Engine Godot a jeho praktické využití

Lubomír Sajič

Bakalářská práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Lubomír Sajič**
Osobní číslo:	**A15031**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační technologie v administrativě**
Forma studia:	**prezenční**
Téma práce:	**Engine Godot a jeho praktické využití**
Téma anglicky:	**The Godot Engine and its Practical Use**

Zásady pro vypracování:

1. **Podrobně se seznamte s enginem Godot.**
2. **V práci popište jeho hlavní vlastnosti a možnosti.**
3. **Vytvořte podrobnou dokumentaci enginu Godot v českém jazyce.**
4. **Navrhněte a vytvořte komplexní aplikaci, která bude demonstrovat širokou škálu funkcí enginu Godot.**
5. **Vytvořenou aplikaci doplňte podrobným popisem její tvorby.**

Rozsah bakalářské práce: -

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **Godot Docs [online]. 2017 [cit. 2017-11-28]. Dostupné z: http://docs.godotengine.org**
2. **SILBER, Daniel. Pixel art for game developers. Boca Raton: CRC Press, 2016. ISBN 9781482252309.**
3. **SCHELL, Jesse. The art of game design: a book of lenses. Boston: Elsevier/Morgan Kaufmann, c2008. ISBN 9780123694966.**
4. **Godot Game Engine Tutorial Series. GameFromScratch [online]. 2015 [cit. 2017-11-28]. Dostupné z: http://www.gamefromscratch.com/page/Godot-Game-Engine-tutorial-series.aspx**
5. **LECARME, Olivier. a Karine. DELVARE. The book of GIMP: a complete guide to nearly everything. San Francisco: No Starch Press, 2013. ISBN 9781593273835.**

Vedoucí bakalářské práce: **Ing. Pavel Pokorný, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **1. prosince 2017**

Termín odevzdání bakalářské práce: **25. května 2018**

Ve Zlíně dne 14. prosince 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

L.S.

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

doc. Ing. Martin Sysel, Ph.D.
garant oboru

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor;
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 17. 5. 2018

.....
podpis diplomanta

ABSTRAKT

Tato bakalářská práce popisuje herní engine Godot, představuje jeho základní možnosti a jeho možné využití v praxi. V teoretické části práce je popsáno rozhraní enginu a jeho ovládací prvky. V další části jsou představeny a popsány jednotlivé nástroje enginu. Praktická část se zabývá tvorbou ukázkové hry a dokumentace k enginu v českém jazyce.

Klíčová slova: Godot Engine, herní engine, vývoj videoher, 2D grafika, 3D grafika, skriptování, částicový systém, herní fyzika

ABSTRACT

This bachelor's thesis describes Godot game engine and introduces its basic capabilities and its possible use in practice. In theoretical part of this thesis the interface of the engine and its control elements are described. Next section introduces and describes individual tools of the engine. The practical part deals with the creation of a demonstration game and documentation for the engine in Czech language.

Keywords: Godot Engine, game engine, game development, 2D graphics, 3D graphics, scripting, particle systém, game physics

PODĚKOVÁNÍ

Tímto bych rád poděkoval mému vedoucímu panu Ing. Pavlu Pokornému za mnoho užitečných rad a připomínek při tvorbě této bakalářské práce.

OBSAH

ÚVOD

Videohry v dnešní době tvoří velkou část naší kultury a jedná se o formu umění, která je na stejné úrovni jako filmy či hudba, proto je třeba herní průmysl dále rozvíjet a posouvat dále. Jedním z nejdůležitějších prvků vývoje každé hry je herní engine.

Herní engine je základní součástí video hry. Každá hra je závislá na svém konkrétním enginu a bez něj nemůže fungovat. Engine musí být přítomen na dané platformě před spuštěním hry, avšak hru netvoří pouze engine, ale je to pouze jedna ze základních částí hry, tak jako je srdce základní částí většího celku (těla) i engine je základní částí hry.
Engine není pouze základní částí hry jako takové, ale také jejího vývoje a měl by předcházet větší části vývojového procesu, jelikož se právě kolem něj točí většina vývoje a vývojového týmu. Funkcí enginu je skrývat běžné funkce spojené se hrou, tak aby se vývojáři mohli soustředit na vytváření grafiky, textur, zvuků, hudby a dalších věcí, které tvoří samotnou hru. Engine pouze vdechuje život hře a stará se například o animace, detekci kolizí mezi objekty, uživatelské vstupy či fyziku.

Tato práce se bude zabývat enginem Godot, který bude blíže představen v teoretické části práce. Budou nejprve popsány jeho základní vlastnosti a možnosti, včetně popisu samotného grafického rozhraní. Následně budou blíže představeny jednotlivé nástroje enginu. Budou přiblíženy možnosti práce ve 2D a 3D prostředí, možnosti jejich kombinace a funkce specifické pro konkrétní prostředí. Zvláštní pozornost bude věnována fyzice v Godotu, přičemž budou opět představeny i rozdíly mezi fyzikou ve 2D a 3D prostředí. Následně bude přiblížena problematika tvorby animace v Godotu a konkrétní techniky animace. Dále budou představeny možnosti skriptování a budou popsány jednotlivé skriptovací jazyky, které Godot nabízí. Následně bude popsáno použití pluginu a možnosti zpracování zvuku. Na konci teoretické části práce bude představen další využitý software.

V praktické části práce bude popsán postup tvorby české dokumentace Godot Enginu, která by měla podrobněji popsat funkce a možnosti enginu. Dokumentace popíše také části, kde engine ještě není zcela bezchybný.

Další část práce popíše podrobně postup tvorby ukázkové hry, která bude demonstrovat širokou škálu možností enginu. Popis postupu tvorby hry se bude snažit nastínit také jakým způsobem v prostředí enginu Godot pracovat a může tak pomoci i začínajícím vývojářům s vytvořením určitých pracovních návyků v Godot Engine. Jednotlivé kapitoly budou chronologicky členěny a popíšou tvorbu hry od konceptu až po hotovou aplikaci.

I. TEORETICKÁ ČÁST

1 Godot Engine

Godot Engine je software určený pro vývoj 2D a 3D aplikací, a to především video her. Nabízí velké množství nástrojů, které jsou pro uživatele k dispozici v jednotném grafickém rozhraní, tak aby se uživatel mohl plně soustředit na vývoj hry. Od roku 2014 je Godot Engine open source a je distribuován pod MIT licencí, to znamená, že veškerý uživatelem vytvořený software je ve vlastnictví uživatele. Godot Engine je vyvíjen a řízen komunitou. Aktualizace enginu jsou vydávány na pravidelné bázi. Co se týče podpory ze strany vývojářů, je možné komunikovat prostřednictvím oficiálních diskuzních fór či sociálních sítí. V dnešní době je tento engine čím dál oblíbenějších u nezávislých vývojářských studií a má velký potenciál do budoucna. [1] [3]

1.1 Základní vlastnosti a možnosti

Godot Engine nabízí inovativní přístup k vývoji video her. Je založený na organizování uzlů a nabízí výběr z velkého množství těchto uzlů. Díky flexibilnímu systému scén, do kterých se jednotlivé uzly vkládají a organizují je vývoj rychlý a snadný. Grafický editor enginu má všechny nástroje přehledně implementovány a je tak vhodný i pro začínající vývojáře. Engine je designován, aby byl uživatelsky přívětivý pro všechny členy vývojového týmu. Úpravy projektu probíhají za běhu enginu bez nutnosti kompilace zdrojového kódu a všechny změny se projevují okamžitě. Godot nabízí i vytváření vlastních nástrojů a tím je ještě flexibilnější.

Od verze 3.0 nabízí ještě lepší možnosti 3D grafiky a samozřejmostí je i tvorba 2D grafických aplikací, včetně možnosti animace.

Pro skriptování aplikace je možné využít širokou škálu jazyků s možností instalace dalších, včetně možností ladění vytvořených aplikací. Engine je multiplatformní a umožňuje i nasazení aplikací na mobilních zařízeních s dotykovou obrazovkou.

Díky uživatelsky přívětivému souborovému systému umožňuje snadnou práci na projektu pro všechny členy vývojového týmu. [5]

1.2 Podporované platformy

Pro vývoj v Godot Enginu, jsou podporovány platformy GNU/Linux, Windows, BSD a Mac OS X. Software vytvořený v Godot Enginu je možné exportovat na platformy

Android, iOS, Windows, Linux, MacOS X, BSD iOS a Web. Je možné software exportovat jak pro 32bit, tak i 64bit architekturu procesorů. [1]

1.2.1 Herní konzole

Godot oficiálně nepodporuje export aplikací pro konzole. Důvodů je hned několik. Prvním důvodem je, že pro vývoj enginu pro herní konzole musí být vývojář enginu registrován jako společnost, zatímco Godot je veden jako open source projekt a ne jako společnost. Dalším důvodem je to, že SDK (Software development kit) pro danou konzoli je licencován danou značkou a i kdyby jej měli vývojáři Godotu k dispozici, nemohli by licencovat Godot jako open source. Posledním důležitým důvodem proč Godot nepodporuje konzole je fakt, že pro vývoj je potřeba specializovaný hardware, který opět poskytuje pouze výrobce dané konzole, tudíž bez něj by nemohl probíhat vývoj. [1]

1.2.2 Porty pro konzole

Ačkoliv Godot oficiálně nepodporuje vývoj her pro konzole a ani jej podporovat nemůže, existují třetí strany, které poskytují služby pro převod her vytvořených v Godot enginu na konzole. Tyto třetí strany vytvořily verzi Godotu přímo pro dané konzole.

Jedním z poskytovatelů této služby je například společnost „Lone Wolf Technology“, která dělá porty her pro konzole Nintendo Switch, Playstation 4 a XBox One. [1]

1.3 Podporované grafické API

API (Application Programming Interface) definuje způsob interakce s komponenty počítače. Rozhraní grafického API je nejčastěji implementováno za pomocí ovladače grafické karty. Při použití standardního grafického API mohou aplikace fungovat na různém grafickém hardwaru. Godot Engine 3.0 podporuje nejrozšířenější grafická API, mezi něž patří OpenGL ES 3.0, OpenGL 3.3 a v budoucích verzích se chystá také podpora Vulkan. [4] [5]

1.3.1 OpenGL

OpenGL (Open Graphics Library) je grafické API pro stolní počítače a pracovní stanice, které umožňuje tvorbu 2D, ale především 3D grafiky. Bylo vyvinuto společností Silicon Graphics, ale v dnešní době jej vyvíjí skupina Khronos, což je průmyslové konsorcium, které se zaměřuje na vývoj různých otevřených API. V současnosti se jedná o nejrozšířenější API na světě. [6] [7]

1.3.2 OpenGL ES

OpenGL ES je multiplatformní API, patřící do rodiny OpenGL. Je určené především pro mobilní zařízení, ale také pro herní konzole. Umožňuje vytvářet pokročilou 3D grafiku. Ačkoli OpenGL ES dominuje na trhu mobilních zařízení, v poslední době se začíná využívat také na stolních počítačích. [6] [8]

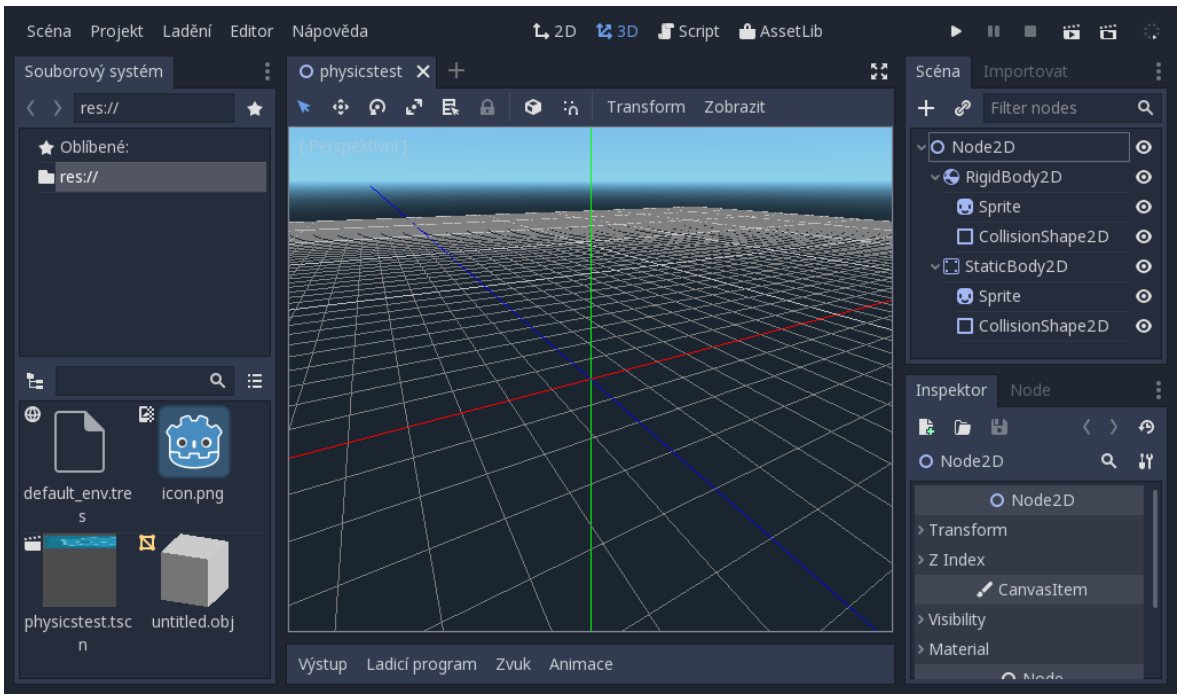
1.3.3 Vulkan

Vulkan je multiplatformní API nové generace, vyvinuté skupinou Khronos. Je to následovník OpenGL, ale používá naprosto odlišný přístup, aby splnilo očekávání dnešních náročných spotřebitelů. Dává mnohem více kontroly do rukou programátora aplikace, toho dosahuje mnohem tenčí vrstvou ovladače oproti jiným API. Oproti předchůdci nabízí navíc nárůst výkonu. [9]

1.4 Popis pracovního prostředí

Po spuštění grafického rozhraní enginu se zobrazí správce projektů. Zde je možné vytvořit nový projekt, importovat existující projekty, smazat projekty nebo zvolit projekt na kterém chce uživatel pracovat. Toto okno také nabízí možnost vytvořit projekt z již připravených šablon a umožňuje tím uživateli rychlejší start vývoje aplikace.

Po zvolení konkrétního projektu k editaci se otevře grafické rozhraní editoru (Obr. 1). Ve výchozím nastavení se otevře okno pro vývoj ve 3D prostředí, které je možné změnit na 2D. V horní části okna se nachází nabídky pro úpravy vlastností projektu a scény. V levé části nalezneme okno pro souborový systém projektu, kde může uživatel jednoduše spravovat všechny soubory projektu. V pravé části okna je panel pro správu scény, kde se nachází všechen aktivní obsah scény nad touto části jsou ovládací prvky pro spuštění či zastavení běhu aplikace.



Obr. 1: Grafické rozhraní editoru

2 NÁSTROJE

Godot poskytuje velké množství vestavěných nástrojů usnadňujících vývoj aplikací. Nástroje jsou velmi dobře integrovány přímo do editoru enginu a umožňují tak uživateli rychlí a efektivní vývoj aplikací.

2.1 Uzly

Uzly v Godot enginu jsou základní stavební kameny pro vývoj hry. Uzly jsou aktivní prvky scény a každý uzel slouží k specifickému účelu, může zobrazit obrázek, přehrávat zvuky, zobrazit 3D model atd. Godot engine nabízí zajímavou organizační strukturu uzlů. Uzly mohou mít pod sebou další uzly a tvoří tak stromovou strukturu, tímto způsobem mohou vznikat mnohem komplexnější funkce spojováním jednotlivých uzlů. [1]

2.2 Scény

Při spuštění hry vytvořené v Godot enginu se spouští tzv. Scéna. Jeden projekt může mít libovolné množství scén, avšak vždy musí být zvolena jedna hlavní scéna, která se zobrazí při spuštění projektu. Scéna se vždy skládá z jednotlivých uzlů, které jsou organizovány do již zmíněné stromové struktury. Každá scéna musí mít kořenový uzel do kterého se vnořují další uzly. Základním konceptem tvorby aplikace v Godot Enginu je editace scén a uzlů, které ji tvoří. [1]

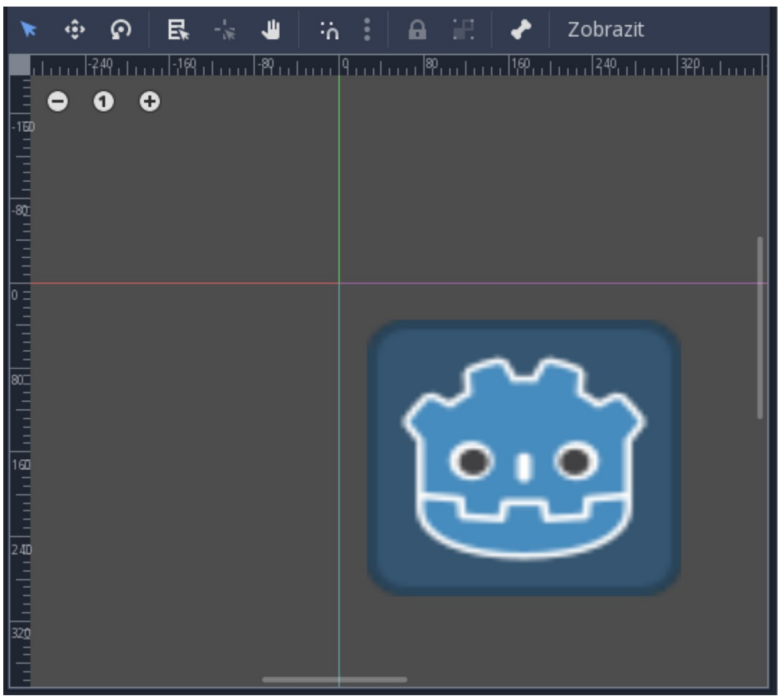
2.3 Správce souborů

Godot obsahuje vestavěného správce souborů, který umožňuje vývojářům efektivně manipulovat se soubory projektů jako jsou 3D modely, textury, obrázky, zvuky, hudba, scény, skripty atd.

Výhodou správce souborů v Godotu je, že spousta dat je obsažena ve scéně samotné a je tím značně snížen objem dat souborů projektu. Pro organizaci souborů se často používají oddělené adresáře pro scény a soubory, které scény používají, takto je organizace souborů mnohem přehlednější. Pro import souborů se do verze 3.0 používaly oddělené adresáře pro zdrojové soubory, bez toho nebylo možné specifikovat parametry importu. Od verze 3.0 Godot využívá modernější a jednodušší postup. Uživatel přesune všechny požadované soubory do adresáře projektu a engine si soubory automaticky importuje a je poté možné určovat další parametry souborů. Godot dělá i automatický reimport souborů, které byli změněny. [1]

2.4 2D prostředí

Godot engine pro vývoj 2D aplikací využívá jednoduchý souřadnicový systém, kde červená osa představuje osu X a zelená osu Y. V levém horním rohu, kde se osy protínají se nachází počátek souřadnicového systému, tedy bod 0, 0 (Obr.2).



Obr. 2: Pracovní prostředí ve 2D prostoru

2.4.1 Důležité uzly 2D prostředí

Základním uzlem ve 2D prostředí je „CanvasItem“ neboli plátno, pod kterým jsou pak všechny 2D uzly. Je důležité si uvědomit, že při změně pozice rodičovského uzlu se automaticky přesouvají také všechny podřadné uzly.

Plátna je možné organizovat do vrstev za pomoci uzlu s názvem „CanvasLayer“, který vytvoří oddělenou vykreslovací vrstvu pro všechny své potomky. Každá vrstva má číselné označení, přičemž výchozí hodnota je 0. Čím větším číslem je vrstva označena, tím výše bude umístěna nad ostatními vrstvami. Číselná hodnota může mít i zápornou hodnotu, taková vrstva se pak bude nacházet až pod výchozí vrstvou. Tato technika se využívá v mnoha situacích, například při tvorbě tzv. „Parallax Backgrounds“, což je pozadí, které se pohybuje pomaleji, než zbytek prostředí nebo při zobrazení nejrůznějších informací, jako je skóre hráče či počet životů, kdy je nutné, aby tyto informace zůstávali na jednom místě na obrazovce a neposouvali společně se zbytkem prostředí.

Dalším důležitým uzlem 2D prostoru v Godot Engine je „Node2D“, který je ve stromové struktuře uzlů vždy potomkem uzlu „CanvasItem“. Tento uzel umožňuje volit pořadí vykreslování svých potomků, stejně jako řízení změn velikosti, pozice či rotace. [1]

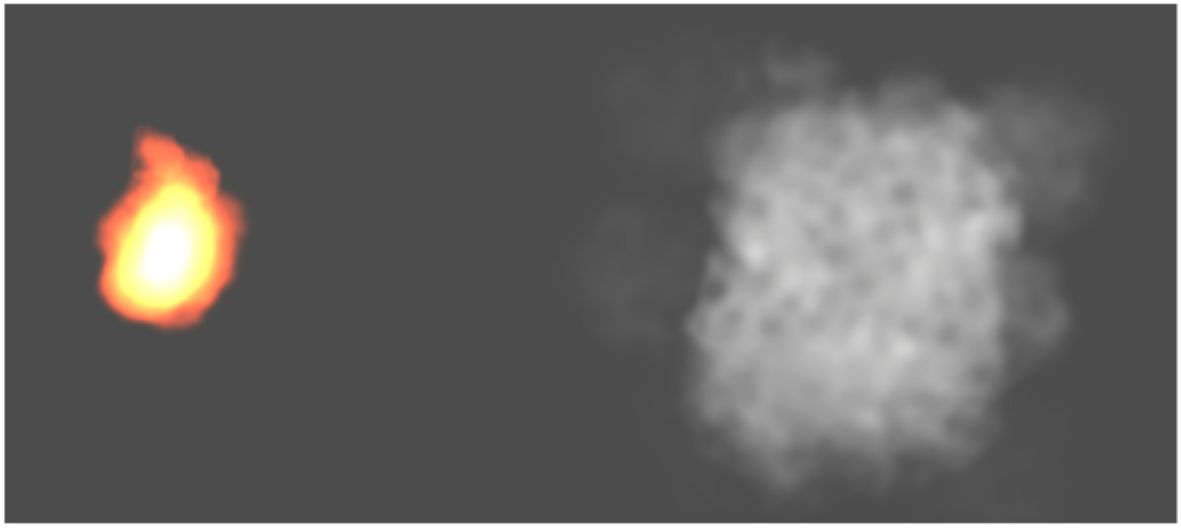
2.4.2 Tilemapy

Tilemapy umožňují rychlou tvorbu herních úrovní. Fungují na principu skládání předem připravených dílů o stejné velikosti do mřížky, jejíž každá buňka odpovídá velikosti jednoho dílu. Soubor takových dílů, ze kterých se tilemapa skládá se nazývá „tile-set“, nejčastěji se díly nacházejí v jednom obrázku, avšak mohou být uloženy i každý zvlášť. [1]

2.5 Částicový systém

Jak již název napovídá, částicový systém se skládá s malých částic, které se pohybují, mění a zanikají v rámci daného částicového systému. Používají se například pro simulaci kouře, ohně, mlhy atd.(Obr. 3) [15]

V Godotu funguje částicový systém tak, že se emituje částice v pevně daném intervalu a s pevně danou dobou existence. Aby se emitované částice chovaly více přiro-zeně jako ve skutečném světě, přidává se každému parametru určitá náhodnost. [1]



Obr. 3: Ukázka použití částicového systému (oheň, kouř)

2.6 3D prostředí

3D prostředí se od 2D liší především přidáním osy Z, která je v náhledu reprezentována modrou barvou. Osa Y znázorňuje pohyb objektů směrem nahoru a dolů, osa X doleva a doprava a osa Z znázorňuje hloubku (dozadu a dopředu). Godot využívá ve všech oblastech metrický systém a je důležité, aby uživatel měl nastaven stejné jednotky také v 3D modelovacím programu, ve kterém se vytvářejí objekty pro Godot, pokud by tak neučinil je možné, že při importu modelu může dojít k chybě ve škálování objektu.

Godot Engine se snaží o co největší podobnost práce se 2D a 3D prostředím a tím usnadnit uživateli přecházení mezi 2D a 3D prostředím. Většina uzlů, které jsou k dispozici ve 2D, mají své protějšky také ve 3D.

Stejně jako byl základním uzlem pro potomky „Node2D“ ve 2D prostoru, ve 3D je tímto základním uzlem „Spatial“. Opět je tento uzel rodičovským uzlem všech 3D objektů a poskytuje možnosti nastavení, pozice, rotace a viditelnosti svých potomků.

Při vytváření 3D projektu je dalším důležitým uzlem „WorldEnvironment“, který se stará o nastavení prostředí scény, jako jsou parametry osvětlení scény, barva pozadí či různé efekty. Je určen především ke zvýšení realističnosti scény. [1]

2.6.1 Import 3D objektů

Oproti importu obrázků do 2D prostoru, je situace ve 3D poněkud složitější, a to především díky ne příliš dobré standardizaci 3D formátů. V Godotu existují dva způsoby jak importovat 3D modely. Prvním způsobem je „OBJ As Mesh“, který se používá pře-

devším u jednodušších modelů ve formátu .OBJ.

Druhým způsobem jak importovat model do Godotu je „Import As Scene“. Jedná se o velmi flexibilní importer, který umožňuje importovat celé scény přímo z 3D modelovacího programu, a to včetně animací, mapovaných kostí a materiálů. Tento způsob je pro import do Godotu nejvíce využíván. [1]

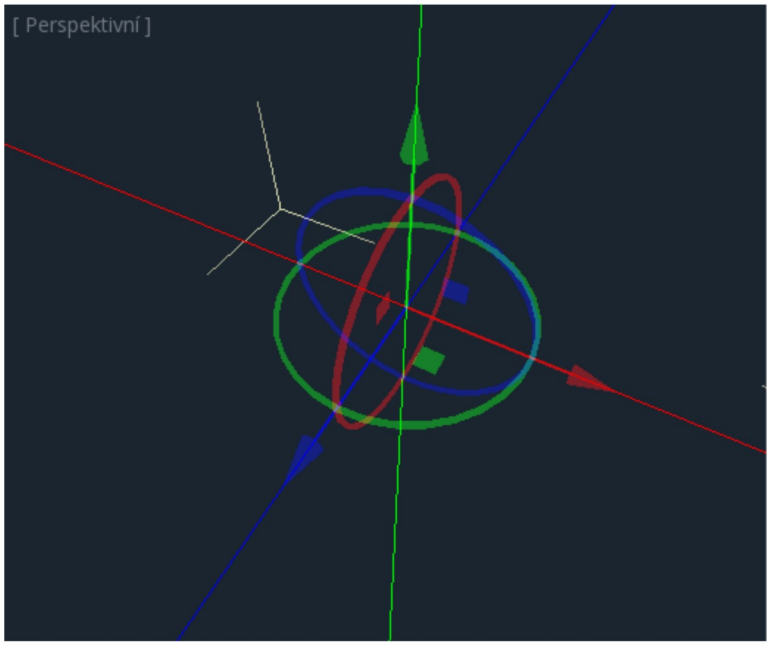
2.6.2 Podporované formáty pro import

Export do formátu, který Godot umí rozpoznat umožňují například Blender, Maya nebo 3DS Max. Následující formáty jsou podporovány importerem scén:

- DAE (Collada) – Doporučovaný formát pro import.
- GLTF 2.0 – Poměrně nový formát, který je však Godotem plně podporován.
- OBJ (Wavefront) – Je také plně podporován, ale je určen spíše pro jednoduché modely, jelikož neumožňuje import společně s mapovanými kostmi na modelu. [1]

2.6.3 Manipulace s 3D objekty

Manipulace s objekty ve 3D náhledu se provádí za pomoci šipek různých barev. Jak již bylo zmíněno, tyto barvy reprezentují jednotlivé osy. Červená pro osu X, zelená pro Y a modrá pro Z (Obr. 4). Táhnutím za jednu z šipek se objekt posouvá ve směru dané osy.



Obr. 4: Manipulace s objekty ve 3D prostoru

2.6.4 Gridmapy

Gridmapy jsou 3D protějškem tilemap z 2D prostředí, to znamená že umožňují také velmi rychlou tvorbu herních úrovní. Rozdílem je, že se tentokrát herní úroveň tvoří ve 3D prostoru, gridmapa se proto skládá s jednotlivých 3D objektů, které se seskupují do mřížky, nejčastěji se jedná o objekty tvaru kvádru. [1]

2.6.5 Světla

Světlo v Godot Enginu může být emitováno různými způsoby. Může být emitováno z materiálu, formou emise určité barvy. Světlo mohou emitovat také speciální světelné uzly nebo světlo, které je součástí uzlu „Environment“.

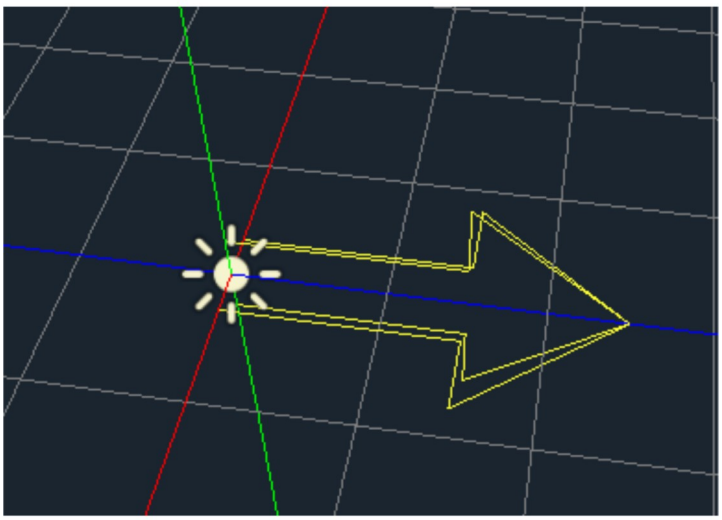
Existují tři typy světelných uzlů, které je možné v Godotu použít, jedná se o „Directional“, „Omni“, a „Spot“. Pro každý světelný uzel je možné nastavit následující vlastnosti:

- Barva (Color) – Barva emitovaného světla
- Energie (Energy) – Využívá se pro vysoký dynamický rozsah (HDR)
- Nepřímá energie (Indirect Energy) – Používá se pro nastavení energie nepřímého světelného zdroje neboli odrazu světla
- Negativ (Negative) – Při povolení této vlastnosti se světlo stává subtraktivní místo výchozího aditivního nastavení.
- Specular – Nastavuje intenzitu odrazu světla od objektů ovlivněných tímto světelným uzlem
- Cull Mask – Objekty, které se nacházejí v uživatelem zvolených vrstvách jsou ovlivněny tímto světelným uzlem [1]

Directional light

„Directional light“ (směrové světlo) je uzel, který představuje světelný zdroj, který vypadá, že svítí z velké dálky jako například sluneční paprsky. Funguje na principu vytváření nekonečného množstvím paralelních světelných paprsků, které pokrývají celou scénu.

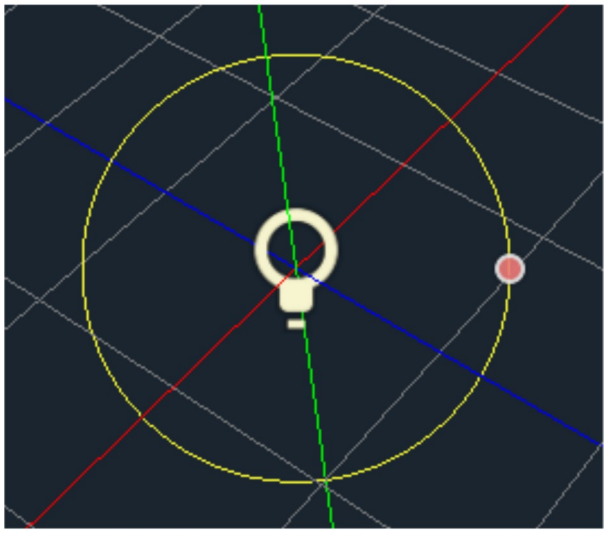
V náhledu Godot Engine je tento světelný zdroj reprezentován velkou žlutou šipkou, která ukazuje směr světelných paprsků (Obr. 5). Pozice uzlu samotného neovlivňuje intenzitu osvětlení scény a muže být umístěn kdekoliv. [1]



Obr. 5: Grafická reprezentace světla typu "Directional light"

Omni light

„Omni light“ je bodový světelný uzel, který emituje světlo ve tvaru koule se zadaným průměrem do všech směrů podobným způsobem jako například žárovka (Obr. 6). Pro nastavení co nejrealističtějšího vzhledu světla slouží dva parametry, a to „Range“ (rozsah) a „Attenuation“ (Utlumení). [1]

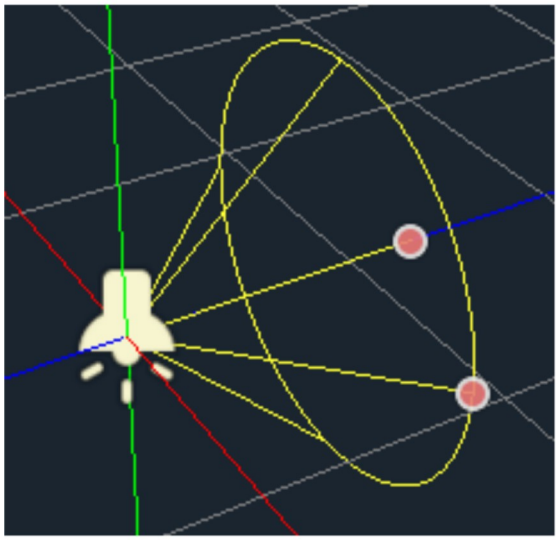


Obr. 6: Grafická reprezentace světla typu "Omni light"

Spot light

„Spot light“ funguje na podobném principu jako „Omni light“, ale neemituje světlo ve tvaru koule, ale v kuželovém tvaru (Obr. 7). Hodí se pro simulaci pouličních lamp, světel automobilů atd. Stejně jako „Omni light“ nabízí k nastavení parametry „Range“ a

„Attenuation“. [1]



Obr. 7: Grafická reprezentace světla typu "Spot light"

2.7 Post-Processing

Post-Processing je metoda využívaná pro přidání nejrůznějších efektů, které engine přidává až po vykreslení určité scény. Používá se k vylepšení vizuální reprezentace video her. [10]

Nejdůležitější uzel týkající se post-processingu je „WorldEnvironment“, který řídí vykreslování prostředí. Lze u něj nastavit parametry jako například oblohu, osvětlení a různé efekty. [1]

2.7.1 Background

„Background“ umožňuje nastavit způsob vyplnění pozadí. Od verze Godotu 3.0 slouží také k nastavení způsobu jakým budou objekty ovlivňovány světlem. Existuje více způsobů, jak nastavit pozadí. „Clear Color“ používá výchozí barvu definovanou projektem, která je konstantní, „Custom Color“ je podobná, jako předchozí, ale je možné nastavit libovolnou barvu pozadí. „Sky“ je velmi užitečný pro nastavení panoramat. Poslední možností nastavení je „Color+Sky“, jedná se o kombinaci „Sky“ a „Color“ a umožňuje definovat oblohu a navíc i barvu. Obloha je použita pro odrazy a „Ambient Light“. [1]

2.7.2 Ambient Light

„Ambient Light“ je typ světla v Godotu, který ovlivňuje každý geometrický prvek stejnou intenzitou a je nezávislý na ostatních typech světel ve scéně. „Ambient Light“ se nejčastěji používá pro jednodušší scény a pro exteriéry velkého rozsahu nebo tam kde hodně záleží na rychlosti aplikace, jelikož tento typ světla není náročný na výpočetní výkon. Existují celkem dva typy.

Prvním je „Ambient Color“, který reprezentuje konstantní barva, která je násobena mírou odrazu světla daného materiálu. Dalším typem je „Ambient Light“ získaný z již zmíněného pozadí typu „Sky“. Je možné nastavit podíl světla a oblohy pomocí parametru „Sky Contribution“, který má ve výchozím nastavení hodnotu 1.0. Dalším užitečným parametrem je „Energy“, což je pouze násobič a využívá se nejčastěji k dosažení HDR (High Dynamic Range) efektu. [1]

2.7.3 Mlha

Mlha umožňuje postupné ztrácení vzdálených předmětů v jednotnou barvu. V Godotu existují celkem dva typy mlhy. Prvním je „Depth fog“, který funguje na základě vzdálenosti od kamery. Druhým typem je „Height Fog“, jedná se o mlhu, která je použita na veškeré objekty, které jsou pod nebo nad určitou výškou a není závislá na vzdálenosti od kamery. Oběma typům je možné upravovat zakřivení či upravovat intenzitu slunečního záření a tím dosáhnout realističtějšího efektu (Obr. 8). [1]

Obr. 8: Rozdíl mezi „Depth Fog“ a „Height Fog“ [1]

2.7.4 Tonemap

Tone maping dokáže vytvořit efekt, kdy jsou tmavé a světlé oblasti více homogenní. Efektu dosahuje za pomocí výběru křivek ze seznamu standardních křivek používaných ve filmu či video herním průmyslu, které jsou následně aplikovány na scénu.

Tone maping je možné nastavit v režimech „Mode“, „Exposure“ a „White“. [1]

2.7.5 Auto expozice

Auto expozice simuluje přizpůsobení oka či kamery různým intenzitám světla. Využívá se především pokud scéna obsahuje kombinaci tmavých a světlých oblastí a přidává tak hře na realismu. Pro využití tohoto efektu je nutné správné nastavení osvětlení, především je nutné nastavit hodnotu energie větší než 1.0 pro osvětlení exteriéru. Stejně jako u ostatních efektů i zde je možné nastavit řadu parametrů, mezi ně patří „Scale“ pro nastavení intenzity osvětlení (čím vyšší hodnota tím světlejší obraz). Dalšími parametry jsou „Min Luma“ a „Max Luma“ , díky kterým je možné specifikovat minimální a maximální osvětlení, kterému se má auto expozice přizpůsobovat. Posledním důležitým parametrem je „Speed“, tedy rychlost jakou se má auto expozice přizpůsobit světelným podmínkám (čím větší hodnota, tím rychlejší je přizpůsobení). [1]

2.7.6 Screen-Space Reflections

„Screen-Space Reflection“, zkráceně SSR, umožňuje vytvořit optické odrazy od jednotlivých předmětů, které jsou mezi sebou v přímém kontaktu (například pokud objekt leží na lesklé podlaze nebo pluje na vodě). Velkou výhodou tohoto efektu je, že pracuje v reálném čase a využívá se často tam, kde jsou objekty v pohybu (herní postavy, auta atd.). [1]

2.7.7 Screen-Space Ambient Occlusion

Screen-Space Ambient Occlusion (SSAO) se používá v situacích kdy je nutné aby světlo nebylo schopno proniknout do dutých nebo konkávních prvků scény (Obr. 9). SSAO funguje nejlépe v kombinaci s nepřímým zdrojem světla jako například „GIProbe“. [1]

Obr. 9: Ukázka použití SSAO [1]

2.7.8 Far Blur

Tento efekt umožňuje simulaci ohniskové vzdálenost kamer, tím že rozmazává objekty v dané vzdálenosti. U efektu je možné nastavit samotnou vzdálenost, ve které budou objekty rozmazány, míru rozmazání a kvalitu efektu. [1]

2.7.9 Near Blur

Na rozdíl od předchozího efektu, který rozmazával vzdálené objekty, „Near Blur“ rozmazává objekty, které jsou blízko kamery a simuluje tak high-end kamery. Parametry, které je možné nastavit jsou stejné jako u „Far Blur“ a často se tyto dva efekty kombinují pro lepší vizuální reprezentaci scény (Obr. 10). [1]

Obr. 10: Kombinace "Far Blur" a "Near Blur" [1]

2.7.10 Glow

V reálném světě se při pořizování snímků či kamerových záběrů stává, že pokud množství světla překročí maximální hodnotu podporovanou danou kamerou či fotoaparátem, světlo se roztáhne do tmavších částí snímku (Obr. 11). Tento efekt simuluje právě „Glow“, který umožňuje nastavit intenzitu, sílu a další parametry efektu. [1]

Obr. 11: Ukázka použití "Glow" efektu [1]

2.8 Kombinace 2D a 3D prostředí

Godot Engine umožňuje kombinovat 2D a 3D objekty, ale je potřeba počítat s mírným úbytkem výkonu aplikace. Kombinace 2D a 3D prostředí se používá například pro 2D hry, které využívají 3D pozadí nebo mohou využívat osvětlení a stíny z 3D prostředí pro lepší vizuální reprezentaci. [1]

2.9 Viewporty

Viewport v Godotu je obdélník do kterého se vykresluje celý interaktivní svět. Viewport by se dal přirovnat k hledáčku kamery. Viewportů může být v jednom projektu více a každá scéna má svůj kořenový.

Viewporty jsou užitečné pro nastavování rozlišení ve kterém se má vykreslovat. Každý viewport má svou výšku a šířku v pixelech. Viewporty, které jsou ve stromové hierarchii pod kořenovým viewportem si berou rozlišení od kořenového. [1]

2.10 Kamery

Kamery jsou speciální typ uzlů, které zobrazují obsah vždy do rodičovského viewportu. V jednom viewportu může být aktivní vždy pouze jedna kamera.

Kamera ve 3D prostředí (Camera) je typ uzlu, který jednoduše zobrazuje 3D obraz do viewportu. Bez aktivní kamery nemůže být 3D scéna vykreslena. Ve 2D prostředí kamera (Camera2D) nutí obrazovku, aby se posouvala a následovala uzel „Camera2D“. Tímto způsobem je jednodušší vytvářet posuvné 2D scény. Při vytváření 2D aplikace nemusí být (na rozdíl od 3D) aktivní ani jedna kamera a engine jednoduše zobrazí obsah kořenového viewportu. [1]

2.11 Fyzikální engine

Fyzikální Engine je software, který umožňuje ve virtuálním prostoru simulovat reálné fyzikální zákony. Takový engine umožňuje aplikaci fyzikálních sil na virtuální objekty a jejich řízení. Další funkcí je správa kolize mezi jednotlivými objekty. Fyzikální engine se využívá nejen při vývoji video her, ale také pro věděcké simulace, které se ale na rozdíl od video her soustředí mnohem více na přesnost a to se odráží na procesní kapacitě, která je velmi vysoká, proto je zpracovávají velice výkonné počítače. Naproti tomu pro použití při vývoji video her se musí počítat i s méně výkonným hardwarem a proto se herní

fyzikální enginy snaží vyhýbat náročnějším simulacím. [11]

2.11.1 Fyzika ve 2D

Godot má pro 2D prostředí svůj vlastní fyzikální engine, jednou z jeho hlavních funkcí je detekce kolize mezi objekty. Pro ověřování zda objekty mezi sebou kolidují je nejprve nutné vyznačit enginu oblast kolize na daných objektech. Základním uzlem pro označení oblasti kolize je „Shape2D“, který poskytuje různé základní tvary, mezi které patří kruh (CircleShape2D), obdélník (RectangleShape2D), tvar kapsle (CapsuleShape2D), trojúhelník (ConvexPolygonShape2D) atd.

V případech, kdy základní tvary nestačí, je k dispozici „CollisionPolygon2D“, který umožňuje vytvoření libovolného mnohoúhelníku, který je nakreslen uživatelem a tím se dá určit tvar objektu, u kterého se má detekovat kolize. [1]

RigidBody2D

Jednou z dalších funkcí fyzikálního enginu v Godotu je simulace reálné fyziky. S tím souvisí důležitý uzel „RigidBody2D“. Tento uzel se nedá přímo kontrolovat, ale dají se na něj aplikovat nejrůznější fyzikální síly a fyzikální engine na základě zadaných hodnot a dalších objektů vypočítá výsledný pohyb objektu. „RigidBody2D“ je aktivní po většinu času, avšak pokud je delší dobu na jednom místě přestanou se provádět fyzikální výpočty, dokud se působením jiného objektu nedá opět do pohybu, tím se šetří výkon procesoru. [1]

StaticBody2D

Jedná se o uzel, který je součástí fyzikálního enginu, to znamená, že je možné u něj detekovat kolizi, avšak není fyzikou ovlivněn (nepohybuje se, nerotuje). Ostatní uzly mohou se „StaticBody2D“ kolidovat a budou ovlivněny fyzikou. Tento uzel se nejčastěji používá pro vytváření částí herních úrovní nebo částí, které mají zůstat na svém místě za každé situace. [1]

KinematicBody2D

„KinematicBody2D“ se svými vlastnostmi podobá předchozímu uzlu „StaticBody2D“. Není ovlivňován fyzikou, ale je součástí detekce kolize. Hlavním rozdílem je, že „KinematicBody2D“ není statický a může se ve scéně pohybovat. Existuje velké množství možných způsobů využití tohoto uzlu. Nejčastěji se používá pro objekty ovládané uživatelem, které mají kolidovat se zbytkem světa nebo pro objekty, které mají vykonávat předem daný pohyb (pohybující se platformy, dveře…) či samotné herní postavy ovládané ze

skriptu. [1]

2.11.2 Fyzika ve 3D

Starší verze Godotu využívali svůj vlastní fyzikální engine pro 3D prostředí, avšak tato situace se ukázala jako velmi komplikovaná z hlediska údržby a především udržením kroku s moderními technikami využití fyziky. Od verze 3.0 využívá Godot open source fyzikální engine Bullet physics, který dodává enginu nové možnosti simulace fyziky, které předchozí verze postrádaly. Z důvodu zpětné kompatibility je stále možno používat původní fyzikální engine. Fyzikální uzly zmíněné u 2D fyziky mají své protějšky také ve 3D prostředí. [5]

2.12 Worlds (světy)

Svět v Godot Enginu je označení pro třídu, která obsahuje vše spojené se světem, tak jak jej známe z reálného světa. Spojuje fyziku, vizuální prostředky a zvukové prostředky do jednoho celku a dává tak virtuálnímu světu život. [1]

2.13 Animace

Animace v počítačové grafice funguje na principu rychlého zobrazování po sobě jdoucích snímků, kdy na každém novém snímku existuje nějaká nepatrná změna oproti předchozímu. Pro 3D animace se často používá technika, kdy jsou nejprve jednotlivé objekty modelovány a následně jim je přidána virtuální kostra, která umožňuje deformaci objektu v určitých místech a tím vytváří pohyb. Pro 2D animaci se používá metoda oddělených snímků s možnou kombinací kostry jako u 3D animace. Důležitým pojmem v oblasti počítačové animace je „key frame“, nebo-li klíčový snímek. Tato metoda funguje tak, že počítač sám spočítá pohyb určitého objektu mezi dvěma klíčovými snímky, to znamená, že uživatel nemusí vytvářen každý snímek zvlášť, ale dělá to za něj software. [12] [1]

2.13.1 2D animace

Godot Engine nabízí nástroj pro tvorbu komplexních 2D animací přímo v prostředí samotného enginu. Animace jsou změny vlastností objektů za pomoci klíčových snímků. Důležitým uzlem v oblasti animace je „AnimationPlayer“, který umožňuje přehrávat vytvořené animace. Jeden uzel muže obsahovat více animací, které na sebe mohou navazovat. Animace v Godotu se tvoří za pomoci časové osy, kam uživatel může vkládat

jednotlivé klíčové snímky. Klíčové snímky v Godotu zachycují parametry objektu v určitém čase. V grafickém prostředí enginu jsou klíčové snímky reprezentovány bílými a modrými kosočtverci na časové ose animace. Lze vytvářet spojité i diskrétní animace (Obr. 12). [1]

Cutout animace

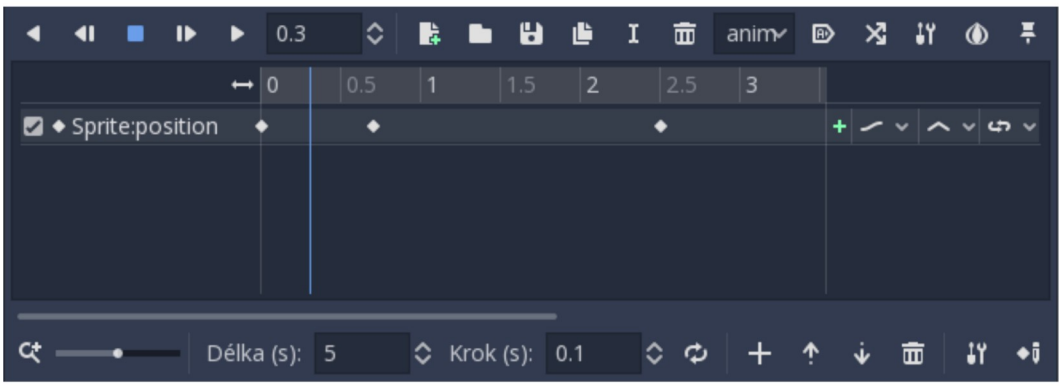
Cutout animaci můžeme často vidět při vytváření animovaných filmů. Je to technika, kdy se na rovný povrch naskládají různé tvary vystřižené z papíru a následně se po povrchu přesouvají po malých částech. Po každém přesunu je vytvořen snímek celé scény a po rychlém změně jednotlivých snímků je vytvořena iluze pohybu. [13]

Tato technika se začala využívat také při tvorbě video her a Godot nabízí několik nástrojů pro tvorby tohoto typu animace. Nejprve je nutné mít animovaný objekt rozdělen na jednotlivé části, které se mají pohybovat (například části těla). Pro animaci jednotlivých částí poskytuje Godot velmi užitečný nástroj a to kostru, stejně jako při 3D animaci. Kosti jsou aplikovány na jednotlivé části objektu a pomocí klíčových snímků animovány. [1]

2.13.2 3D animace

Pro potřeby animace 3D objektů poskytuje Godot několik možností. Nejpoužívanější metodou je import 3D objektů i s animací vytvořenou v externím modelovacím programu. V takovém případě Godot animaci rozpozná a automatický vytvoří nový uzel typu „AnimationPlayer“, se kterým je možné dále pracovat a upravovat parametry animace. [2]

Pokud má importovaný model již vytvořenou kostru, Godot importuje model i s kostrou a vytvoří uzel typu „Skeleton“ se kterým je možné pracovat a vytvářet pohyby objektu. Kosti jsou hierarchicky uspořádány. Každá kost má své jméno (hlava, ruka atd.) a unikátní identifikační číslo (ID). [1]



Obr. 12: Časová osa animace v uzlu typu „AnimationPlayer

2.14 Uživatelské vstupy

Správa vstupů uživatele je velmi komplexní problematika. Musíme brát v úvahu různé možnosti vstupu v závislosti na platformách, pro které bude aplikace určena. Nemusí se jednat pouze o klávesnici a myš, ale také například herní ovladač pro konzole nebo dotykovou obrazovku pro chytré telefony a tablety.

Godot zachytávání uživatelských vstupů usnadňuje použitím datového typu „InputEvent“, ten dokáže zachytávat různé typy vstupních událostí, se kterými se dá poté dále v enginu pracovat. „InputEvent“ obsahuje informace jako například identifikační číslo události (ID) či index vstupního zařízení. „InputEvent“ je reprezentován různými typy v závislosti na typu vstupního zařízení. [1]

2.15 Skriptování

Skriptovací jazyk je vysokoúrovňový programovací jazyk. Jeho velkou výhodou je fakt, že dokáže běžet i bez předchozího zkompilování (přeložení) do strojového kódu, místo toho je každý příkaz překládán za běhu programu pomocí tzv. interpreteru, což je program, který se stará o samotný překlad. Nevýhodou skriptovacích jazyků je jejich nižší rychlost běhu v porovnání s kompilovanými jazyky, jelikož jednotlivé instrukce nezpracovává přímo procesor. V oblasti výkonu skriptovacích jazyků se postupem času udělal velký pokrok a v dnešní době již není mezi rychlostí kompilovaných a skriptovacích jazyků znatelný rozdíl. Velkou výhodou skriptovacích jazyků je jejich jednoduchost a lepší strukturovanost. Mezi skriptovací jazyky patří například Python, Perl, PHP či Ruby. [14]

Godot Engine nabízí celkem čtyři jazyky pro skriptování aplikace. Těmi hlavními jsou GDScript a VisualScript. Od verze 3.0. jsou podporovány také C# a GDNative (C++). [1]

2.15.1 GDScript

GDScript je hlavním skriptovacím jazykem Godot Enginu. Z důvodu velmi dobré integrace s enginem má řadu výhod oproti ostatním jazykům. Umožňuje automatické dokončování kódu, má vestavěné datové typy pro vektory, transformace atd. Využívá více vláken a je velmi rychlý. Jeho syntaxe je velmi podobná Pythonu a pro skriptování v Godotu je doporučován.

Skript v Godoru lze rozdělit na tři hlavní části. Na úplném začátku skriptu se definují proměnné a konstanty, které se v kódu dále mohou použít prostým napsáním jména

konstanty či proměnné. Proměnná jde v průběhu skriptu měnit, avšak konstanta je vždy stejná.

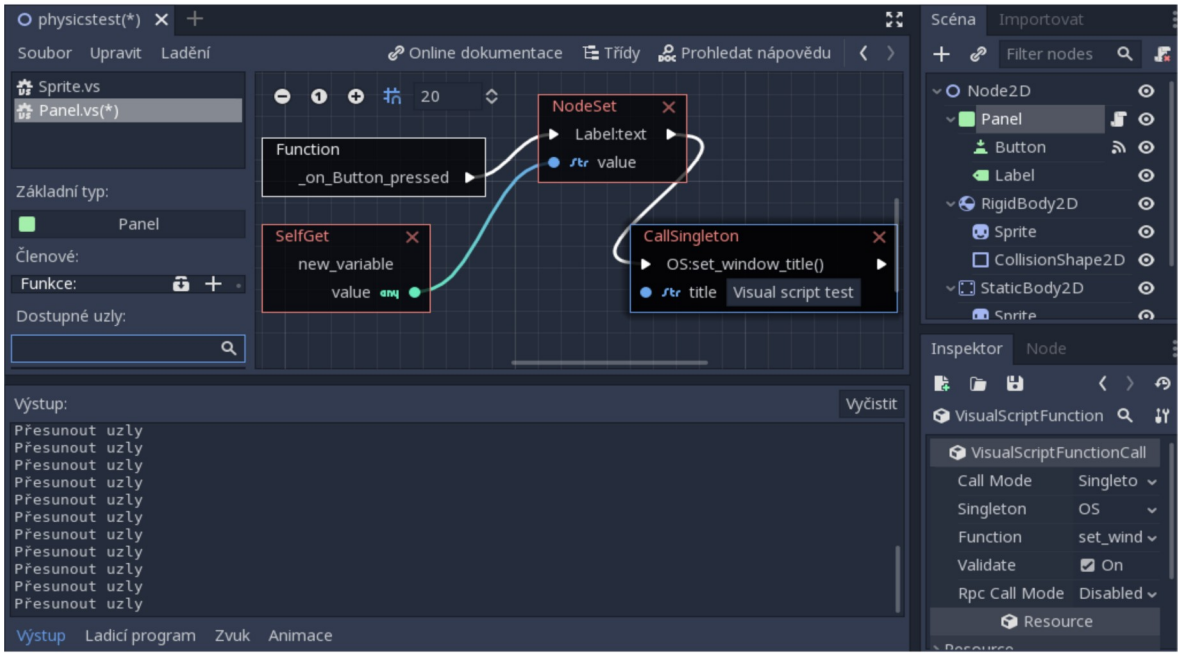
Další částí je funkce „_ready()“. Ta se volá vždy, když uzel vstoupí do scény, tedy pouze jednou. Hodí se při akcích, které mají být spuštěny pouze na začátku skriptu a pouze jednou.

Hlavní částí skriptu je funkce „_physics_process(delta):“, zde se odehrává veškerá logika hry a všechny akce jsou vykonávány synchronizovaně s fyzikálním procesem.

Kromě dvou zmíněných funkcí si může uživatel definovat, také jakékoli vlastní funkce a volat je ve vlastním skriptu či dokonce ve skriptech jiných uzlů. [1]

2.15.2 VisualScript

Godot Engine od verze 3.0. nabízí možnost využití vizuálního skriptování, které funguje na principu spojování jednotlivých bloků (Obr. 13). Je to prakticky vývojový diagram uzlů. Práce s VisualScriptem je velmi jednoduchá. Po otevření nového VisualScriptu se ukáže prázdné plátno, kam je možné skládat jednotlivé uzly, jejich vlastnosti, jednotlivé funkce a další skripty pro statické funkce a konstanty. Spojováním jednotlivých funkcí a uzlů poté vzniká samotný skript. VisualScript je užitečný pro uživatele bez znalosti programování nebo pro programátory, kteří chtějí aby byly jejich skripty dobře čitelné pro všechny. [1] [2]



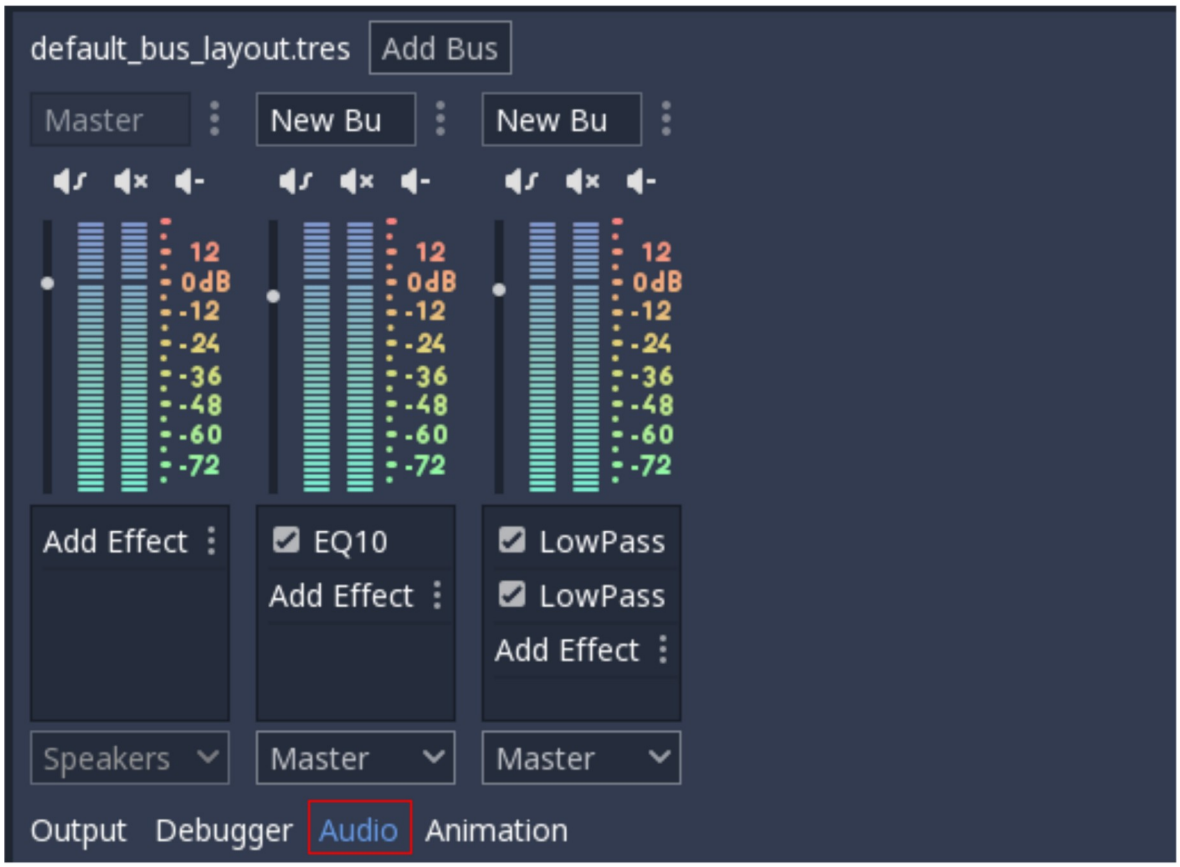
Obr. 13: Ukázka jednoduchého skriptu ve VisualScript

2.16 Pluginy

Pluginy (zásuvné moduly) umožňují rozšiřovat funkcionalitu editoru. Mohou být vytvořeny za pomoci GDScriptu a standardních scén. Mohou přidávat nové uzly nebo dockovací panely a tím vytvořit editor se speciálními funkcemi pro potřeby daného vývojáře. Pluginy se skládají ze dvou základních souborů, plugin.cfg pro nastavení a souboru GDScriptu pro vlastní funkcionalitu. [1]

2.17 Audio

Od verze 3.0 obsahuje Godot kompletně přepracovaný nástroj pro zpracování zvuku a to včetně procesoru pro správu efektů. Ovládání bylo navrženo, tak aby bylo přívětivé pro audio profesionály. Umožňuje kombinaci neomezeného množství kanálů. Každý kanál se zobrazuje pomocí „VU-meteru“, který ukazuje průběh hlasitosti daného kanálu v jednotkách decibelů (14. Obr). Každý kanál může obsahovat nejrůznější audio efekty. [1]



Obr. 14: Nástroj pro zpracování zvuku [1]

3 PŘEDSTAVENÍ DALŠÍHO POUŽITÉHO SOFTWARU

Při zpracování praktické části této práce byl použit, kromě Godot Enginu i další software. Jedná se o grafický editor GIMP a 3D modelovací program Blender. Tento software bude v následující kapitole blíže představen.

3.1 GIMP

GIMP (GNU Image Manipulation Program) je multiplatformní software určený pro tvorbu a úpravy obrázků. Je distribuován pod licencí GNU GPL a je kompletně zdarma i pro komerční užití. Nabízí přehledné a plně nastavitelné grafické rozhraní plné nejrůznějších nástrojů pro zpracování obrázků a podporu velkého množství grafických formátů. Je k dispozici pro platformy GNU/Linux, Windows, Mac OS, OpenSolaris a FreeBSD.

3.1.1 Vlastnosti a možnosti

GIMP nabízí možnosti úpravy vzhledu a chování rozhraní programu podle potřeb uživatele. Má pokročilé možnosti úprav fotografií a digitálního retušování. Podpora hardwaru je na velmi vysoké úrovni a GIMP podporuje i velké množství vstupních metod jako jsou grafické tablety. GIMP podporuje množství běžných i speciálních formátů souborů a je velmi užitečný pro tvorbu textur ke 3D objektům, jelikož nabízí širokou škálu filtrů, které mohou být na obrázek aplikovány. Dále obsahuje množství nejrůznějších nástrojů pro kreslení a úpravy obrázků. Obrázky tvořené v GIMPu je možné organizovat do jednotlivých vrstev a tím je snadnější jejich editace. GIMP nabízí také široké možnosti úpravy barev a správu barevných palet. [18]

3.2 Blender

Blender je software pro tvorbu a zpracování 3D grafiky. Umožňuje modelování a animaci 3D objektů, stejně jako renderování, fyzikální simulace a tvorbu video her. Pro tvorbu specializovaných nástrojů Blender využívá skriptů v jazyce Python. Blender je multiplatformní a je k dispozici pro GNU/Linux, Windows a Mac OS. Je zcela zdarma a distribuován pod licencí GNU GPL.

3.2.1 Vlastnosti a možnosti

Blender umožňuje modelování za pomocí široké škály nástrojů. Má velmi pokročilé možnosti tvorby animací, které se používají i pro tvorbu filmů a to včetně možností editace

video souborů. Postavy mohou být animovány za pomoci kostí, které je možné organizovat do vrstev. Blender používá vlastní render s názvem Cycles, který umožňuje výsledné 3D scény renderovat ve vysoké kvalitě. Další důležitým nástrojem Blenderu je možnost simulace fyziky. Umožňuje simulaci kouře, ohně, tekutin, vlasů, látek, destruktivních objektů a částic. Dále obsahuje vestavěný herní engine, který se často využívá pro tvorbu prototypů her, díky snadné tvorbě herní logiky. Uživatelské rozhraní může být upraveno podle potřeb uživatele a díky API pro Python je možné vytvářet vlastní moduly a tím rozšiřovat možnosti programu. [19]

II. PRAKTICKÁ ČÁST

4 TVORBA ČESKÉ DOKUMENTACE

Při tvorbě české dokumentace bylo v některých kapitolách vycházeno přímo z oficiální anglické dokumentace (http://docs.godotengine.org/en/3.0/). Dokumentace je logicky členěna do jednotlivých kapitol, přičemž každá kapitola představuje blíže určitou část či funkci enginu Godot.

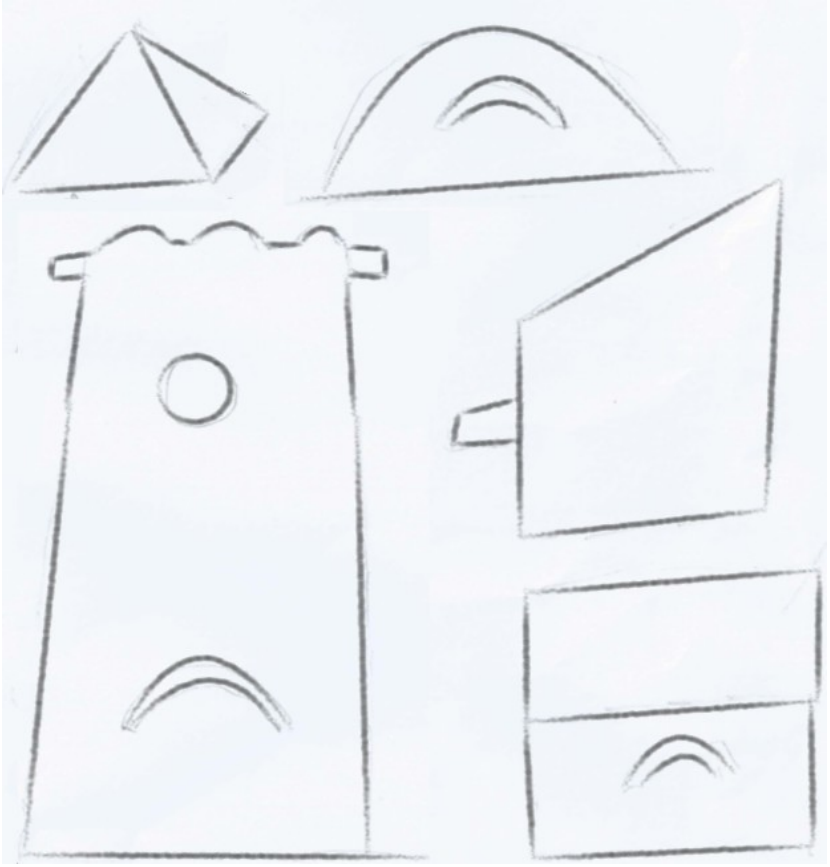
5 TVORBA UKÁZKOVÉ HRY

Následující kapitola popíše postup tvorby hry, na které bude demonstrována široká škála funkcí a možností Godot Enginu. Jako ukázková hra byla vybrána arkáda z pohledu ze shora, kdy má hráč za úkol likvidovat nepřátelské lodě a přitom se vyhýbat či také ničit asteroidy. Při tvorbě byly využity především programy Blender pro tvorbu 3D objektů a GIMP pro tvorbu textur a obrázků.

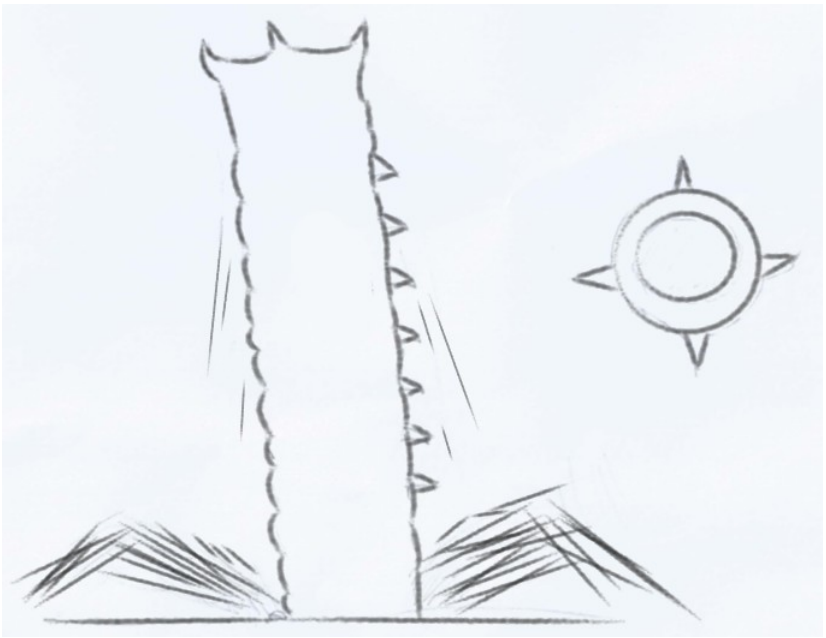
Ve hře se hráč ujímá role pilota vesmírné lodě a snaží se probojovat na konec hry, přičemž v cestě mu stojí celá řada překážek. Tou první jsou nepřátelské lodě, které se snaží hráče zničit pomocí různých typů zbraní. Tou další překážkou jsou nástrahy samotného prostředí, ať už se jedná o asteroidy v hlubokém vesmíru či obří červy, kteří vystřelují ze země směrem k hráči.

5.1 Návrh základního konceptu hry

Ze všeho nejdříve bylo nutné navrhnout koncept hry. Základní vizí bylo vytvořit klasickou arkádovou hru s moderní 3D grafikou a efekty. Prostředí, ve kterém se hra bude odehrávat bylo navrženo ve stylu Sci-Fi, přičemž hráč bude ovládat vesmírnou loď a bude se snažit probojovat vlnami nepřátelských lodí. Jako název hry byl vybrán „Quadrant Z“ a jedná se o označení části vesmíru, ve které se hra odehrává. Hra byla rozdělena do dvou úrovní, ve kterých bude mít hráč za úkol dostat se na konec a přitom získat co nejvyšší skóre likvidací nepřátel. Jako první úroveň byl zvolen hluboký vesmír s hvězdami, asteroidy a planetami. Druhá úroveň byla navržena tak, aby vypadala mnohem komplexněji a především, aby na ní šly demonstrovat grafické možnosti Godot Engine. Úroveň byla zasazena do prostředí pouštní mimozemské planety, na které se budou nacházen rozlehlé oceány, pouštní duny a především velké město plné nejrůznějších budov, přičemž každá budova byl navržena s ohledem na dodržování specifického grafického stylu (Obr. 15). Další věcí, která byla navržena pro druhou úroveň byli obří červy, kteří budou vystřelovat z písku a budou se snažit hráče zasáhnout (Obr. 16). Hra záměrně nemá daný příběh a je na fantazii hráče, aby si mohl domyslet svůj vlastní.



Obr. 15: Koncept budov druhé úrovně



Obr. 16: Koncept červa

5.2 Nastavení projektu

Než bylo možné začít se samotným vývojem, bylo nutné nejdříve v Godotu nastavit projekt podle daných požadavků. Aby mohl Godot vůbec něco zobrazit po spuštění, byla vybrána hlavní scéna, kterou je hlavní menu. Následně byla zvolena ikona projektu, která byla vytvořena v programu GIMP a zobrazuje hlavní vesmírnou loď. Dalším důležitým nastavením bylo rozlišení výsledné aplikace, to bylo nejprve zvoleno jako FullHD (1920x1080 pixelů), avšak to se ukázalo jako zbytečně vysoké a bylo změněno na HD (1280x720 pixelů). Posledním parametrem týkajícím se displeje byl „Stretch_mode“, který umožňuje roztažení obrazu na displeje s vyšším rozlišením. Zde byl nastaven mód roztažení na „2d“ a ignorování poměru stran, tím se obraz vždy roztáhne na celou obrazovku bez ořezání obrazu (Obr. 17.).

Obr. 17: Nastavení zobrazení aplikace

5.3 Přepínání aktivních scén

Jelikož se bude v projektu nacházet několik scén, je důležité aby šlo přepínat z jedné aktivní scény na druhou. Godot nabízí jednoduchou funkci „change_scene“ pro přepínání scén, avšak ta se hodí spíše pro menší scény, proto byla využita sofistikovanější metoda, při jejíž tvorbě bylo vycházeno z návodu v oficiální dokumentaci Godotu. [1]

Ze všeho nejdříve byla vytvořena globální scéna, která bude vždy načtena a přístupná z kterékoli jiné scény. K tomu byl využit „AutolLoad“, který se nachází v nabídce nastavení projektu. V „AutoLoad“, lze definovat buď samostatný skript nebo celou scénu. V tomto případě se jedná o scénu, jelikož ta se bude zobrazovat při načítání ostatních scén. Proto byla vytvořena scéna s jednoduchým pozadím s hvězdami a textem „Loading...“, které byl následně přidán skript, jenž bude obstarávat přepínání scén a načítání jejich dat. Ve skriptu je důležitá funkce „goto_scene“, která obstarává samotné přepínání scén. Nejprve se načtou data nové scény a stará scéna se vymaže, aby se uvolnilo místo v operační paměti, přičemž se při tomto procesu zobrazí obrazovka s načítáním. Scény poté mohou být přepínány s kterékoli scény zavoláním zmíněné funkce.

5.4 První úroveň

Pro první úroveň hry bylo zvoleno prostředí vesmíru a byla navržena tak, aby se v ní daly vytvářet a testovat základní mechanizmy hry, jako je pohyb hráče, detekce kolize mezi objekty či tvorba animací.

5.4.1 Pozadí a textury

Nejprve bylo nutné vytvořit pozadí, které bude reprezentovat vesmír. Ve hře vzniká iluze pohybu tak, že hráč je na jednom místě, ale pohybuje se svět okolo něj, proto byly nejprve vytvořeny dvě plochy v Blenderu, na které byly aplikovány textury. První plocha je statické pozadí vesmíru a mraků a druhá plocha jsou hvězdy, které se budou pohybovat v protisměru hráče a vytvářet tak iluzi, že se hráč pohybuje směrem dopředu.

Pro pohyblivou vrstvu, tedy hvězdy, byla nejprve v programu GIMP vytvořena textura, která byla následně namapována na plochu v Blenderu. Textura je průhledná, aby byla vidět statická plocha vesmíru pod hvězdami. Stejným způsobem byla vytvořena také druhá statická plocha s texturou vesmíru, která již ale není průhledná.

5.4.2 Export objektů z Blenderu

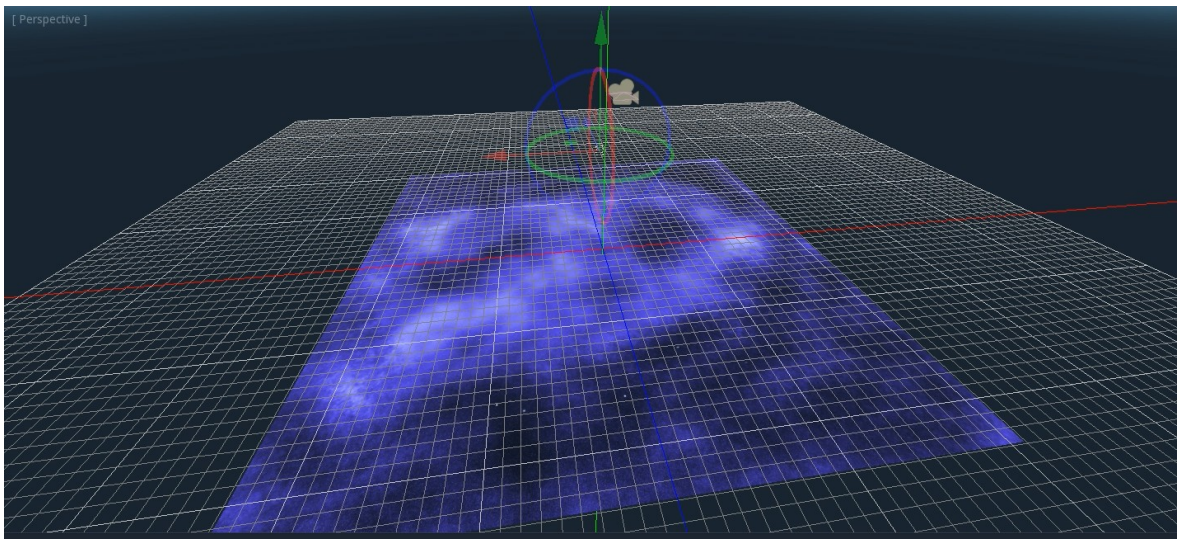
Po vytvoření obou ploch bylo nutné je exportovat z Blenderu a importovat do Godot Enginu. Pro export a import celých scén z Blenderu bylo využíváno formátu glTF 2.0, jedná se o poměrně nový formát, který je ale velmi dobře standardizován a je určen přímo k převodu 3D scén. Formát je v poslední době čím dál více prosazován vývojáři počítačových her a je doporučován samotnými tvůrci Godotu. Blender ve výchozí podobě nepodporuje export do glTF, proto bylo nutné nainstalovat plugin „glTF 2.0 Exporter“

(https://github.com/KhronosGroup/glTF-Blender-Exporter). Po instalaci je Blender připraven k exportu modelů.

5.4.3 Import objektů a příprava scén

Po vytvoření nového projektu v Godot Enginu, byly exportované scény z Blenderu přesunuty do adresáře projektu a Godot si scény importuje automaticky, avšak tyto scény nelze upravovat dokud se neuloží v nativním formátu Godotu, proto je nutné scénu v Godotu nejprve otevřít a následně uložit jako „.tscn“ soubor. Oba objekty byly tedy uloženy jako scény. Následně byla vytvořena hlavní scéna, do které se z dalších scén bude sestavovat samotná úroveň. Jedná se o princip vnořování scén do jiných scén a ačkoli se tento princip zdá matoucí, jedná se ve skutečnosti a velmi jednoduchý a rychlý způsob tvorby scén. Jednou z hlavních výhod tohoto přístupu, je úprava objektů. Při úpravě objektů ve vedlejší scéně se po jejím uložení projeví změny automaticky i ve všech scénách ve kterých je vnořena. Do hlavní scény, která bude reprezentovat první úroveň hry byla vložena kamera, aby mohl být obraz vykreslen do viewportu.

Do hlavní scény byly vnořeny scény obsahující dvě vytvořené plochy (v Godotu se tento proces nazývá „vytváření instancí“) (Obr. 18).



Obr. 18: Plochy vesmíru importované do enginu

5.4.4 Planety

Aby scéna vypadala zajímavěji byly vytvořeny planety které se budou pohybovat ve scéně stejně jako hvězdy a dokreslí tak celkovou atmosféru. Nejprve byla v Blenderu vytvořena koule, u které byl označen šev, tedy předěl, který má brát Blender v úvahu při

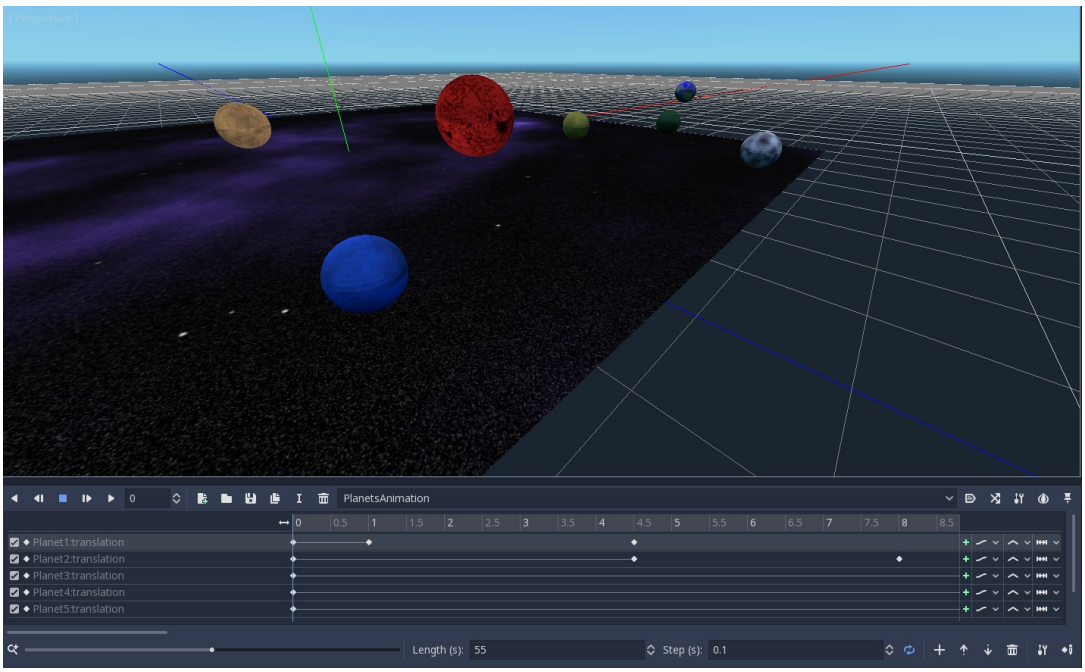
mapování textury. Pomocí „UV unwrap“ bylo vytvořeno rozvržení objektu v 2D prostoru, které bylo následně importováno do GIMPu, kde probíhala tvorba textur.

Celkem bylo vytvořeno sedm odlišných planet, avšak ve skutečnosti se jedná pouze o sedm odlišných textur, přičemž byly mapovány na jeden objekt a poté exportovány jako oddělené scény.

5.4.5 Animace scény

Když byla hlavní scéna připravena a všechny vnořené scény byly na svém místě, začalo se s animací hvězd a planet. Jak již bylo zmíněno, hvězdy a planety se posouvají a vytváří tak iluzi pohybu vesmírných lodí. U hvězd byl vytvořen uzel typu „Animation-Player“ a následně byly vloženy klíčové snímky u počáteční pozice hvězd a konečné pozice hvězd, přičemž animace se přehrává ve smyčce. Výsledný efekt je, že hvězdy se pohybují pod kamerou a po uplynutí určité vzdálenosti se vrací zpět na startovní pozici.

Stejně byly animovány také planety. Zde však byl jejich pohyb o něco složitější klíčové snímky startovních a konečných pozic byly zvoleny, tak že některé planety se pohybují pod kamerou stejně jako hvězdy a jiné čekají na své pozici a poté začínají svůj pohyb taktéž, zatímco planety na konečných pozicích se pomalu vrací zpátky na své pozice mimo záběr kamery. Až jsou všechny planety na svých pozicích začíná druhá vlna oběhu, ale ta je odlišná od té první, aby bylo dosaženo jisté variace. Po dokončení druhé vlny se animace ve smyčce opakuje (Obr. 19).

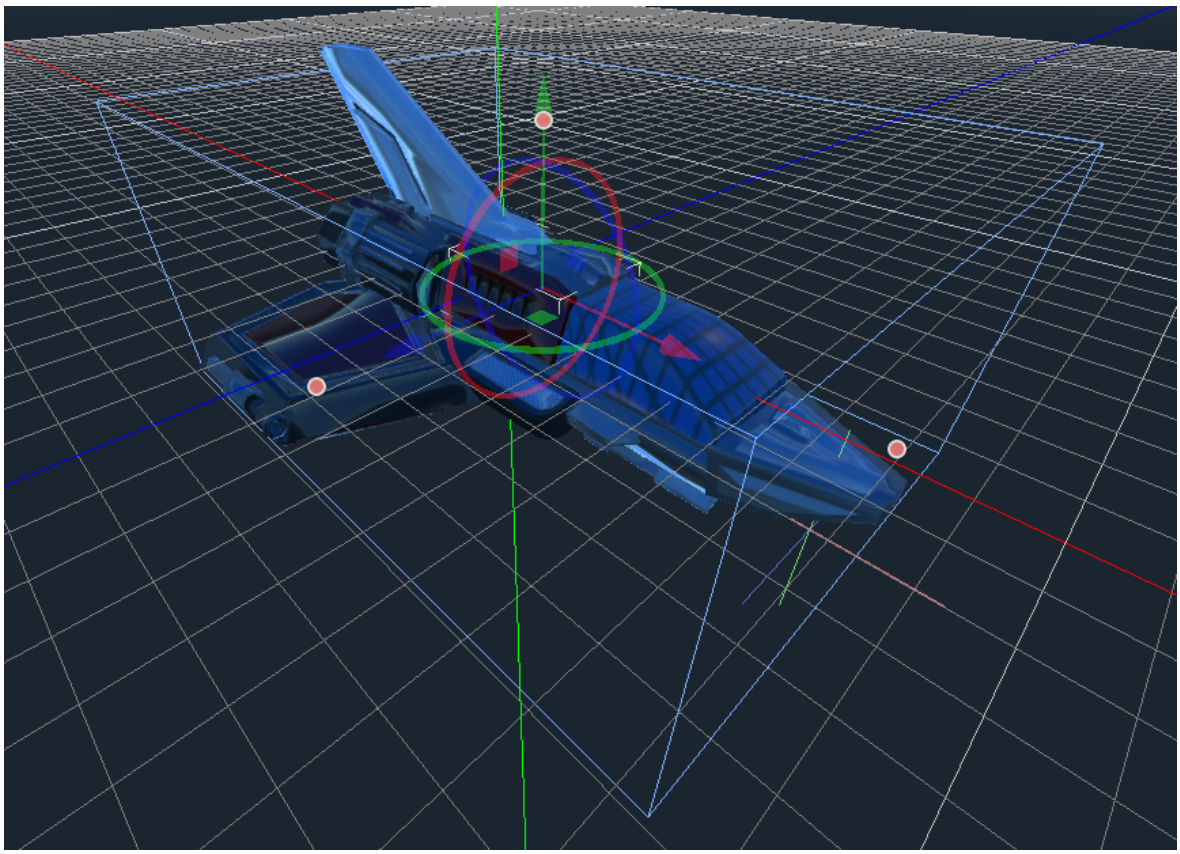


Obr. 19: Tvorba animace planet

5.5 Hlavní loď

Hlavní vesmírnou loď ovládá hráč a obsahuje hlavní skript scény, který řídí téměř všechno dění ve scéně. Pro model lodi byl využit jeden ze sady pěti lodí ze stránky opengameart.org, autorem modelů je Viktor Hahn a jsou distribuovány pod licencí CC BY-SA 3.0. Model byl exportován z Blenderu a importován do Godotu. Samotný model, neboli „mesh“ byl vložen do uzlu typu „KinematicBody“, který umožňuje kontrolu nad jeho pohybem a navíc umožňuje správu kolize s ostatními objekty. Aby kolize správně fungovala, je potřeba enginu vyznačit oblast, která má podléhat kolizi, proto byl přidán uzel „CollisionShape“, jako tvar kolize byl zvolen kvádr a pomocí něj byl označen samotný model lodi (Obr. 20). Následně byla v hlavní scéně vytvořena instance scény s hlavní lodí a začalo skriptování uzlu s lodí. Níže je uveden autor použitého modelu, včetně licence:

- Fighters by Viktor Hahn (Viktor.Hahn@web.de), is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. http://creativecommons.org/licenses/by-sa/3.0/



Obr. 20: Vyznačení oblasti kolize u hlavní lodi

Jedna z prvních funkcí, která byla vytvořena byl úbytek zdraví lodi. Byla definována proměnná pro zdraví z hodnotou 100 a následně vytvořena samotná funkce poškození,

kde se zdraví snižuje o určitou hodnotu (Obr. 21).

```
func _damage(): #Funkce poškození (Stejně jako u nepřátel)
    if health>=0:
        health = health - 2 #Úbytek zdravý o daný počet jednotek
        ArmorLabel.set_text(String(health)) #Výpis zdravý do HUD(přepis hodnoty v daném labelu)
    if health<=0:
```

Obr. 21: Funkce poškození u hlavní lodi

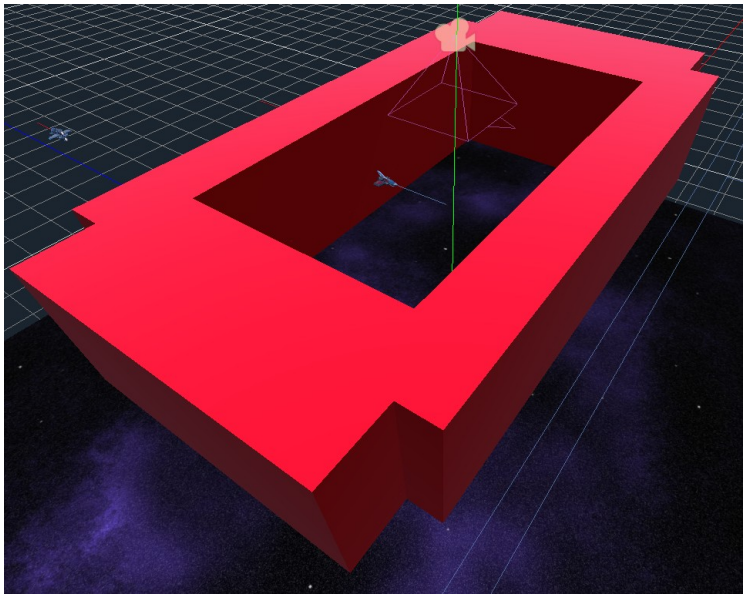
5.5.1 Pohyb lodi

Před skriptováním samotného pohybu lodi byly nejprve definovány proměnné, které budou pro pohyb potřeba, a to rychlost pohybu a směr. Poté se na začátku funkce „_physics_process(delta)“ definoval směr jako Vector3(0,0,0), je to proto, aby se směr vždy resetoval na tuto hodnotu. Následně bylo nutné zjistit jakou klávesu uživatel stiskl a jaká akce se má vykonat po stisku příslušné klávesy. Godot nabízí velice užitečnou funkci pro kontrolu stisknuté klávesy „Input.is_action_pressed()“. Za pomoci této funkce byly vytvořeny podmínky, pokud je určitá klávesa stisknuta, změní se proměnná „direction“ v určité ose o určitý počet bodů (Obr. 22). Body se v jednotlivých osách mohou přičítat nebo odečítat a vzniká tak pohyb uzlu. Posledními kroky pro pohyb hlavní lodi bylo použití funkcí „normalized()“ pro normalizaci rychlosti pohybu a vynásobením směru rychlostí a časem a následně použití funkce „move_and_slide“, která umožňuje po definování směru pohyb tělesa a jeho klouzání po různých površích, jako jsou stěny či rampy.

```
if Input.is_action_pressed("ui_up"): #Pukdu uživatel stiskne šipku nahoru...
    direction.z -= 1 #...zmenši směr na ose Z o 1 (je to stejné jako direction.x = direction.x - 1)
if Input.is_action_pressed("ui_down"):
    direction.z += 1
if Input.is_action_pressed("ui_left"):
    direction.x -= 1
if Input.is_action_pressed("ui_right"):
    direction.x += 1
```

Obr. 22: Podmínky pro pohyb hlavní lodi

Po úspěšném vytvoření funkčního pohybu hráče se vyskytl zásadní problém, a to že hráč může libovolně opustit obrazovku a tím i oblast hry. Tomu bylo zabráněno vytvořením stěn okolo hráče a následně jejich zneviditelnění (Obr. 23). Tím hráč nemůže opustit pluchu vyznačenou stěnami a v kombinaci s výše uvedenou funkcí „move_and_slide“ klouže po stěnách při kontaktu s nimi.



Obr. 23: Viditelné hranice oblasti pohybu hráče

5.5.2 Zbraně

Hlavní loď má k dispozici dvě zbraně, a to laser a plasmu. Laser byl navržen, tak aby bylo možné střílet s větší kadencí, avšak má malé poškození. Plasma střílí s nižší kadencí, ale má mnohem větší poškození.

Pro model laseru byl nejprve v Blenderu vytvořen válec a v GIMPu byla vytvořena textura, která byla na připravený objekt namapována a objekt byl importován do Godotu. V hlavní scéně byla následně vytvořena instance scény s laserem jako potomek hlavní lodi. Pro kontrolu kolize byl zvolen uzel typu „RayCast“, jehož velikost byla nastavena stejná jako velikost laseru. „RayCast“ vysílá paprsek v daném směru a umožňuje tak kontrolu kolize. Ve skriptu byla u „RayCastu“ využita funkce „ is_colliding():“ pro kontrolu, zda paprsek koliduje z nějakým objektem ve scéně. Pokud koliduje, pak se do proměnné pro kolidující objekt uloží pomocí funkce „get_collider()“ název objektu se kterým koliduje. Zde je nutné podotknout, že název je interní název uzlu v enginu a nekoresponduje se samotným názvem definovaným uživatelem, proto se dále využila funkce „get_name()“, pro získání samotného názvu uzlu, tak jak jej vidí uživatel a je možné poté dále definovat akce, které se mají při kolizi spustit. Tím byl získán název nejbližšího uzlu, se kterým paprsek koliduje. V tomto stavu je model laseru stále viditelný ve scéně. Požadovaný efekt ale je, že až po stisknutí mezerníku hráčem se objeví model laseru a začne se kontrolovat kolize, proto byly zmíněné funkce pro kontrolu kolize zabaleny do podmínky a vykonávají se až po stisku mezerníku, zároveň byl do podmínky přidán parametr „visible“ u modelu

laseru, jehož hodnota je „true“. Pokud mezerník stisknut není hodnota „visible“ se nastaví na „false“ (Obr. 24). Pro realističtější reprezentaci byla ve scéně laseru vytvořena animace pulzování, které bylo dosaženo změnou průhlednosti materiálu.

```
if Input.is_action_pressed("ui_select"):
    if ray.is_colliding(): #Pokud paprsek ("RayCast") koliduje...
        collidingobject = ray.get_collider() #...Vezmi collider(obsahuje název uzlu v Enginu)
        print(collidingobject.get_name())
        if collidingobject.get_name()=="Enemy": #Zjisti definované jméno uzlu v collideru a pokud se shoduje se jménem "Enemy"...
            enemy._damage() #... Zavolej funkci "_demadge" v uzlu Enemy
        elif collidingobject.get_name()=="Enemy2":
            enemy2._damage()
        elif collidingobject.get_name()=="Enemy3":
            enemy3._damage()
        elif collidingobject.get_name()=="Asteroid1":
            get_parent().get_node("Asteroid1")._damage()
        elif collidingobject.get_name()=="Asteroid2":
            get_parent().get_node("Asteroid2")._damage()
        elif collidingobject.get_name()=="Asteroid3":
            get_parent().get_node("Asteroid3")._damage()
        elif collidingobject.get_name()=="Asteroid4":
            get_parent().get_node("Asteroid4")._damage()
    else:
        print("Žádná kolize s laserem")
    laser.visible = true
else:
    laser.visible = false
```

Obr. 24: Skript pro kontrolu kolize laseru

Pro vytvoření druhé zbraně s názvem plasma byl zvolen zcela odlišný přístup, jelikož plasma využívá projektilů, které se pohybují určitou rychlostí v určitém směru a po kontaktu s objektem kontrolují kolizi, laser je v tomto směru jednodušší na implementaci, jelikož pouze vysílá paprsek a kontroluje kolizi. Pro model projektilu plasmy byl využit již dříve vytvořený model pro planetu, na který byla namapována textura vytvořená v GIMPu. Po importu do Godotu jako scény bylo nutné nastavit parametr „Unshaded“, je to proto aby nebyla plasmová koule stínována a měla stále stejný kontrast. Jako rodičovský uzel ve scéně plasmy byl zvolen typ „Area“, ten umožňuje kontrolu kolize v dané oblasti, avšak není součástí fyziky hry, což je pro účely projektilu ideální. Plasma oproti laseru, má svůj vlastní skript, který určuje její chování po vstupu na scénu. Pro samotný pohyb směrem kupředu byly využity stejné proměnné jako pro pohyb hlavní lodi, zde je však směr pevně daný a pohyb samotný se odehrává pomocí funkce „translate“, ve které se násobí rychlost a směr pohybu. Tím se po vstupu uzlu na scénu začne pohybovat daným směrem a danou rychlostí. Pro kontrolu kolize s jakýmkoli objektem byla využita funkce „overlaps_area“, díky které je možné zjistit zda se dvě oblasti navzájem překrývají a pomocí podmínek pak určit akce, které se po kolizi mají vykonat. Jednou z funkcí, která se po kolizi vykonává je „queue_free()“, jedná se o velmi důležitou funkci, která slouží k vymazání uzlu ze scény, proto při kolizi uzel automaticky zmizí. Dalším problémem týkajícím se plasmy byla situace, kdy plasma opustí obrazovku. Ačkoli uživatel již plasmu nevidí, ta se stále nachází ve scéně a obsazuje místo v operační paměti. Aby se této situaci zamezilo, byl k plasmě přidán

uzel „VisibilityNotifier“, ten umožňuje dohled nad viditelností uzlu. Pro jeho správnou funkci bylo nutné spojit tzv. signál se samotným skriptem. V záložce „node“ v uzlu „VisibilityNotifier“ se spojil signál „screen_exited“ s vytvořeným skriptem. Tím se ve skriptu vytvořila funkce, která spouští definované akce pokud uzel opustí obrazovku, v tomto případě tato akce byla funkce „queue_free()“ (Obr. 25).

```
    translate(Vector3(direction*speed))
    #Detekce kolize plasmy s nepřáteli:
    if overlaps_area(enemy1Area): #Pokud se překrývá oblast plasmy s oblastí nepřítele1...
        queue_free() #...Zruší se instance plasmy
        enemy._damage_plasma() #...a zavolá se funkce _damage uvnitř nepřítele1
    elif overlaps_area(enemy2Area):
        queue_free()
        enemy2._damage_plasma()
    elif overlaps_area(enemy3Area):
        queue_free()
        enemy3._damage_plasma()
    elif overlaps_area(enemy_type2Area):
        queue_free()
        enemy_type2._damage_plasma()
    elif overlaps_area(asteroid1Area):
        queue_free()
        asteroid1._damage_plasma()
    elif overlaps_area(asteroid2Area):
        queue_free()
        asteroid2._damage_plasma()
    elif overlaps_area(asteroid3Area):
        queue_free()
        asteroid3._damage_plasma()
    elif overlaps_area(asteroid4Area):
        queue_free()
        asteroid4._damage_plasma()
    elif overlaps_area(enemy_type2):
        queue_free()
        enemy_type2._damage_plasma()
    elif overlaps_area(enemy_type2_1):
        queue_free()
        enemy_type2_1._damage_plasma()
#   pass


func _on_VisibilityNotifier_screen_exited():
    queue_free()
```

Obr. 25: Kontrola kolize plasmy s nepřáteli

Nyní je scéna s plasmou připravena k implementaci do hlavní lodi, jelikož ta bude projektily vystřelovat. Opět bylo nejprve nutné vložit podmínku, která detekuje kterou klávesu uživatel stisknul. Na rozdíl od laseru či pohybu, zde byla využita funkce „is_action_just_pressed“, která danou akci provede pouze jednou, po stisknutí příslušné klávesy. Samotná akce která se má provádět je vytvoření instance scény plasmy. Ještě než se instance vytvoří, je nutné nejprve vytvořit konstantu s načtenou scénou, tak aby byla připravena k instanci. Samotná instance se provádí ve dvou krocích prvním je funkce „instance()“, ta

vytvoří instanci, ale engine neví čí potomek instancovaný uzel bude, proto je nutné ještě použít funkci „add_child“ a zvolit uzel, který bude rodičem instancovaného uzlu. Posledním krokem vytvoření instance scény je nastavení její pozice. Projektil se má objevit na pozici hlavní lodě, proto je využit parametr „translation“, a to ve tvaru „Plasma.translation=(translation)“, tím se přiřadí pozice lodi k pozici projektilu (Obr. 26).

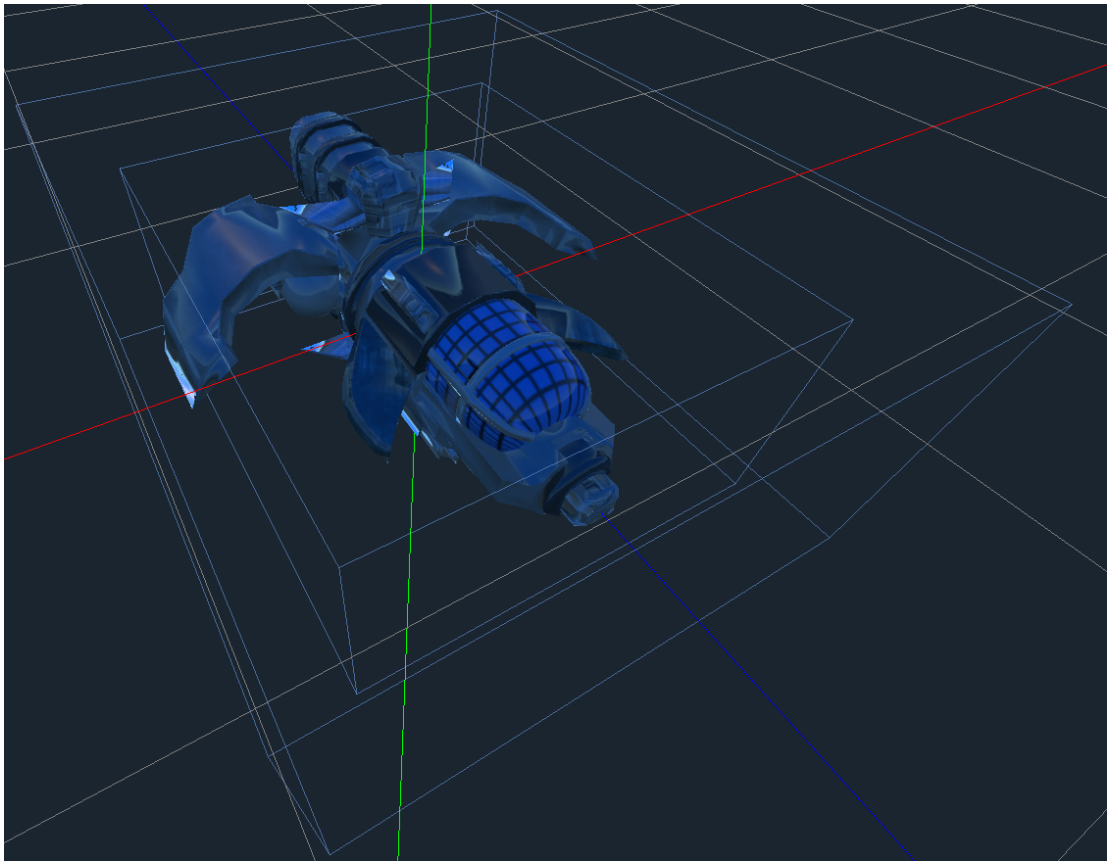
```
if Input.is_action_just_pressed("ui_select"):
    var Plasma=PlasmaScene.instance() #Vytvoř
    get_parent().add_child(Plasma) #Přiřadí P
    Plasma.translation=(translation) #Přiřadí
```

Obr. 26: Vystřelení plasmy hlavní lodí

5.6 Nepřátelské lodě

Nepřátelské lodě přilétají do scény v různém počtu a provádějí různé manévry, přičemž hráč se je snaží zničit. Po zničení nepřítele je určitá šance, že z něj vypadne bonus v podobě zdraví, nebo dvou různých zbraní.

Pro tvorbu nepřátelských lodí byl využit opět model lodi ze stejné sady modelů jako při tvorbě hlavní lodi. Model byl opět exportován z Blenderu a importován do Godotu jako scéna. Postup tvorby je totožný s tvorbou hlavní lodi. Základem je uzel typu „KinematicBody“ a následně byl zvolen „CollisionShape“ pro vyznačení oblasti kolize. Oproti hlavní lodi mají nepřátelské lodě uzel typu „Area“ a k němu přiřazený „CollisionShape“. Uzel „Area“ má dvě funkce. Za prvé umožňuje detekci kolize s projektilem plasmy a za druhé hlídá stav, kdy se hlavní loď dostane do těsné blízkosti nepřátelské lodě (Obr. 27).



Obr. 27: Vyznačení kolizních oblastí nepřátelské lodě

Stejně jako hlavní loď i ta nepřátelská obsahuje funkci pro poškození, ve které se snižuje hodnota zdraví, zde jsou ale dva typy poškození, a to laser a plasma, podle toho se kterým prvkem nepřátelská loď koliduje. Laser odečítá 3 a plasma odečítá 30 z proměnné „Health“.

5.6.1 Zbraň

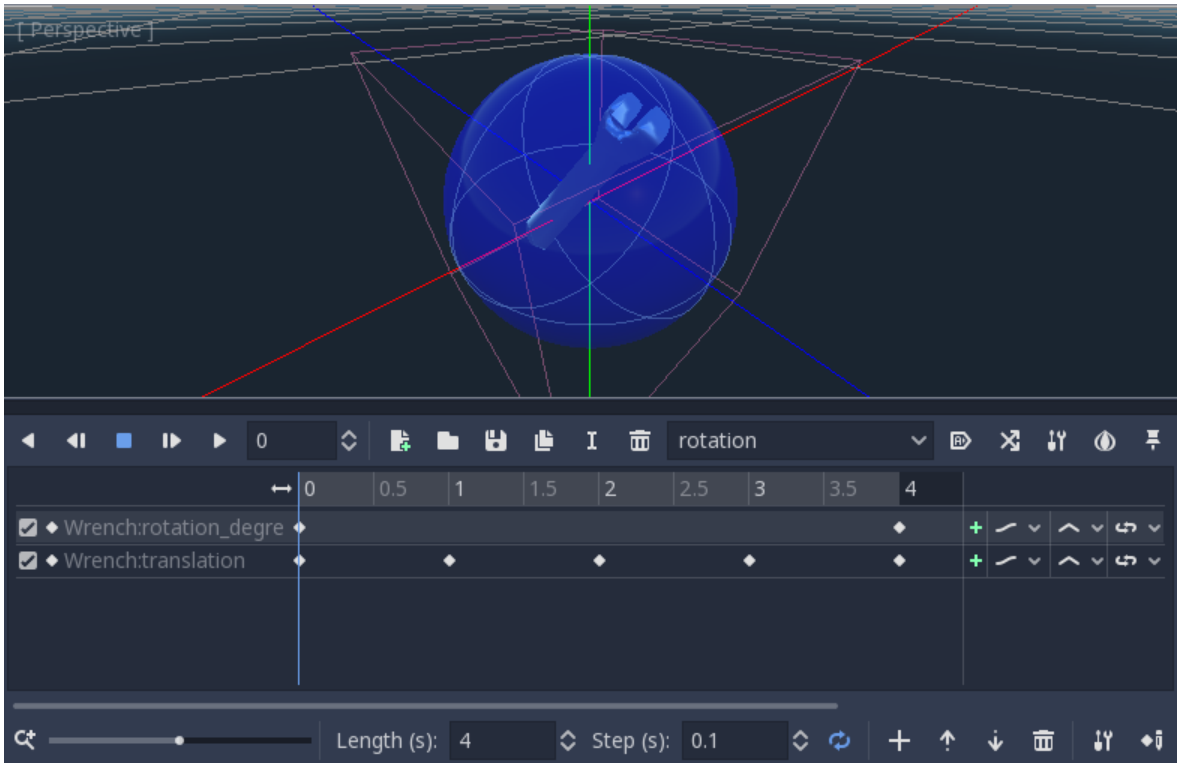
Nepřátelé mají k dispozici jako zbraň laser, stejně jako hlavní loď. Detekce kolize probíhá stejně jako u hlavní lodi. Pokud je název kolizního objektu stejný jako název hlavní lodi, zavolá se funkce pro poškození hlavní lodi. Stejně jako u hlavní lodi jsou laser i paprsek stále aktivní, což není požadovaný efekt. Laser by se měl objevovat a mizet v určitém časovém intervalu, stejně jako jako paprsek pro kolizi. Toho bylo dosaženo za pomoci animace. Byl vytvořen uzel typu „AnimationPlayer“ a vytvořena nová animace. Jednotlivé klíčové snímky jsou od sebe vzdáleny 0,5s a každý lichý snímek obsahuje vypnutou viditelnost u laseru a vypnutý paprsek, každý sudý snímek má tyto parametry zapnuty, tím pádem je dosaženo požadovaného efektu a kolize se kontroluje pouze v případě, že je laser viditelný. Výsledná animace se spouští ve skriptu nepřítele uvnitř funkce „_ready“, to znamená, že se spustí vždy automaticky při vstupu nepřítele do scény.

5.6.2 Animace nepřátel

Pro pohyb nepřátel po scéně byla zvolena metoda animace pomocí klíčových snímků, stejně jako při tvorbě laseru nepřátel. Ve scéně jsou celkem tři stejné kopie nepřátelských lodí a v průběhu hry přilétají do záběru kamery ve skupinkách maximálně po třech. Pro tento účel má každá loď svůj „AnimationPlayer“, který obsahuje animace jednotlivých vln, které pomocí klíčových snímků mění pozici nepřátel. Nepřátelé obsahují ještě jeden „AnimationPlayer“, který slouží ke změně pozice nepřítele mimo záběr kamery při jejich zničení. Při vytváření jednotlivých vln bylo využito také stěn, které zabraňují hlavní lodi v opuštění záběru kamery, jelikož přesně kopírují záběr kamery a bylo tak jednoduší lodě v jednotlivých vlnách rozmisťovat.

5.6.3 Bonusy

Bonusy může hráč sebrat, pokud vypadnou z nepřátelské lodě po jejím zničení. Samotný model pro každý bonus byl tvořen v Blenderu. Prvním tvořeným bonusem byl bonus pro zdravý a jako model, který jej zastupuje byl vybrán klíč. Po importu modelu do Godotu, byl do scény přidán ještě objekt tvaru koule, ve kterém bude klíč umístěn z důvodu lepší viditelnosti. Koule byla obarvena na modrou barvu pomocí parametru „Albedo“ a byla jí přidána průhlednost, aby byl vidět klíč uvnitř. Samotný klíč byl v enginu taktéž upraven, přesněji řečeno jeho materiál byl upraven přidáním parametru „Metallic“ a nastavením hodnoty na 0,5. Tím vznikl kovový materiál klíče. Dalším faktorem přispívajícím k lepší viditelnosti klíče je jeho animace. Proto byl přidán uzel typu „AnimationPlayer“ a byla vytvořena animace. Klíč v kouli rotuje a pohybuje se nahoru a dolů ve smyčce. Toho bylo dosaženo vytvořením počátečního klíčového snímku rotace a poté vytvořením koncového klíčového snímku z rotací 360° okolo osy Y. Pro pohyb nahoru a dolů byla využita další stopa animace, kde se stejným způsobem animuje pozice klíče, která se mění každou 1s (Obr. 28). K bonusu byl nakonec přidán uzel „Area“ a „CollisionShape“, aby bylo možné detekovat kolizi.



Obr. 28: Animace bonusu se zdravím

Ve skriptu bonusu je animace spuštěna ve funkci „_ready()“, tedy při vstupu uzlu do scény. Ve skriptu bylo dále nutné vytvořit pohyb bonusu směrem pryč ze scény, aby nebyl stále na jednom místě. Pohybu bylo dosaženo naprosto stejným způsobem jako pohybu projektilu plasmy, jediným rozdílem je jiná hodnota směru (opačný než u plasmy) a nižší rychlost, aby jej hráč měl šanci sebrat. Pro sběr bonusu bylo nejprve nutné zjistit zda hlavní loď překrývá oblast bonusu pomocí funkce „overlaps_body()“. Zároveň se ověřuje zda zdraví lodi je mezi hodnotami 0 a 100. Pokud jsou obě podmínky splněny, ukončí se instance bonusu funkcí „queue_free()“ a přičte se ke zdraví hlavní lodi hodnota 30, následuje ještě jedna vnořená podmínka a to pokud je zdraví po přičtení větší než 100, potom se uloží hodnota pouze 100, tím je ošetřeno, že zdraví hráče nepřekročí hodnotu 100. Stejně jako u projektilu plasmy je zde vyřešena situace, kdy bonus opustí záběr kamery. Po vyslání signálu uzlem „VisibilityNotifier“, ukončí se instance bonusu (Obr. 29).

```
func _ready():
    get_node("Area/AnimRepair").play("rotation")

func _physics_process(delta):
#    # Called every frame. Delta is time since last frame.
#    # Update game logic here.
    translate(Vector3(direction*speed)) #Pohyb bonusu
    if RepairArea.overlaps_body(Main_char): #Pokud oblast bonusu překrývá oblast hlavní lodi...
        if Main_char.health<100 and Main_char.health>0: #...a pokud je zdraví hlavní lodi mezi 0 a 100
            queue_free() #...ukonči instanci bonusu
            Main_char.health = Main_char.health+30 #přičti hlavní lodi ke zdraví 30
            if Main_char.health>100: #Pokud bude po přičtení výše uvedené hodnoty zdraví větší než 100
                Main_char.health=100 #Potom se zdraví rovná 100(aby hráč nemohl mít Armor více než 100
            ArmorLabel.set_text(String(Main_char.health)) #Nastav label na hodnotu zdraví
    


#    pass


func _on_VisibilityNotifier_screen_exited():
    queue_free()
```

Obr. 29: Skript pro sebrání bonusu hráčem

Další dva bonusy slouží k přepínání zbraní hlavní lodi. Pro jejich modely byly využity již dříve vytvořené modely laseru a plasmového projektilu. Dalším rozdílem byla barva koule obklopující modely, aby se daly snáze rozeznat. Plasma má zelenou barvu a laser červenou. Dalším rozdílem je akce při sebrání těchto bonusů. Místo přidání zdraví hlavní lodi se mění hodnota její proměnné, která určuje jakou zbraň používá, hodnota 1 je laser a hodnota 2 je plasma. Samotné vypadnutí bonusu ze zničeného nepřítele je řešeno ve skriptu nepřítele, tím že se vygeneruje náhodné číslo od 1 do 8 a poté je pomocí podmínek určeno jaký bonus má vypadnou podle čísla (Obr. 30).

```
BonusNumber=randi()%9+1 #Vygeneruje náhodné čísl
if BonusNumber==1: #Pokud bude náhodné číslo 1..
    var RepairBonus=RepairScene.instance() #...P
    get_parent().add_child(RepairBonus) #Přídá b
    RepairBonus.translation=(translation) #Přesu
elif BonusNumber==2: #Pokud bude náhodné číslo 2
    var LaserBonus=LaserBonusScene.instance() #.
    get_parent().add_child(LaserBonus) #Přídá bo
    LaserBonus.translation=(translation) #Přesur
elif BonusNumber==3: #Pokud bude náhodné číslo 3
    var PlasmaBonus=PlasmaBonusScene.instance()
    get_parent().add_child(PlasmaBonus) #Přídá b
    PlasmaBonus.translation=(translation) #Přesu
```

Obr. 30: Generování bonusu při zničení nepřítele

5.7 Asteroidy

Asteroidy jsou založeny na podobném principu jako nepřátelské lodě, ale prolétají pouze jedním směrem, nestřílí a nepadají z nich bonusy proto jsou na úplném začátku hry, aby si hráč mohl osahat základní herní mechaniky.

Asteroidy byly v Blenderu vymodelovány a byla na ně mapována textura vytvořená v GIMPu.

Po importu modelů asteroidů bylo nutné v Godotu lehce upravit jejich materiál, jelikož odrážely světlo a nevypadaly tak realisticky. Pro odstranění odrazu byla ve vlastnostech materiálu zvýšena hodnota parametru „Roughness“ na hodnotu 1. Čím větší hodnota tím méně světla objekt odráží. Uspořádání uzlů v asteroidu i jejich skripty jsou prakticky totožné s nepřátelskými loděmi, ale na rozdíl od nich nemají zbraň. Technika animace je také shodná s nepřáteli, ale je mnohem jednodušší, jelikož asteroidy putují pouze jedním směrem. Dalším podstatným rozdílem je, že z důvodu odlišných modelů asteroidů, nemohly být v hlavní scéně pouze duplikovány jako nepřátelské lodě, namísto toho je každý asteroid oddělená scéna s vlastním skriptem.

5.8 Kolize mezi objekty

Kolizní systém je jednou z nejdůležitějších součástí hry. Umožňuje detekovat kolizi mezi objekty ve scéně a popřípadě na ni reagovat. Nejprve byl u hlavní lodi a nepřátel vytvořen uzel „CollisionShape“, jemuž byl přiřazen tvar kvádru a jeho velikost byla nastavena na velikost lodí. Tento uzel zajišťuje, aby lodě nemohli prolétat skrz ostatní, ale aby se zastavily pokud narazí jedna na druhou a engine tuto situaci hlídá automaticky. „CollisionShape“ byl využit také při tvorbě stěn, aby hráč nemohl opustit danou oblast. Zde nastává problém, kdy hráč sice nemůže vyletět z oblasti, ale to stejné platí také pro nepřátele, proto zde bylo využito kolizních vrstev, které Godot nabízí. Nepřátelské lodě se nachází ve stejné vrstvě jako hlavní loď, ale nenachází se ve vrstvě se stěnami a hlavní loď je ve vrstvě se stěnami a nepřáteli.

Další kolize, kterou je třeba kontrolovat je pokud se hráč přiblíží příliš blízko k nepříteli a dotkne se jej, pak je nutné zavolat funkci poškození. Tato situace je řešena za pomoci uzlů typu „Area“, které se nacházení okolo nepřátelských lodí a kontrolují, zda objekt hlavní lodi vstoupí do této oblasti.

5.9 Částicový systém

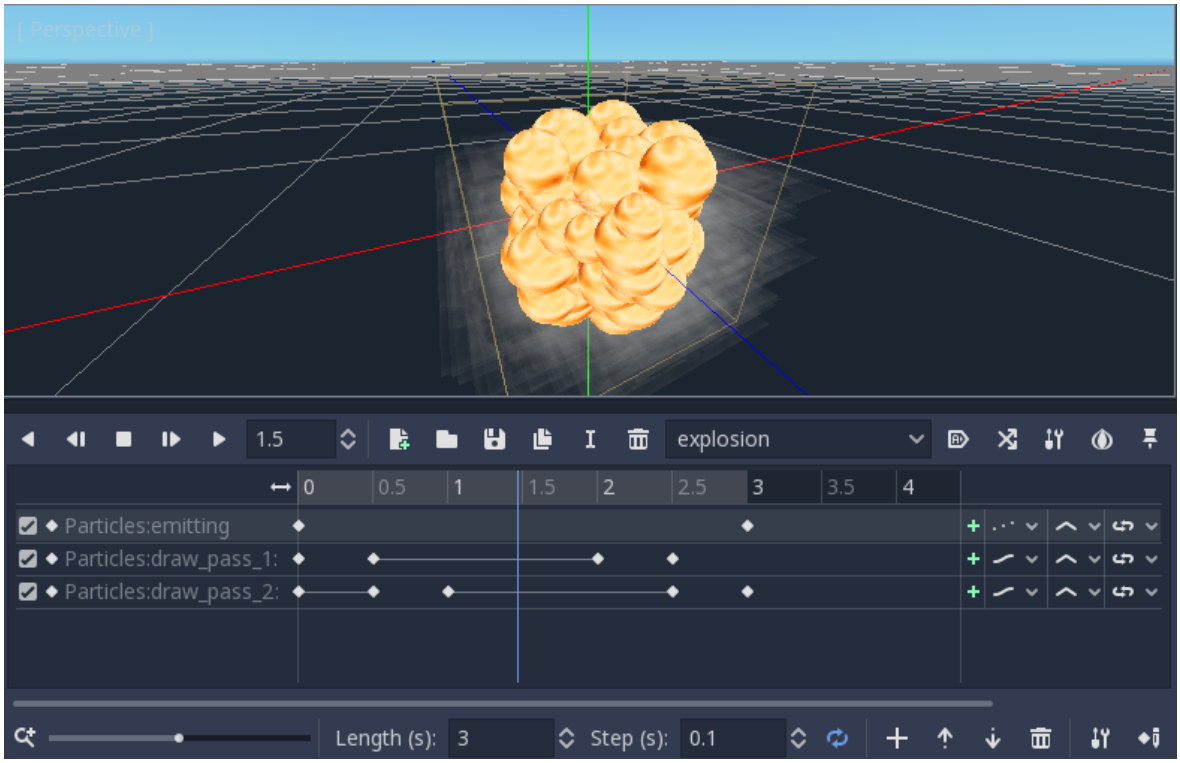
V mnoha částech hry bylo využito částicového systému, který Godot nabízí. Byl použit při tvorbě výbuchu nepřátelské lodě, asteroidu či v kombinaci s projektilem plasmy.

5.9.1 Výbuch

Výbuch se skládá ze dvou typů částic, z ohně a kouře. Před tvorbou výbuchu samotného byly nejprve vytvořeny textury pro částice v GIMPu.

Pro tvorbu částic výbuchu byla nejprve vytvořena nová scéna. Do scény byl přidán jako kořenový uzel „Spatial“ a jako jeho podomek následně „Particles“. Nejdůležitější částí procesu je správné nastavení uzlu „Particles“. Nejprve byl nastaven nový „process material“ typu „particles material“, ten umožňuje nastavení nejrůznějších parametrů samotného částicového systému. Dalším krokem bylo vytvoření dvou „draw passes“, které určují tvar a materiál částic. V tomto případě byla jedna částice nastavena na tvar koule, které byl přidán nový materiál a v záložce „albedo“ jí byla přidána již vytvořená textura ohně. Aby bylo dosaženo realističtějšího efektu ohně, byl materiálu ještě přidán parametr „unshaded“, aby nebyl stínován. Druhé částici byl zvolen tvar „plane“, jehož velikost byla zvětšena aby přesahoval částici ohně a následně mu byl přidán nový materiál a textura kouře společně s parametrem „unshaded“. Materiálu byla nakonec přidána průhlednost, aby byl kouř realističtější.

Dalším krokem bylo vytvoření animace pro explozi. Byl vytvořen nový uzel typu „AnimationPlayer“ a čas animace byl nastaven na 3 vteřiny. První klíčový snímek obsahuje spuštění emise částic a poslední ukončení emise částic. Dále byly vytvořeny další dvě stopy animace. První stopa obsahuje postupné objevení a mizení první sady částic (ohnivé koule), tohoto efektu bylo dosaženo přidáním průhlednosti v parametru „albedo“ a na dalším klíčovém snímku opět odebrání průhlednosti a v poslední částí opět zprůhledněním částic. Tím vzniká efekt, kdy se ohnivá koule plynule objeví a plynule zmizí. Stejný postup byl aplikován i v druhé stopě animace na kouř. Aby se animace spustila při vstupu uzlu do scény, byl vytvořen skript, který animaci automaticky spustí (Obr. 31).



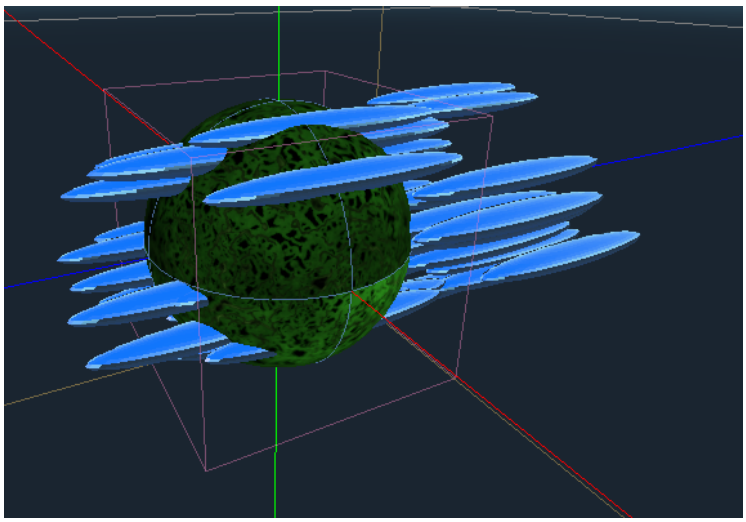
Obr. 31: Tvorba animace výbuchu

Dále bylo nutné výbuch zakomponovat také do skriptu nepřátelské lodi. Instance výbuchu se vytvoří vždy pokud je zdraví lodi menší nebo rovno 0.

Postup tvorby výbuchu asteroidu byl totožný a jediným rozdílem je, že u něj chybí ohnivá koule.

5.9.2 Plasma

Stejně jako výbuch i plasmová koule využívá částicový systém k vykreslení stopy za tělesem. Model plasmy a také model částice byl vytvořen v Blenderu a textura v GIMPu. Následně byly modely importovány do Godotu. A byl vytvořen nový částicový systém kterému byl přiřazen model částice vytvořený v Blenderu. Bylo nejprve třeba změnit některé parametry aby částicový systém vypadal věrohodně. Byla změněna gravitace na ose Z na hodnotu -1, aby částice plynuly správným směrem, dále množství částic je 50 a náhodnost má hodnotu 1. Tvar emise byl nastaven na kouli (Obr. 32).



Obr. 32: Plasma s částicovým systémem

5.10 Druhá úroveň

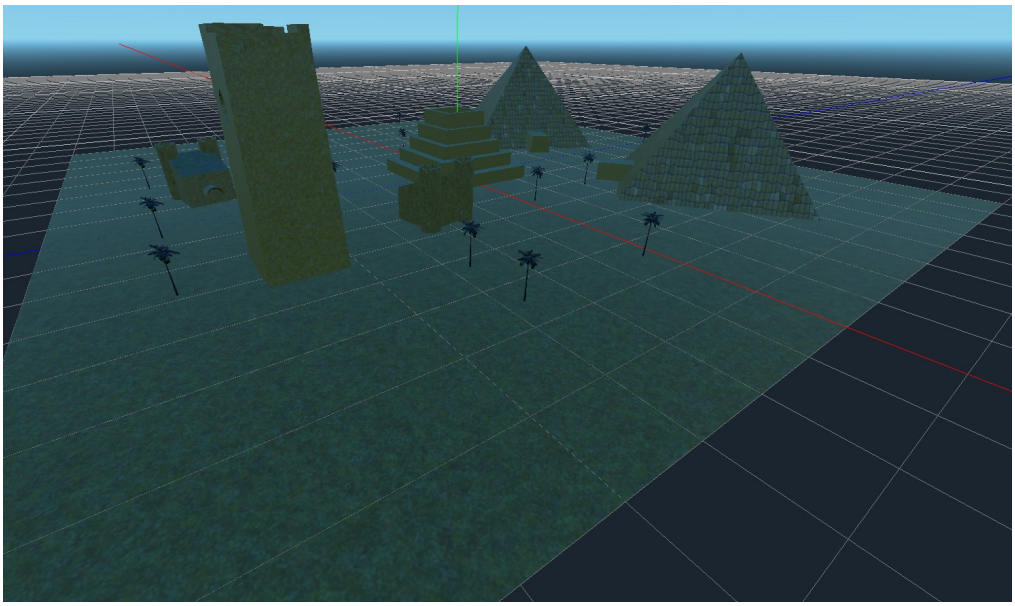
Druhá úroveň byla navržena, tak aby ukázala co nejvíce grafických možností Godot Enginu. Jako lokace byla zvolena planeta s pouští, velkým městem plným budov a písečnými dunami. Na hráče zde čekají nejen nástrahy v podobě nepřátelských lodí, ale také nástrahy prostředí, v podobě velkých červů, kteří vystřelují z písku.

Tvorba druhé úrovně započala v Blenderu, kde bylo vytvořeno pět stejně velkých části pouště, které na sebe plynule navazovaly. Jedná se o část s písečnými dunami, plochou částí pouště, vstupní bránou do města, pláží a jezírkem. Následně byly vytvořeny také budovy a červí díry. Palmy použité v druhé úrovni byly získány z: https://opengameart.org/content/palm-tree-v2-low-poly-edition. Veškeré textury pro modely byly vytvořeny v programu GIMP.

Druhá úroveň používá stejnou základní logiku jako první úroveň, proto byla scéna s první úrovně duplikována, přejmenována a použita jako základ pro tvorbu druhé úrovně. Duplikován a přejmenován musel být také skript hlavní lodi, jelikož by se při změně skriptu projevily změny také v první úrovni. Dále byly vymazány všechny uzly, které nebudou potřeba pro druhou úroveň. Následně byly všechny modely importovány do Godotu. Oproti importům do první úrovně, byl tentokrát využit formát DAE (Collada), jelikož u formátu glTF byl problém s deformací objektů při použití modifikátorů.

V Godotu byly skládány dohromady jednotlivé části pouště a ostatní objekty. Vždy se začalo otevřením scény s částí pouště, do které následně byly instancovány požadované objekty jako budovy a palmy (Obr. 33). Každá hotová scéna s části úrovně byla uložena

zvlášť a vzniklo, tak celkem 40 částí úrovně. Tyto části pak byly instancovány do separované scény s názvem „ground_anim“. Všechny instancované scény byly poté spojeny dohromady rodičovským uzlem typu „Spatial“, aby se s nimi lépe manipulovalo. Nakonec se vytvořila animace pomocí „AnimationPlayer“, kdy se všechny díly úrovně posouvaly po ose Z (směrem ke hráči). Výsledná scéna s animací byla instancována do hlavní scény úrovně. Aby se spustila animace, byla ve skriptu hlavní lodi změněna funkce pro spuštění pohybu vesmíru na spuštění pohybu podkladu druhé úrovně.

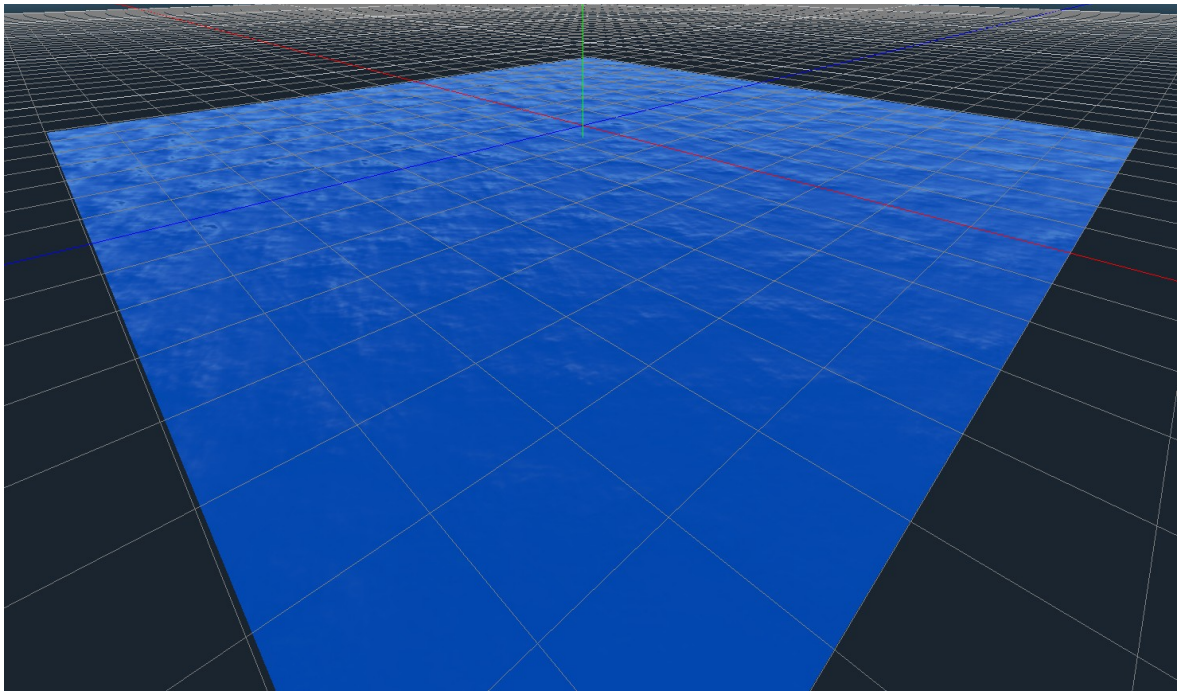


Obr. 33: Hotová část druhé úrovně

5.10.1 Voda

V této úrovni se nachází jak poměrně rozsáhle vodní plochy, tak i malá jezírka, proto bylo nutné vytvořit vodní plochu, která by vypadala co nejvíce realisticky. Nejprve byla v GIMPu vytvořena normálová mapa a ta byla importována do Godotu.

Nejprve byla vytvořena scéna s kořenovým uzlem „Spatial“, jeho potomkem je „MeshInstance“, kterému byl zvolen model typu „Plane“. Vytvořený model byl obarven modrou albedo barvou a parametr „Roughness“ byl nastaven na hodnotu 0,2, aby se voda leskla. Pro tvorbu realisticky vypadajících vln byla využita připravená normálová mapa (Obr. 34).



Obr. 34: Vodní plocha s aplikovanou normálovou mapou

V tomto stavu je ale vodní hladina bez pohybu, proto byla vytvořena animace, kde byl na prvním klíčovém snímku nastaven „Offset“ normálové mapy na hodnotu 0,0,0 a na koncovém snímku na 0,1,0. Tím se plynule posouvá normálová mapa a vzniká tak iluze pohybu vln. Stejným způsobem byla vytvořena také voda pro jezírko, ale s jinou a především průhlednější barvou, aby bylo vidět na dno.

5.10.2 Druhý typ nepřátelské lodí

V druhé úrovni se objevuje nový typ nepřátelské lodi, ten byl navržen, tak aby zabíral více prostoru a bylo tak pro hráče těžší jej obletět a tím narůstá obtížnost. Tato loď je navíc vybavena plasmou, která má větší poškození než laser.

Pro tvorbu byl využit opět model lodě ze sady modelů, která byla využita při tvorbě hlavní lodi a nepřátelských lodí. Základní princip kontroly kolize je totožný s prvním typem lodi, proto byla scéna s první lodí duplikována a použita jako základ pro druhý typ. Po úpravě kolizních oblastí a velikosti modelu, byly upraveny údaje o kolizi také ve skriptu hlavní lodi, aby nový typ bral v potaz. Samotný skript druhého typu opět vychází ze skriptu prvního typu.

Když byla kontrola kolize a loď funkční, začalo skriptování plasmové střely, kterou bude loď střílet. Vycházelo se ze scény s plasmou, kterou používá hlavní loď, ale byla změněna gravitace částicového systému na opačnou stranu, stejně tak byl ve skriptu

změněn směr pohybu směrem ke hráči a změněna kontrola kolize na kontrolu kontaktu s hráčem. Tím byla scéna s plasmou připravena. Pro samotné střílení lodi byl využit totožný princip jako u hlavní lodi, tentokrát ale byla loď doplněna o uzel „Timer“ neboli časovač, kdy při uplynutí jedné vteřiny vyšle signál do skriptu lodi a provede se instance scény s plasmou, tím loď každou vteřinu vystřelí plasmu (Obr. 35).

```
func _on_Timer_timeout(): #Po vypršení času v časov
    if get_node("Waves").is_playing(): #Pokud je an
        var EnemyPlasma=EnemyPlasmaScene.instance()
        get_parent().add_child(EnemyPlasma) #Přiřad
        EnemyPlasma.translation=(translation)
```

Obr. 35: Skript pro střelbu plasmy nepřátelskou lodí

Pohyb nepřátelských lodí byl stejně jako v první úrovni řešen animací nepřátel. Rozdílem je spouštění jednotlivých vln. Spouštění vln zde neprobíhá na základě času, ale na základě pozice, proto jsou v druhé úrovni, v různých částech rozmístěny oblasti, do kterých jakmile hráč vstoupí, spouští se určitá vlna. Spouštění se již neřeší ve skriptu hráče, ale v dané části úrovně.

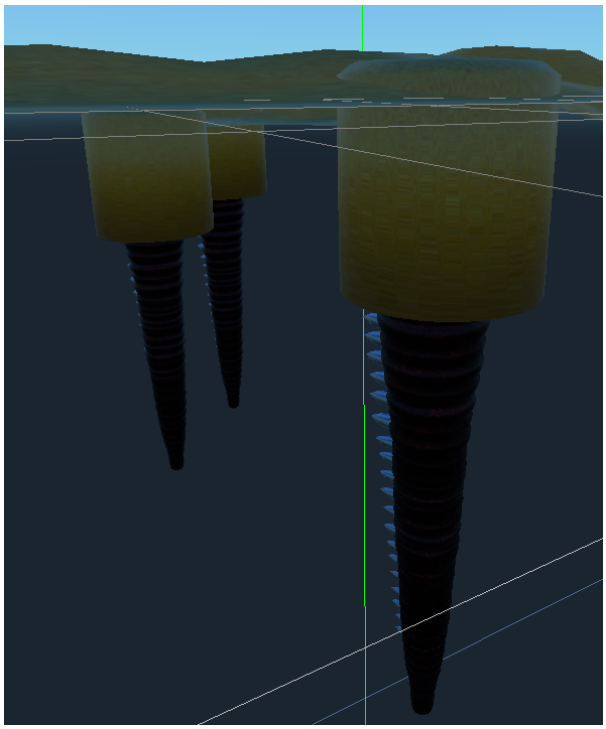
5.10.3 Červy

Kromě druhého typu lodě je ve druhé úrovni další nepřítel v podobě červa, který vystřeluje z písku směrem k hráči. Během hraní úrovně se dá poznat odkud červ vystřelí podle kruhových útvarů v písku.

Nejprve byl v Blenderu vytvořen model červa a kruhových útvarů, které reprezentují červí díry. Následně byly modely importovány do Godotu. Ve scéně s červem byly vytvořeny dvě animace první je jednoduchá rotace zubů červa (ta se spouští při vstupu červa do scény) a druhou je pohyb červa směrem vzhůru ke hráči. Dále se ve scéně nachází dvě oblasti. Jedna míří vysoko do vzduchu a slouží ke spuštění animace útoku (pokud hráč vstoupí do oblasti, spustí se animace). Druhá oblast se nachází těsně nad hlavou červa a slouží ke spuštění funkce poškození u hráče. Hotová scéna s červem musela být duplikována, jelikož pokud by se prostě ve scéně rozkopírovala pak by došlo při kontaktu s jedním červem ke spuštění animace všech ostatních (stejný skript). Proto se červy nachází v separovaných scénách.

Rozmístění červů probíhalo nejprve rozmístěním jednotlivých červích děr do úrovně (nachází se v poslední části úrovně) a následným instancováním červů na jejich

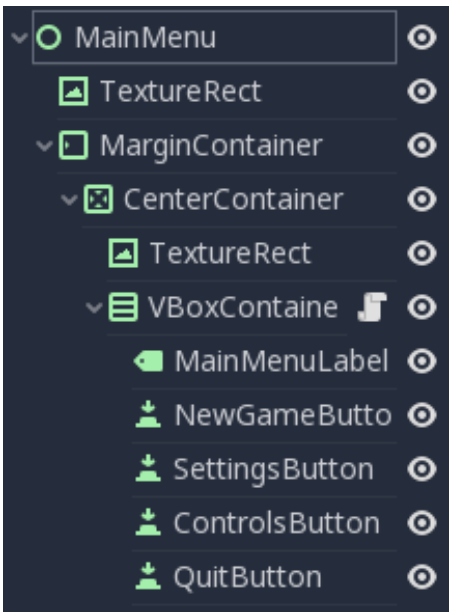
pozici do scény (Obr. 36).



Obr. 36: Rozmístění červů ve scéně

5.11 Uživatelské rozhraní

Celkem bylo do hry zakomponováno šest scén s uživatelským rozhraním a ovládacími prvky pro rozhraní. Jedná se o Hlavní menu, Nastavení, Ovládání a dvě scény po splnění úrovní. Nejprve bylo vytvořeno hlavní menu. Do nové scény byl vložen uzel typu „Control“, kterému se pak přidal potomek „TextureRect, jemuž byla přiřazena textura vesmíru, tato textura má za úkol vyplnit prostor, pokud by bylo uživatelské rozhraní zvětšeno nad hranici HD rozlišení (1280x720 pixelů). Dalším potomkem uzlu „Control“ je pak „MarginContainer“, který určuje hranice uživatelského rozhraní. Poté následuje uzel „VboxContainer“, který zarovnává své potomky do řádků, jeho potomky jsou tlačítka a text. Uživatelské rozhraní má toto uspořádání uzlů z důvodu možné změny rozlišení obrazovky, kdy prvky po změně zůstanou na svých místech (Obr. 37). Všechny ostatní scény uživatelského rozhraní vychází právě z této hierarchie. Pro kontrolu kliknutí na jednotlivá tlačítka je využita funkce „pressed“. Ve skriptu scény s nastavením hry „Settings“ je možné přepínat mezi zobrazením na celou obrazovku (full screen) nebo zobrazením v okně, pomocí funkce „OS.window_fullscreen“. Font použitý při tvorbě rozhraní byl získán z: http://www.1001fonts.com/autobus-bold-font.html a tvůrcem je NimaVisual.



Obr. 37: Struktura uzlů hlavního menu

5.11.1 HUD

HUD (Heads up display) vypisuje hráči na obrazovku informace o zdraví, skóre a aktuální zbrani. Základem HUD je stejně jako u scén uživatelského rozhraní uzel „MarginContainer“ pro nastavení celkové velikosti. Potomky „MarginContainer“ jsou „HboxContainer“ a „VboxContainer“ pomocí nich jsou textové prvky zarovnány do rohů obrazovky (Obr. 38).

Obr. 38: Hotový HUD zobrazený v editoru

5.12 Audio

Audio se skládá ze tři zvukových stop. Jedna pro hlavní menu, druhá pro první úroveň a třetí pro druhou úroveň. Zvukové soubory byly převedeny na formát OGG pomocí programu Audacity. Zvuk byl implementován v každé ze scén pomocí uzlu „AudioStreamPlayer“, kterému byl přiřazen zvukový soubor a pomocí parametru „Autoplay“ jsou zvukové soubory přehrávány vždy při vstupu do scény.

Jedná se o hudební doprovod hry. Všechny tři nahrávky byly získány z: dig.ccmixter.org a jsou distribuovány pod licencí CC BY 3.0. Všechny nahrávky a jejich autoři jsou uvedeni níže:

- Orc March by basematic (c) copyright 2011 Licensed under a Creative Commons Attribution (3.0) license. http://dig.ccmixter.org/files/basematic/33579 Ft: snowflake, wolf sebastian, spinning merkaba
- Welcome in the intox by Bluemillenium (c) copyright 2018 Licensed under a Creative Commons Attribution (3.0) license. http://dig.ccmixter.org/files/Bluemillenium/57202
- Weathered Faces by @nop (c) copyright 2016 Licensed under a Creative Commons Attribution (3.0) license. http://dig.ccmixter.org/files/Lancefield/54886

5.13 Export hry

Výsledná ukázková hra bude k dispozici pro platformy GNU/Linux a Windows Desktop. Před exportem aplikace bylo nejprve nutné nainstalovat „export templates“, neboli šablony pro export. Šablony pro export si Godot stáhne a nainstaluje automaticky po potvrzení uživatelem a pokud ne, dají se získat z oficiální stránky Godotu (https://godotengine.org/download/linux).

Následně v nabídce „Export“ byly zvoleny požadováné platformy tlačítkem „Add...“. Veškerá nastavení zůstala na výchozích hodnotách, avšak při vybírání umístění jednotlivých souboru byla vypnuta možnost „Export With Debug“, tím se aplikace neexportuje s ladícím programem a tím se zmenší i výsledná velikost. Výsledné spustitelné soubory jsou ve formátu EXE pro Windows a x86_64 pro GNU/Linux. Hra běží pouze na 64bitových operačních systémech.

ZÁVĚR

Cílem této práce bylo blíže představit Godot Engine, jeho funkce, vlastnosti a možnosti jeho praktického využití, které byly demonstrovány vytvořenou aplikací.

V teoretické části práce byl nejprve engine blíže představen, včetně jeho základních vlastností a možností. Dále byla přiblížena problematika exportu aplikací a platformy, které jsou pro exportu podporovány, včetně jednotlivých grafických API, které jsou enginem podporovány nebo ty které budou podporovány v budoucnu. Dále bylo popsáno pracovní prostředí enginu a jeho ovládací prvky.

Další část práce se blíže zaměřila na jednotlivé nástroje, které Godot Engine nabízí. Nejprve byla vysvětlena role uzlů a následně scén, ze kterých se projekt skládá. Poté byl představen správce souborů Godotu a možnosti importu souborů do projektu. Následně bylo popsáno 2D prostředí, včetně důležitých uzlů, které do něj spadají a funkce specifické pro 2D prostředí. Čtenář byl seznámen se 3D prostředím a možnostmi importu 3D modelů, kde byly také popsány jednotlivé formáty pro import, které Godot v současnosti podporuje. Dále byla popsána manipulace s 3D objekty. Ke 3D prostředí neodmyslitelně patří také světla, která byla blíže popsána. U 3D prostředí byly také představeny efekty post-processingu a jejich možné využití k dosažení realistické grafiky. Byly vysvětleny možnosti kombinace 2D a 3D prostředí. Dále se práce zabývala problematikou viewportů a kamer. Důležitou částí enginu je také fyzikální engine, u kterého byl vysvětlen rozdíl mezi 2D a 3D fyzikou a důležité uzly, které k ní patří. Čtenář byl seznámen s animacemi v Godotu a jejich možnostmi. Práce se dále zabývala problematikou uživatelských vstupů a typy vstupů. Následně bylo popsáno skriptování v Godotu a byly představeny jednotlivé jazyky, které mohou být ke skriptování využity. Další část práce se věnovala pluginům a možnostmi zpracování zvuku. V závěru teoretické části byl blíže představen další software použitý při tvorbě praktické části práce.

V praktické části byla nejprve popsána tvorba české dokumentace ke Godot Engine a bylo popsáno její členění.

V další části byl čtenář seznámen s postupem tvorby ukázkové hry, na které byla demonstrována široká škála nástrojů Godotu. Nejprve byla hra představena a byla popsána tvorba konceptu hry. Následně byl vysvětlen postup tvorby první úrovně, včetně nepřátel, hráčem řízené lodi, zbraní, detekce kolizí a částicového systému. Následně byl stejným způsobem popsán postup tvorby druhé úrovně, navíc byl vysvětlen postup při tvorbě nových typů nepřátel a vodních ploch. Dále byla rozebrána tvorba uživatelského rozhraní

hry a postup přidání hudby. V poslední části praktické práce byl přiblížen export hotové aplikace.

Hra samozřejmě není dokonalá a dala by se v budoucnu dále rozšiřovat a zlepšovat. Daly by se přidat především nové úrovně a nepřátelé, přičemž stávající úrovně by mohly být lépe optimalizovány. Dále by šla hra rozšířit například na mobilní platformy jako jsou Android či iOS.

SEZNAM POUŽITÉ LITERATURY

[1] Godot Docs [online]. Brooklyn, NY: Juan Linietsky, 2018 [cit. 2018-05-16]. Dostupné z: http://docs.godotengine.org/en/3.0/

[2] GameFromScratch.com [online]. 2018 [cit. 2018-05-16]. Dostupné z: http://www.gamefromscratch.com/

[3] Mostly codeless game development: new school game engines. New York, NY: Springer Science Business Media, 2017. ISBN 978-1-4842-2969-9.

[4] What Is a Graphics API?. Samsung Developers [online]. Soul, Jižní Korea: Samsung, 2018 [cit. 2018-05-16]. Dostupné z: https://developer.samsung.com/tech-insights/vulkan/what-is-a-graphics-api

[5] Godot Engine [online]. Brooklyn, NY: Juan Linietsky, 2018 [cit. 2018-05-16]. Dostupné z: http://www.gamefromscratch.com/

[6] OpenGL ES 3.0 Programming Guide: Edition 2. 2. Boston, Massachusetts, USA: Addison-Wesley Professional, 2014. ISBN 9780133440126.

[7] Grafická knihovna OpenGL (1). Root.cz [online]. Praha: Internet Info, s.r.o, 2018 [cit. 2018-05-16]. Dostupné z: https://www.root.cz/clanky/graficka-knihovna-opengl-1/

[8] EDITED BY PATRICK COZZI a Christophe RICCIO. OpenGL Insights. Hoboken: CRC Press, 2012. ISBN 978-143-9893-777.

[9] SINGH, Parminder. Learning Vulkan. Birmingham, UK: Packt Publishing, 2016. ISBN 9781786460844.

[10] Post-Processing Video Effects. TVTropes [online]. San Diego: Tropes HQ, 2018 [cit. 2018-05-16]. Dostupné z: http://tvtropes.org/pmwiki/pmwiki.php/Main/PostProcessingVideoEffects

[11] JONES, M. Tim. Open source physics engines. IBM DeveloperWorks [online]. New York, USA: IBM, 2018 [cit. 2018-05-16]. Dostupné z: https://www.ibm.com/developerworks/library/os-physicsengines/

[12] Computer animation. ScienceDaily [online]. Rockville, MD: ScienceDaily, 2018 [cit. 2018-05-16]. Dostupné z: https://www.sciencedaily.com/terms/computer_animation.htm

[13] Learn About Cutout Animation. Lifewire [online]. New York: Dotdash, 2018 [cit. 2018-05-16]. Dostupné z: https://www.lifewire.com/what-is-cutout-animation-140519

[14] Scripting language. TechTarget [online]. Newton, MA: SearchWinDevelopment, 2018 [cit. 2018-05-16]. Dostupné z: https://searchwindevelopment.techtarget.com/definition/scripting-language

[15] Intro to particle systems. Khan Academy [online]. United States: Khan Academy, 2018 [cit. 2018-05-16]. Dostupné z: https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-particle-systems/a/intro-to-particle-systems

[16] Game Engine Design and Implementation. Burlington, Massachusetts, USA: Jones & Bartlett Publishers, 2011. ISBN 9781449666484.

[17] What is a Game Engine?. GameCareerGuide [online]. San Francisco, CA: UBM, 2018 [cit. 2018-05-16]. Dostupné z: https://www.gamecareerguide.com/features/529/what_is_a_game_.php

[18] GIMP [online]. Orinda Way Ste. C Orinda, CA: GNOME Foundation, 2018 [cit. 2018-05-16]. Dostupné z: https://www.gimp.org/

[19] Blender [online]. Amsterdam: Blender Foundation, 2018 [cit. 2018-05-16]. Dostupné z: https://www.blender.org/foundation/

[20] SILBER, Daniel. Pixel art for game developers. Boca Raton: CRC Press, 2016. ISBN 978-148-2252-309.

[21] SCHELL, Jesse. The art of game design: a book of lenses. Boston: Elsevier/Morgan Kaufmann, c2008. ISBN 978-012-3694-966.

[22] LECARME, Olivier. a Karine. DELVARE. The book of GIMP: a complete guide to nearly everything. San Francisco: No Starch Press, 2013. ISBN 978-159-3273-835.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
Ssao	Screen-Space Ambient Occlusion
SSR	Scree-Space-Reflections
MIT	Massachusetts Institute of Technology
GNU	GNU's Not Unix
GPL	General Public License
CC	Creative Commons
SA	Share-alike
HDR	High Dynamic Range
HD	High Definition

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P 1: ADRESÁŘOVÁ STRUKTURA PŘÍLOH

-Binaries – obsahuje spustitelné soubory pro Windows a GNU/Linux

-Development – soubory použité pro vývoj hry (3D modely, textury, zvuky…)

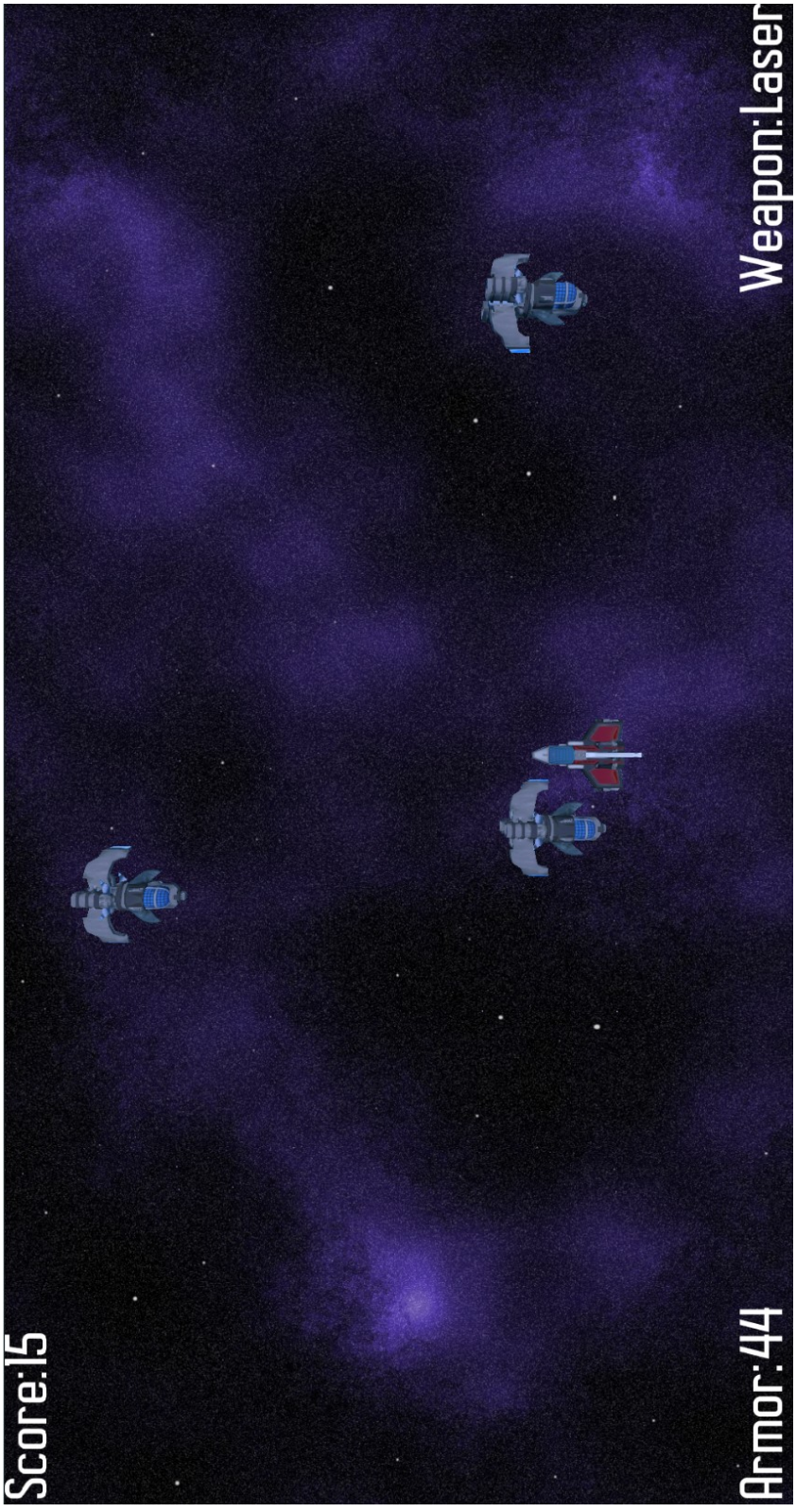
- Bonuses – soubory spojené s tvorbou bonusů
- Effects – soubory spojené s tvorbou efektů
- Fonts – fonty použité pro text
- Icons – ikona aplikace
- Levels – veškeré soubory využity při tvorbě první a druhé úrovně
- Posters – obrázky využity při tvorbě uživatelského rozhraní
- Ships – sada lodí použitá pro tvorbu lodí
- Weapons – soubory spojené s tvorbou zbraní

-Godot project – obsahuje samotný projekt vytvořený v Godotu, který lze do Godotu importovat (níže je popsán obsah adresáře „Assets“, který obsahuje všechny soubory projektu)

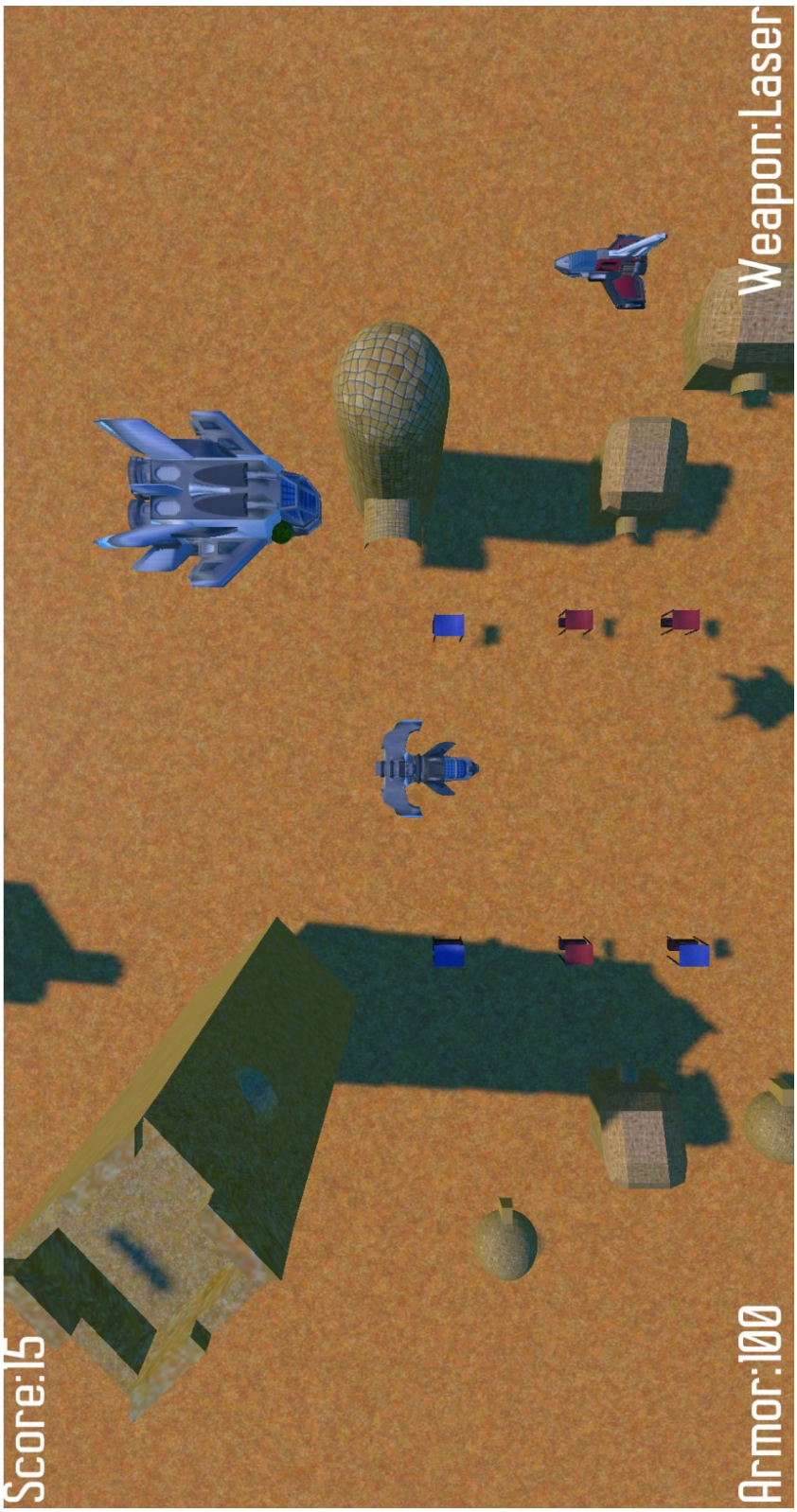
- Fonts – fonty využité pro text
- Scenes – scény v nativním formátu Godotu (.tscn) + další soubory spojené se scénami
- Scenes_raw – adresář byl využit pro importované scény, které se dále uložily do nativního formátu Godotu a následně byly smazány, nyní obsahuje pouze použité materiály a některé scény.
- Scripts – Obsahuje globální skript
- Sound – obsahuje zvukové soubory

Text – obsahuje samotný text bakalářské práce a dokumentace ve formátu PDF a ODT.

PŘÍLOHA P 2: PRVNÍ ÚROVEŇ HRY



PŘÍLOHA P 3: DRUHÁ ÚROVEŇ HRY



PŘÍLOHA P 4: HLAVNÍ MENU HRY