarů a operačních systémů na superpočítačích, vestavěných systémech, to znamená stolních počítačích, nebo v PLC, což pochází z anglického *programmable logic controller*, což je laicky řečeno „jednoduchý počítač různých automatů nebo továrních strojů“. [22]

```
#include <stdio.h>
int main (void)
{
    printf("Hello World!\n");
    return 0;
}
```

2.2 C++

C++ je jedním z mnoha multiplatformních, univerzálních programovacích jazyků, který je již využíván méně, ale stále je v malém objemu používán k vývoji vysoce výkonných aplikací. Byl vyvinut Dánem Bjarnem Stroustrupem v roce 1985 jako rozšíření jazyka C. C++ poskytuje programátorům vysokou kontrolu nad systémovými prostředky a pamětí. Tento jazyk byl za dobu své existence aktualizován pouze třikrát, a to v letech 2011, 2014 a 2017. [23]

```
#include <iostream>
int main (int argc, char *argv []) {
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

2.3 C#

C#, čteno stejně jako v anglickém hudebním zápisu, tedy *C Sharp*, česky „sí šárp“ je vysokoúrovňový objektově orientovaný programovací jazyk od firmy Microsoft vytvořený v roce 2000. Tento jazyk vychází z programovacích jazyků C++, C a Java. C# lze použít pro tvorbu webových stránek, webových aplikací, webových služeb, databázových programů, softwaru pro mobilní telefony a počítačových her. Byl vytvořen tak, aby byl co nejjednodušší, mnohoúčelový, a aby co nejvíce zohledňoval strukturu CLI (*Common Language Infrastructure*). CLI popisuje vlastnosti proveditelného kódu a prostředí, ve kterém program probíhá. [24]

Programy tohoto jazyka fungují přes platformu .Net Framework, což je platforma od společnosti Microsoft, která umožňuje spouštět a vyvíjet aplikace a webové služby Windows. Mnoho jiných programovacích jazyků také používá .Net Framework. Platforma poskytuje množství služeb užitečných pro spuštěné aplikace, jako například správu paměti nebo rozsáhlou knihovnu tříd. [24]

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello, world!");
    }
}
```

2.4 Java

Java je jeden z objektově orientovaných programovacích jazyků. Byl vytvořen firmou Sun Microsystems v roce 1995. Jedná se o jeden z nejpoužívanějších programovacích jazyků na světě podle TIOBE indexu. TIOBE index vyjadřuje popularitu programovacích jazyků zjištěnou na základě analýzy výsledků dotazů na různých vyhledávačích, jako

například Google, Yahoo!, Baidu, Wikipedia nebo YouTube. V roce 2020 však Javu předběhly v TIOBE indexu jazyky C a Python. [25] [26]

Tento jazyk se stal populární díky své přenositelnosti, lze ho používat pro programy pracující na základě čipových karet, na mobilních telefonech, v aplikacích pro desktopové počítače, ale dokonce i v distribuovaných systémech pracujících na řadě propojených počítačů rozprostřených po celém světě. Dne 8. května 2007 Sun Microsystems uvolnil zdrojové kódy Javy, což je asi 2,5 miliónů řádků kódu. Od té doby je Java dále vyvíjena jako *open source*, kdy odborná veřejnost z celého světa se může podílet na dalším vývoji Javy. [25] [26]

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

2.5 JavaScript

JavaScript je multiplatformní, objektově orientovaný skriptovací jazyk. Byl vytvořen americkým programátorem jménem Brendan Eich pracujícím pod firmou Netscape. Zápis zdrojového kódu jazyka má mnoho společných znaků s jazyky C, C++ a Java, ale zároveň se od nich zásadně sémanticky liší. Slovo Java je součástí jména JavaScript pouze z marketingových důvodů. Původně byl ve společnosti Netscape vyvíjen pod názvem Mocha a později LiveScript. Managment společnosti Netscape se rozhodl, že přidání slova Java do názvu programovacího jazyka zvýší prodejnost vzhledem k popularitě Javy od společnosti Sun Microsystems. [27] [28]

JavaScript byl ohlášen v prosinci roku 1995 společně s jazykem Java. JavaScript je doplňkem k jazyku Java a k jazykům HTML. [27] [28]

JavaScript byl v roce 1997 standardizován asociací EMCA, *European Computer Manufacturers Association*, a v roce 1998 asociací ISO, *International Organization for Standardization*. [27] [28]

Dnes bývá JavaScript často využíván k tvorbě webových stánek, bývá součástí předinstalovaných balíčků v operačních systémech Window, ale také se využívá jako DHTML a k tvorbě nebo rozšíření mnohých aplikací jako například Adobe Acrobat. [27] [28]

```
<!DOCTYPE HTML>
<html>

<body>

  <script>
    alert( 'Hello, world!' );
  </script>

  </body>

</html>
```

2.6 Lua

Jazyk Lua je odlehčený, reflexivní, imperativní a procedurální. Byl navržen jako skriptovací jazyk s rozšiřitelnou sémantikou. Jeho název je odvozen z portugalského slova pro měsíc. Lua vytvořili v roce 1993 Roberto Lerusalimshy, Luiz Henrique de Figueiredo a Waldemar Celes, což byli členové společnosti Computer Graphics Technology Group. [29]

Lua je svým designem srovnatelný s jazykem Icon a svou jednoduchostí podobný programu Python, a proto pracovně velmi intuitivní i pro člověka nezkušeného v programování. Lua je dostatečně objemově malý jazyk, aby mohl být zařazen na nejrůznější hostitelské platformy a významně nezatížil jejich úložiště, ale nevýhodou je podpora jen malého množství atomárních datových struktur jako jsou například *boolovské* hodnoty, čísla a řetězce. Pro ostatní operace využívá Lua nativní datové tabulky. Použitím minimálního počtu datových typů se snaží Lua dosáhnout rovnováhy mezi silou a velikostí. [29]

```
print("Hello world!")
```

2.7 Python

Python se řadí mezi vysokoúrovňové programovací jazyky. Nabízí dynamickou kontrolu datových typů a mnoho různých programovacích paradigmat, například objektově orientovaných, procedurálních, funkcionálních nebo imperativních. Vytvořil ho v roce 1991 nizozemec Guido van Rossum. [30]

V roce 2018 se stal velice populárním a je nyní zařazen mezi nejoblíbenější programovací jazyky na světě; v žebříčcích popularity obsazuje většinou jednoho z prvních tří míst. Python byl a je vyvíjen jako open source, který je možno zdarma stáhnout na většinu běžných platforem, jako například Unix, Windows, macOS nebo Android, a ve většině distribuovaných systémů GNU a Linux bývá Python dokonce součástí základní instalace. [30]

```
print("Hello world!")
```

2.8 Scratch

Zcela atypickým mezi programovacími jazyky je Scratch. Jedná se o vizuální programovací jazyk, pracuje se s ním pomocí manipulace s grafickými programovými elementy, a nepíší se žádné skripty. Ve Scratchi se pracuje s objekty v angličtině označovanými jako „sprites“, což v češtině znamená postavy. Scratch má vlastní jednoduchý editor, ve kterém lze tyto postavy nakreslit jak ve vektorové, tak v bitmapové grafice. Další možností může být vložení obrázku z vnějšího webového zdroje nebo z webové kamery. [31] [32]

V dnešní době se Scratch používá přes oficiální internetové stránky, ale dříve tomu bylo jinak. První verze vyšla v roce 2005 a jednalo se o desktopovou aplikaci, ke které byly v roce 2007 vytvořeny webové stránky, aby mohli uživatelé ukládat a sdílet svoje výtvory s ostatními tvůrci. V roce 2013 byla pak vydána verze 2.0. Od té doby není nutno si Scratch nainstalovat, ale je možno pracovat pouze na internetových stránkách. Nejnovější verzí je Scratch 3.0 vydaný v květnu roku 2019. [31] [32]

Obrázek 5 Scratch: Hello World! [Autor]

3 Vývojářské Enginy

3.1 Blender

Blender je volně přístupný a využitelný software pro modelování a vykreslování trojrozměrné počítačové grafiky a animací. Bývá využíván i ke komerčním účelům. Tento engine využívá techniky sledování paprsků, globálního osvětlení scény nebo *scanline rendering*. Grafické rozhraní Blenderu je vykreslováno pomocí knihovny OpenGL. OpenGL zajišťuje zrychlení vykreslování dvourozměrných a trojrozměrných objektů. Umožňuje snadné přenášení souborů mezi všemi podporovanými platformami. Mezi ně patří například Microsoft Windows, mac OS X, FreeBSD, nebo Linux. [33] [34]

Jádro tohoto enginu je psáno v jazycích C, C++ a Python. Seznámit se s prací v tomto enginu je možno z různých zdrojů. Existují edukační volně přístupná videa, odborná literatura a je vydáván časopis BlenderArt Magazine, který je zdarma přístupný na internetu. BlenderArt Magazine v každém čísle popisuje a rozebírá jinou oblast 3D vývoje. [33] [34]

V minulosti byl Blender využíván také jako herní engine, ve kterém se daly vytvářet jednoduché interaktivní hry. Tvorba v tomto enginu byla jednoduchá a intuitivní; mohli ji využívat i lidé s nevelkými programátorskými zkušenostmi. Zkušenější uživatelé mohli využít možnosti skriptování pomocí jazyka Python a tvořit i složitější hry. Nyní je funkce herního enginu již odstraněna, z důvodu zastaralosti ve srovnání s jinými open source herními enginy. [33] [34]

Obrázek 6 Ukázka animace v Blenderu [39]

3.2 Microsoft Visual Studio

Microsoft Visual Studio je engine vyvinutý společností Microsoft. Jedná se o programátorské prostředí vyvinuté za účelem zrychlení vývoje počítačových programů, webových stránek a aplikací. [35]

Ve Visual Studiu je možné programovat jak v strojovém kódu, tak v spravovaném kódu. Jádro tohoto enginu je napsáno v C++, ale další části jsou sepsány v C#. V základu Visual Studio operuje na platformách Microsoft Windows a .Net Framework. Je možné v něm psát šestatřiceti různými programovacími jazyky, mezi které patří například C, C++, C# nebo Basic .NET. Sám o sobě však nepodporuje žádný programovací jazyk – je třeba do něj vždy přidat rozšíření s vybraným jazykem. [35]

První verze Visual studia byla vydána v roce 1997. Vznikla spojením mnoha programátorských nástrojů Microsoftem. Jednalo se o verzi Visual Studio 97, obsahovala Visual Basic 5.0 pro programování ve Windows a Visual J++ 1.1 pro programování v Javě. [35]

Následovala verze Visual Studio 6.0, která vyšla v roce 1998, která byla podporovaná do roku 2002 pro Windows 9x, protože se Microsoft rozhodl v letech 1998 až 2002 soustředit na vývoj platformy .Net Framework. Ve verzi Visual Studio 6.0 byla do Visual

Studia Microsoftem přidána možnost stažení rozšíření s programátorským jazykem C#. [35]

V roce 2002 vydal Microsoft verzi Visual Studio .NET. Nejvýraznějším vylepšením bylo přidání vývojového prostředí pro spravovaný kód, což je kód, který je nejdříve napsán, poté zkomprimován do strojového kódu a až pak je teprve spuštěn. Po verzi Visual Studio .NET přišlo do dnešní doby ještě devět dalších verzí. Nejnovější verzí je Visual Studio 5.12. [35]

Jeden z nejvíce využívaných i oceňovaných prvků běžnými uživateli Microsoft Visual Studia je editor kódu. Ten poskytuje zvýraznění chyb v syntaxi, neboli chyb ve struktuře kódu, což uživateli usnadňuje čtení. Engine má nastavenou funkci automatického dokončování kódu, k čemuž používá IntelliSense. Jedná se o soubor funkcí k dokončování kódů pomocí nejpravděpodobnějších zakončení. Při zadání části příkazu je zobrazena nabídka možných členů skriptu nebo informace o parametrech. Další funkcí Visual Studia je takzvaný *debugger*, který pracuje se spravovaným i strojovým kódem a dokáže kódy mezi sebou vzájemně převádět. [35]

Praktická část práce

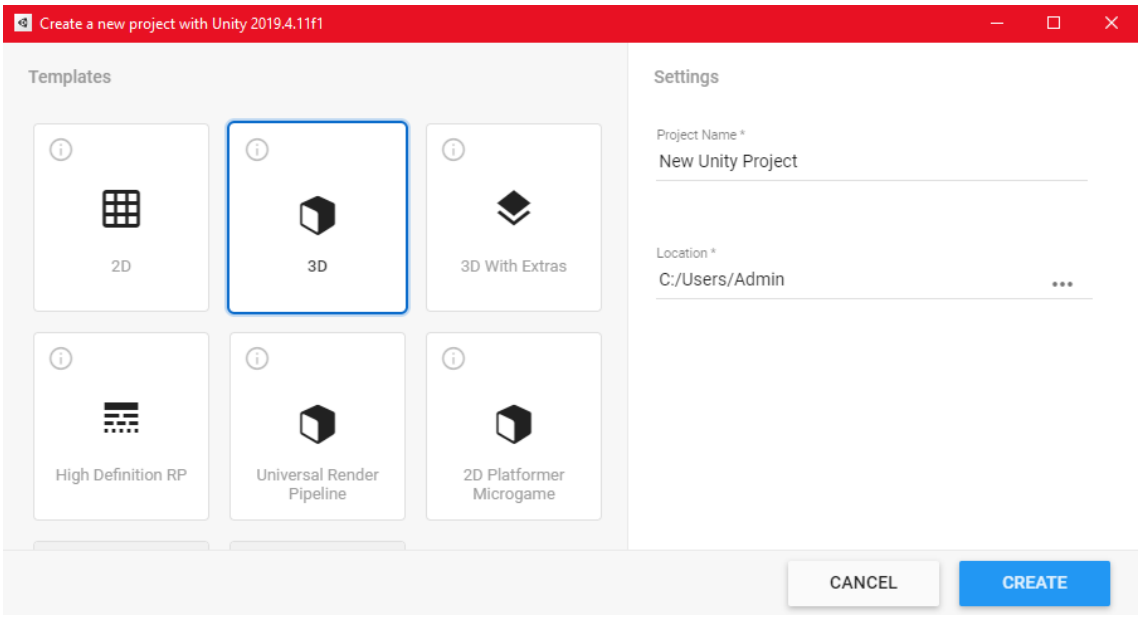
4 Metodika

V této části své práce budu podrobně popisovat postup tvorby hry, kterou jsem vytvořil v Unity v 64-bitové verzi 2019.4.22f1. Mimo to budu také popisovat, co jaká část skriptu hry umožňuje a vysvětlovat, proč tomu tak je. Protože je však v oboru tvorby her nespočet možností, a protože překypuje rozmanitostí, budu popisovat pouze ty funkce, které jsem použil při tvorbě své hry. Ta byla vytvořena v *Low-poly* grafice, což znamená, že neobsahuje mnoho jednotlivých vykreslovaných detailů, a zaměřuje se spíše na vizuální stránku her než na tu pohybovou, kterou ale samozřejmě nejde vynechat úplně, a proto o ní budu mluvit i zde. Celý program Unity používám v anglickém jazyce, tudíž budu v praktické části práce popisovat jednotlivé kroky částečně právě v angličtině. Kromě toho jsem na psaní skriptů používal vývojářský engine Visual studio. V něm jsem psal v programovacím jazyce C#, který je odvozený právě také z angličtiny. Jeho části a úryvky skriptů v něm psané se také v práci objevují.

5 Tvorba hry v herním enginu Unity

5.1 Vytvoření a pojmenování souboru

Prvním krokem u každé hry je vytvořit si nějakou složku a pojmenovat ji, aby se dalo v projektu později dobře vyznat. Unity naštěstí usnadňuje spoustu takovýchto technických záležitostí, takže si samo najde v počítači složky, kam musí být soubory uloženy, aby šlo vše správně spustit. Stačí tedy pouze aplikaci Unity spustit, vytvořit nový projekt a zadat několik informací do předpřipravených kolonek. Mezi takové informace patří například – vybrat, jestli má být hra ve 2D nebo 3D grafice, její jméno. Je zde také možno změnit místo uložení, ale to jak již bylo zmíněno, není nutné, protože předem navržené umístění je naprosto bezproblémové. Okno, ve kterém se tyto změny provádí může vypadat například jako na obrázku číslo 7.

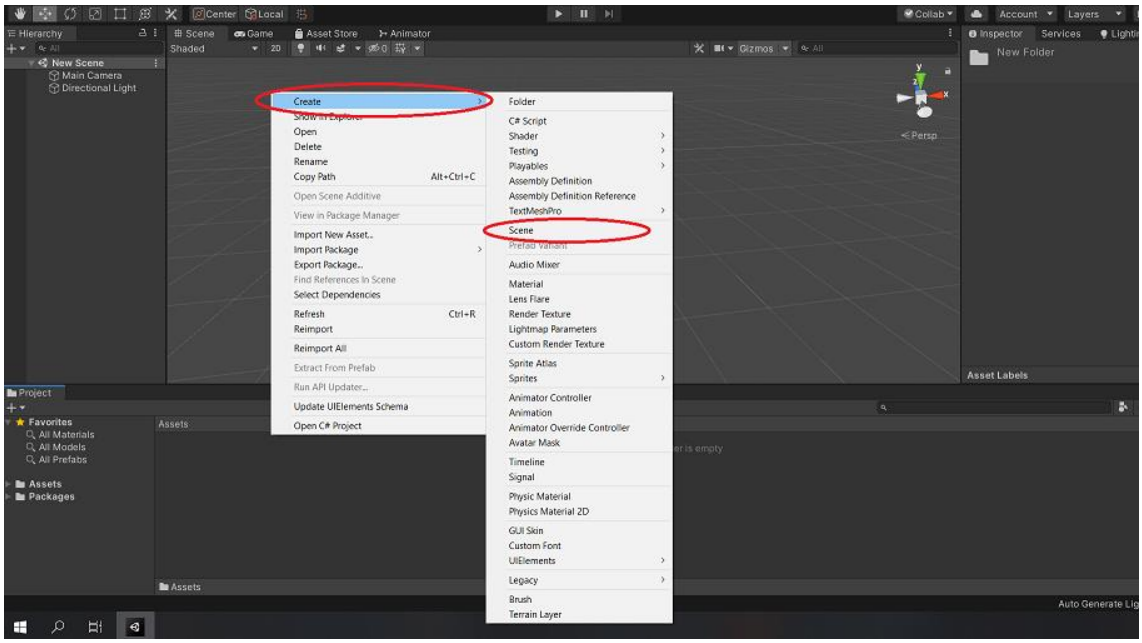


Obrázek 7 Tvorba nového projektu v Unity [Autor]

5.2 Začátek samotné tvorby

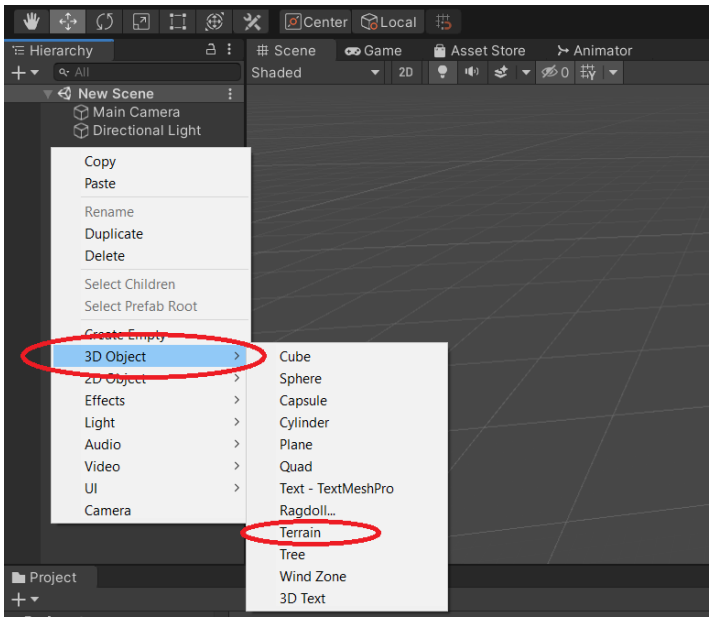
Po otevření projektu se musí uživatel rozhodnout, jak chce, aby jeho hra nakonec vypadala – nejen vzhledově, ale i příběhově a částečně i funkčně. V případě hry, která je zde popisována, jsem si vybral 3D *Low-poly* grafiku s jednoduchým příběhem a širokou škálou funkcí, abych mohl lépe ukázat, co vše lze v Unity vytvořit.

Mezi první kroky samotné tvorby tedy patří vytvoření scény. První scénu vytvoří samotné Unity, a proto není třeba se o ni starat v tuto chvíli, ale později v práci jsem však scény přidával. Pro přidání scény je třeba kliknout pravým tlačítkem myši do spodní části editoru, kde, po rozvinutí záložky, tvůrce přejede na *Create*, a pak na *Scene*. (Ukázáno na obrázku 8.) Každá nová scéna se jmenuje *New Scene*, dokud není přejmenována. Pojmenovávat si jednotlivé scény v projektu je velice praktické a sám jsem tak v celém projektu dělal, protože je poté znatelně snazší se při práci orientovat.



Obrázek 8 Vytváření nových scén [Autor]

Dalším velice důležitým krokem je vytvořit si terén, po kterém se postava hráče bude pohybovat. Terén je možno jednoduše vložit tím, že je kliknuto pravým tlačítkem myši do levé části obrazovky. Ve sloupci se jménem scény (na obrázku 9 "New Scene"), je nutno najet kurzorem na kolonku *3D Object* a po rozvinutí další záložky kliknout na *Terrain*.



Obrázek 9 Vytváření terénu [Autor]

5.3 Vytvoření modelu hráče a pohyb postavy

Při vytváření postavy je potřeba nejdříve vytvořit model s texturami. Ve svém případě jsem tak učinil v aplikaci Blender. Tento model jsem poté vložil do Unity tak, že jsem si v souborech našel, kde mám projekt uložený. Na tomto místě jsem si pak našel složku *Assets* a do ní jsem přetáhl svůj hotový model. Když už má tvůrce ale model připravený, tak ho musí nějak rozpohybovat. Právě to jsem udělal pomocí svého prvního skriptu, který jsem si pojmenoval *Player Movement*. (Viz následující skript.) Po dokončení kódu je třeba v Unity připojit k jednotlivým proměnným příslušná čísla, nebo objekty tak, aby každá proměnná pracovala se správným druhem informací.

```
public class PlayerMovement : MonoBehaviour
{
    public CharacterController controller;

    public float speed = 12f;
    public float gravity = -9.81f;
    public float jumpHeight = 3f;
    public Transform groundCheck;
    public float groundDistance = 0.4f;
    public LayerMask groundMask;

    Vector3 velocity;
    bool isGrounded;

        void Update()
    {
        isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance,
groundMask);

        if(isGrounded && velocity.y < 0)
        {
            velocity.y = -2f;
        }

        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");

        Vector3 move = transform.right * x + transform.forward * z;

        controller.Move(move * speed * Time.deltaTime);

        if(Input.GetButtonDown("Jump") && isGrounded)
        {
            velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravity);
        }

        velocity.y += gravity * Time.deltaTime;

        controller.Move(velocity * Time.deltaTime);
    }
}
```

Nyní zde bude vysvětleno co, která funkce dělá nebo znamená. Bude postupováno od horní části skriptu dolů – jinými slovy, bude popsáno vše zásadní od patnáctého řádku. Řádky nad tím jsou pouze proměnné, kterým byly přiřazeny zmiňované hodnoty, ale bude na ně odkazováno, když bude třeba je zmínit z důvodu přítomnosti v určitých funkcích.

Jako první funkce bude vysvětlena funkce *void Update()*, pod kterou je zbytek skriptu. Tato funkce spouští vše, co je pod ní, při každém snímku – to znamená tolikrát za sekundu, na kolik FPS (*frames per second*, neboli snímků za sekundu) je počítač hru schopný spustit.

Dále následuje funkce *Physics.CheckSphere*, která kontroluje, zda se hráč dotýká země, aby, jak je později napsáno v kódu, mohl skákat. Tu, včetně jejích parametrů, jsem pro jednoduchost používání pojmenoval *isGraounded*. Tato funkce vytváří imaginární kouli, ke které je potřeba připojit tři proměnné, respektive parametry. První proměnnou je přiřazení této koule k bodu, respektive objektu s proměnlivými souřadnicemi, kterého se mé tato koule držet, druhou je, jak velká má tato koule být být, a nakonec, co má toto funkce považovat za "zem". Velikost koule má ve skriptu jistou číselnou hodnotu, ale ostatní dvě proměnné jsem musel později sám přiřadit v editoru – ne v kódu.

To úzce souvisí s následující podmínkou. Ta říká, že jestli se zmíněná imaginární koule nedotýká ničeho označeného jako "zem", tak se postavě zvyšuje rychlost při tomto pádu.

Poté přichází řádky 25 až 30, které se zaobírají pohybem hráče v rovině os x a z. Řádky 25 a 26 kontrolují pohyb po těchto osách, ale pro usnadnění tvůrcům her v sobě má Unity tyto osy napojené na klávesy W a S pro pohyb dopředu a dozadu, A a D pro pohyb doleva a doprava. Jsou to tedy proměnné, které obě získávají hodnotu -1 až 1 podle zmáčknuté klávesy a ty potom řádek 28 převádí na vektor směru pohybu postavy, který je nakonec násoben zadanou rychlostí pohybu a zároveň započítáme čas pomocí *Time.deltaTime*, což je hojně používaný způsob, jak zajistit, aby výkon zařízení, tedy FPS ve hře neměly vliv na rychlost pohybu postavy.

Předposlední částí tohoto skriptu je podmínka, která ovládá skákání. Ta se nejprve zeptá, jestli platí obě její podmínky, a teprve až pak se případně spustí. Těmito podmínkami jsou, zda je stlačena klávesa určená pro funkci skákání, v mém případě mezerník, a jestli platí, již dříve zmiňovaná, funkce *isGrounded*. Pokud jsou obě splněny, postava vyskočí.

Poslední dva řádky tohoto skriptu se zabývají pouze změnou výsledných hodnot procesem *Time*.deltaTime, aby, jak již bylo řečeno, neměl výkon počítače dopad na rychlosti ve hře.

5.4 Pohled Hráče

Kromě toho, aby se mohl hráč hýbat, by měl mít možnost se otáčet a dívat kolem sebe. K pohledu jako takovému slouží v Unity kamera s názvem *Main Camera*. Ta se ale nijak nehýbe a je pořád na jednom místě, proto je třeba napsat skript, ve kterém je připojena k hráči a napojena na pohyb myši. Tento skript vypadá následovně.

```
public class MouseLook : MonoBehaviour
{
    public float mouseSensitivity = 100f;

    public Transform playerBody;

    float xRotation = 0f;
    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    void Update()
    {
        }
        float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity;
        float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity;

        xRotation -= mouseY;
        xRotation = Mathf.Clamp(xRotation, -80f, 65f);

        transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
        playerBody.Rotate(Vector3.up * mouseX);
    }
}
```

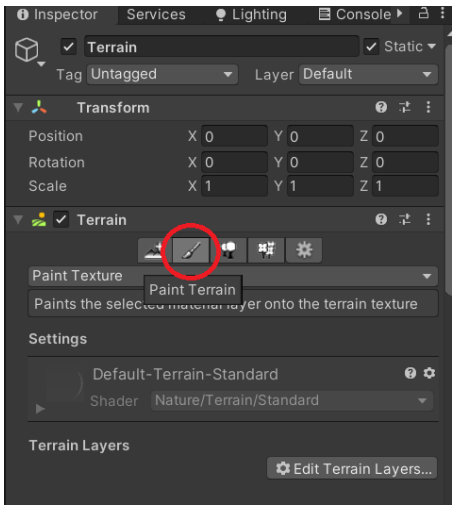
U tohoto skriptu, oproti minulému, přibyla funkce *void Start()*, která spouští akce pod ní ještě před prvním snímkem. V mém případě dělá to, že mi zamkne kurzor myši uprostřed obrazovky a zneviditelní jej.

Poté následuje již jen *void Update()*, který byl zmíněn u pohybu hráče. V tomto skriptu se pod ní schovávají úkony řídící rotaci již zmiňované kamery i modelu samotného hráče. První dva řádky sledují pohyb myši a násobí ho konstantou, kterou jsem pojmenoval *mouseSensitivity*.

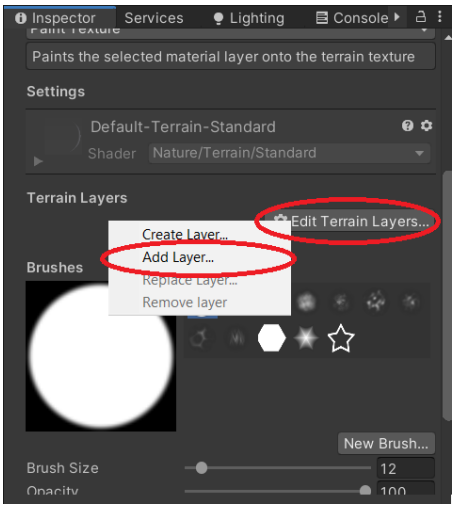
Následují další dva řádky, které řídí rotaci kamery po ose y, tedy nahoru a dolu. První řádek přebírá informace z řádků minulých a ten druhý omezuje pohled jen na určité úhly. Unčinil jsem tak, abych napodobil realistické pohyby hlavy, jinými slovy, aby se nedala hlava protočit "kolem dokola" o 360°. Předposlední řádek určuje to, jak je kamera natočená oproti osám x a z a fixuje ji ve správné poloze. Poslední řádek otáčí celým modelem hráče doprava a doleva podle pohybu myši. Kamera se v tomto případě otáčí s modelem a je v něm zafixovaná, takže není třeba řešit její samostatný pohyb.

5.5 Prostředí

V tuto chvíli má tvůrce při shodném postupu postavu, která se může hýbat a dívat se kolem sebe, ale pořád je na ploše sama. Na začátku přidaný terén se dá v Unity upravovat mnoha způsoby, ale to jsem ve svém projektu nedělal, a proto bude zmínka o této možnosti pouze velice krátká. Jediná změna, co jsem dělal na terénu byla, že jsem ho nabarvil. K tomu jsem použil pouze tři barvy, a to hnědou na cesty, zelenou na trávu a černou pro vizuální ohraničení farmy. Terén lze barvit, pokud si ho označíme kliknutím levým tlačítkem myši a poté na pravé straně, v záložce *Inspector*, klikneme, opět, levým tlačítkem myši na obrázek štětce. Potom už jen stačí sjet v této záložce níže a kliknout na *Edit Terrain Layers*, pak na *Add Layer*, a nakonec vybrat 2D texturu, kterou chceme barvit.



Obrázek 10 Výběr úpravy terénu [Autor]



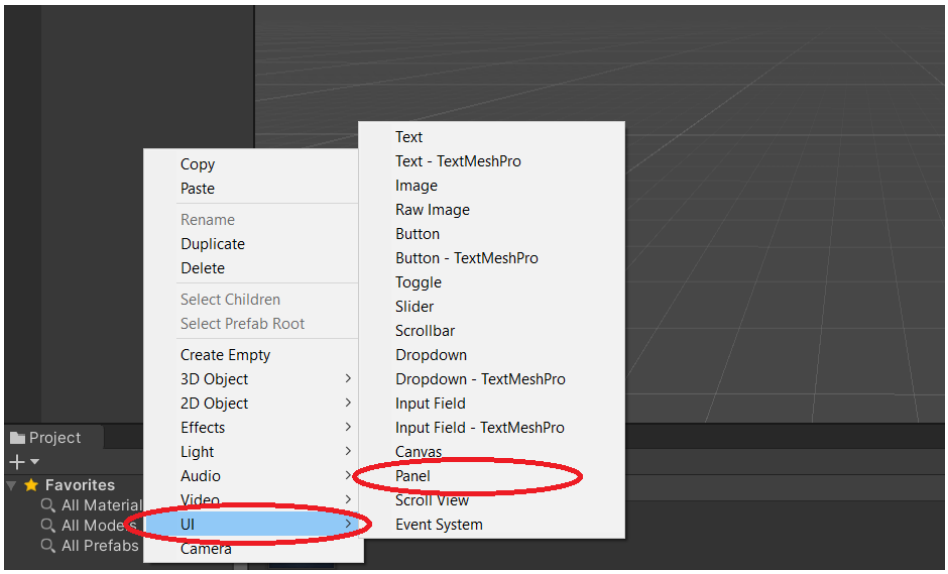
Obrázek 11 Vytváření nové barvy terénu [Autor]

Co jsem ale naopak ve své práci udělal, bylo přidání mnoha modelů do tohoto prostředí a začal jsem na tomto terénu postupně tvořit jednotlivé prvky hry, které pak vidí hráč. Prvně je nutné, aby si každý tvůrce určil měřítko, aby mohl velikosti jednotlivých věcí porovnávat a mohl tak dosáhnout co nejlepšího vzhledu v rámci zvoleného stylu. Například, má hra je vytvořená v *Low-poly* grafice a vypadá proto jako kreslená, ale vzájemné velikosti jednotlivých předmětů jsou realistické. Jako prostředí své hry jsem zvolil farmu. Podle své představy jsem si tedy načrtnul návrh, a když jsem s ním byl spokojený položil jsem první modely. Začal jsem komplikovanějšími stavbami, jako je například malý domeček, který jsem poskládal z různých modelů, stodola nebo silo. Poté jsem se přesunul k menším modelům jako jsou třeba dva traktory, studna nebo stromy. Takto jsem se propracoval až k nejmenším detailům jako je tráva nebo kameny podél cest. Tento postup je i často doporučován designery – začít velkými prvky a propracovat se k detailům.

Obrázek 12 Farma ptačí pohled [Autor]

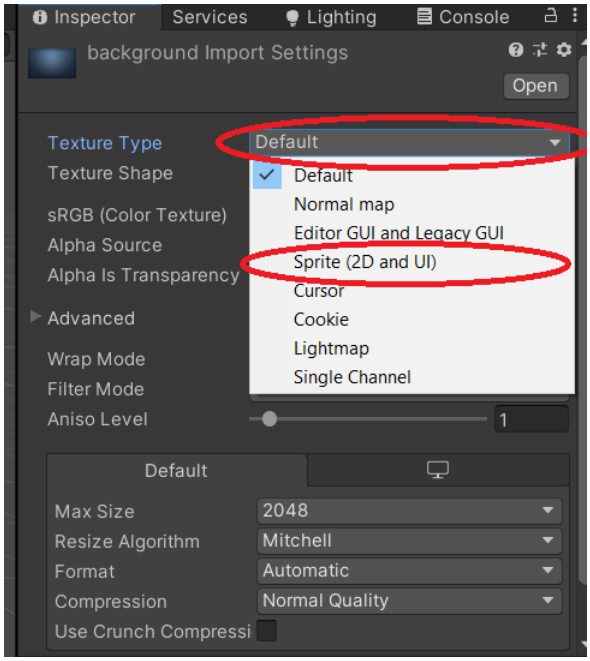
5.6 Začáteční menu a úvod do hry

Po dokončení hlavní scény se většina autorů přesouvá k vytvoření menu, které se ukáže hned po zapnutí hry. I já jsem tak učinil. U tvorby takového menu jsou nejdůležitější dva prvky, kterými jsou pozadí a samotný text. První, co musí každý tvůrce udělat před tím, než začne tyto dvě věci řešit, je vytvořit si plochu kam může tyto dvě věci vložit. To jsem udělal pravým kliknutím na levou část editoru do sloupce se scénou, najel na *UI* a zde na jsem kliknul na *Panel*. (Viz obrázek 13) To ve scéně vytvoří již zmiňovanou plochu, do které je poté možno vložit zvolené pozadí.



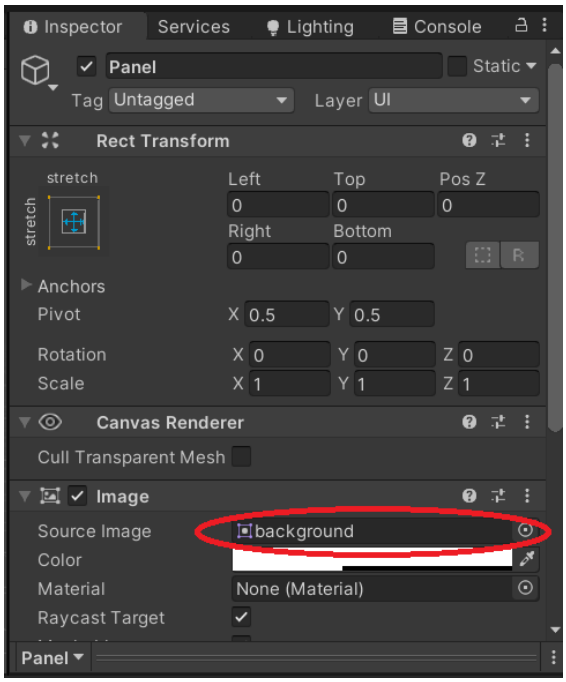
Obrázek 13 Vytváření Panelu [Autor]

Při tvorbě pozadí je potřeba řešit, aby nebylo moc rušivé pro oči, ale zároveň, aby na něm byl dobře vidět text, proto je dobré si vybrat tmavé pozadí se světlým textem nebo obráceně, já si vybral tmavě modré, které směrem ke středu přechází ke světlejšímu, aby hráč nekoukal jen na jednobarevnou plochu. Když má tvůrce obrázek vybraný, musí ho po vložení do Unity ještě upravit, aby ho mohl použít jako pozadí. Je potřeba z takzvaného *Default* 2D obrázku udělat *Sprite (2D and UI)*. (Viz obrázek 14.) Taková změna se provádí následovně. Nejdříve je třeba si v Unity svůj obrázek najít, kliknout na něj a v pravém horním rohu editoru rozbalit záložku *Texture Type*, a najít si již zmíněný *Sprite (2D and UI)*.



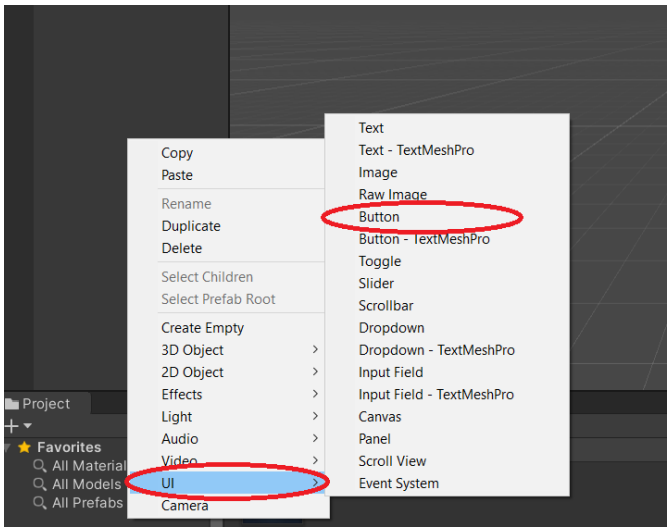
Obrázek 14 Změna typu obrázku [Autor]

Po změně typu obrázku je potřeba ho ještě vložit do panelu, který je předem vytvořen dříve. Ve scéně je pak nutné si tento panel vyhledat, kliknout na něj levým tlačítkem myši a tím ho označit, aby se o něm v pravé části editoru, v záložce *Inspector*, ukázali informace. Tam je třeba přesunout svůj zrak trochu níže a najít kolonku, vedle které se nachází nápis *Source Image*, a do této kolonky vybrané pozadí přesunout.



Obrázek 15 Vkládání obrázku na panel [Autor]

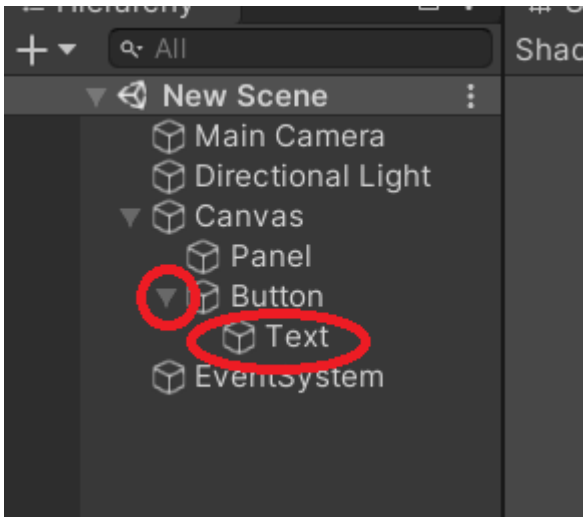
Tím je pozadí hotové, ale tvůrce musí na toto pozadí vložit alespoň nějaká tlačítka, aby se hráč po spuštění hry odsud někam dostal. Tlačítka jsou vkládána velice jednoduše, hodně podobně jako panel, ale místo na *Panel*, pod záložkou *UI*, je nutno kliknout na *Button*. Tato tlačítka jsem si pro svoje hlavní menu vytvořil tři – *Play, How to play* a *Quit*.



Obrázek 16 Vytváření tlačítka [Autor]

Aby ale tlačítka vypadala tak, jak si autor hry představuje, musí je většinou upravit. U všech těchto tlačítek jsem udělal vizuální úpravy, kromě textu a velikosti, stejné. To znamená, že jsem si u každého tlačítka v pravé části editoru, v záložce *Inspector*, upravil

jeho velikost, barvu textu na bílou a barvu pozadí na černou, k čemuž se potom ale vázalo také to, že jsem změnil normální barvu na transparentní. To znamená, že pokud není na tomto tlačítku kurzor, černá barva vidět není. Takto jsem postupoval u všech tří tlačítek. Změnu textu zde ale není možno nalézt, a proto jsem přejel doleva, do seznamu objektů ve scéně, a tam jsem, pomocí malé šipky, rozbalil seznam věcí pod tlačítkem, kde se nachází text. Po jeho označení se mi v *Inspektoru* otevřelo veškeré nastavení tohoto textu. V tomto nastavení už si pak může autor vyhrát, jak se mu zachce.



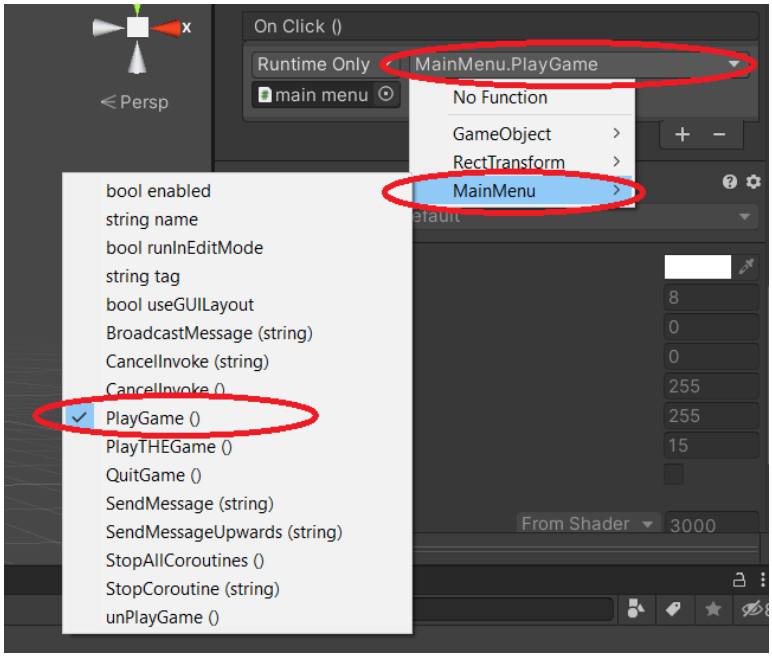
Obrázek 17 Text tlačítka [Autor]

Všechna tři tlačítka jsem si potom vložil do jedné složku, abych se ve scéně lépe vyznal. Poté jsem si vytvořil další jedno tlačítko a dvě textová pole a ty vložil pod novou složku. Tyto položky jsem použil později v této práci pro objasnění některých dalších skutečností. Aby ale tlačítka mohla fungovat a k něčemu byla, je třeba pro ně napsat skript.

```
public class MainMenu : MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
    public void PlayTHEGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
        PauseMenu.GameIsPaused = false;
    }
    public void unPlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
    }
    public void QuitGame()
    {
        Debug.Log("QUIT!");
        Application.Quit();
    }
}
```

Ten je určen převážně k přepínání scén, což by ale v Unity nefungovalo samo o sobě, proto je nutné přidat na začátek kódu část *enginu*, která toto přepínání podporuje. To znamená, že je třeba dopsat na tento, zmíněný, začátek *using UnityEngine.SceneManagement*. Nyní ale už ke kódu samotnému – opravdu často se zde objevuje sousloví *public void*, a to je ve skriptu, právě protože pomocí toho lze vidět jména funkcí napsaných za tím, dále, v editoru, když budou připojovány k jednotlivým tlačítkům. První z těchto funkcí načítá scénu, která přichází po scéně, ve které bylo toto tlačítko stlačeno. Druhá dělá téměř to samé, ale kromě již zmíněného ještě přepíná jistou hodnotu funkce odkazující na jiný skript, o kterém budu mluvit v jedné z následujících kapitol této práce. Třetí funkce ale, na rozdíl od prvních dvou, posouvá scénu o jednu nazpět. Poslední, ze zde zmíněných, funkcí je ale zase něco zcela jiného než tyto tři. U ní je nejdůležitější druhý řádek, který vypne celou aplikaci, kde ten první, zatím, jen do konzole editoru vypíše frázi uvedenou v uvozovkách. V mém případě to je „QUIT!“, což se do kódu píše, aby, když je hra testována a tato funkce tedy zatím nemůže vypnou editor, tvůrce věděl, zda toto tlačítko funguje. Kromě hlavního menu používám tento kód i v další scéně, protože pro mě bylo jednodušší spojit dva skripty dohromady, a proto jsem také hned na první scéně nepoužil všechny funkce, které bych býval mohl. Po dopsání tohoto skriptu jsem si ho připojil ke složce, ve které mám tlačítka uložena.

Tlačítka *Play* a *Quit* jsem napojoval stejně, pomocí kódu, který byl již zmíněn dříve. V tuto chvíli si ale, tvůrce hry možná říká, jak má tlačítka, která právě vytvořil napojit. V editoru si autor označí jedno tlačítko, které chce právě spravovat, v záložce *Inspector* sjede úplně dolů, a klikne na plus u kolonky s názvem *On Click ()*, což přidá možnost přidání funkce, co se stane, když je tlačítko zmáčknuto. Zde jsem si já do prázdné kolonky přetáhl složku, ve které jsem měl tlačítka a na kterou byl připojený můj skript. Potom už mi stačilo jen přidat kýženou funkci. Tu je možné přidat přes kolonku, na které je, do autorem provedené změny, napsáno *No Function*. Tu je třeba rozbalit, pod ní najít jméno svého skriptu a v něm funkci, kterou chce tvůrce přidat.



Obrázek 18 Přidání funkce tlačítku [Autor]

U tlačítka *How to play* byl můj postup sice hodně podobný jako u předchozích tlačítek, ale přece se trochu lišil. Zde jsem, totiž, nepracoval se svým vlastním skriptem ale s funkcemi přidanými samotným Unity. Mým cílem zde bylo přepínat viditelnost dvou, jinak se překrývajících, textů. Nový text a tlačítko jsem si vložil do jiné složky než své hlavní menu. Poté už stačilo pouze přidat funkce těmto tlačítkům, obdobně k tomu, co již bylo řečeno. Tentokrát jsem si přidal dvě místa na funkce a do každého z nich umístil jednou složku. Tentokrát jsem však nenastavoval tyto funkce přes svůj skript, ale přes funkci *GameObject.SetActive*, která, jak již bylo řečeno, vypíná a zapíná viditelnost a možnost používání určitého objektu, ke kterému je připojena. U každého z těchto tlačítek

jsem vypnutí a zapnutí složek nastavil vždy přesně obráceně oproti té druhé, aby se nemohlo stát, že by byly obě zapnuté, nebo naopak obě vypnuté ve stejnou chvíli.

5.7 Pauza

Pro případ, že musí hráč někam během hraní odejít, opustit tak počítač, je téměř vždy vhodné mít ve hře možnost hru pozastavit. Z tohoto důvodu jsem i já do této hry takovou možnost přidal. Kromě smyslu zastavení hry, je důležité i vizuální zobrazení tohoto stavu, aby si hráč hru omylem nepozastavil a pak si nemyslel, že hra nefunguje. Pauzou, respektive její možností, je třeba ošetřit téměř všechny scény hry. Musel jsem proto vložit do každé scény panel se třemi tlačítky. Panel, který je automaticky nastaven na celou obrazovku, jsem ve svém případě nabarvil na černo, ale barvu jsem nenechal sytou, nýbrž jsem ji trochu zprůhlednil, aby, když se hráč vrátí k počítači, alespoň částečně věděl, kde se jeho postava zrovna nachází. Tlačítka *Resume,* znamenající pokračovat v hraní, *Menu,* sloužící k návratu do menu, a *Quit*, sloužící k vypnutí aplikace, jsem si, stejně jako u hlavního menu, vložil do jedné složky, ke které jsem připojil vlastní skript, který byl již napsán dříve. (Viz skript na následující straně.)

První, co jsem udělal při tvorbě této pauzy, bylo, že jsem vytvořil takzvaný *bool*, což je hodnota pracující pouze se vstupy *true* a *false* – jinými slovy, pravda a lež, nebo 1 a 0, a ten nastavil na výchozí hodnotu *false*, tedy 0. Funkce *bool* v tomto skriptu pouze mění své hodnoty, což, v jiném skriptu, zamyká a odemyká kurzor uprostřed obrazovky a zviditelňuje ho a nechává mizet. Jedná se o část kódu viditelnou níže na této straně. U jiných skriptů jsem tuto funkci používal pouze z důvodů jejich zjednodušení, abych kvůli pouhé změně jména funkce nemusel psát další skripty.

```
if (PauseMenu.GameIsPaused)
        {
            Cursor.lockState = CursorLockMode.None;
            Cursor.visible = true;
            mouseSensitivity = 0f;
        }
        else
        {
            Cursor.lockState = CursorLockMode.Locked;
            Cursor.visible = false;
            mouseSensitivity = 10f;
        }
```

```
public class PauseMenu : MonoBehaviour
{

    public static bool GameIsPaused = false;

    public GameObject pauseMenuUI;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (GameIsPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }
    public void Resume ()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
    }
    void Pause ()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
    }
    public void LoadMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(0);

    }
    public void QuitGame()
    {
        Debug.Log("QUIT!");
        Application.Quit();

    }
}
```

Nyní zde bude dokončen popis tohoto skriptu. Tedy, následovalo vytvoření kolonky do Unity, ke které jsem později přiřadil svůj panel s tlačítky, se kterými kód pracuje. Na každém snímku hry tento skript kontroluje, zda nebyla stisknuta klávesa *Escape*. Pokud je tato klávesa stisknuta, spustí se kontrola podmínky, a to takové, jakou hodnotu má zrovna *bool*. Pokud je hodnota zrovna *true*, spustí se funkce *Resume()*, která je zapsána níže v kódu, ale pokud je naopak v tu chvíli hodnota *false*, spustí se funkce *Pause()*, která je taktéž níže.

Po podmínkách jsem se dostal k nastavení funkce pro fungování prvního tlačítka, kterým je *Resume*. Abych se ve skriptu mohl dobře orientovat, pojmenoval jsem si funkce stejně, nebo alespoň podobně jako tlačítka. Jak tlačítko, tak ale i funkci, zmiňovanou dříve, jsem naprogramoval tak, aby, když bude spuštěna, se přerušila viditelnost a funkčnost panelu s tlačítky a znovu se spustil čas scény, tedy jinými slovy, aby se vše dalo znovu do pohybu a aby se *bool* přepnul na hodnotu *false*.

Další funkcí v tomto kódu je funkce *Pause(),* která neodkazuje na žádné tlačítko, a proto se vyskytuje pouze v podmínce, výše ve skriptu. Díky té se, v podstatě, děje přesný opak toho, co dělá funkce *Resume()*. To znamená, že se zde zviditelní panel s tlačítky, zastaví se čas ve scéně a *bool* se nastaví na hodnotu *true,*

Poté je třeba nastavit tlačítko pro návrat do hlavního menu. Aby měl hráč přístup k tomuto tlačítku během hry, musí nejdříve stisknout klávesu *Escape* a pozastavit tak hru, stejně jako pro tlačítko *Resume*. Díky tomu, že kurzor myši je zde volně pohyblivý, stačilo napsat do skriptu pouze několik málo úkonů – změnit scénu a spustit čas.

Poslední nastavované tlačítko je *Quit*. To jsem zde nastavil úplně stejně jako u hlavního menu. To znamená, že pro kontrolu fungování jsem si do konzole editoru nechával vypisovat frázi, která ale fungování skriptu nijak neovlivňuje. K tomuto jsem ještě připsal funkci vypínající celou aplikaci.

5.8 Minihra – Bludiště

Minihra Bludiště není technicky náročná, ale i jednoduché věci se při tvorbě her tu a tam objevují. V této minihře jsem se snažil hlavně poukázat na to, že není třeba složitého programování, aby bylo možno vytvořit jednoduchou hru, takže zde není o hře samotné moc co říct.

Protože se ale musí hráč pohybovat mezi hlavní scénou farmy a minihrami, využiji tohoto prostoru a popíši zde, jak tento přesun funguje. (Pro přesun do a ze všech miniher je využíván totožný způsob.)

```
public class bludisteTP : MonoBehaviour
{

    void OnCollisionEnter(Collision collisionInfo)
    {
        if (collisionInfo.collider.tag == "praseTAM")
        {
            SceneManager.LoadScene(3);
            PauseMenu.GameIsPaused = true;

        }
    }
}
```

Tento skript je používán pro přesun z farmy na další scénu, kterou je vždy vysvětlení příběhového aspektu a cíle dané minihry, do které se snaží hráč přesunout. U každého zvířete na farmě jsem tedy nastavil unikátní *tag*, neboli štítek, na který jsem potom v kódu odkazoval, když byly zrovna popisovány jisté kolize.

Nyní bude vysvětleno, co a jak tento skript dělá. Podmínka napsaná ve skriptu při každé kolizi s jakýmkoliv objektem na scéně zjistí, zda se jedná o takový objekt, název jehož štítku by byl shodný s frází v uvozovkách. Pokud je tento název stejný, načte se další scéna, kde se nachází vysvětlení minihry, jak bylo řečeno, a je zviditelněn kurzor myši.

```
public class bludisteVysvetleni : MonoBehaviour
{
    public void PlayMaze()
    {
        SceneManager.LoadScene(4);
        PauseMenu.GameIsPaused = false;
    }
    public void unPlayMaze()
    {
        SceneManager.LoadScene(2);
        PauseMenu.GameIsPaused = false;
    }
}
```

Na každé scéně s vysvětlením jsou vždy dvě tlačítka, *Continue*, znamenající pokračovat, a *Back*, znamenající zpět. Obě fungují na stejném principu, a to takovém že načtou jinou scénu a znovu zneviditelní kurzor myši.

```
public class bludistePryc : MonoBehaviour
{

    void OnCollisionEnter(Collision collisionInfo)
    {
        if (collisionInfo.collider.tag == "praseZPATKY")
        {
            SceneManager.LoadScene(2);
        }
    }
}
```

Do každé minihry jsem také přidal cestu, jak se vrátit zpět na hlavní scénu farmy pomocí kolize s jistými objekty v minihrách. Tento skript načte scénu farmy, a protože je již v minihře kurzor neviditelný, nic jiného nebylo nutno psát.

```
public class bludisteVyhra : MonoBehaviour
{

    void OnCollisionEnter(Collision collisionInfo)
    {
        if (collisionInfo.collider.tag == "mazeVyhra")
        {
            SceneManager.LoadScene(5);
            PauseMenu.GameIsPaused = true;
        }
    }
    public void leaveMazeforever()
    {
        SceneManager.LoadScene(2);
        PauseMenu.GameIsPaused = false;
    }

}
```

Po dosažení cíle minihry se načte „děkovací“ scéna, kde zvíře z dané minihry nějakým způsobem poděkuje za laskavost, kterou mu hráč prokázal. Na této scéně se nachází pouze jedno tlačítko a to hráče vrátí zpět na farmu.

5.9 Minihra – Létání

Jak se do minihry dostat, bylo vysvětleno u hry bludiště. Jeden z rozdílu oproti hře bludiště je však to, že tato minihra je hrána z pohledu třetí osoby, takže bylo třeba napsat zcela nový skript pro kameru.

```
public class CameraFollow : MonoBehaviour
{
    public Transform player;
    public Vector3 offset;

    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }
    void Update()
    {
        transform.position = player.position + offset;
        if (PauseMenu.GameIsPaused)
        {
            Cursor.lockState = CursorLockMode.None;
            Cursor.visible = true;

        }
        else
        {
            Cursor.lockState = CursorLockMode.Locked;
            Cursor.visible = false;

        }
    }
}
```

Ve funkci *void Start()*, která se spouští hned při načtení scény, byl zamčen a zneviditelněn kurzor, stejně jako v jiných skriptech pro tuto hru. Možná nejdůležitější řádek tohoto skriptu je první řádek funkce *void Update()*, který, pomocí předem nastavené hodnoty vzdálenosti kamery od hráče a proměnné, kterou je zde poloha hráče na scéně, určuje polohu kamery v prostoru. Zmíněná hodnota vzdálenosti kamery od hráče je zadávaná do editoru v posunu po prostorových osách x, y a z.

Pro tuto minihru byl také naprogramován odlišný způsob pohybu, protože zde bylo potřeba, aby se hráč pohyboval svým přičiněním právě nahoru a dolů.

```
public class Floppymovement : MonoBehaviour
{
    public static bool playerMovement = true;

    public Rigidbody rb;

    public float ForwardForce = 2000f;
    public float UpwardForce = 500f;

    void FixedUpdate()
    {
        if (playerMovement)
        {
            rb.AddForce(ForwardForce * Time.deltaTime, 0, 0);

            if (Input.GetKey("space"))
            {
                rb.AddForce(0, UpwardForce * Time.deltaTime, 0,
ForceMode.VelocityChange);
            }
        }
    }
}
```

Tento pohyb spočívá v působení sil na objekt – hráče. Aby se hráč stále pohyboval dopředu, je mu neustále dodávána energie. Toto přidávání lze vidět v první podmínce tohoto skriptu. Síla je zde objektu dodávána pouze ve směru osy x, proto také hodnoty y a z jsou zde reprezentovány nulami. Druhá podmínka je té první velice podobná, ale síla není hmotnému bodu dodávána ve směru x, ale y. Právě proto jsou zde hodnoty x a z reprezentovány nulami. Rozdíl oproti první podmínce je také to, že detail na konci této podmínky mění fungování setrvačné síly v tomto směru. Tato, druhá, podmínka ale není splněna pořád – platí, pouze pokud hráč drží stisknutou klávesu *Space*, neboli mezerník.

Jedním z posledních v této minihře je kód, který vrací hráče na start, pokud narazí do překážky, nebo spustí další scénu, když hráč doletí do cíle.

```
public class PlayerCollision : MonoBehaviour
{
    void OnCollisionEnter(Collision collisionInfo)
    {
        if (collisionInfo.collider.tag == "floppyVyhra")
        {

            SceneManager.LoadScene(8);
            PauseMenu.GameIsPaused = true;

        }
        if (collisionInfo.collider.tag == "klada")
        {

            FindObjectOfType<Gamemanager>().EndGame();

        }
    }
}
```

V tomto kódu se vyskytují dvě podmínky, kde první slouží k ukončení minihry, a to pomocí kolize s objektem se štítkem *floppyVyhra*. Druhá podmínka se zabývá vracením hráče na začátek minihry, znovu, pomocí kolize hráče s objektem, který má ale tentokrát štítek *klada*. V tomto případě však nestačí načíst další scénu – je třeba odkázat na *Gamemanager*, ve kterém je kód, který vypadá následovně.

```
public class Gamemanager : MonoBehaviour
{

    bool GameHasEnded = false;

    public float restartDelay = 1f;


    public void EndGame()
    {
        if (GameHasEnded == false)
        {
            GameHasEnded = true;
            Debug.Log("GAME OVER");
            Invoke("Restart", restartDelay);
        }

    }
    void Restart()
    {
        SceneManager.LoadScene(7);
    }
}
```

Zde se nachází podmínka, která kontroluje, zda je *bool* s názvem *GameHasEnded* na hodnotě *false*. Pokud je, přepne se tato hodnota na *true* a je zahájen restart scény, u kterého bylo ještě nastaveno, jak dlouho má program čekat, než scénu restartuje.

5.10 Minihra – Hledání

V této minihře bylo oproti hlavní scéně farmy nutno přidat pouze dva skripty. Jeden z nich kontroluje a počítá nalezené objekty a druhý je nechává mizet. Do scény bylo tedy pouze přidáno prázdné textové pole, do kterého byl následně první skript připojen. (Nyní budou tyto dva kódy vyobrazeny a popsány.)

```
public class Score : MonoBehaviour
{
    public static Score instance;

    public Text scoreText;

    int score = 0;

    private void Awake()
    {
        instance = this;
    }
    void Start()
    {
        scoreText.text = score.ToString() + "/4";
    }
    public void AddPoint()
    {
        score += 1;
        scoreText.text = score.ToString() + "/4";
    }
    void Update()
    {
        if(score == 4)
        {
            SceneManager.LoadScene(11);
            PauseMenu.GameIsPaused = true;
        }
    }
}
```

Tento, první, skript, jak již bylo zmíněno, se zabývá kontrolou skóre a konečným přepnutím na následující scénu. Pod funkcí *void Start()* je řádek, který zajišťuje zobrazení skóre, na začátku 0/4. Funkce *public void AddPoint()* zajišťuje přičítání bodů. Zde poslední funkce, *void Update(),* kontroluje zda není skóre plné, tedy 4/4. Pokud je, načte následující scénu a zviditelní kurzor.

```
public class AddPointAndBeGone : MonoBehaviour
{
    public GameObject podkova;

    void OnCollisionEnter(Collision collisionInfo)
    {

        if (collisionInfo.collider.tag == "Player")
        {
            podkova.GetComponent<MeshRenderer>().enabled = false;
            podkova.GetComponent<MeshCollider>().enabled = false;
            Score.instance.AddPoint();
        }
    }
}
```

V tomto, druhém, skriptu se objevuje pouze jedna podmínka. Ta kontroluje kolizi hráče s určeným objektem, v tomto případě s podkovou. Když této kolizi dojde, viditelnost a možnost další kolize podkovy se vypnou, a je spuštěna funkce *AddPoint()*, nacházející se v minulém skriptu, která přičte hráči bod.

Diskuse

V praktické části práce jsem se snažil naplnit téma zadání – demonstrovat tvorbu hry s důrazem na tvorbu prostředí hry, a formou jednoduchého příběhu s vloženými minihrami zaujmout a motivovat k hraní této hry. Tématem mé hry je zemědělská farma se zvířaty; hlavním hrdinou je husa, se kterou se hráč může ztotožnit. Na své cestě po farmě musí husa „Goosey“ (hráč) pomoci třem zvířatům splnit jednoduché úkoly, aby hráč mohl úspěšně hru dokončit.

Ve hře bylo vytvořeno prostředí o dvanácti scénách včetně hlavního menu, základní scény farmy, tří scén miniher a sedmi scén s příběhem. Prvně byla vytvořena postava, která je ovládána pomocí kláves W, A, S, D, a mezerníku a pohybu myši, která zajišťuje, aby se mohla postava různě otáčet. Poté bylo vytvořeno do scén farmy a miniher příslušné prostředí. Prostředí bylo vytvořeno v *Low-poly* grafice, se zaměřením na nejmenší detaily, jako je tráva nebo malé houby v jedné z miniher. Dále byla vytvořena scéna hlavního menu. Zde bylo poprvé zapotřebí využít 2D pohledu, skriptů zaměřujících se na fungování tlačítek a přepínání mezi scénami. Obdobně byla vytvořena i pauzy. Pauza byla přidána také do všech scén, kde se hráč může volně pohybovat. Funkčnost pauzy byla nastavena na principu zapínání a vypínání panelu s tlačítky a zastavování času ve hře. Nakonec byly vytvořeny minihry. Pro tvorbu bludiště byly využity již dříve používané skripty; pro minihru létání byl vytvořen unikátní skript pro pohyb; a pro minihru hledání byl vytvořen skript pro zaznamenávání skóre.

V práci by se dalo pokračovat a vylepšovat ji o množství dalších prvků, jako jsou vytvoření detailnějších a reálnějších textur prostředí, vytvoření větší mapy pro volný pohyb hráče, propracovanější zobrazení zvířat, a v neposlední řadě zařazení více miniher. Bylo by možné, a vhodné, vylepšit pohyb postavy, aby pohyb působil realističtěji a nebylo možné provádět nereálné pohyby, jako například oscilace postavy při skoku. Přínosem by bylo rozšířit nabídku postav, za které bude hráč moci hrát, a jednotlivým postavám nadefinovat i speciální vlastnosti.

Pravděpodobně jsou nyní některé parametry nastaveny ne zcela efektivně a po detailní analýze by bylo možno dosáhnout zjednodušení a zefektivnění některých skriptů.

Závěr

Cílem práce bylo seznámit čtenáře se základními principy tvorby počítačových her.

Teoretická část je zaměřena na možnosti volby různých herních enginů a výběr počítačového jazyka při tvorbě her, a je pouze stručným informativním přehledem. První část je věnována popisu a rozdělení herních enginů. Druhá část hovoří o vybraných programovacích jazycích a v poslední části teoretického bloku jsou popsány některé vývojářské enginy. Jedná se o velmi zjednodušený a, zcela jistě, nekompletní přehled, jelikož zásadní byla snaha o zdůraznění předností popisovaných typů enginů a jazyků s ohledem na možnost práce s těmito produkty pro začínající programátory. Je zde zmíněna i dostupnost některých komponentů a možnost jejich využití v závislosti na typu platforem.

V praktické části byla uskutečněna a popsána tvorba hry v Unity v *Low polygon* grafice v jazyce C#, protože právě herní engine Unity je pro začínající tvůrce často doporučován. V praktické části práce byla cíleně využita jiná forma popisu než v teoretické části práce, aby byly tvorba hry, funkce jednotlivých skriptů a způsob práce, přiblíženy začínajícím tvůrcům. Tvorba hry je velmi kreativní aktivitou, ale je časově náročná, kde například tvorba opakujících se detailů pozadí se ukázala být zdlouhavá. Osvědčilo se práci rozdělit do několika pracovních bloků a rutinní zápisy střídat s kreativní tvorbou, aby výrazně neklesala produktivita práce. Hra by mohla být rozšířena o řadu dalších herních prvků a vhodné by bylo zdokonalit prostředí hry; velkým přínosem by bylo zjednodušení a zefektivnění některých skriptů.

Z pohledu začínajícího tvůrce počítačových her, kterým je i sám autor práce, byly cíle naplněny a, zda se podaří touto prací zaujmout a motivovat k práci podobné i ostatní, ukáže čas. Například u této práce se časové omezení, ale i hloubka teoretických a praktických znalostí tvůrce, ukázaly být zásadním limitem, ale nemusí tomu tak být pro každého a výsledek může být i přes to výborný.

Seznam použité literatury

[1]: *Unity* [online]. USA: 2021 Unity Technologies, 2005 [cit. 2021-12-05]. Dostupné z: https://unity.com/

[2]: *Unity Learn* [online]. USA: 2021 Unity Technologies, 2005 [cit. 2021-12-05]. Dostupné z: https://learn.unity.com/

[3]: Unity User Manual 2020.3 (LTS). *Unity Manual* [online]. USA: 2021 Unity Technologies, 2021 [cit. 2021-12-05]. Dostupné z: https://docs.unity3d.com/Manual/index.html

[4]: *Unreal Engine* [online]. USA: Epic Games, 2004 [cit. 2021-12-05]. Dostupné z: https://www.unrealengine.com/en-US/

[5]: Welcome. *Wayback Machine* [online]. San Francisco: Internet Archive, 2014 [cit. 2021-12-05]. Dostupné z: https://web.archive.org/web/20110207205744/http://www.unrealtechnology.com/

[6]: Unreal Engine 2. *Wayback Machine* [online]. San Francisco: Internet Archive, 2014 [cit. 2021-12-05]. Dostupné z: https://web.archive.org/web/20081210001754/http://www.unrealtechnology.com/features.php?ref=past-versions

[7]: Current technology – Unreal Engine 3. *Wayback Machine* [online]. San Francisco: Internet Archive, 2014 [cit. 2021-12-05]. Dostupné z: https://web.archive.org/web/20081217133451/http://www.unrealtechnology.com/features.php?ref=technology-overview

[8]: Legacy:Unreal Engine Versions. *Wayback Machine* [online]. San Francisco: Internet Archive, 2014 [cit. 2021-12-05]. Dostupné z: https://web.archive.org/web/20081015020051/http://wiki.beyondunreal.com/Legacy:Unreal_Engine_Versions

[9]: *CryEngine* [online]. Německo: 2021 Crytek, 2002 [cit. 2021-12-05]. Dostupné z: https://www.cryengine.com/

[10]: *CryTek* [online]. Německo: 2021 Crytek, 1999 [cit. 2021-12-05]. Dostupné z: https://www.crytek.com/

[11]: CryEngine. *Steam* [online]. Washington, USA: Valve corporation, 2003 [cit. 2021-12-05]. Dostupné z: https://store.steampowered.com/app/220980/CRYENGINE/?l=czech

[12]: *CRYENGINE V Manual* [online]. USA: 2021 Crytek, 2003 [cit. 2021-12-06]. Dostupné z: https://docs.cryengine.com/

[13]: Release Notes. *CryEngine* [online]. USA: 2021 Crytek, 2003 [cit. 2021-12-06]. Dostupné z: https://docs.cryengine.com/display/RN/Release+Notes

[14]: RAGE. *Mod DB* [online]. Austrálie: DBolical, 2002 [cit. 2021-12-06]. Dostupné z: https://www.moddb.com/engines/rage

[15]: Rockstar Advanced Game Engine. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2021-12-06]. Dostupné z: https://en.wikipedia.org/wiki/Rockstar_Advanced_Game_Engine

[16]: Ubisoft Anvil. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2021-12-06]. Dostupné z: https://en.wikipedia.org/wiki/Ubisoft_Anvil

[17]: Director (Anvil Pipeline). *Ubisoft Montréal* [online]. Francie: 2021 Ubisoft Entertainment, 1986 [cit. 2021-12-06]. Dostupné z: https://montreal.ubisoft.com/en/jobs/director-anvil-pipeline-2/

[18]: 4A Engine. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2021-12-06]. Dostupné z: https://en.wikipedia.org/wiki/4A_Engine

[19]: *4A Games* [online]. Kyjev, Ukrajina: 4A Games, 2020 [cit. 2021-12-06]. Dostupné z: http://www.4a-games.com/

[20]: Source (game engine). *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2021-12-06]. Dostupné z: https://en.wikipedia.org/wiki/Source_(game_engine)

[21]: Source 2. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2021-12-06]. Dostupné z: https://en.wikipedia.org/wiki/Source_2

[22]: C (programming language). *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2021-12-06]. Dostupné z: https://en.wikipedia.org/wiki/C_(programming_language)

[23] .*w3schools. C++ Introduction* [online]. [cit. 2021-12-05]. Dostupné z: https://www.w3schools.com

[24]: CSharp. *C# Tutorial and source code* [online]. [cit. 2021-12-05]. Dostupné z: http://csharp.net-informations.com/

[25] *Printable Version What is Java technology and why do I need it?* [online]. 2021 [cit. 2021-12-05]. Dostupné z: https://www.java.com

[26] Kubu. *Úvod do programovacího jazyka Java* [online]. 2021 [cit. 2021-12-05]. Dostupné z: http://kubu.wz.cz

[27]: *JavaScripct.com* [online]. USA: JavaScript.com, 2016 [cit. 2021-12-06]. Dostupné z: https://www.javascript.com/

[28]: JavaScripct. *Jak Psat Web* [online]. Česká Republika: jakpsatweb [cit. 2021-12-06]. Dostupné z: https://www.jakpsatweb.cz/javascript/

[29]: Lua: about. *Lua* [online]. Brzílie: PUC-RIO, 1993 [cit. 2021-12-06]. Dostupné z: https://www.lua.org/about.html

[30]: Python. *What is Python? Executive Summary* [online]. [cit. 2021-12-06]. Dostupné z: https://www.python.org

[31]: *Scratch* [online]. USA: Scratch Foundation, 2007 [cit. 2021-12-06]. Dostupné z: https://scratch.mit.edu/

[32]: O Scratchi. *Scratch* [online]. USA: Scratch Foundation, 2007 [cit. 2021-12-06]. Dostupné z: https://scratch.mit.edu/about

[33]: *Blender* [online]. [cit. 2021-12-06]. Dostupné z: https://www.blender.org

[34]: Blender. *Steam* [online]. Washington, USA: Valve corporation, 2003 [cit. 2021-12-05]. Dostupné z: https://store.steampowered.com/app/365670/Blender/

[35]: Microsoft. *Welcome to the Visual Studio IDE* [online]. [cit. 2021-12-05]. Dostupné z: https://microsoft.com

Seznam použitých obrázků

Obr. 2 [36]: Epic's next-gen Unreal Engine 5 is now available in early access. *Engadget* [online]. USA: Yahoo 2021, 1994 [cit. 2021-12-05]. Dostupné z: https://www.engadget.com/unreal-engine-5-early-access-152553496.html?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAAZ2QfMOxij02mG-D0dpdthTTS0t0ahYz_NEs4NmpJoUSpJ-F4iaFL3KXWFPd9QALzU9MBjJ5yQmNE1JMQj9g9gd_41xIJhJYeLaNoPqv1luXTLHs-Qa3Xd0y491avKWtORZ070SKGiOFwVWgAxUVwuSTOxESjZhvQtLoeZBcv9g

Obr. 3 [37]: CryEngine. *Steam* [online]. Washington, USA: Valve corporation, 2003 [cit. 2021-12-05]. Dostupné z: https://store.steampowered.com/app/220980/CRYENGINE/?l=czech

Obr. 4 [38]: DIE GESCHICHTE GEHT WEITER: METRO: LAST LIGHT. *Metro Exodus* [online]. Kyjev, Ukrajina: 4A Games, 2020 [cit. 2021-12-05]. Dostupné z: https://www.metrothegame.com/de/die-geschichte-bisher/

Obr. 6 [39]: Blender. *Steam* [online]. Washington, USA: Valve corporation, 2003 [cit. 2021-12-05]. Dostupné z: https://store.steampowered.com/app/365670/Blender/