|  |
| --- |
|  |
| Maturitní práce – **Informační technologie** |
| Vývoj 3D videoher ve Unity enginu |
| Autor práce:  Vojtěch Lahola, 4. AI |
| Vedoucí práce  Bc. Veronika Vyvlečková |
| 2023/2024 |

**Záznam o průběhu maturitní práce (VLOŽIT ORIGINÁL, DO ELEKTRONICKÉ VERZE NASCANOVAT)**

Školní rok 2023/2024.

**Jméno a příjmení autora práce: Vojtěch Lahola**

**Jméno a příjmení vedoucího práce: Veronika Vývlečková**

**Název práce: Vývoj 3D videoher ve Unity enginu**

**Počet příloh:**

**Podpis autora:**

Plnění stanovených termínů:

|  |  |  |  |
| --- | --- | --- | --- |
| **TERMÍN** | **STUPEŇ ROZPRACOVANOSTI** | **DNE** | **PODPIS** |
| do 1. 12. 2023 | První povinná konzultace s vedoucím práce |  |  |
| do 5. 1. 2024 | Druhá povinná konzultace s vedoucím práce |  |  |
| do 9. 2. 2024 | Třetí povinná konzultace s vedoucím práce |  |  |
| do 15. 3. 2024 | Povinná konzultace před dokončením práce |  |  |
| do 29. 3. 2024 | Odevzdání maturitní práce vedoucímu práce |  |  |
| do 1. 4. 2024 | Předání maturitní práce řediteli školy |  |  |
| do 24. 5. 2024 | Veřejná obhajoba maturitní práce |  |  |

**Při nedodržení dvou a více termínů konzultace je maturitní práce automaticky hodnocena známkou nedostatečná.**

Vyplněný a vedoucím práce podepsaný protokol je podmínkou pro obhajobu maturitní práce v řádném maturitním termínu. Tento protokol se rovněž vkládá do práce, za titulní stranu.

Nebudou-li řádně vyplněny veškeré údaje, především podpisy konzultanta (vedoucího práce) u stanovených termínů, nemůže student práci obhajovat.

|  |  |
| --- | --- |
| Prohlášení | |
| Prohlašuji, že jsem maturitní práci na téma \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ vypracoval(a) samostatně, s využitím poznatků získaných během studia a studiem odborné literatury a pramenů, uvedených na seznamu zdrojů, který tvoří přílohu této práce. | |
| Ve Zlíně dne: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Podpis |

**Poděkování**

Obsah

[Obsah 5](#_Toc158963411)

[Úvod 7](#_Toc158963412)

[I. Teoretická část 8](#_Toc158963413)

[1. Programovací jazyk c# 9](#_Toc158963414)

[1.1. Výhody vývoje v jazyce C# 9](#_Toc158963415)

[1.2. Základní informace 10](#_Toc158963416)

[1.2.1. Objektově orientované programování 10](#_Toc158963417)

[1.2.2. Přístupové modifikátory 11](#_Toc158963418)

[1.2.3. Objektově orientováné designové principy 11](#_Toc158963419)

[2. Herní enginy 14](#_Toc158963420)

[2.1. Hlavní představitelé 14](#_Toc158963421)

[2.1.1. Unreal Engine 14](#_Toc158963422)

[2.1.2. Unity engine 15](#_Toc158963423)

[3. Herní design 18](#_Toc158963424)

[3.1. Obecné informace 18](#_Toc158963425)

[3.2. Proces designování 18](#_Toc158963426)

[3.2.1. Výzkum a analýza 18](#_Toc158963427)

[3.2.2. Koncept 18](#_Toc158963428)

[3.2.3. Blocking 18](#_Toc158963429)

[3.2.4. Finalizace 18](#_Toc158963430)

[3.3. Využití psychologie při vytváření designu videoher 19](#_Toc158963431)

[3.3.1. Motivace 19](#_Toc158963432)

[3.3.2. Úspěchy 19](#_Toc158963433)

[3.3.3. Pokrok 19](#_Toc158963434)

[II. Praktická část 21](#_Toc158963435)

[4. Vývoj herní mapy 22](#_Toc158963436)

[4.1. Trenchbroom 22](#_Toc158963437)

[4.2. Osvětlení 23](#_Toc158963438)

[4.3. Implementace mapy 24](#_Toc158963439)

[4.4. Texturování 25](#_Toc158963440)

[5. Vývoj herních modelů 26](#_Toc158963441)

[5.1. Modelování 26](#_Toc158963442)

[5.2. Texturování 26](#_Toc158963443)

[5.3. Implementace modelů 27](#_Toc158963444)

[6. Vývoj hry 28](#_Toc158963445)

[6.1. Skriptování 28](#_Toc158963446)

[6.1.1. Střelba hráče 28](#_Toc158963447)

[6.1.2. Pohyb hráče 29](#_Toc158963448)

[6.1.3. Nepřátelé 30](#_Toc158963449)

[6.1.4. Inventář 31](#_Toc158963450)

[6.2. Zvuk 31](#_Toc158963451)

[6.2.1. Ukládání zvuků 31](#_Toc158963452)

[6.2.2. Přehrávání zvuků 32](#_Toc158963453)

[6.3. Uživatelské rozhraní 32](#_Toc158963454)

[6.3.1. Rozhraní hráče 32](#_Toc158963455)

[6.3.2. Hlavní menu 34](#_Toc158963456)

[6.4. Animace 35](#_Toc158963457)

[6.4.1. Mixamo 35](#_Toc158963458)

[6.4.2. Aplikace animací 36](#_Toc158963459)

[Závěr 37](#_Toc158963460)

[Seznam použité literatury 38](#_Toc158963461)

[Seznam obrázků 39](#_Toc158963462)

[Seznam tabulek 40](#_Toc158963463)

[Seznam příloh 41](#_Toc158963464)

Úvod

|  |
| --- |
| 1. Teoretická část |

1. Programovací jazyk c#

C# je moderní, vysoko úrovňový, objektově orientovaný programovací jazyk (OOP). Umožňuje vývojářům vytvářet stabilní a robustní aplikace, které primárně běží na .NET frameworku. C# pochází z jazykové rodiny C, proto sdílí, několik podobností s jazyky jako jsou C, C++, Java a Javascript. Díky flexibilitě jazyka lze v něm vyvíjet nejen mobilní a desktopové aplikace, ale dokonce i například videohry pomocí Unity Enginu nebo Godotu.

Programovací jazyk byl na designován zaměstnancem Microsoftu Andrers Hejlsbergem v roce 2000. Je to Dánský softwarový inženýr, který již dříve měl historii s účastněním se na vývoji dalších programovacích jazyků a nástrojích jako jsou Typescript a Delphi.

**Populární aplikace, které byli postaveny v jazyce C#:**

* Microsoft Visual Studio
* Paint.NET
* Pinta
* KeePass

<https://www.pluralsight.com/blog/software-development/everything-you-need-to-know-about-c->

Top of Form

<https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>

<https://www.geeksforgeeks.org/introduction-to-c-sharp/>

* 1. Výhody vývoje v jazyce C#

C# díky svému intuitivnímu designu je jazyk relativně jednoduchý pro naučení v porovnání s dalšími jazyky v rodině C. I když jeho křivka učení není srovnatelná k Pythnu, i tak je ideální pro začátečníky i zkušené programátory, kteří se chtějí naučit nový jazyk.

Jedna z dalších výhod je efektivita programovacího jazyku. Je to staticky psaný jazyk, což ujišťuje nejen jeho čitelnost a jednoduchost pro vyhledávání chyb v kódu, ale i k pochopení kódu, který uživatel píše.

Není to nejen jeho design, který exceluje při stavění různých aplikací, C# má jednu z největších komunit na trhu, která umožňuje najít podporu a odpověď na různé otázky velice jednoduše oproti ne tak populárních jazyků.

<https://www.codeguru.com/csharp/benefits-of-c/>

* 1. Základní informace
     1. Objektově orientované programování

OOP se může popsat jako způsob vyvíjení softwarových aplikací pomocí tříd, které spolu interagují pomocí objektů. Díky tomuhle způsobu vytváříme kód softwaru více flexibilní, jednoduše rozšiřitelný a znovu použitelný, což nám eliminuje redundaci a zvyšuje čitelnost.

Většina programovacích jazyků poskytuje vývojáři následující základní funkce, které lze popsat jako podstaty pro využívání OOP:

* **Třída**
  + Třída definuje strukturu pomocí metod a hodnot, které napodobují objekty z reálného světa.
* **Metoda**
  + Metoda udává konkrétní chování a provádí různé akce, které například mohou vracet informace o objektu, aktualizovat jeho data...
* **Objekt**
  + Objekt je instance třídy, která obsahuje všechny data a metody, které jsou v ní definovány. Tyhle data mohou být různě upravovány, pokud to dovoluje přístupový modifikátor
* **Rozhraní**
  + Rozhraní neboli interface definuje soubor pravidel, které zajišťují určitou funkcionalitu. Rozhraní se využívají v souladu se třídami pomocí OOP principů Jako polymorphismus a dědictví, které dělají kód více flexibilní a znovu použitelný

<https://stackify.com/oop-concepts-c-sharp/>

* + 1. Přístupové modifikátory
* **Public**
  + Člen s modifikátorem public, může být přistoupen z kamkoliv a jakýmkoliv kódem ve stejném projektu.
* **Private**
  + Člen s modifikátorem private může být přistoupen pouze v kódu ve stejné třídě nebo structu.
* **Protected**
  + Člen s modifikátorem protected může být použit pouze ve stejné třídě nebo v třídě, která je zděděná z jiné třídy.
* **Internal**
  + Člen s modifikátorem internal může být použit pouze v kódu stejné kompilace

<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers>

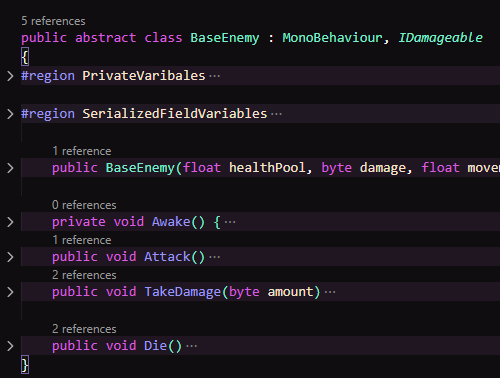
* + 1. Objektově orientováné designové principy

Těchto principů a technik designu existuje několik. Ujišťují vývojáři, že kód, který staví je jednoduše udržovatelný a rozšiřitelný.

Následující principy jsou hlavní pro objektově orientované programování:

**Abstrakce**

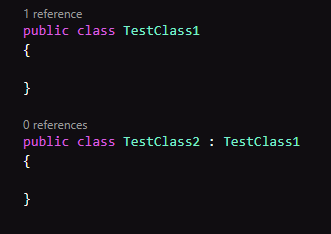
* Abstrakce je koncept myšlení při designování aplikace, který identifikuje třídy a jejich vlastnosti (metody, data...). Abstrakce vytváří základ logiky a dat bez přisazování jakýkoliv hodnot. Tento základ je pak přezván pomocí dědičnosti a umožňuje poté pracovat s veškerými daty, které byli převážný.
* Příklad takové Abstrakce může být vytvoření třídy ve videohře pro **nepřátele.**  Tato třída (např. BaseEnemy) by obsahovala veškeré funkce, které nepřátel může provádět (útok, chození...) a jeho data (životy, rychlost chození...). Táhle abstraktní třída by se poté mohla převzít pomocí **dědičnosti** do třídy druhé (např. Knight), kde by se poté přiřadily určité hodnoty...



<https://www.tutorialsteacher.com/csharp/abstraction>

**Dědičnost**

* Dědičnost je druh vztahu mezi třídami, který umožňuje dědění vlastností a funkcí z jedné třídy do třídy druhé. Díky takovému vztahuje kód získává větší čitelnost, zmenšení redundance a celkovou smysluplnou strukturu.



<https://www.tutorialsteacher.com/csharp/inheritance>

**Polymorphismus**

* Polymorphismus je slovo pocházející z Řeckého slova, které znamená více forem nebo tvarů. Vývojář může využít polymorphismus pro využití několik forem jedné metody ve stejné třídě se stejným názvem, ale jinou signaturou (např. přidání o jednu proměnnou více nebo změnění jejího druhu)
* Může být dosáhnut **dvěma způsoby**:
* **Přetěžování metod**
  + Při přetěžování metod se určuje, která metoda se stejným jménem bude zavolána během kompilování, což je krok před během programu. Která z těchto metod bude zavolána závisí čistě na signatuře metody tzn., že zvolí metodu pomocí parametrů.
* **Vyvolání přetížených metod**
  + Druhý způsob funguje na tom, že vývojář přímo může zavolat na přetíženou hodnotu, a to pomocí přiřazování přímo parametrů, které metoda potřebuje. Např. pokud přetížená hodnota potřebuje pouze parametr string, vývojář zavolá metodu s parametrem string.

<https://www.tutorialsteacher.com/csharp/polymorphism>

**Zapouzdření**

* Zapouzdření je technika pro implementaci abstrakce v kódu. Je tvořena pomocí tříd a jejich členy, které mají správně přiřazené přístupové modifikátory. Zapouzdření tedy umožňuje uschovat nebo ukázat data a funkce jiným třídám, což může přidat extra vrstvu zabezpečení.

<https://www.tutorialsteacher.com/csharp/encapsulation>

<https://www.tutorialsteacher.com/csharp/oop>

1. Herní enginy

Herní engine je software, který je primárně nedesignován pro vývoj videoher. Vývojáři dokážou zutilizovat funkce těchto enginů, aby nemuseli „začínat od nuly“ a vytvořit si vlastní engine, což jim ušetří čas a pracovní sílu pro vývoj hry. Engine přináší vývojáři obrovský nespočet předem vytvořený funkcí a zautomatizovaných procedur, příklad takových funkcí se může skládat z optimalizace projektu až po jednoduchého implementování projektu na více platform.

<https://www.gameopedia.com/game-engines-all-you-need-to-know-about/>

* 1. Hlavní představitelé

Aktuálně prosazeným industriálním standardem jsou enginy jako Unreal Engine a Unity. Přímo tyhle dva „velcí hráči na trhu,“ vytváří infrastrukturu nejpopulárnějších her na světě. V kapitole ní

* + 1. Unreal Engine

UE je nástroj pro vyvíjení videoher v reálnem čase od společnosti Epic Games. Jedna z jeho prvních verzí byla vyvíjená přímo od zakladatele firmy Tim Sweeney pro hru Unreal, která měla vyjít v roce 1998. EU byl licencován pro další vývojáře v roce 1996, což jim umožnilo volnému užití nástroje k vývoje dalších her.

Zpočátku UE byl nedesignován pro softwarové (procesorově založené) renderování, ale postupem času se začal zaměřovat na dedikovány hardware. Tvorba videoher není jediná věc, co UE dokáže. Je v něm možné dělat dokonce i simulace, rendrovat animace... UE pracuje v jazyku C++, který zajišťuje jeho portabilitu na jiné platformy a stabilitu.

Jedna z hlavních funkcí, na kterých UE funguje je vizuálně skriptovací systém s názvem Blueprint. Systém funguje na principů nodů, které vývojář dokáže pokládat jednotlivě na graf. Tyto nody reprezentují funkce, podmínky… Vývojář dokáže nody libovolně spojovat a propojovat, což umožňuje vytvářet komplexní systémy. Vizuální skriptování funguje jako alternativa manuálního psaní kódu, kterou mohou uvítat začáteční vývojáři. Celý systém lze porovnat k vývojovému diagramu, díky jeho vizualizace kódu.

Příklady her, které byli vyvíjeni v UE: Batman Arkham Knight, Fornite, Star Wars Jedi: Fallen Order...

<https://vagon.io/blog/top-10-games-made-with-unreal-engine/>

<https://www.bairesdev.com/blog/what-is-unreal-engine/>

https://www.unrealengine.com/en-US/license

https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/

* + 1. Unity engine

Stejně jako již zmíněný UE je Unity nástroj, který je stavěn pro vyvíjení her v reálném čase od společnosti Unity technologies, která byla založena roku 2004. První verze unity otevřená pro všechny vývojáře byla v roce 2005.

Oproti UE Unity je psáno v jazyce C#, který je o něco přívětivější pro začínající programátory než C++, například díky svému automatickému garbage collectoru. Stejně jako UE v Unity lze využít možnost vizuálního skriptování.

Unity umožňuje vývoj videoher nejen pouze v 2D, ale i v 3D prostoru. 2D hry využívají sprity, což jsou dvou dimenzionální obrázky nebo animace, které reprezentují herní postavy, objekty...

Narozdíl od 2D videoher hlavní stavební block pro vyvíjení v 3D prostoru Unity jsou GameObjects. Jak již z názvu lze vyčíst je to herní objekt, který je třeba využít, pokud vývojář chce pokládat něco na herní scénu. Samotné objekty toho moc nedokážou, ačkoliv spíše slouží jako kontejnery pro komponenty, které implementují funkcionalitu.

Příklady her, které byli vyvíjeni v Unity: Among Us, Ori and the Will of the Wisps, Cuphead, Beat Saber...

|  |  |
| --- | --- |
| **Výhody** | **Nevýhody** |
| Unity Asset Store obsahující několik zdarma využitelných modelů... | Mále vývojařské teamy nemají přístup k zdrojovému kódu Unity |
| Velké škála platforem na kterých může být využit | Menší výkon oproti dalším enginům |
| Výborná sada nástroju pro mobilní i indie vývoj | Je potřeba license pro projekty genurující více než 100 000$ měsíčně |

https://kevurugames.com/blog/unity-what-makes-it-the-best-game-engine/

<https://docs.unity3d.com/Manual/GameObjects.html>

<https://www.create-learn.us/blog/top-games-made-with-unity/>

<https://medium.com/@wota_mmorpg/unity-development-history-and-the-influence-of-this-game-engine-on-the-game-development-36dc7a7a3b9d>

<https://unity.com/pricing#plans-student-and-hobbyist>

* 1. Rozdíly mezi UE a Unity

1. Herní design

V téhle kapitole si zkráceně vysvětlíme pár informaci ohledně video herním designu a jeho různých procesů

* 1. Obecné informace

<https://en.wikipedia.org/wiki/Video_game_design>

* 1. Proces designování
     1. Výzkum a analýza

Před vytváření konceptu videohry, je důležité si udělat výzkum ohledně designu, rozložení a mechanik úrovní a blíže se seznámit s žánrem na kterém budeme stavět hru. Díky výzkumu a analýze získáme dostatečné znalosti, které nám pomohou v pozdějších fázích designu, jelikož vývojář bude seznámen, co se pro daný žánr hodí, jak by měla hra zhruba vypadat...

* + 1. Koncept

Poslední krok před samotným designovaným je koncept, který nám předá čistou představu ohledně tématu hry a základním wireframu. V téhle části je ještě jednoduché dělat změny oproti pokročilým fázích. Pokud již budeme vědět téma je mnohokrát jednoduší rozšiřovat na téhle myšlence, jelikož máme jednotný styl.

Koncept je hlavní část pro designery a vývojáře, která jim pomáhá při finálním designu a rozhodování co by se mohlo vylepšit, ba naopak přidat na základě konceptu. Díky němu lze si již představit různé herní mechaniky, atmosféra...

* + 1. Blocking

Blocking je fáze, kde se vytváří zjednodušená verze hry pomocí primitivních tvarů. Táhle verze nám dá hrubý rámec, který je založen na konceptu. Díky tomuhle si vývojář již vizuálně představit, jak bude finální verze vypadat. Blocking nepředá vývojáři čistě vzhledový output, který vychází z konceptu, ale také přibližný výkon a flow hry před finalizací.

* + 1. Finalizace

Finalizace je finální blok designu, kde se předpokládá, že rozhraní, herní mechaniky, téma… Jsou již ve verzi kde se neplánuje nic měnit a budou se jen přidávat a mírně upravovat před publikováním.

<https://300mind.studio/blog/game-level-design-guide/>

* 1. Využití psychologie při vytváření designu videoher

Pomocí psychologie v herním designu dokážou vývojáři hráči více porozumět. Tyto informace jim slouží pro nasměrování hráče cestou, která je bude udržovat při hraní a nebude je nudit. Každý uživatel má svůj vlastní důvod proč danou hru hraje a co je motivuje, proto je důležité udržovat balanc mezi těmito hlavními psychologickými technikami.

* + 1. Motivace

Motivace může být popsána jako důvod proč by měl hráč pokračovat v hraní hry. Zajišťuje, že hráče stále něco udržuje při hrání např. pomocí úkolů, které motivují hráče je plnit čistě pro získání požitku nebo za účelem získaní odměny. Tyto úkoly nebo činnosti se mohou lišit od nekomplexního rybaření až po obtížní hlavní úkol plný zvratů. Mohou to být také možnosti přístupu, které udržují hráče při hraní, jelikož přidávají variaci, pomocí které hráč musí přemýšlet nad svými akcemi.

* + 1. Úspěchy

Úspěchy jsou reprezentace dosáhnutí nějakého milníku, což může klidně být splnění úkolu, získání nové úrovně… Tyto úspěchy většinou hráče odmění, pomocí předmětů nebo schopností, díky kterým hráč dokáže postupovat hrou o něco lehčeji než doposud, což hráči dá euforický pocit. Tento pocit z úspěchu udržuje hráče při hraní a motivuje ho k pokračování.

* + 1. Pokrok

Pokroková technika nějakým způsobem zobrazuje hráči jeho pokrok ve hře. Může to být ve stylu úrovní, kde hráč začíná na úrovní 1 a postupem investovaného času, díky plnění úkolů se dokáže dostat až na úrovně 60. Úrovně nejsou jediná mechanika, co může zvizualovat pokrok, co hráč udělal, ale může to být například i vybavení co hráč má na sobě, kde postupem času získává honosnější vzhled…

<https://www.gamedeveloper.com/design/the-psychological-perspective-on-game-design>

|  |
| --- |
| 1. Praktická část |

1. Vývoj herní úrovně

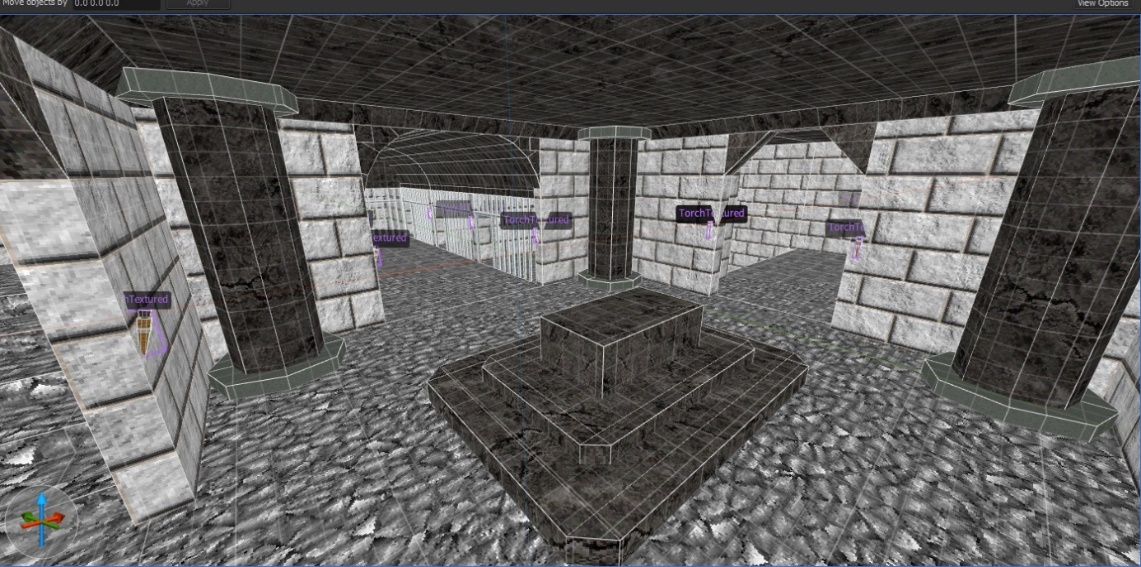
V nacházející kapitole budu popisovat, jaké nástroje jsem využil pro tvorbu herní úrovně. Dále zmíním postup tvorby různých modelů nacházejících se v úrovni společně s jejich implementací.

* 1. Trenchbroom

Celá úroveň je vytvořena pomocí open source nástroje Trenchbroom. Je zaměřen na tvorbu herních úrovní pro old school retro hry v Quake-Enginu. Ovšem poskytuje možnost vytvářet mapy na další hry jako Half-Life, Doom... Ačkoliv je nástroj specializovaný na hry s retro enginem je v něm možnost tvořit úrovně i pro další enginy. Zvolil jsem si tento nástroj oproti softwaru pro 3D grafiku z důvodu, že je na tvorbu herních úrovní přizpůsoben, což dělá celý vývoj rychlejší, jednoduší a intuitivní.

Celý 3D prostor v nástroji je rozdělen do linek horizontálních a vertikálních. Tyto linky utváří tvary čtverečků, na které lze pokládat co nástroj označuje jako entity. Entitu si lze představit jako jednu krychli nacházející se na čtverečku v 3D prostoru.

Vytvořené entity je možné různě upravovat. Tyto upraví mohou být velikostí, tvarové… Každá entita je rozdělena do vertexových bodů, díky kterým se otvírá nespočetní škála možností, co se s entitou dá vytvářet (tvorba různých tvarů z původní krychle). Program obsahuje několik nástrojů, které dělají úpravu entit snazší. Nejvíce jsem využil nástroj cut, díky kterému je možné různě rozřezávat entity na více dílů nebo nástroj pro pohyb vertexových bodů entit.



https://trenchbroom.github.io/

* 1. Osvětlení

Herní úroveň je osvětlena pomocí herního objektu point light. Tento objekt dokáže vygenerovat zdroj světla. Světlo lze v 3D prostoru zobrazit více způsoby, tyto způsoby jsou realtime, baked a mixed. Jako způsob vyobrazení světla jsem zvolil baked. Tento způsob, jak z názvu lze vyčíst „vpeče“ světlo na objekty nacházející se v jeho blízkosti. V úrovni se těchto světel nachází několik, což bylo odůvodnění zvolení tohohle způsobu, jelikož realtime vyobrazení by mohlo spočívat za menší snímky za sekundu při hraní hry.

Tyto světla jsou doprovázeny particle systémem, který generuje jednoduchý obrázek ohně. Díky konstantnímu replikování tohoto obrázku s náhodným generování jeho velikosti, dráhy pohybu a životnosti, dokážeme simulovat lowpoly, nenáročný oheň.

Díky spojení světla a simulaci ohně, dokážeme vytvořit louč. Model, který zastupuje louč je hlavním objektem, který má jako pod objekty již zmíněné světlo a particle systém. Tento model byl vytvořen i texturován v již zmíněném nástroji Trenchbroom.

Obsah obrázku PC hra, snímek obrazovky, Počítačová hra, text

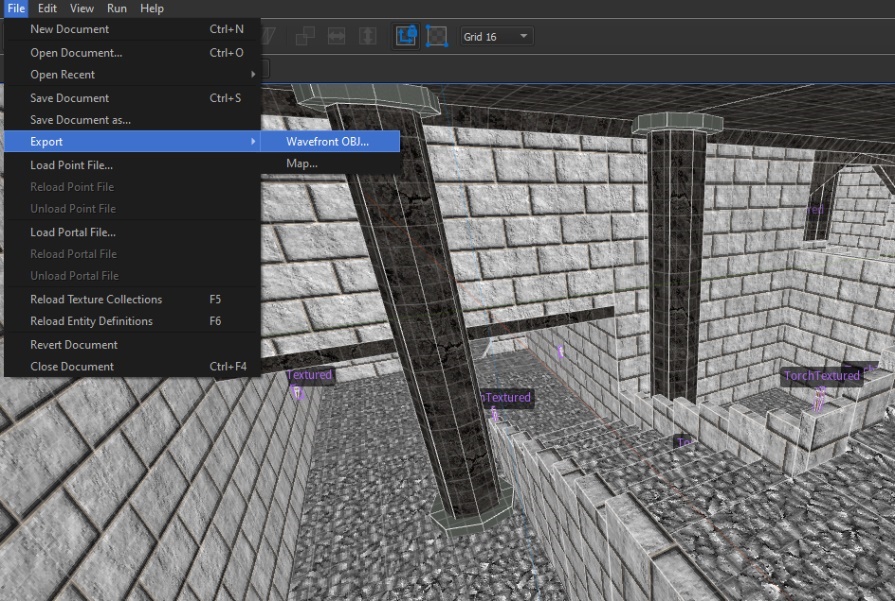
Popis byl vytvořen automaticky

* 1. Implementace úrovně

Po vytvoření úrovně v nástroji Trenchbroomu je třeba úrovně implementovat do Unity. Samotná implementace lze udělat několika způsoby. Já jsem zvolil možnost exportu úrovně do formát WaveFront. Nadále mohu vytvořenou mapu exportovat do programu Blender ve kterém lze udělat jakékoliv finalizační úpravy před importováním do Unity.

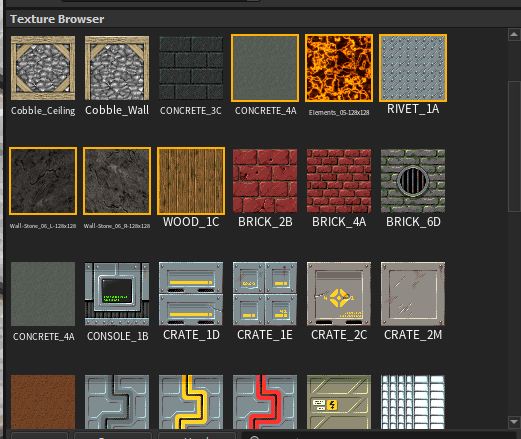
Poslední krok pro vložení úrovně do Unity je třeba úroveň znovu exportovat v Blenderu do formátu FBX. Tento formát již je podporován Unity oproti WaveFront, tudíž je možné importovat mapu do Unity.

Pokud byla importovaná mapa již předem texturována v nástroji trenchbroom a byla implementována zmíněným způsobem, textury budou také importovány s mapu. Ovšem, aby se textury aplikovaly, je třeba změnit způsob načítání textur na FBX souboru v Unity. Veškeré textury na úrovni budou importovány do vygenerovaného souboru. Tento soubor se vkládá do stejné složky jako importovaná úroveň. Aby se textury aplikovaly na model úrovně je třeba zapnout na FBX souboru úrovně v sekci materials možnost import via MaterialDescription.



* 1. Texturování

Implementace textur do Trenchbroomu funguje na principu vybrání složky obsahující textury a dále stačí pouze zvolit entity na které je potřeba dané textury aplikovat. Textury lze v programu různě upravovat. Takové upraví se pohybují od velikosti, šířky... až po hustotu opakování textury na entitě.



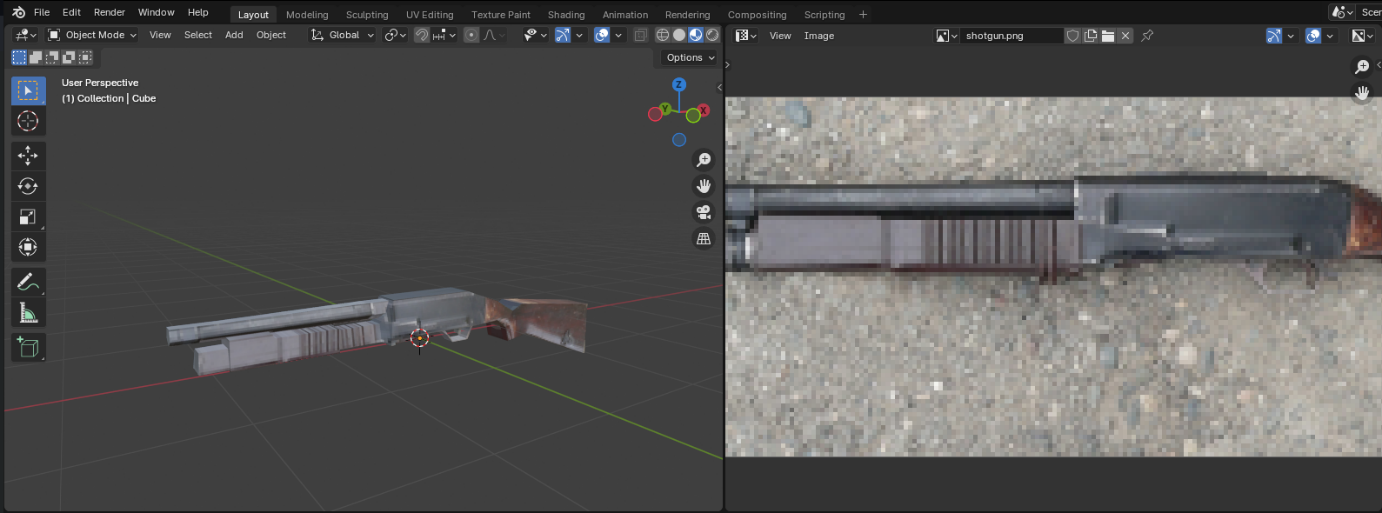
1. Vývoj herních modelů

V kapitole níže se dozvíte, jak jsem postupoval při modelování, texturování a samotnému implementování modelů do Unity

* 1. Modelování

Pro vývoj herních modelů jsem nepoužil již dříve zmíněný Trenchbroom, ale Blender. Odůvodnění tohohle rozhodnutí je takové, že ačkoliv je Trenchbroom lépe stavěn pro tvorbu herních úrovní je v něm za to komplikovanější tvořit určité modely, které potřebují moc specifický tvar a texturování.

Můj projekt Carnage je založen na retro žánru hry. Tyto hry se vyznačují jejich jednoduchým low poly modelováním a rozpixelovaným stylem. Proto pro správného dosáhnutí replikace pocitu hrání retro videohry, je třeba vytvářet objekty s co nejmenším počtem polygonů. Vzhled modelu lze popsat jako hrubý návrh modelu před „sculptingem“. Ovšem objekt se jinak více modelovat nebude a veškeré detaily jsou zobrazeny na samotné textuře modelu, proto pro dosáhnutí správného pocitu retro hry je důležité zvolit vhodné textury.

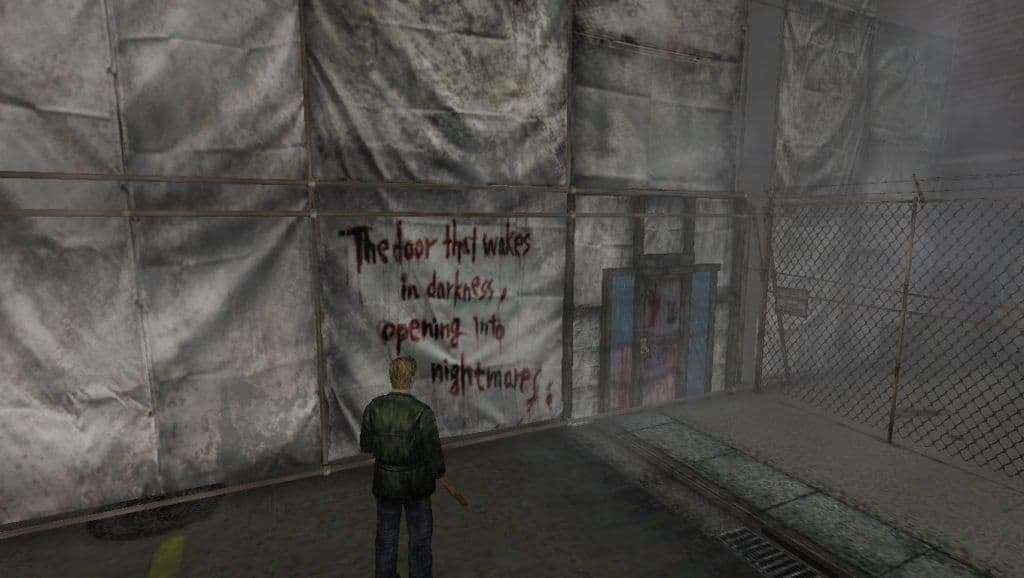


* 1. Texturování

Jak již bylo zmíněno je třeba pro replikování retro pocitu ze hry zvolit vhodné textury, které tohohle pocitu dokáží dosáhnout. Před aplikování textury jsem strávil relativně delší čas s výzkumem správné textury. Je důležité, aby vybraná textura seděla tvaru modelu. Není důležitý nejen tvar, ale i kvalita obrázku a celkový vzhled.

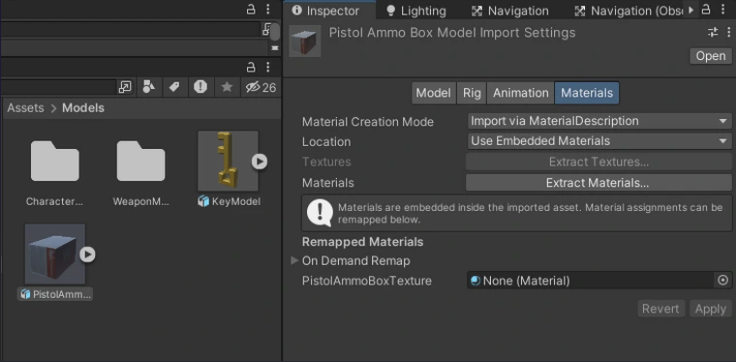
Při tomhle způsobu vývoje se setkáme s paradoxem, jelikož nehledáme nejkvalitnější 4K textury, ba naopak horší kvalitu obrázku. Odůvodnění tohohle rozhodnutí je takové, že při aplikování bude textura sedět žánru hry díky jejího rozpixelovanému vzhledu. Méně kvalitní obrázky nám také zajistí menší velikost a náročnost při renderování modelů do 3D prostoru.

Po správném výběru textury je samotné aplikování jednoduché. Potřebná textura je vložena do Blenderu jako nový materiál. Po vložení textury se stačí přepnout do módu „UV mapping“ a na celý model aplikovat texturu. Díky zvolenému módu vidíme všechny strany modelu. Dále jenom tyto strany upravíme pomocí posouvání jejich vertexových bodů, změny velikosti... než textura bude sedět na modelu.



* 1. Implementace modelů

Implementace herních modelů funguje na úplně stejném principu jako implementace herní úrovně. Tyto modely se exportují do formátu FBX, který Unity podporuje. Po vložení modelu do Unity projektu je třeba změnit způsob načítání textur nebo velikosti dle potřeby.

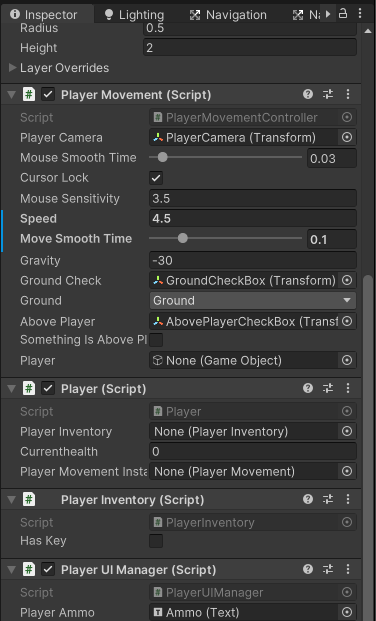


1. Vývoj hry

V kapitole níže se dozvíte, jak jsem vyřešil problematiku různých skriptů a jak vůbec skriptování funguje v Unity enginu.

* 1. Skriptování

Jak již bylo zmíněno v kapitole pro engine Unity skriptování v něm funguje na systému aplikování komponent. Tyto komponenty obsahující kód se přiřazují na herní objekty, se kterýma bude kód interagovat. Ovšem není to pravidlem, že komponent musí vždy být přiřazen na herní objekt, jelikož lze vytvořit skript, který se může čistě starat o tok dat…



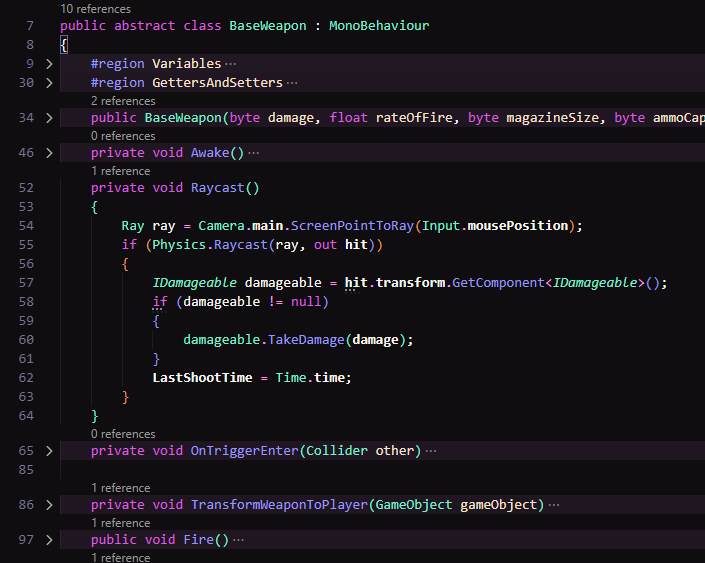
* + 1. Střelba hráče

Skript, který má za účel řešit střelbu hráče je přímo součástí abstraktní třídy pro zbraně. Tohle řešení umožní každé třídě pro další zbraně, které dědí z třídy abstraktní možnost využívat nejen funkci střílení, ale všechny její vlastnosti a funkce. Díky tomuhle se zutilizují veškeré hlavní pilíře OOP, což zajišťuje robustnost, čitelnost a menší redundanci kódu.

Jedna z těchto funkcí, která umožňuje sřelbu je metoda Raycast Tato metoda si definuje startovní pozice, z které neviditelný paprsek reprezentující trajektorii střely poletí a nadále jeho směr, který je reprezentován objektem na který se hráč díve. Tato pozice je určena jako pozice kamery hráče a směr je směr kde se kamera hráče díve. Vše je možno díky jedné funkci Unity Physics.Raycast. Metoda nám vrátí objekt, který byl zasáhnut. Poté se zkontroluje pomocí jednoduché podmínky, jestli objekt má vrstvu „Enemy“, kterou má každý nepřátel na mapě.

Pokud zasáhnutý objekt je skutečně nepřátel je mu přiřazeno poškození dle zbraně, kterou hráč aktuálně drží v ruce. Hráči se při zasáhnutí jakěhokoliv objektu odečte náboj.

Aby hráč nemohl spamovat tlačítko střelby je potřeba udělat výjimku. Tato výjimka bude mít na starost rychlost střelby, což zajistí, že hráč nebude střílet, tak rychle jak kliká. Rychlost střílení je provede pomocí jednoduché podmínky. Tahle podmínka má jako argument čas poslední střely. **DOPSAT POTOM**

****

* + 1. Pohyb hráče

Pohyb hráče je umožněn díky komponentě s názvem PlayerController, která od chvíle, co hráč incializuje úroveň bere jeho klávesoví vstup a dle klávesy provádí danou činnost. Tento způsob řešení je umožněn pomocí Unity funkce s názvem Update, do které se vstupuje každý snímek od inicializace projektu, pokud je povolen.

Tato funkce je jedna z funkcí, kterou lze získat pomocí zdědění třídy MonoBehaviour. Tato třída obsahuje funkce života programu, které zajišťují snadnější práci s Unity. Pokud je tato třída zděděna Unity počítá, že třída je jako komponent na herním objektu. Díky tomuhle můžeme přímo v kódu pracovat s objektem, na kterém se komponent nachází.

Aby bylo možné se pro hráče pohybovat je třeba zavolat metody, které zajišťují jeho pohyb. Tyto metody se volají na základě již zmíněného vstupu hráče. **(potom dopsat)**

Hodnoty, které jsou potřeba při výpočtu jako například rychlost chození nebo síla gravitace mají na sobě atribut serializable, který zajišťuje jednodušší práci pro vývojáře. Díky němu může editovat hodnoty přímo v Unity editoru a nemusí jít do kódu, aby hodnotu změnil.



* + 1. Nepřátelé

Každý nepřátel má svůj komponent, který mu přidává funkce jako jsou útok, pohyb, detekce hráče... Aby byl redukován počet redundance kódu, bylo použito abstrakce. Abstrakní třída s názvem BaseEnemy vytváří základ každého nepřítele, který se pomocí dědictví předá třídě pro specifického nepřítele.

Parametry, které určují různé statisticky nepřítele (rychlost chůze, síla útoku...), jsou přiřazeny v konstruktoru třidy pro nepřítele, který má již zděděn základ díky naší třídě BaseEnemy.

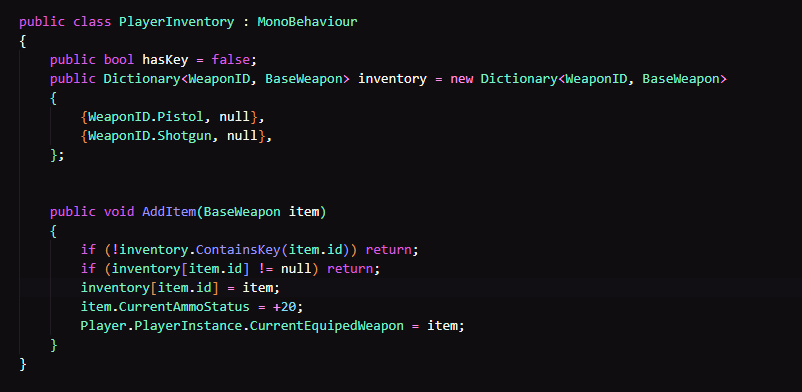
Nepřátelé fungují na jednoduchém principu. Každý z nich má svůj AggroRange, který určuje prostor, do kterého když hráč vejde, tak se aktivuje funkce, které nepřátele požene přímo k hráči, který vstoupil do prostoru. Od chvíle, co hráč vstoupí do prostoru, se také aktivuje skript, který zajistí, že se nepřátel pozoruje hráče.

Pohyb nepřátele je vytvořen díky umělé inteligence, která má přímo v herní úrovni „Vpečené“ cesty kam může chodit. Tato „vpečená“ cesta nejen zvolí plochu po které může chodit, ale také překážky, přes které chodit nemůže. Tuto umělou inteligenci lze získat pomocí nainstalování zdarma unity balíčku.



* + 1. Inventář

Hráčský inventář je vytvořen pomocí jedné dicionary, která má jako klíč danou hodnotu enum. Tento enum v sobě obsahuje několik názvů druhů zbraní. Díky tomuhle dokážeme udělat systém inventáře, kde před definujeme pozici, kterou zbraň bude mít v dictionary ještě něž jí hráč získá. Jako hodnota klíče je akce. Tato akce **(DOPSAT POTOM)**

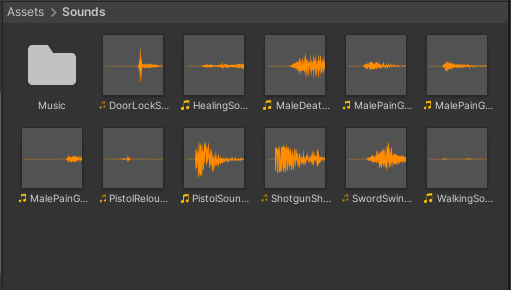
****

* 1. Zvuk

Nacházející kapitola obsahuje informace ohledně ukládání a přehrávání zvuků v Unity projektu.

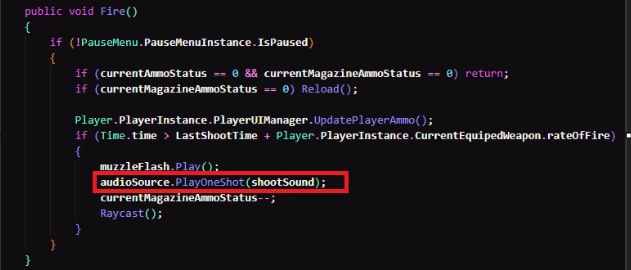
* + 1. Ukládání zvuků

Zvuky, které se přehrávají se jednoduše ukládají do souboru hry specificky určen pro zvuky. Ukládám je ve formátu MP3, ale je možné je ukládat i v jiných jako jsou například: WAV, AIFF nebo OGG.



* + 1. Přehrávání zvuků

Systém přehrávání zvuku je založen na uložení potřebného zvuku do třídy kde se bude využívat a nadále jeho přehrání při dané situaci. Aby bylo možné zvuk přehrát je potřeba mít audio source. **(DOPSAT POTOM)**

****

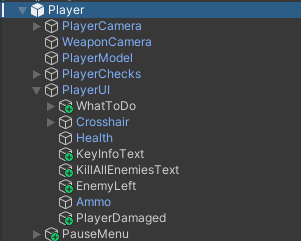
* 1. Uživatelské rozhraní

Kapitola zahrnuje mé řešení pro tvorbu uživatelského rozhraní hráče a menu.

* + 1. Rozhraní hráče

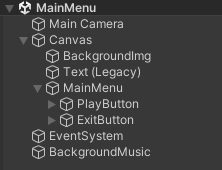
Hráčské rozhraní je vytvořeno pomocí herního objektu Canvas. Tento objekt je přímo vložen do prefabu hráče. Díky němu můžeme hráči na kameru zobrazovat různě texty, obrázky, tlačítka... Hráč může různě s těmito předměty interagovat a měnit jejich stav nebo mohou být čistě statické.

Na UI je přiřazen komponent PlayerUIManager, který obsahuje funkce, které aktualizují stav UI. Do těchto funkcí jsou vstupní parametry (Stav nábojů, poškození...), které jsou poté díky jednoduché kalkulaci přičteny nebo odečteny textu uživatelského rozhraní.



* + 1. Hlavní menu

Stejně jako hráčské rozhraní funguje na stejném principu hlavní menu nebo menu jiné. Ovšem, že hlavní menu je zakomponován do kompletně jiné scény než již zmíněné uživatelské rozhraní. Celé hlavní menu se tvoří pomocí hlavního objektu Canvas. Do tohohle objektu je vložen nadpis hry a tlačítka. Tyto tlačítka obsahují komponenty s jednoduchým skriptem. Např. při kliknutí tlačítka „New Game“, skript jenom načte scénu obsahující první úrovně.



* 1. Animace

Animace jsou obrovská část herního vývoje. Dávají objektům, postavám a různým akcím nádech života. Bez nich by postava nesprintovala ale jenom klouzala po obrazovce. V kapitole níže si vysvětlíme, jak animce vytvářet a aplikovat.

* + 1. Mixamo

Mixamo je webová platforma od firmy Adobe obsahují několik nástrojů vhodných pro vytváření a animování 3D postav. Mixamo poskytuje velkou škálu 3D modelů a animací, které mohou být zdarma použity.

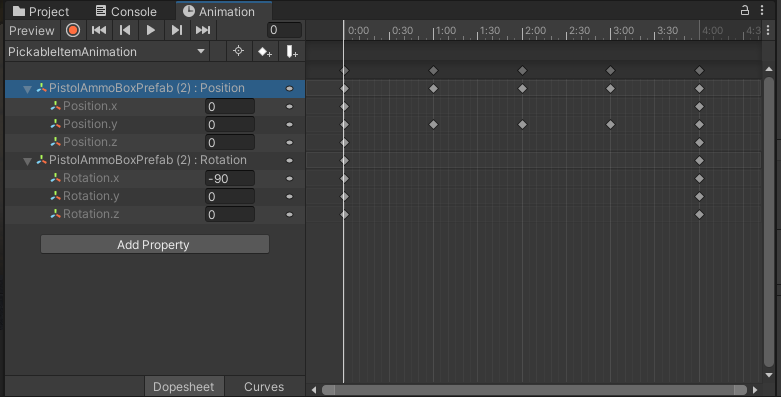
Využil jsem Mixamo, abych si vzal potřebné animace. Ovšem i přes to, že stránka obsahuje automatické riggování modelů postav v mém případě byl rig rozbitý, proto jsem si model postav riggoval sám v programu Blender. „Rigging“ znamená přidávání modelu postavy kostru, pomocí které lze s modelem interagovat jako kdyby měl reálnou kostru.

Animace lze na stránce upravovat. Mezi tyto úpravy patří zrychlení, zpomalení nebo dokonce i stříhání animace pro exportování pouze potřebné části.



* + 1. Aplikace animací

Vytváření a aplikování animací v Unity lze udělat několika způsoby. Jeden z těchto způsobů je pomocí animování časové osy, kde lze upravovat objekty v různých časových sekvencí, které vytvoří animaci. Tenhle způsob není úplně optimální, pokud chceme aplikovat komplexnější animace, jelikož by to bylo moc zdlouhavé a nedostalo by se k perfektnímu výsledku. Místo něho lze animace vytvářet v externích 3D softwarech a poté jen pomocí ovladače animací danou animaci aplikovat.



Na objekty, které je třeba animovat se musí vložil komponent animátor, aby bylo možné animace ovládat a vůbec je aplikovat. Pokud je objekt, na který chceme vložit animaci model postavy je třeba, aby byla tzv. „Rigged“. Avšak tento komponent vyžaduje referenci na kontrolér animací, který definuje, kdy se animace přehrají nebo jaké mají mezi sebou přechody (přepsat)

Závěr

Seznam použité literatury

|  |  |
| --- | --- |
| [1] | A tour of the C# language. Online. Learn.microsoft. 2023. Dostupné z: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>. [cit. 2023-12-14]. |
| [2] | Introduction to C#. Online. Geeksforgeeks. 2008. Dostupné z: <https://www.geeksforgeeks.org/introduction-to-c-sharp/>. [cit. 2023-12-14]. |
| [3] | OOP Meaning – What is Object-Oriented Programming? Online. Freecodecamp. 2022. Dostupné z: <https://www.freecodecamp.org/news/what-is-object-oriented-programming/>. [cit. 2023-12-14]. |
| [4] | What is Dynamic Programming? Working, Algorithms, and Examples. Online. Spiceworks. 2022. Dostupné z: <https://www.spiceworks.com/tech/devops/articles/what-is-dynamic-programming/>. [cit. 2023-12-14]. |
| [5] | PLANS AND PRICING. Online. Unity. 2023. Dostupné z: <https://unity.com/pricing>. [cit. 2023-12-14]. |
| [6] | Další dílo |
| [7] | Další dílo |
| [8] | Další dílo |
| [9] | Další dílo |
| [10] | Další dílo |
| [11] | Další dílo |
|  |  |

Seznam obrázků

[Obrázek 1 ukázkový obrázek [1] 10](#_Toc52904178)

[Obrázek 2 Nastavení titulku 11](#_Toc52904179)

Seznam tabulek

[Tabulka 1 ukázková tabulka 11](#_Toc52713261)

Seznam příloh

|  |  |
| --- | --- |
| Příloha I | Tištěné propagační materiály firmy Nokia |
| Příloha II | Diagram zapojení tištěného spoje |