|  |
| --- |
|  |
| Maturitní práce – **Informační technologie** |
| Implementace počítačové hry z pohledu první osoby |
| 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 Vyvlečková**

**Název práce: Implementace počítačové hry z pohledu první osoby**

**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**

Rád bych poděkoval vedoucí mé maturitní práce, paní Bc. Veronice Vyvlečkové, za vedení, pomoc, připomínky a čas strávený za zpracování maturitní práce.

Obsah

[Obsah 5](#_Toc161862870)

[Úvod DOPSAT 7](#_Toc161862871)

[I. Teoretická část 8](#_Toc161862872)

[1. Programovací jazyk c# 9](#_Toc161862873)

[1.1. Výhody vývoje v jazyce *C#* 9](#_Toc161862874)

[1.2. Základní informace 10](#_Toc161862875)

[1.2.1. Objektově orientované programování 10](#_Toc161862876)

[1.2.2. Přístupové modifikátory 13](#_Toc161862877)

[1.2.3. Objektově orientované designové principy 14](#_Toc161862878)

[2. Herní enginy 18](#_Toc161862879)

[2.1. Hlavní představitelé 18](#_Toc161862880)

[*2.1.1.* *Unreal Engine* 18](#_Toc161862881)

[*2.1.2.* *Unity Engine* 19](#_Toc161862882)

[3. Herní design 21](#_Toc161862883)

[3.1. Obecné informace 21](#_Toc161862884)

[3.2. Proces designování 21](#_Toc161862885)

[3.2.1. Výzkum a analýza 21](#_Toc161862886)

[3.2.2. Koncept 21](#_Toc161862887)

[3.2.3. Blocking 22](#_Toc161862888)

[3.2.4. Finalizace 23](#_Toc161862889)

[3.3. Využití psychologie při vytváření designu videoher 23](#_Toc161862890)

[3.3.1. Motivace 23](#_Toc161862891)

[3.3.2. Úspěchy 24](#_Toc161862892)

[3.3.3. Pokrok 24](#_Toc161862893)

[II. Praktická část 25](#_Toc161862894)

[4. Obecný přehled projektu 26](#_Toc161862895)

[4.1. Obecné informace 26](#_Toc161862896)

[4.2. Popis přílohy 26](#_Toc161862897)

[4.3. Vybraní pojmenování hry 26](#_Toc161862898)

[4.4. Plánování projektu 26](#_Toc161862899)

[5. Vývoj herní úrovně 27](#_Toc161862900)

[5.1. *Trenchbroom* 27](#_Toc161862901)

[5.2. Osvětlení 28](#_Toc161862902)

[5.3. Implementace úrovně 28](#_Toc161862903)

[5.4. Texturování 30](#_Toc161862904)

[6. Vývoj herních modelů 31](#_Toc161862905)

[6.1. Modelování 31](#_Toc161862906)

[6.2. Využití externích modelů DOPSAT 31](#_Toc161862907)

[6.3. Texturování 31](#_Toc161862908)

[6.4. Využití externích textur DOPSAT 32](#_Toc161862909)

[6.5. Implementace modelů 33](#_Toc161862910)

[7. Vývoj hry 34](#_Toc161862911)

[7.1. Skriptování 34](#_Toc161862912)

[7.1.1. Střelba hráče 34](#_Toc161862913)

[7.1.2. Pohyb hráče 36](#_Toc161862914)

[7.1.3. Nepřátelé 37](#_Toc161862915)

[7.1.4. Inventář 38](#_Toc161862916)

[7.2. Zvuk 39](#_Toc161862917)

[7.2.1. Ukládání zvuků 39](#_Toc161862918)

[7.2.2. Přehrávání zvuků 39](#_Toc161862919)

[7.3. Uživatelské rozhraní 40](#_Toc161862920)

[7.3.1. Rozhraní hráče 40](#_Toc161862921)

[7.3.2. Hlavní menu 41](#_Toc161862922)

[7.4. Animace 41](#_Toc161862923)

[7.4.1. *Mixamo* 41](#_Toc161862924)

[7.4.2. Aplikace animací 42](#_Toc161862925)

[Závěr DOPSAT 44](#_Toc161862926)

[Seznam použité literatury 45](#_Toc161862927)

[Seznam obrázků 49](#_Toc161862928)

[Seznam tabulek 50](#_Toc161862929)

[Seznam příloh 51](#_Toc161862930)

Úvod

Videohry dosahují stále rostoucí popularitu, přičemž umožňují hráčům zážitky, které jsou v rámci reálného světa nedosažitelné. S touhle popularitou neroste pouze zájem o samotné hry, ale i o jejich tvorbu. Taková tvorba her nejvíce láká samotné hráče, které je hrají, ovšem pro jejich tvorbu je třeba nejen herní znalosti, ale i velkých technických znalostí programování.

Cílem práce je implementace počítačové hry z pohledu první osoby. Teoretická část práce seznámí nejen s programovacím jazykem *C#*, společně s jeho důležitými funkcemi a principy, ale i herní enginy ve, kterých je tvorba her mnohem snažíš až po jejich samotný design a údržbu.

V praktické částí práce budou popsány postupy pro tvorbu vlastní hry od jejího začátku až po konečný výsledek společně s nástroji využité pro vývoj. Dále nás práce seznámí s problémy, s kterými se při vývoji lze setkat. Finální výstup bude hra, která obsahuje hráčský inventář, pohyb hráče, nepřátele, nastavení, ukládání pokroku a možnost postupovat do dalších úrovní. Dále bude ke hře vytvořen herní manuál obsahující hlavní informace, které se ke hře týkají jako jsou například: ovládání hry, instalace hry nebo popis menu, které hra obsahuje.

|  |
| --- |
| 1. Teoretická část |

1. Programovací jazyk c#

*C#* je moderní, vysoko úrovňový, objektově orientovaný programovací jazyk. 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 vysoké flexibilitě lze v jazyce vyvíjet nejen mobilní a desktopové aplikace, ale dokonce i například videohry pomocí *Unity enginu* nebo *Godotu*.[1]

Programovací jazyk byl nedesignován 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*.[1;2]



Obrázek 1 - C# logo[35]

https://en.wikipedia.org/wiki/C\_Sharp\_%28programming\_language%29

**V článku od *Muthu Kumara* jsou označeny jako populární aplikace vytvořené v jazyce C#:**

* *Microsoft Visual Studio*
* *Skype*
* *Photoshop*
* *Unity engine*
  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 *Pythonu*, i tak je ideální pro začátečníky i zkušené programátory, kteří se chtějí naučit nový jazyk.[4]

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.[4]

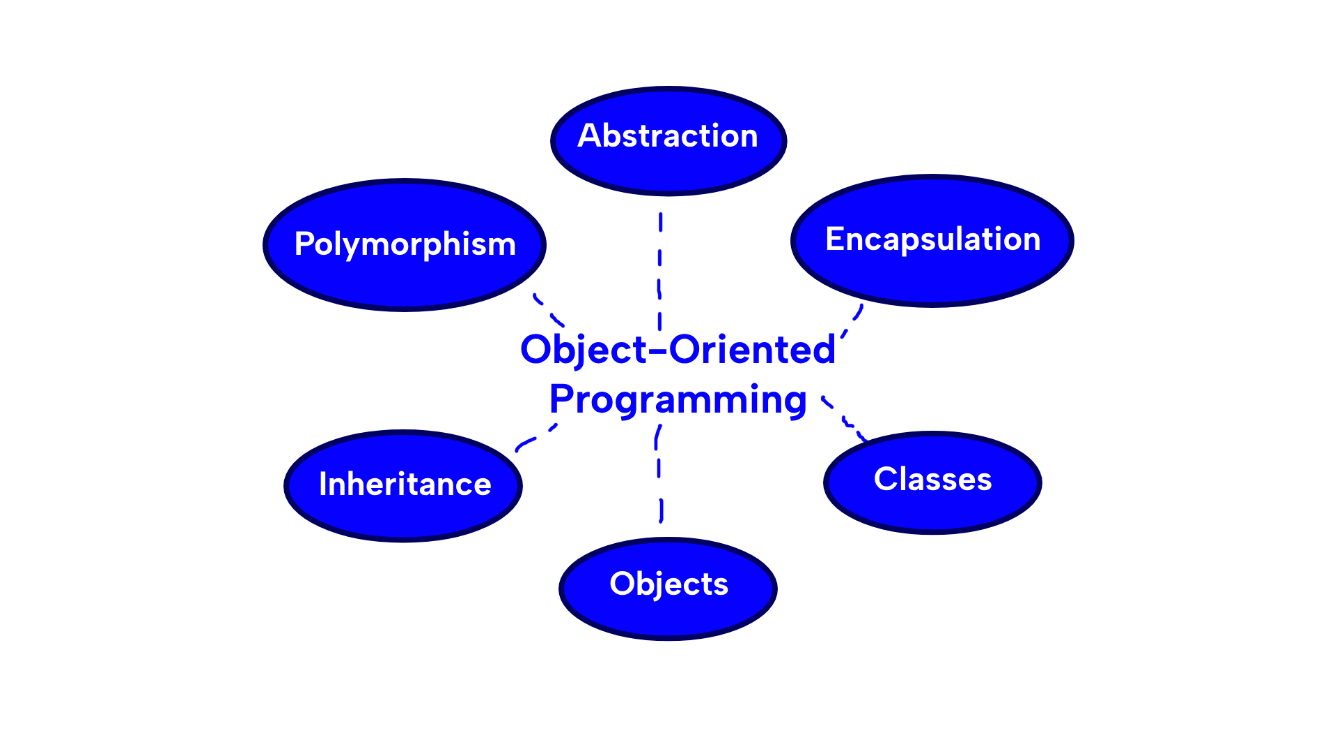
Není to nejen jeho design, který exceluje při stavění různých aplikací, *C#* má jednu z největších komunit na trhu, která umožňuje najít podporu a odpověď na různé otázky velice jednoduše oproti ne tak populárních jazyků.[4]

* 1. Základní informace

Nižší části budu poskytovat základní informace, které se jednají programovacího jazyka *C#*.

* + 1. Objektově orientované programování

Objektově orientované programování je imperativní programovací paradigma. Tohle paradigma rozdělí software na objekty a třídy, díky tomuhle se formuje kód softwaru více flexibilní, jednoduše rozšiřitelný a znovu použitelný, což eliminuje redundanci a zvyšuje čitelnost. Třídy reprezentují znovu použitelné šablony kódu a objekty jsou individuální instance těchto tříd. Další programovací jazyky, které využívají OOP jsou *C++*, *Java* nebo *Python*.[5]

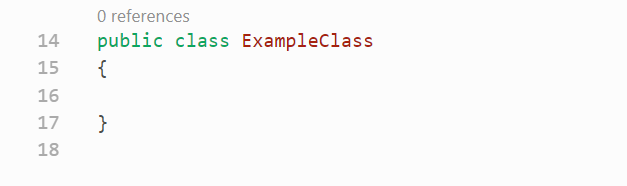


Obrázek 2 - Rozdělení OOP

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.[6]

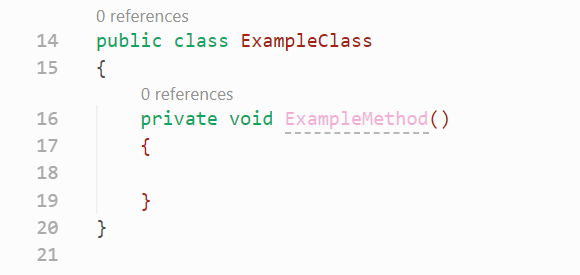


Obrázek 3 - Ukázka kódu třídy

Na obrázku se nachází příklad syntaxu pro vytvoření prázdné třídy s názvem *ExampleClass*.

**Metoda**

Metoda je část kódu udávající konkrétní chování nebo provedení různých akcí, které například mohou vracet informace o objektu nebo aktualizovat jeho data.6]



Obrázek 4 - Ukázka kódu metody

Na obrázku se nachází příklad již zmíněné třídy *ExampleClass* a prázdné metody *ExampleMethod*.

**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.[6]

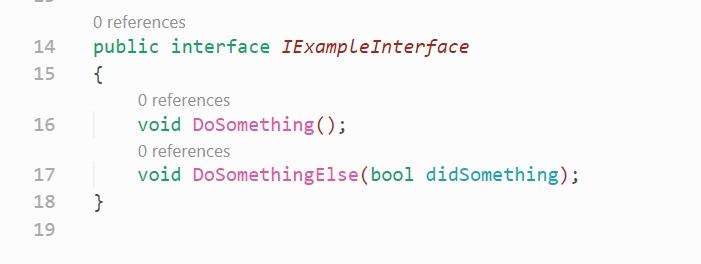


Obrázek 5 - Ukázka kódu objektu

Obrázek vyobrazuje způsob, jak se může vytvořit instance neboli objekt třídy *ExampleClass*.

**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.[6]



Obrázek 6 - Ukázka kódu rozhraní

Na obrázku se nachází jednoduché rozhraní udávající, že každá třída, do které se bude rozhraní dědit musí obsahovat metody *DoSomething* a *DoSomethingElse*.

* + 1. Přístupové modifikátory

Kapitola níže obsahuje informace ohledně přístupových modifikátorů v jazyce *C#*.[7]

Všechny členy v kódu mají svoji úroveň přístupnosti. Přístupnost ovládá, jestli mohou být použití v jiném kódu ve stejném assembly. Tato úroveň lze upravovat a měnit pomocí přístupových modifikátoru. Mezi tyto modifikátory patří:

**Public**

Člen s modifikátorem public, může být přistoupen z kamkoliv a jakýmkoliv kódem ve stejném projektu.

**Private**

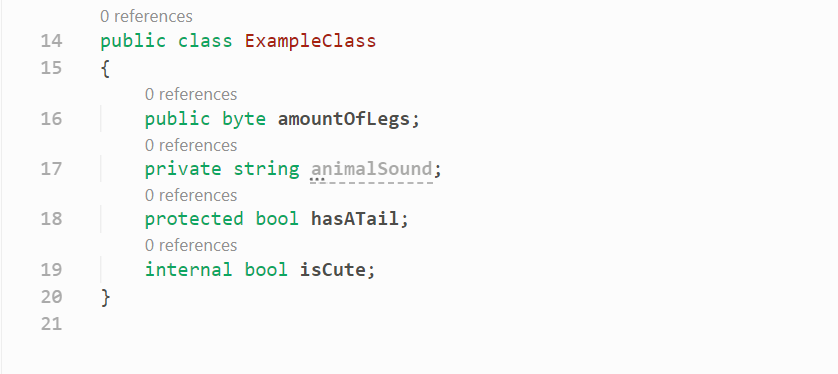
Člen s modifikátorem private může být přistoupen pouze v kódu ve stejné třídě nebo datové struktuře.

**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.



Obrázek 7 - Ukázka kódu přístupových modifikátorů

Obrázek vyobrazuje, jak se různé přístupové modifikátory mohou aplikovat na proměnné nacházejíce ve třídě *ExampleClass*.

* + 1. Objektově orientované designové principy

Tyto principy lze popsat jako hlavní pilíře objektového programování. Jsou to tedy stavební kameny, na kterých celé OOP funguje.

**Abstrakce**

Abstrakce je způsob tvoření metody a tříd. Z abstraktní třídy nelze být vytvořena instance, jelikož slouží jako šablona pro tříd, do kterých se dědí. Abstraktní metoda neobsahuje žádnou logiku a může být použita pouze v abstraktní třídě, jelikož se počítá, že třída, do které se metoda dědí jí předá logiku.[8]

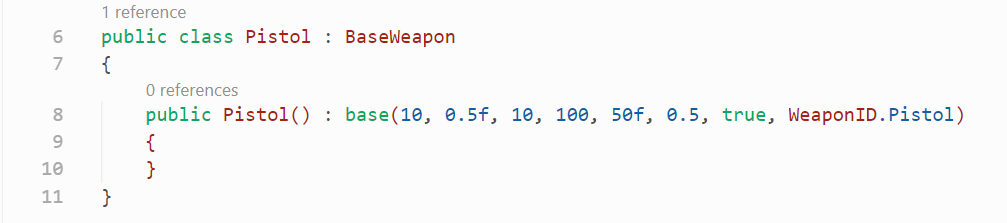


Obrázek 8 - Ukázka kódu abstrakce

Abstraktní třída *Animal* obsahuje veškeré základní vlastnosti a metody co budou potřebovat všechny třídy dědící z této třídy. Druhá třída *Dog* specifikuje dané zvíře a pomocí konstruktoru mu přidává individuální hodnoty do proměnných.

**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.[9]



Obrázek 9 - Ukázka kódu dědičnosti

Na obrázku se vyobrazuje třídu Pistol, která má veškeré její vlastnosti a metody zdědění z třídy *BaseWeapon*.

**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).[10]

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.[10]

**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.[10]

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

Obrázek 10 - Ukázka kódu polymorphismu

Obrázek ukazuje dvě třídy, na kterém se jedna zdědí do druhé, Díky tomuhle jevu druhá třída obsahuje veškeré vlastnosti a metody první třídy. I když druhá třída má zděděnou metodu *Something* je pomocí polymorphismu přetížena pro změnu výsledku.

**Zapouzdření**

Zapouzdření je technika pro implementaci abstrakce v kódu. Je tvořena pomocí tříd a jejich členy, které mají správně přiřazené přístupové modifikátory. Zapouzdření umožňuje uschovat nebo ukázat data a funkce jiným třídám, což přidává extra vrstvu zabezpečení.[11]



Obrázek 11 - Ukázka kódu zapouzdření

Obrázek vyobrazuje funkčnost zapouzdření, díky objektu instance, který je instancí třídy *ExampleClass*. Tahle instance má dvě proměnné, z kterých má jedna přístupový modifikátor *private* a druhá public. Metoda *DoSomething* ve druhé třídě se snaží přistoupit na proměnné instance, ale díky jejich modifikátorům je k jedné přístup odepřen a druhé ne.

1. Herní enginy

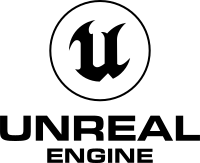
Herní engine je software, který je primárně navržen pro vývoj videoher. Vývojáři dokážou využít funkce těchto enginů, aby nemusely „začínat od nuly“ a vytvořit si vlastní engine, což jim ušetří čas a pracovní sílu pro vývoj hry. engine přináší vývojáři obrovský nespočet předem vytvořený funkcí a procedur. Příklad takových funkcí se může skládat z optimalizace projektu až po jednoduchého implementování projektu na více platforem.[12]

* 1. Hlavní představitelé

V textu níže popisuji dva nejvíce vývojáři používané představitele herních enginů.

* + 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.[13]



Obrázek 12 - Logo *Unreal* enginu[29]

Zpočátku UE byl navržen pro softwarové vykreslování, ale postupem času se začal zaměřovat na hardware akcelerování. [13]

Tvorba videoher není jediná věc, co UE dokáže. Je v něm možné dělat dokonce i simulace nebo vykreslovat animace. UE pracuje v jazyku *C++*, který zajišťuje jeho portabilitu na jiné platformy a stabilitu.[13]

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 a vlasntosti. 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.[14]

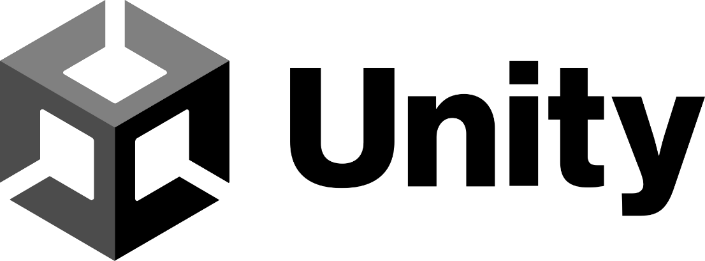
Příklady her, které byli vyvíjeni v UE: *Batman Arkham Knight, Fornite, Star Wars Jedi: Fallen Order*.[15]

Tabulka 1 - Výhody a nevýhody *Unreal enginu*[30]

|  |  |
| --- | --- |
| **Výhody** | **Nevýhody** |
| Výborná výkonnost při práci s komplexními prostředí | Komplexní uživatelské rozhraní a strmá křivka učení pro začátečníky |
| Možnost využití nejmodernějších vizuálů | Povinnost platit 5 % výdělku z produktu, pokud vydělá více než 1mil.$ |
| Výborná sada nástrojů pro implementaci na virtuální realitu nebo projektů pro více hráčů | Je potřeba velkých systémových požadavků při používání plného potencionálu enginu |

* + 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.[16]



Obrázek 13 - Logo *Unity enginu[31]*

Oproti UE *Unity* je psáno v jazyce *C#*, který je o něco přívětivější pro začínající programátory oproti *C++*, například díky svému intuitivnímu návrhu, který na uživatele působí jednodušším a přívětivým učením.[17]

*Unity* umožňuje vývoj videoher nejen pouze v 2D, ale i v 3D prostoru. Tyto hry lze spouštět na obrovské škále platforem, kterou *Unity* engine pokrývá. Mezi tyto platformy patří: Virtuální realita, desktop, mobilní zařízení i konzole.[17]

Hlavní stavební blok 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 jsou využívány jako kontejnery pro komponenty, které implementují funkcionalitu.[18]

Příklady her, které byly vyvíjeny v *Unity*: *Among Us, Ori and the Will of the Wisps, Cuphead, Beat Saber*.[19]

Tabulka 2 - Výhody a nevýhody *Unity* enginu[32]

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

1. Herní design

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

* 1. Obecné informace

Herní design může být považována jako plánovací část celého projektu. Žádná videohra není vytvořená bez plánu a herní design je proces tvoření tohohle plánu. Je potřeba, aby při tomhle procesu návrháři využili velké kreativity a technického myšlení, aby byl plán co nejkvalitněji zpracován pro ulehčení práce vývojářům, kteří budou hru přímo vyvíjet.[20]

* 1. Proces designování

Text nacházející se níže obsahuje krátké vysvětlení, jaký je proces při navrhování videoher.

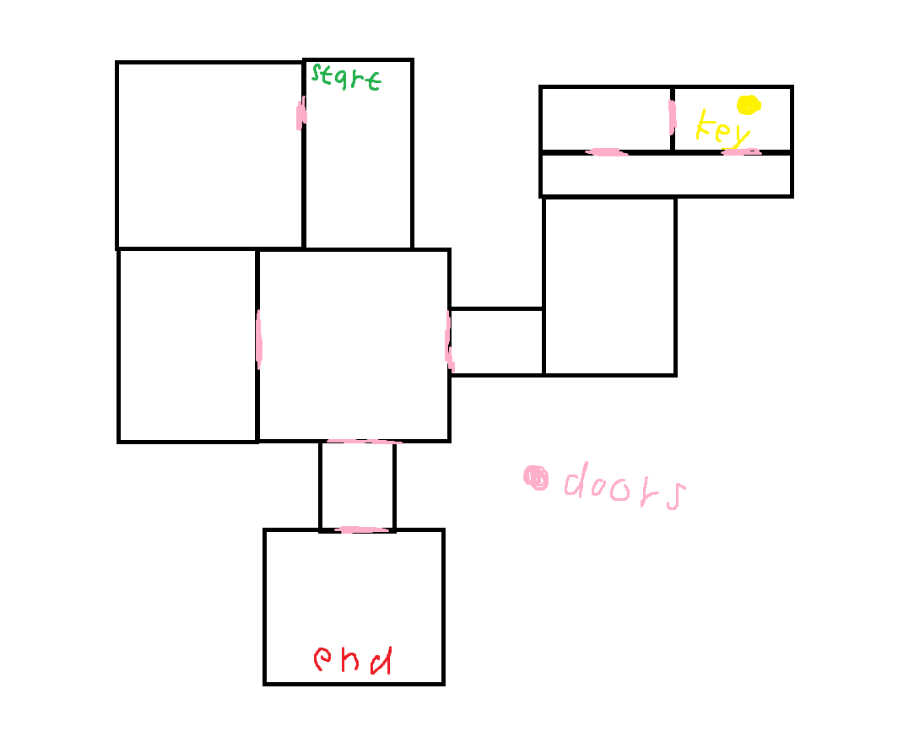
* + 1. Výzkum a analýza

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 nebo jaké tempo by měla hráči dávat..[21]

* + 1. Koncept

Poslední krok před samotným návrhem je koncept, který nám předá čistou představu ohledně tématu a základní kostry hry. 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.[21]

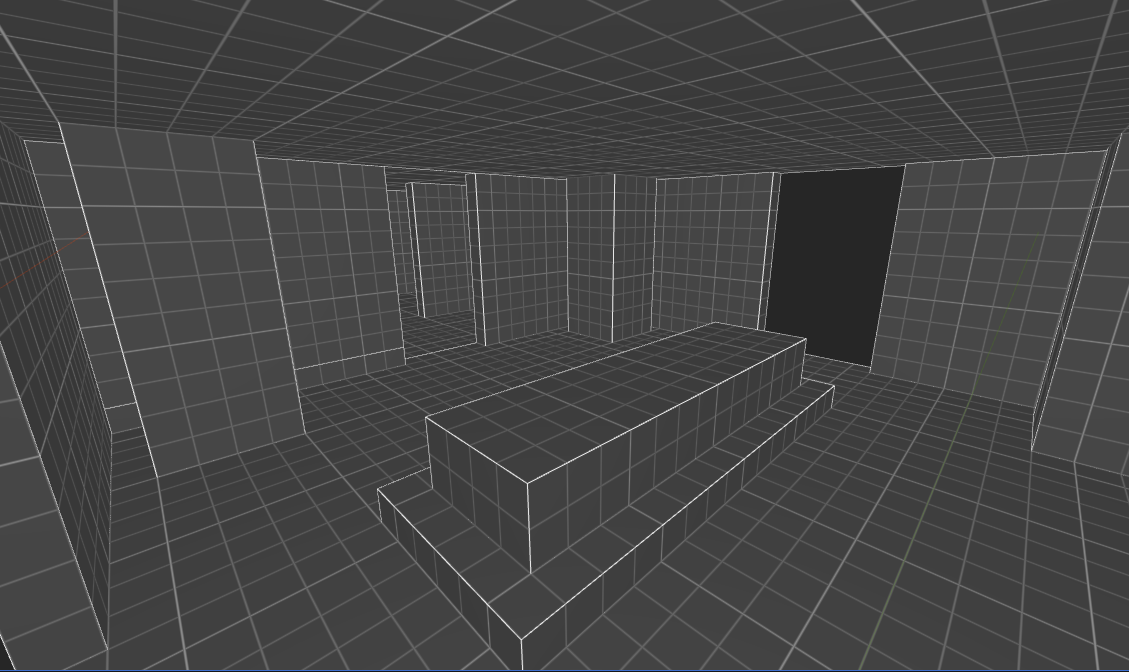
Koncept je hlavní část pro designery a vývojáře, která jim pomáhá při finálním designu a rozhodování co by se mohlo vylepšit, ba naopak přidat na základě konceptu. Díky němu si lze představit různé herní mechaniky, atmosféru atd.[21]



Obrázek 14 - Koncept herní úrovně

* + 1. Blocking

Blocking je fáze, kde se vytváří zjednodušená verze hry pomocí primitivních tvarů. Tahle 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 průtok hry před finalizací. [21]



Obrázek 15 - Blokování herní úrovně

* + 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. [21]

Obsah obrázku snímek obrazovky, mozaika

Popis byl vytvořen automaticky

Obrázek 16 - Finalizace herní úrovně

* 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.[22]

* + 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.[22]

* + 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.[22]

* + 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 vizualizovat pokrok, co hráč udělal, ale může to být například i vybavení co hráč má na sobě, kde postupem času získává honosnější vzhled, silnější vybavení nebo nové schopnosti.[22]

|  |
| --- |
| 1. Praktická část |

1. Obecný přehled projektu

V kapitole níže budu popisovat krátký úvod do projektu.

* 1. Obecné informace

*Carnage* je 3D FPS 1st person intenzivní střílečka, které vizuálně reprezentuje retro videohry. Celé premisa hry je, že hráč bojuje ve velké nevýhodě proti obrovskému množství nepřátel a snaží se přežít. Hra je inspirována hrami jako jsou *Doom, Dusk a Ultrakill*. *Carnage* obsahuje prvky hororu i akce.

* 1. Popis přílohy

Složka hry obsahuje herní manuál, který obsahuje veškeré potřebné informace ohledně hry. Tyhle informace se týčí nejen ovládání hry, ale i jak se vůbec hra instaluje, spustí nebo co to vůbec *Carnage* je.

Dále obsahuje spustitelný soubor, který spustí samotnou hru. Tento soubor lze identifikovat pomocí ikony hry, koncovkou .exe nebo názvem *Carnage*.

* 1. Vybraní pojmenování hry

Potřeboval jsem, aby se z názvu hry dalo vyčíst jaký o jaký druh hry se jedná, jako to je u videohry *Doom*. Tedy čistě z názvu hry by mělo hráči dojít, že jde o hru na téma: smrt, vražda a temnota.

Při vyhledávání jména jsem využil anglického online slovníku, do kterého jsem zadával různé již zmíněná témata a díval jsem se na různé synonyma a archaické významy. Dále jsem kontroloval, jestli již neexistují známe videohry se stejným názvem a aby byl název přitažlivý pro hráče.

* 1. Plánování projektu

Pro co nejvíce efektivní práci na projektu, bylo třeba si udělat pečliví plán. Toho jsem dosáhnul pomocí rozdělení projektu na několik částí, které jsem postupně plnil. Takové části se skládali z: vybraní tématiky, vývoj hlavních funkcí, tvorba modelů a herní úrovně a finalizace.

1. Vývoj herní úrovní

V nadcházející 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 herní úroveň je vytvořena pomocí open source nástroje *Trenchbroom*. Je zaměřen na tvorbu herních úrovní pro retro hry v *Quake-Enginu*. Ačkoliv je nástroj specializovaný na hry s retro enginem je v něm možnost tvořit úrovně i pro další enginy.[23]

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.[23]

Vytvořené entity je možné různě upravovat. Tyto upraví mohou být velikostní, tvarové i délkové. Každá entita je rozdělena na vertexové body, 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.[23]

Obsah obrázku nádoba, krabice

Popis byl vytvořen automaticky

Obrázek 17 - Logo nástroje *Trenchbroom*[23]

* 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.[24]

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 system*.

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

Popis byl vytvořen automaticky

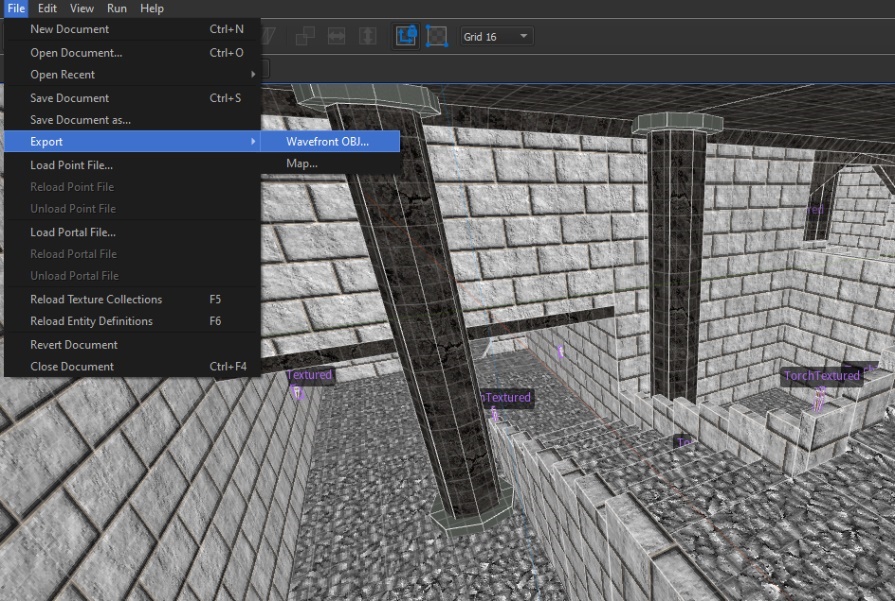
Obrázek 18 - Osvětlení v *Unity* prostředí

* 1. Implementace úrovní

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*.

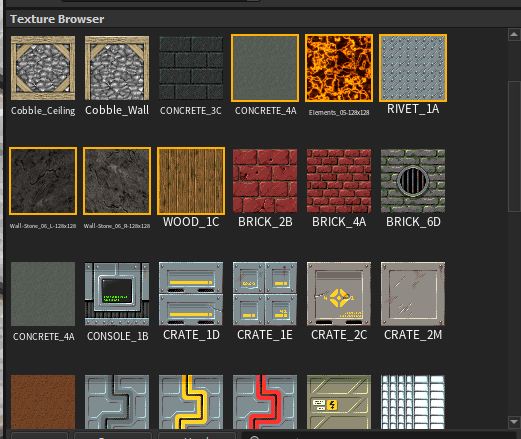
Pokud byla importovaná mapa již předem texturována v nástroji *Trenchbroom* a byla implementována zmíněným způsobem, textury budou také importovány s mapu. Ovšem, aby se textury aplikovaly, je třeba změnit způsob načítání textur na *FBX* souboru v *Unity*. Veškeré textury na úrovni budou importovány 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*.



Obrázek 19 - Exportování herní úrovně v nástroji *Trenchbroom*

* 1. Texturování herních úrovní

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 nebo šířky až po hustotu opakování textury na entitě.



Obrázek 20 - Složka textur v nástroji *Trenchbroom*

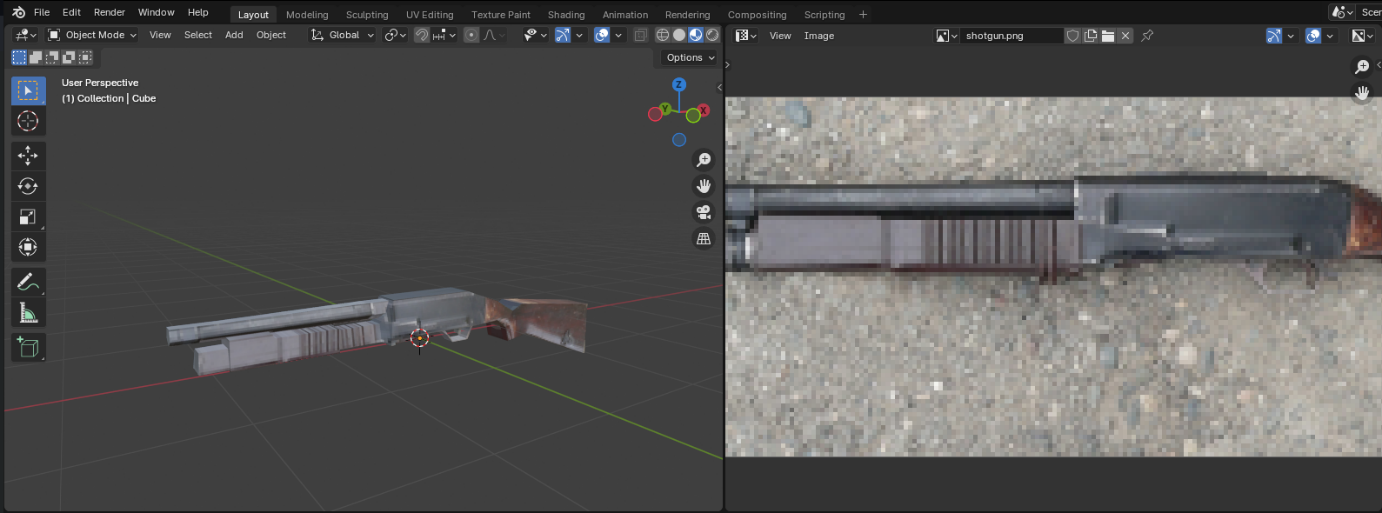
1. Vývoj herních modelů

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.

LowPoly hry se vyznačují jejich jednoduchým modelováním a „pixel artem“. 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.



Obrázek 21 - Herní model zbraně v nástroji *Blender*

* 1. Využití externích modelů DOPSAT
  2. Texturování modelů

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.

S takovým způsobem 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 „pixel art“ vzhledu. Méně kvalitní obrázky nám také zajistí menší velikost a náročnost při renderování modelů do 3D prostoru.

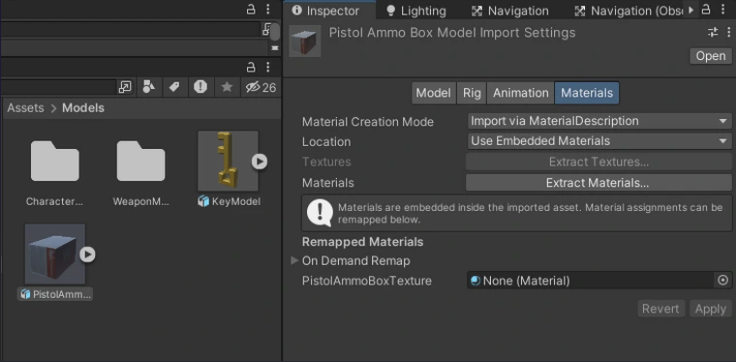
Po správném výběru textury je samotné aplikování jednoduché. Potřebná textura je vložena do *Blenderu* jako nový materiál. Po vložení textury se stačí přepnout do módu *UV* *mapping* a na celý model aplikovat texturu. Díky zvolenému módu vidíme všechny strany modelu. Dále jenom tyto strany upravíme pomocí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



Obrázek 22 - Retro videoherní textury[33]

* 1. Využití externích textur DOPSAT
  2. 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.



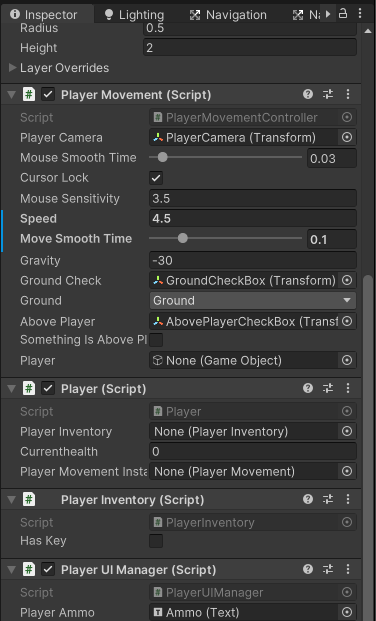
Obrázek 23 - Implementace textur na importované modely v *Unity*

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.[25]



Obrázek 24 - Využití komponent v *Unity*

* + 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.

Funkce *Fire*, které inicializuje střelu pouze kontroluje, jestliže hráč má dostatečný počet nábojů v zásobníku, aby mohl vystřelit a zda čas posledního výstřelu je větší než čas rychlosti střílení zbraně. Pokud vše projde v pořádku odečte se náboj ze zásobníku, přehraje zvuk střely s animací a ke konci spustí metodu řešící, zda nepřítel byl střelen.

Zmíněna metoda, která má za úkol zjistit, jestli byl trefen nepřítel a odebrat mu životy se nazývá *Raycast*. Tahle metoda si pomocí matematického postupu vypočítá neviditelný paprsek, který poletí z hráčské kamery. Díky tomuhle dokážeme zjistit první herní objekt, který paprsek trefil. Jediné, co metoda nadále potřebuje zjistit je, jestli daný objekt má na sobě komponent *iDamagable*. Tento komponent je interface, který se nachází na každém nepříteli, tudíž zjišťujeme, jestli zasáhnutý objekt je nepřátel. Pokud je tahle podmínka pravda nepříteli se odečte počet životů dle poškození, které zbraň dělá.

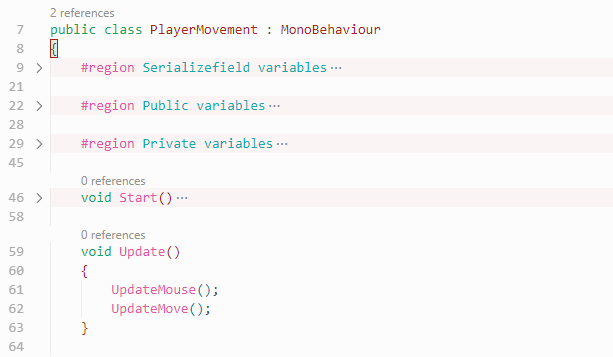
****

Obrázek 25 - Ukázka kódu pro střelbu hráče

Ovšem, aby bylo možné tyto metody inicializovat, je třeba, aby hráč zmáčknul dané tlačítko pro střelbu a aby měl aktuálně v ruce zbraň. To vše má nestarosti komponent *PlayerWeaponManager*, který je přiřazen na objektu hráče.

* + 1. Pohyb hráče

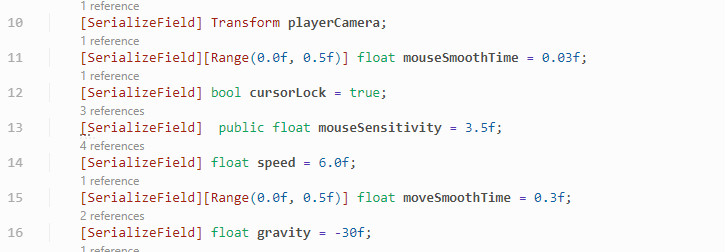
Pohyb hráče je umožněn díky komponentě s názvem *PlayerController*, která od chvíle, co hráč inicializuje úroveň bere jeho klávesoví i myšoví vstup. Metoda Update obsahuje metody řešící zpracování dat hráčského vstupu, jelikož se metoda spouští na každém snímku od doby inicializace scény.



Obrázek 26 - Ukázka kódu pro pohyb hráče

Metoda Update je jedna z metod, které lze získat pomocí zdědění třídy *MonoBehaviour*. Tato třída obsahuje metody, které se spouští v různých částí života objektu, které zajišťují snadnější práci s *Unity*. Další metody, které třída obsahuje jsou: *Start, OnDisable, OnEnable, LastUpdate*.

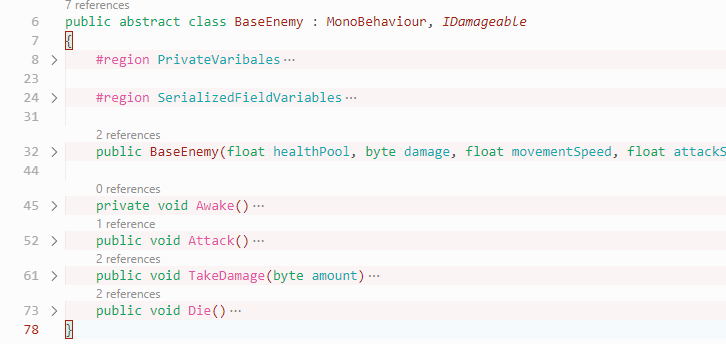
Hodnoty, které jsou potřeba při výpočtu jako například rychlost chození nebo gravitace mají na sobě atribut *SerializeField*. Díky atributu lze editovat hodnoty, které proměnná, je reprezentuje přímo v *Unity* editoru. To umožňuje rychlejší práci při testování.



Obrázek 27 - Ukázka proměnných využité pro pohyb hráče

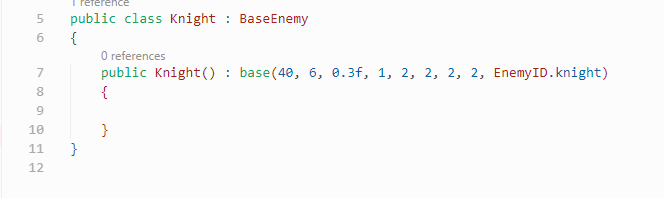
* + 1. Nepřátelé

Kód nepřátel je opět postaven na hlavních pilířích OOP (abstrakce, dědičnost…), stejně jako již bylo využito u třídy pro zbraně. Tudíž jsem vytvořil abstraktní třídu s názvem *BaseEnemy*, která obsahuje funkce a vlastnosti, jež budou mít všichni nepřátele. Mezi tyto funkce patří metody pro útok, získání poškození a smrt…



Obrázek 28 - Ukázka kódu abstraktní třídy pro nepřátele

Parametry, které určují různé vlastnosti specifických nepřátel jako: rychlost chůze, síla útoku. Tyto proměnné jsou přiřazeny v konstruktoru třidy, který má již zděděn základ díky abstraktní třídě *BaseEnemy*.



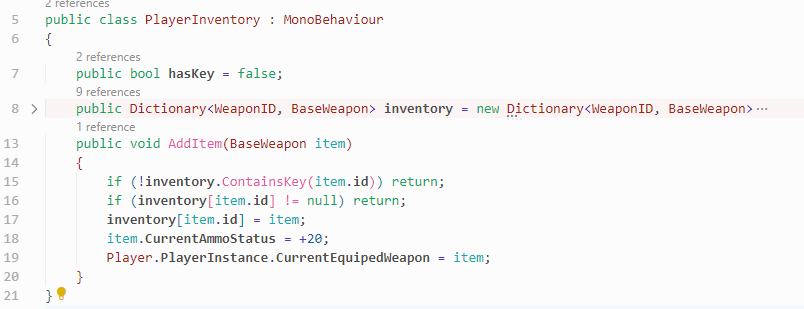
Obrázek 29 - Ukázka kódu individuálního nepřátele

Nepřátelé fungují na jednoduchém principu. Každý z nich má svůj rozsah, který určuje prostor, do kterého když hráč vstoupí, tak se aktivuje funkce, které nepřátele požene přímo k hráči.

Pohyb nepřátele je vytvořen díky umělé inteligence, která má přímo v herní úrovni baked 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 *dictionary*, která má jako klíč hodnotu *enum*. Tento *enum* v sobě obsahuje názvy druhů zbraní a jako hodnotu má třídu zbraní. Díky tomuhle dokážeme udělat systém inventáře, kde před definujeme pozici, kterou zbraň bude mít v *dictionary* ještě něž jí hráč získá.

****

Obrázek 30 - Ukázka kódu pro inventář hráče

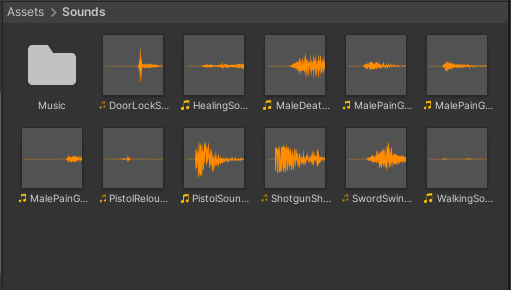
Aby bylo možné přidávat zbraně do *dictionary*, vytvořil jsem metodu, která má jako vstupní parametr proměnou abstraktní třídu zbraní. Metoda zkontroluje, jestli zbraň již je v inventáři a pokud ne, tak bude do něj přidána společně s její nastavení jako aktuálně používána zbraň hráče.

* 1. Zvuk

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

* + 1. Ukládání zvuků

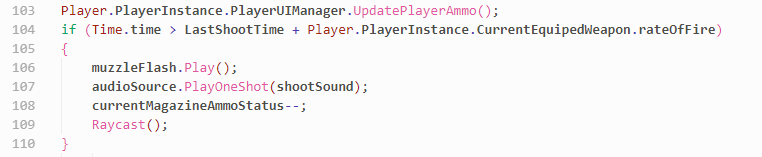
Zvuky, které se přehrávají ve scénách se ukládají do souboru hry specificky určen pro zvuky. Ukládám je ve formátu *MP3*, ale je možné je ukládat i v jiných jako jsou například: *WAV*, *AIFF* nebo *OGG*.[26]



Obrázek 31 - Souboru určen pro herní zvuky a hudbu

* + 1. Přehrávání zvuků

Systém přehrávání zvuků je založen na uložení potřebného zvuku a audio zdrojů do třídy kde se bude využívat. Aby bylo možné zvuk přehrát je potřeba zvuk přehrát z daného audio zdroje. Zvuk přehrávám pomocí unity metody *PlayOneShot*, která zvuk přehraje pouze jednou.

****

Obrázek 32 - Ukázka kódu pro přehrávání herních zvuků

* 1. Uživatelské rozhraní

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

* + 1. Rozhraní hráče

Hráčské rozhraní je vytvořeno pomocí herního objektu *Canvas*. Tento objekt je přímo vložen do objektu hráče. Díky němu můžeme hráči na kameru zobrazovat různě texty, obrázky, tlačítka.[27]

Obsah obrázku text, snímek obrazovky, software, Multimediální software

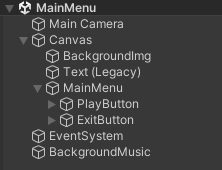
Popis byl vytvořen automaticky

Obrázek 33 - Rozložení rozhraní hráče

Na UI je přiřazen komponent *PlayerUIManager*, který obsahuje metody aktualizují stav UI. V těchto metodách se používají proměnné, které mají pomocí atributu *SerializedField* danou hodnotu, která je rovna hodnotě UI komponent.

* + 1. Hlavní menu

Stejně jako hráčské rozhraní funguje na stejném principu hlavní menu nebo každé jiné menu ve hře. Narozdíl od uživatelského rozhraní je hlavní menu zakomponován do kompletně jiné scény. 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.



Obrázek 34 - Rozložení scény hlavního menu

* 1. Animace

Animace jsou obrovská část herního vývoje. Dávají objektům, postavám a různým akcím nádech života. Bez nich by postava nesprintovala, ale jenom pouze klouzala po zemi. V kapitole níže si vysvětlíme, jak jsem animace vytvářel a aplikoval.

* + 1. *Mixamo*

*Mixamo* je webová platforma od firmy *Adobe* obsahují několik nástrojů vhodných pro animování 3D modelů postav. *Mixamo* poskytuje velkou škálu 3D modelů a animací, které mohou být zdarma použity.[28]

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*.[28]

„Rigging“ znamená přidávání modelu postavy kostru, pomocí které lze s modelem interagovat jako kdyby měl reálnou kostru.[34]

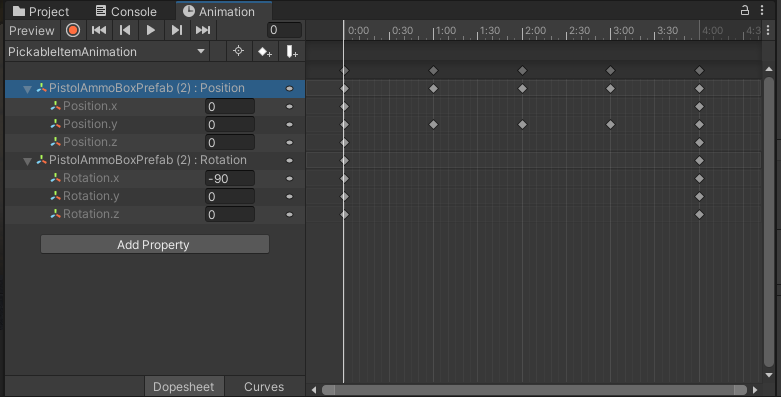
Animace lze na stránce upravovat. Mezi tyto úpravy patří zrychlení, zpomalení nebo dokonce i stříhání animace pro exportování pouze potřebné části.[28]



Obrázek 35 - List animací na stránce *Mixamo*[28]

* + 1. Aplikace animací

Vytváření a aplikování animací v *Unity* lze udělat několika způsoby. Jeden z těchto způsobů je pomocí animování časové osy, kde je možno 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.



Obrázek 36 - Způsob aplikace animací v *Unity*

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.

Závěr DOPSAT

Seznam použité literatury

|  |  |
| --- | --- |
| [1] | What is C# Programming? A Beginner's Guide. Online. Pluralsight. C2004-2024. Dostupné z: <https://www.pluralsight.com/blog/software-development/everything-you-need-to-know-about-c->. [cit. 2024-03-19]. |
| [2] | A tour of the C# language. Online. Microsoft Learn. C2024. Dostupné z: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>. [cit. 2024-03-19]. |
| [3] | Introduction to C#. Online. Geeksforgeeks. C2009. Dostupné z: <https://www.geeksforgeeks.org/introduction-to-c-sharp/>. [cit. 2024-03-19]. |
| [4] | Benefits of C#. Online. Codeguru. 2021. Dostupné z: <https://www.codeguru.com/csharp/benefits-of-c/>. [cit. 2024-03-19]. |
| [5] | What exactly is a programming paradigm? Online. Freecodecamp. 2019. Dostupné z: <https://www.freecodecamp.org/news/what-exactly-is-a-programming-paradigm/>. [cit. 2024-03-19]. |
| [6] | OOP Concepts in C#: Code Examples and How to Create a Class. Online. Stackify. 2024. Dostupné z: <https://stackify.com/oop-concepts-c-sharp/>. [cit. 2024-03-19]. |
| [7] | Access Modifiers (C# Programming Guide). Online. Microsoft Learn. C2024. Dostupné z: <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers>. [cit. 2024-03-19]. |
| [8] | Abstraction in Object-oriented Programming. Online. Tutorialsteacher. C2024. Dostupné z: <https://www.tutorialsteacher.com/csharp/abstraction>. [cit. 2024-03-19]. |
| [9] | Inheritance in C#. Online. Tutorialsteacher. C2024. Dostupné z: <https://www.tutorialsteacher.com/csharp/inheritance>. [cit. 2024-03-19]. |
| [10] | Polymorphism. Tutorialsteacher [online]. c2024 [cit. 2024-03-19]. Dostupné z: https://www.tutorialsteacher.com/csharp/polymorphism |
|  |  |
| [11] | Encapsulation in Object-oriented Programming. Online. Tutorialsteacher. C2024. Dostupné z: <https://www.tutorialsteacher.com/csharp/encapsulation>. [cit. 2024-03-19]. |
| [12] | Game Engines: All You Need to Know. Online. Gameopedia. C2023. Dostupné z: <https://www.gameopedia.com/game-engines-all-you-need-to-know-about/>. [cit. 2024-03-19]. |
| [13] | What Is Unreal Engine? Online. Bairesdev. C2009-2024. Dostupné z: <https://www.bairesdev.com/blog/what-is-unreal-engine/>. [cit. 2024-03-19]. |
| [14] | Introduction to Blueprints. Online. Epic Games Dev. C2004-2024. Dostupné z: <https://docs.unrealengine.com/5.0/en-US/introduction-to-blueprints-visual-scripting-in-unreal-engine/>. [cit. 2024-03-19]. |
| [15] | Top 10 Games Made With Unreal Engine. Online. Vagon. C2019. Dostupné z: <https://vagon.io/blog/top-10-games-made-with-unreal-engine/>. [cit. 2024-03-19]. |
| [16] | Unity: Development History and the Influence of This Game Engine on the Game Development Industry. Online. Medium. 2023. Dostupné z: <https://medium.com/@wota_mmorpg/unity-development-history-and-the-influence-of-this-game-engine-on-the-game-development-36dc7a7a3b9d>. [cit. 2024-03-19]. |
| [17] | Unity vs. Unreal: Which Game Engine is Best For You? Online. Udemy. C2024. Dostupné z: <https://blog.udemy.com/unity-vs-unreal-which-game-engine-is-best-for-you/>. [cit. 2024-03-19]. |
| [18] | GameObjects. Online. Unity User Manual. 2024. Dostupné z: <https://docs.unity3d.com/Manual/GameObjects.html>. [cit. 2024-03-19]. |
| [19] | Top Games Made with Unity: Unity Game Programming. Online. Create-learn. 2022. Dostupné z: <https://www.create-learn.us/blog/top-games-made-with-unity/>. [cit. 2024-03-19]. |
| [20] | What is Game Design? Critical Skills for Video Games. Online. Gamedevacademy. C2024. Dostupné z: <https://gamedevacademy.org/what-is-game-design/>. [cit. 2024-03-19]. |
| [21] | Comprehensive Guide to Game Level Design: From Architects to Adventure. Online. 300mind. 2023. Dostupné z: <https://300mind.studio/blog/game-level-design-guide/>. [cit. 2024-03-19]. |
| [22] | The Psychological Perspective on Game Design. Online. Gamedeveloper. 2019. Dostupné z: <https://www.gamedeveloper.com/design/the-psychological-perspective-on-game-design>. [cit. 2024-03-19]. |
| [23] | TrenchBroom. Online. Github. 2012. Dostupné z: <https://github.com/TrenchBroom/TrenchBroom>. [cit. 2024-03-20]. |
| [24] | Lights. Online. Unity User Manual. 2024. Dostupné z: <https://docs.unity3d.com/2023.2/Documentation/Manual/class-Light.html>. [cit. 2024-03-20]. |
| [25] | Component. Online. Unity User Manual. 2024. Dostupné z: <https://docs.unity3d.com/ScriptReference/Component.html>. [cit. 2024-03-20]. |
| [26] | Audio files. Online. Unity User Manual. 2024. Dostupné z: <https://docs.unity3d.com/2023.2/Documentation/Manual/AudioFiles.html>. [cit. 2024-03-20]. |
| [27] | Canvas. Online. Unity User Manual. 2024. Dostupné z: <https://docs.unity3d.com/ScriptReference/Canvas.html>. [cit. 2024-03-20]. |
| [28] | Mixamo | Common questions. Adobe [online]. 2024 [cit. 2024-03-20]. Dostupné z: https://helpx.adobe.com/uk/creative-cloud/faq/mixamo-faq.html |
| [29] | Unreal Engine. Online. In: Wikipedia. 2008. Dostupné z: <https://cs.wikipedia.org/wiki/Unreal_Engine>. [cit. 2024-03-20]. |
| [30] | Unity vs Unreal engine: pros and cons. Online. Kevurugames. 2023. Dostupné z: <https://kevurugames.com/blog/unity-vs-unreal-engine-pros-and-cons/>. [cit. 2024-03-20]. |
| [31] | Unity (game engine). Online. In: Wikipedia. 2007. Dostupné z: <https://en.wikipedia.org/wiki/Unity_%28game_engine%29>. [cit. 2024-03-20]. |
| [32] | Unity − What Makes It the Best Game Engine? Online. Kevurugames. 2023. Dostupné z: <https://kevurugames.com/blog/unity-what-makes-it-the-best-game-engine/>. [cit. 2024-03-20]. |
| [33] | Critically-Acclaimed FPS Horror Game Leaked for PlayStation Consoles. Online. In: Gamerant. 2023. Dostupné z: <https://gamerant.com/dusk-playstation-xbox/>. [cit. 2024-03-20]. |
| [34] | Fundamentals of Rigging - Learn How to Rig in Blender. Online. Cgcookie. C2024. Dostupné z: <https://cgcookie.com/courses/learn-how-to-rig-anything-in-blender-fundamentals-of-rigging>. [cit. 2024-03-20]. |
| [35] | C Sharp (programming language). Online. In: Wikipedia. 2003. Dostupné z: <https://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29>. [cit. 2024-03-20]. |

Seznam obrázků

[Obrázek 1 - C# logo 9](#_Toc161862087)

[Obrázek 2 - Rozdělení OOP 10](#_Toc161862088)

[Obrázek 3 - Ukázka kódu třídy 11](#_Toc161862089)

[Obrázek 4 - Ukázka kódu metody 11](#_Toc161862090)

[Obrázek 5 - Ukázka kódu objektu 12](#_Toc161862091)

[Obrázek 6 - Ukázka kódu rozhraní 12](#_Toc161862092)

[Obrázek 7 - Ukázka kódu přístupových modifikátorů 13](#_Toc161862093)

[Obrázek 8 - Ukázka kódu abstrakce 14](#_Toc161862094)

[Obrázek 9 - Ukázka kódu dědičnosti 15](#_Toc161862095)

[Obrázek 10 - Ukázka kódu polymorphismu 16](#_Toc161862096)

[Obrázek 11 - Ukázka kódu zapouzdření 17](#_Toc161862097)

[Obrázek 12 - Logo *Unreal* enginu[29] 18](#_Toc161862098)

[Obrázek 13 - Logo *Unity enginu[31]* 19](#_Toc161862099)

[Obrázek 14 - Koncept herní úrovně 22](#_Toc161862100)

[Obrázek 15 - Blokování herní úrovně 22](#_Toc161862101)

[Obrázek 16 - Finalizace herní úrovně 23](#_Toc161862102)

[Obrázek 17 - Logo nástroje *Trenchbroom*[23] 27](#_Toc161862103)

[Obrázek 18 - Osvětlení v *Unity* prostředí 28](#_Toc161862104)

[Obrázek 19 - Exportování herní úrovně v nástroji *Trenchbroom* 29](#_Toc161862105)

[Obrázek 20 - Složka textur v nástroji *Trenchbroom* 30](#_Toc161862106)

[Obrázek 21 - Herní model zbraně v nástroji *Blender* 31](#_Toc161862107)

[Obrázek 22 - Retro videoherní textury[33] 32](#_Toc161862108)

[Obrázek 23 - Implementace textur na importované modely v *Unity* 33](#_Toc161862109)

[Obrázek 24 - Využití komponent v *Unity* 34](#_Toc161862110)

[Obrázek 25 - Ukázka kódu pro střelbu hráče 35](#_Toc161862111)

[Obrázek 26 - Ukázka kódu pro pohyb hráče 36](#_Toc161862112)

[Obrázek 27 - Ukázka proměnných využité pro pohyb hráče 36](#_Toc161862113)

[Obrázek 28 - Ukázka kódu abstraktní třídy pro nepřátele 37](#_Toc161862114)

[Obrázek 29 - Ukázka kódu individuálního nepřátele 37](#_Toc161862115)

[Obrázek 30 - Ukázka kódu pro inventář hráče 38](#_Toc161862116)

[Obrázek 31 - Souboru určen pro herní zvuky a hudbu 39](#_Toc161862117)

[Obrázek 32 - Ukázka kódu pro přehrávání herních zvuků 39](#_Toc161862118)

[Obrázek 33 - Rozložení rozhraní hráče 40](#_Toc161862119)

[Obrázek 34 - Rozložení scény hlavního menu 41](#_Toc161862120)

[Obrázek 35 - List animací na stránce *Mixamo*[28] 42](#_Toc161862121)

[Obrázek 36 - Způsob aplikace animací v *Unity* 42](#_Toc161862122)

Seznam tabulek

[Tabulka 1 - Výhody a nevýhody *Unreal enginu*[30] 19](#_Toc161862123)

[Tabulka 2 - Výhody a nevýhody *Unity* enginu[32] 20](#_Toc161862124)

Seznam příloh

|  |  |
| --- | --- |
| Příloha I | Herní manuál |