|  |
| --- |
|  |
| 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 22](#_Toc158963438)

[4.3. Implementace mapy 23](#_Toc158963439)

[4.4. Texturování 23](#_Toc158963440)

[5. Vývoj herních modelů 24](#_Toc158963441)

[5.1. Modelování 24](#_Toc158963442)

[5.2. Texturování 24](#_Toc158963443)

[5.3. Implementace modelů 25](#_Toc158963444)

[6. Vývoj hry 26](#_Toc158963445)

[6.1. Skriptování 26](#_Toc158963446)

[6.1.1. Střelba hráče 26](#_Toc158963447)

[6.1.2. Pohyb hráče 26](#_Toc158963448)

[6.1.3. Nepřátelé 27](#_Toc158963449)

[6.1.4. Inventář 28](#_Toc158963450)

[6.2. Zvuk 28](#_Toc158963451)

[6.2.1. Ukládání zvuků 28](#_Toc158963452)

[6.2.2. Přehrávání zvuků 28](#_Toc158963453)

[6.3. Uživatelské rozhraní 28](#_Toc158963454)

[6.3.1. Rozhraní hráče 28](#_Toc158963455)

[6.3.2. Hlavní menu 29](#_Toc158963456)

[6.4. Animace 29](#_Toc158963457)

[6.4.1. Mixamo 29](#_Toc158963458)

[6.4.2. Aplikace animací 29](#_Toc158963459)

[Závěr 30](#_Toc158963460)

[Seznam použité literatury 31](#_Toc158963461)

[Seznam obrázků 32](#_Toc158963462)

[Seznam tabulek 33](#_Toc158963463)

[Seznam příloh 34](#_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 dokáží běžet i 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 i například videohry pomocí Unity Enginu.

Programovací jazyk byl nadesignován 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 ve stejné třídě jako jsou například jazyky C, C++ a Java. Ikdyž jeho křivka učení není srovnatelná k Pythnu, i tak je ideální pro začátečníky i zkušené programátory, kteří se chtějí naučit nový jazyk.

Jedna z jeho 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 pochopení kódu, který uživatel píše.

Není to nejen jeho design, který exceluje při stavění ruzných 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 podstaty pro využívání OOP:

* **Třída**
  + Třídá definuje strukturu pomocí metod a hodnot, které napodobují objekty z reálného světa.
* **Metoda**
  + Metoda udává konkretní 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 přísupový modifikátor
* **Rozhraní**
  + Rozhraní neboli interface definuje soubor pravidel, které zajišťují určitou funkcionalitu. Rozhraní se využívájí 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>

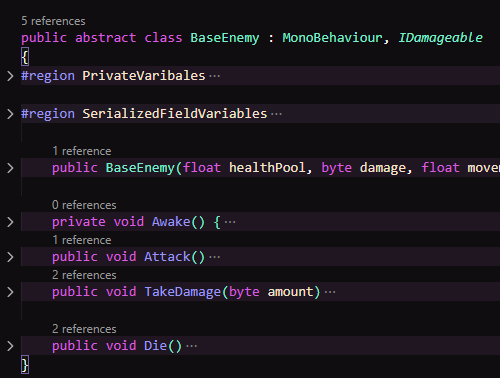
* + 1. Objektově orientováné designové principy

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

Následující principy jsou hlavní pro objektově orientované programování:

**Abstrakce**

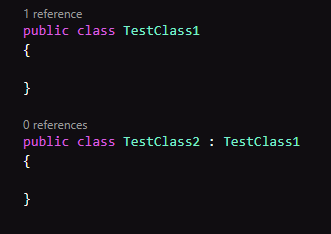
* Abstrakce je koncept myšlení při designování aplikace, který identifikuje třídy a jejich vlastnosi (metody, data...). Abstrakce výtváří základ logiky a dat bez přižazování jakýkoliv hodnot. Tento základ je pak převzán pomocí dědičnosti a umožňuje poté pracovat s veškerými daty, které byli převzány.
* Příklad takové Abstrakce může být vytvoření třídy ve videohře pro **nepřátele.**  Tato třída (např. BaseEnemy) by obsahovala veškere 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 method**
  + 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 závolá 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 přistupové modifikátory. Zapozdření tedy umožňuje uschovat nebo ukázat data a funkce jiným třídám, což může přidat extra vrstvu zapezpečení.

<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 vývojí 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 využívájí 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 nedokáží, 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 mapy

V nacházející kapitole budu popisovat, proč jsem si zvolil jiný nástroj pro tvorbu herní mapy než blender a jak jsem v něm tvořil mapu. Dále zmíním, jak různé komponenty nacházející na mapě byly vytvořeny s její samotnou implementací.

* 1. Trenchbroom

Celá herní mapa je vytvořena pomocí open source map builderu trenchbroom. Je zaměřen na tvorbu herních map na old school hry jako Quake, Half-Life, Doom... Zvolil jsem si tento nástroj oproti například blenderu, jelikož je práce s ním o mnoho jednodušší.

Celý 3D prostor v programu je rozdělen do linek horizontálních i vertikálních. Tyto linky tvoří kostky na které lze vytvářet objekty. Objekty mění velikost dle hustoty linek.

Vytvořené objekty můžeme různě upravovat pomocí nástrojů, které trenchbroom obsahuje. Nejvíce jsem využil nástroj cut, díky kterého jsem dokázal rozřezávat objekty na různé díly nebo nástroj pro pohyb vertexových bodů objektu.

* 1. Osvětlení

Herní mapa je osvětlena pomocí herního objektu point light. Tento objekt nám dokáže vygenerovat zdroj světla. Světlo lze generovat více způsoby, mezi tyto způsoby patří: realtime, baked a mixed. Zvolil jsem baked, jelikož se na mapě nachází několik světel a to by mohlo ztrácet snímky za sekundu u slabších zařízení.

Tyto světla jsou doprovázeny particle systémem, který má za úkol generovat jednoduchého obrázeku ohně. Díky konstantnímu replikování tohoto obrázku a různé měnení jeho velikosti, dráhy pohybu a životnosti, dokážeme simulovat lowpoly, nenárořný 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 a particle effekt. Tento model byl vytvořen i texturován v programu trenchbroom

* 1. Implementace mapy

Po vytvoření mapy v trenchbroomu je třeba mapu implementovat do Unity. Samotná implementace lze udělat několika způsoby. Já jsem zvolil možnost exportu mapy do formátu Wavefront. Díky téhle akci mohu vytvořenou mapu exportovat do Blenderu, kde lze udělat jakékoliv potřebné úpravy před finálního importu do Unity.

Poslední krok pro vložení mapy do Unity je třeba mapu znovu exportovat v Blenderu do formátu FBX. Tento formát již je podporován Unity tudíž je možné importovat mapu do Unity.

Pokud byla importovaná mapa již texturována v programu trenchbroom a byla implementována již zmíněným způsobem. Textury budou také importovány s mapu. Ovšem, aby se textury aplikovaly, je třeba změnit způsob načítání textur na FBX souboru v Unity, jelikož tyto textury budou importovány do souboru a vloženy do Unity automaticky. Tento soubor se vloží do stejné složky jako importovaná mapa.

* 1. Texturování

Implementace textur do trenchbroomu funguje na principu vybrání složky obashující textury a dále stačí jenom vzvolit objekty na které je potřeba danou texturu aplikovat. Textury lze v programu různě upravovat. Takové uprávy se pohybují od velikosti, šířky... až po změnu počtu opakování textury na objektu.

1. Vývoj herních modelů

Veškeré herní modely jsou vytvoření v programu Blender. V kapitole níže se dozvíte jak jsem postupoval při modelování, texturování a samotnému implementování těchto modelů do Unity

* 1. Modelování

Pro vývoj samotných herních modelů jsem ku podivu nepoužil již dřívě zmíněný trenchbroom, ale Blender. Odůvodnění tohohle rozhodnutí je absordňě jednoduché. Ačkoliv je Trenchbroom lépe stavěn pro tvorbu herních map je v něm více komplikované tvoření skutečných modelů.

Můj projekt je založen na retro žánru hry. Tyto hry se vyznačují jejich jednoduchým low poly modelováním a rozpixelovaným stylem. Proto pro dosáhnutí repliky pocitu hrání retro videohry, je třeba vytvářet objekty s co nejmenším počtem polygonů. Modely jsou stavěny jako hrubý návrh modelu před „sculptingem“. Ovšem objekt se jinak více modelovat nebude a veškere detaily jsou zobrazeny na samotné textuře zbraně, proto pro dosáhnutí správněho pocitu retro hry je důležité zvolit správné textury.

* 1. Texturování

Jak již bylo zmíněno je třeba pro replikování retro pocitu ze hry zvolit textury, které tohole pocitu dokáží dosáhnout. Před aplikování textury jsem strávil relativně delší čas s výzkumem správné textury. Je důležité, aby vybraná textura seděla tvaru herního objektu. 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 seděť žánru hry díky jejího rozpixelovanému vzhledu. Méně kvalitní obrázky nám také odůvodní menší velikost hry a náračonost renderování modelů.

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

* 1. Implementace modelů

Implementace herních modelů funguje na úplně stejem principu jako u implementace mapy, Tyto texturováné modely se exportují do formátu FBX, který Unity podporuje. Po vložení modelu do Unity projektu stačí změnit jenom způsob načítání textur nebo velikosti dle potřeby.

1. Vývoj hry

Celý vývoj byl proveden v herním enginu Unity. V kapitole níže se dozvíte, jak jsem vyřešil problematiku různých skriptů.

* 1. Skriptování

Jak již bylo zmíněno v kapitole pro herní engin Unity skriptování funguje na systému komponent. Tyto komponenty se přiřazují herním objektům se kterýma bude script interagovat. Ovšem není to pravidlem, že komponent musí vždy být na herním objektu, lze vytvořit skripty, které se např. jenom starají o tok dat.

* + 1. Střelba hráče

Střelba hráče lze rozdělit do několik části: raycast...

Raycast funguje na principu definování startovní pozice z které „paprsek“ poletí a poté jeho směr, kterým poletí. 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 zbraňě kterou hráč aktuálně drží v ruce. Hráči se při zasáhnutí jakěhokoliv objektu odečte náboj.

Aby hráč nemohl spamovat tlačítko střelby je potřeba udělat vyjímku. Tato vyjímka bude mít na starost rychlost střelby, což zajistí, že hráč nebude střílet, tak rychle jak kliká. Rychlost střílení je proveda 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 klávesi 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 zdědení 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 základe 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 přidáva 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 speficického nepřítele.

Parametry, které určují různé statisticky nepřítele (rychlost chůze, sílá 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 vždy díve na hráče.

Pohyb nepřátele je vytvořen díky umělé inteligence, která má přímo v herní úrovni „Vpečené“ cesty kama může chodit. Tato „vpečená“ cesta nejen určí 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 inventářový systém, 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 přehrávájí 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 užtivetelského rohraní 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 nábujů, poškození...), které jsou poté díky jednoduché kalkulaci přižazeny nebo odečteny textu uživatelského rozhraní.

* + 1. Hlavní menu

Stejně jako hráčské rozhraní funguje hlavní menu na stejném principu. 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 úrověn

* 1. Animace

Animace jsou obrovská část herního vývoje. Dávájá 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 poskituje velkou škálu 3D modelů a animací, které mohou být zdarma použity.

Využil jsem Mixamo, aby jsem si z tama vzal potřebné animace. Ovšem i přes to, že stránka obsahuje automatické riggování modelů v mém případě byl rig rozbitý, proto jsem si model postav riggoval sám.

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 může být pomocí časové osy, kde můžeme upravovat objekty v různých časových sekvencí, které vytvoří animaci. Tenhle způsob není úplně optimální pokud animce již máme vytvořené z jiných zdrojů.

Na objekty na které potřebuji přidat animaci jsem vložil komponent animator. Aby bylo možné vložit na tento objekt animaci, je třeba aby byl tzv. „Rigged“. Rigging znamená přidávání objektu kostru, pomocí, které lze z objektem různé interagovat. Avšak tento komponent vyžaduje referenci na kontroler animací, který definuje, kdy se animace přehrají nebo jaké mají mezi sebou přechody

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 |