Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

2D hra žánru Space Combat

2025

Vojtěch Jiříček

Vedoucí práce:
Mgr. Radek Janoštík, Ph.D.

Studijní program: Informatika,
Specializace: Programování a vývoj software

Bibliografické údaje

Autor:	Vojtěch Jiříček
Název práce:	2D hra žánru Space Combat
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2025
Studijní program:	Informatika, Specializace: Programování a vývoj software
Vedoucí práce:	Mgr. Radek Janoštík, Ph.D.
Počet stran:	34
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Vojtěch Jiříček
Title:	2D Space Combat game
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2025
Study program:	Computer Science, Specialization: Programming and Software Development
Supervisor:	Mgr. Radek Janoštík, Ph.D.
Page count:	34
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Počítačové hry, mobilní hry, arkádové kabinety; podoba je různá, poslání jednotné: zabavit člověka v nejinteraktivnější formě digitální zábavy dnes. Každý titul je osobitý, a to nejen svou podobou, ale i vývojovým procesem, který odráží jedinečnou vizi a cíle tvůrce. Tato práce se zaměřuje na historii her a vývojové postupy, které byly nebo stále jsou běžně používány při jejich tvorbě. Dále je představen herní engine Godot a v něm implementovaná videohra, která prezentuje využití těchto postupů v praxi. Jsou zmíněny všechny aspekty hry, od audiovizuálního zpracování až po interní funkcionalitu jednotlivých systémů a jejich propojení.

Synopsis

Computer games, mobile games, arcade cabinets; forms vary, but purpose unified: to entertain people in the most interactive form of digital entertainment to date. Each title is unique, not only in its appearance, but also in the development process, which reflects creator's distinct vision and goals. This thesis focuses on the history of games and the development practices that have been or still are commonly used in their creation. Additionally, the Godot game engine is introduced, along with a video game implemented within it, demonstrating the practical use of these practices. All aspects of the game are discussed, from audiovisual design to the internal functionality of individual systems and their integration.

Klíčová slova: video hra; historie video her; vývoj počítačové hry; herní enginy; herní mechaniky; Godot Engine

Keywords: videogame; videogame history; computer game development; game engines; game mecahnics; Godot Engine

Rád bych poděkoval vedoucímu práce Mgr. Radku Janoštíkovi, Ph.D. za jeho vstřícnost a trpělivost. Rovněž bych chtěl poděkovat své rodině za jejich trvalou podporu.

Odevzdáním tohoto textu jeho autor/ka místopřísežně prohlašuje, že celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

Seznam obrázků

1 Videohry

Od experimentů s prvními počítači na akademické půdě, například na univerzitách MIT, Cambridge a dalších [1]. Videohry velmi rychle přerostly rámec nekomerčních aplikací v jednu z nejvlivnějších forem moderní kultury. V posledních letech se herní průmysl stal jedním z nejvýnosnějších odvětví zábavního průmyslu. Svou monetární silou [2] dokonce předčil tradiční média jako filmová [3] a hudební [4]. Ať už jde o počítačové, konzolové, arkádové, nebo nejnovější platformy jako mobilní hry a virtuální realita, s hrami je možné se setkat opravdu kdekoliv.

Počet hráčů je obrovský, jen na platformě Steam, jedné z hlavních distribučních platforem, je denně přihlášeno přes 30 milionů různých uživatelů [5]. Za zmínku stojí i počty prodaných her, příkladem může být herní série Grand Theft Auto, která zaznamenala prodeje přesahující stovky milionů kusů, nebo značky firmy Nintendo jako Mario, Zelda a Pokémon, které v prodejích také nezaostávají a každý nový díl série má vždy na kontě desítky milionů prodaných kopií.

Jednou z prvních her byla Spacewar!, poprvé vytvořená na MIT roku 1962 pro počítač PDP-1. Její koncept je velmi jednoduchý, dva hráči ovládají kosmické lodě, které se navzájem snaží zničit. Ozvláštnění přidává hvězda uprostřed, která svou gravitací ovlivňuje pohyb lodí. Spacewar! byla díky své jednoduchosti a popularitě na půdě MIT velkou inspirací pro první hry, například Computer Space, která je její přímou napodobeninou a někdy neprávem označována za první videohru. Klon Spacewar! je součástí této práce, a to nejen jako technický projekt, ale také jako příklad starého herního designu, který stále může ovlivnit moderní videohry.

Obrázek 1: Obrázek jednoho z prvních hraní hry Spacewar! [6]

Pro mě je hraní her nedílnou součástí mého života. Hry nevnímám jen jako příležitost k odreagování, ale také jako možnost poznávání nových konceptů, získání inspirace a naučení se nových poznatků. Takže jsem měl možnost promítnout své nabyté zkušenosti do práce, která nebyla jen povinností, ale i zábavou.

2 Vývoj her

Vytváření her je značně komplikovaný proces zahrnující široké spektrum oborů, například grafiku, UI (User Interface) a UX (User Experience) design, navrhování mechanik, modelování, vytváření textur, skriptování a další. Každý tento aspekt hraje svou roli ve finálním produktu. Programování je pak stěžejním základem všeho, protože právě to orchestruje všechny zmíněné elementy.

2.1 Herní engine

Herní engine je jádro hry, starající se o vnitřní funkce a poskytující systémy potřebné pro její fungování. To zahrnuje zejména rendering, animace, fyzikální a kolizní systémy, správu zvuku a kostru uživatelského rozhraní. Díky těmto systémům se lze více zaměřit na kreativní proces, zatímco nízkoúrovňové technické aspekty jsou automaticky zajišťovány enginem samotným. Samotný výběr enginu je taktéž jedním z aspektů vývoje her, a to poměrně zásadním, protože se od něj odvětvují dostupné poskytované funkce, architektura aplikace, programovací jazyky nebo kompatibilní platformy.

Většina **AAA**, označení pro vývojářská studia nebo vydavatele s velkým rozpočtem a vysokým počtem zaměstnanců, studia mají, nebo měly vlastní enginy, příkladem mohou být EA (Electronic Arts) s enginem Frostbite [7], nebo Valve se Source Enginem [8]. Tyto jsou většinou veřejnosti buď zcela nedostupné, nebo ve velmi omezené míře v podobě nástrojů pro úpravu specifických částí nebo modifikaci her.

Vytvoření vlastního enginu přináší řadu výhod, jako je plná kontrola nad architekturou a možnost optimalizací. Díky tomu lze implementovat mimořádně specifickou funkcionalitu a přizpůsobit výkon přesně podle požadavků projektu. Finská společnost Remedy Entertainment, specializující se na příběhové akční hry, je příkladem studia, které těží z těchto benefitů a jejich engine Northlight je jedním z průkopníků moderních metod [9].

Udržování **in-house** enginů je nesmírně náročné. Reagování na změny v hardwaru a podpora všech nejnovějších grafických funkcí zabírá velké množství prostředků. Při dnešním rychlém pokroku technologií se vlastní řešení velmi rychle stávají zastaralými a jejich technologický dluh výrazně brzdí všechny možnosti spojené s vývojem hry. Tento problém se stává čím dál tím větší výzvou i pro velkorozpočtové firmy, ty často nemají jinou možnost než stávající engine definitivně opustit. Příkladem je herní série Arma od českého studia Bohemia Interactive, která dlouhá léta využívala engine Real Virtuality, původně vyvinutý před téměř 30 lety. S postupem času však tento engine dosáhl svých limitů, což začalo brzdit jak technický pokrok, tak ambice vývojářů a simulační hloubku. Ohlasy na třetí díl této série byly převážně pozitivní, nicméně komunita vyjádřila nespokojenost se zastaralým rozhraním a nedostatečnou podporou pro pokročilé modifikace. Jako řešení se objevila potřeba vytvořit zcela nový engine, což vedlo k vývoji enginu Enfusion [10].

Velký problém bývá i v možnosti náboru nových vývojářů. Přestože je mnoho postupů je ustálených, při využití vlastního enginu je zaučovací doba nových členů týmu mnohem delší. Dnes si mnoho studií vývoj nebo údržbu enginu outsourcuje, což vede k odklonu od vlastních enginů a přechodu k univerzálním řešením [11] [12]. Pro jednočlenné vývojáře a malé týmy, obecně nazývané **Indie**, bývá tato bariéra často nepřekonatelná, a tvorba vlastního enginu se v tomto prostředí vidí velice zřídka.

Existuje široká škála nástrojů, které se liší svou komplexností a účelem. Od lehkých a jednoduchých nástrojů, často zaměřených na specifické žánry, patří například RPG Maker, zaměřený na tvorbu titulů s nádechem konzole SNES, nebo Ren'Py, pro vytváření vizuálních novel. Na opačném konci spektra stojí komplexní sady provázaných nástrojů a softwarových celků, jako jsou Unity a Unreal Engine, které jsou jedněmi z největších a nejznámějších zástupců volně dostupných enginů.

2.2 Objektově orientované programování

S příchodem programovacích jazyků jako jsou C++ a Smalltalk se samozřejmě prvky OOP (Objektově orientované programování) přenesly i do vývoje her. Zde je jejich výskyt velmi přirozený, protože herní prvky jsou ve své podstatě objekty. Například postava hráče může být samostatný objekt obstarávající uživatelský vstup. Přesnější využití jednotlivých principů OOP ve vývoji her je popsáno níže.

3 Godot Engine

Pro účely této práce byl zvolen Godot Engine. Svou komplexností spadá někde mezi právě zmiňované, není pouze specializovaný na jeden účel, ale zároveň není přeplněný funkcemi, které by pro začínající vývojáře byly bariérou pro naplnění jejich vizí [13].

Godot Engine je open source herní engine. Nabízí možnost vytvářet 2D a 3D hry. Obsahuje všechny nutné nástroje pro vývoj her, včetně podpory hry více hráčů po síti a nejmodernějších renderovacích metod. Díky benevolentní licenci MIT, má každý naprosto volnou ruku v jeho užití či úpravě v soukromých nebo komerčních účelech. Další předností je jeho malá velikost a nízké nároky na systém, to platí nejen pro samotné hraní her na něm postavených, ale i pro počítač, který dané hry exportuje. Je tedy vhodný jak pro začínající, tak pokročilé vývojáře, malé či velké projekty, kdy například editor je vytvořen v Godotu samotném.

Nejvýjimečnější vlastnost Godotu, oproti jiným enginům je jeho systém uzlů a scén. Uzly zastupují atomické prvky, které jsou skládány do stromových struktur zvaných scén. Vzniká pak hierarchická struktura, kde kořenová scéna může mít potomky v podobě jednotlivých koncových uzlů, nebo dalších stromů reprezentovaných dalšími scénami. V editoru je možné tyto stromy skládat a propojovat bez potřeby psaní jakéhokoliv kódu. Jako příklad je přiložen obrázek: 2 scény hráče z minihry a jeho podružných uzlů, takto vytvořená scéna může být opět vložena do úrovně.

Obrázek 2: Ukázka scény hráče

3.1 Uzly

Uzly jsou nejmenší stavební komponenty hry, jejichž skládáním do stromů postupně vzniká celá hra. Předpřipravené uzly je možné kombinovat, upravovat a rozvíjet za pomoci skriptů, každý má své jméno, typ, nastavitelné vlastnosti a výchozí chování. V řeči OOP by se samostatný uzel dal nazvat třídou. Strom dědičnosti v Godotu je velmi košatý. Jeho rozsáhlost je lépe znázorněna na přiloženém diagramu: 3.

Všechny předdefinované typy dědí ze základní `Object`, která slouží jako univerzální základ pro interakci s interními systémy v Godotu. Tato třída poskytuje funkce pro správu paměti, která je pak řešena manuálně, reagování na signály, pokročilou správu vnitřních stavů a možnost nést skript [14].

Na `Object` navazuje `Node`, základ pro všechny uzly. Jak už název napovídá, `Node` může být přidána do stromu scén a plně s ním interagovat, nejčastěji automatickým zavoláním metody `_ready` po přidání do stromu [15]. Mezi příklady uzlů odvozených od `Node` patří `Timer`, `AudioStreamPlayer`, nebo `CanvasItem`, který poskytuje metody pro vykreslování ve 2D. Na něj navazují `Control`, pro práci s uživatelským rozhraním a hlavně `Node2D`.

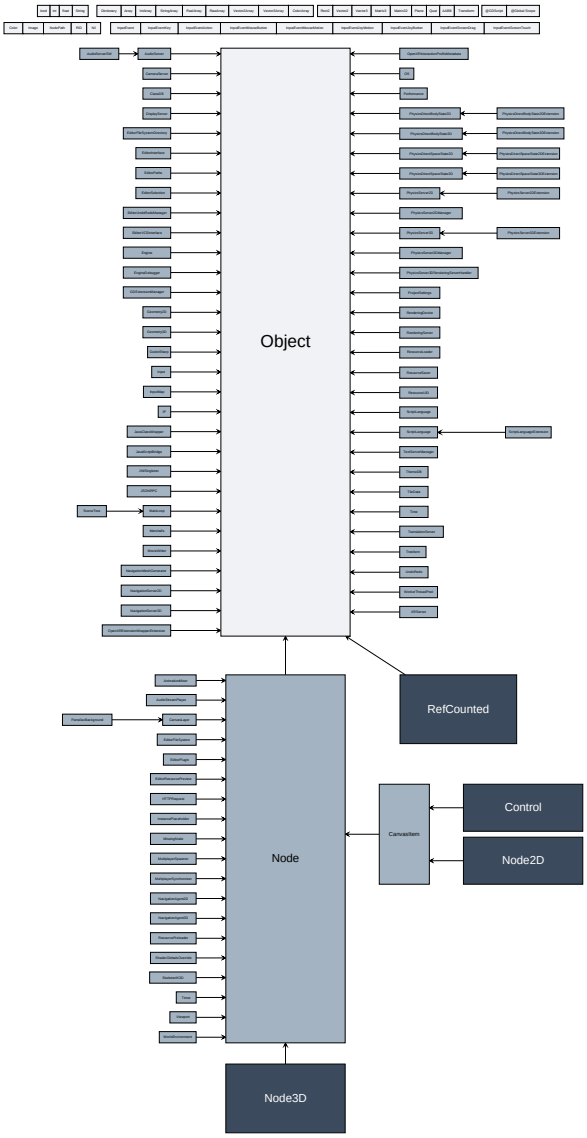
`Node2D` slouží jako základní objekt pro práci ve 2D prostoru a zajišťuje základní transformace, jako je pozice, rotace a měřítko. Tato báze je pak využívána všemi 2D objekty, například `Sprite2D`, `Camera2D`, nebo `Area2D`.

Každý uzel lze použít přímo, nebo jej upravit podle potřeb projektu. Přidáním skriptu k uzlu vzniká nový typ, který dědí veškerou funkcionalitu svého rodiče. Například ze základního `Sprite2D` lze vytvořit obrázek reagující na uživatelský vstup a pohybující se podle jednoduchých pravidel.

3.2 Scény

Spojení uzlů tvoří scénu [17]. Scéna je stromová struktura s jedním kořenovým uzlem a dalšími navazujícími uzly, toto uskupení je pak možné uložit do souboru s příponou .tscn a dále ji používat jako jeden nezávislý uzel, někdy označovaný termínem **prefab**, který je využíván v Unity [18], v kombinaci s dalšími uzly anebo scénami. Tam kde uzel plní jednu funkcionalitu, scéna díky spojení několika uzlů představuje složitější konstrukt s dalšími pravidly. A například pro 2D uzly ve scéně platí, že dědí transformace svého rodiče, což znamená, že pokud rodič provede například posun, posunou se i jeho potomci.

Systém scén také navádí k využívání komponentového principu [19, str. 289], kdy poskládáním nezávislých uzlů vznikne například hráčský charakter s uzly pro počet životů, texturou a zpracovávání kolizí. Scény jako takové tedy mohou představovat jednoduché i složité objekty, úroveň a v konečném důsledku je složení všech těchto jednotlivých scén hra samotná.



Obrázek 3: Dědičnost v Godot Engine [16]

3.3 SceneTree a herní smyčka

`SceneTree` [20] je jedním z nejdůležitějších objektů vůbec. Stará se o všechny uzly ve scéně, o jejich přidávání a odebírání. Lze jej využít k organizaci uzlů do skupin, které jsou adresovatelné na základě svého jména. `SceneTree` je vytvořen vždy při startu programu, poté je vytvořena kořenová scéna, na kterou jsou jeden po druhém navázány zapnuté AutoLoady a poté právě běžící scéna. `SceneTree` dědí a implementuje pro hry primární koncept, **Game Loop**, v češtině herní smyčku [19, str. 169], v Godotu zvanou `MainLoop`.

Hlavním posláním `MainLoop` je rozpojení běhu hry od uživatelského vstupu a procesorového času. Skoro každý herní engine ho nějakým způsobem implementuje. V principu herní smyčka běží neustále bez přestání a při každém prů-

běhu bez blokace zpracuje vstup od uživatele a aktualizuje herní systémy. Buďto v přesně daných intervalech, nebo co nejdříve je to možné, obecně při vykreslení dalšího snímku. Herní fyzika je často oddělena od zobrazování, což minimalizuje riziko situací, kdy by mohlo dojít k nepředvídatelnému chování hry.

Nechvalně proslulý příklad špatně provedeného oddělení těchto systémů může být populární The Elder Scrolls V: Skyrim. Creation Engine, na kterém je hra postavena, se při překonání hranice 60 snímků za sekundu nedokáže rozhodovat o správné pozici objektů a kolizí ve hře. To vede k nestabilitě, která hru činí prakticky nehratelnou, případně způsobí její kompletní pád [21].

3.4 GDScript

Dále je nutné zmínit možnosti využití interního skriptovacího jazyka GDScript, který byl navržen a vyvinut přímo pro Godot. Jedná se o vysokoúrovňový, objektově orientovaný jazyk s graduálním typováním. Jeho syntax je založena na odsazování, podobně jako v jazyce Python. Je velmi úzce spjatý s vnitřním fungováním Godot Enginu a tím poskytuje jak výkon, tak flexibilitu.

Interně je GDScript zaveden do Godotu jakožto modul a Godot může fungovat zcela bez něj. GDScript pouze komunikuje s vnitřními třídami [22]. Skripty, soubory s příponou .gd, jsou kompilovány do bytekódu [19, str. 213] interpretovaném vnitřním virtuálním strojem. To výrazně snižuje časové nároky na kompilaci, a i oproti konvenčnímu C# je mnohem rychlejší na sestavení. Co se samotné rychlosti týče, GDScript není určený pro kritické systémy a rychlost není primární. Což neznamená, že by nějakým způsobem omezoval funkčnost.

Pro účely skriptování a velmi rychlého prototypování je velmi vhodný. Navíc, díky přímé integraci do Godot Enginu, kdy je jeho součástí i textový editor pro práci s GDScript, není tedy nutné používat externí program pro psaní, úpravu a debugování kódu. Napsaný kód je navíc přímo propojený s prostředím editoru, a vestavěné proměnné je možné nastavovat graficky, což usnadňuje ladění. Z editoru je pak možné nahlédnout do uživatelské příručky, obsahující kompletní dokumentaci.

3.5 AutoLoad

`AutoLoad` v Godotu představuje implementaci **Singleton** patternu [19, str. 102]. Systém scén v Godotu je silný, ovšem má jednu slabinu. Co když je potřeba komunikovat s jednou scénou z vícero míst, nebo nezávisle posílat specifické scéně informace? To řeší právě `AutoLoad`, který vytvoří globální objekt viditelný pro všechny scény. Pokud je nějaký uzel designován jako `AutoLoad`, je automaticky přítomen bez ohledu na právě běžící scénu. Díky tomu je možné ukládat globální proměnné, nebo přepínat mezi scénami bez nutnosti neustále ukládat a načítat data z persistentního úložiště.

3.6 Signály

Signály jsou zprávy, které uzly vysílají, pokud se stane nějaká specifická událost, třeba stisknutí tlačítka. Ostatní uzly se mohou k signálu připojit a při jeho zachycení spustit určitou funkci. Uvnitř objektu tak nemusí být referencován druhý objekt, stačí pouze reagovat na změny jím provedené. Příkladem může být třeba uživatelské rozhraní, které nemusí neustále sledovat počet životů hráče, ale upraví ho na základě signálu zaslaného hráčem. Signály jsou jednou z nejsilnějších funkcí, protože stejně jako **Observer** princip [19, str. 62], snižují nutnost propojení mezi objekty a zvyšují jejich nezávislost.

3.7 Virtuální metody uzlů

Jak bylo zmíněno, uzly mohou mít skripty, ve kterých mimo čisté své nové funkce, mohou také přepsat virtuální metody volané při specifických událostech typických pro hru a její fundamentální fungování nebo pro Godot samotný.

3.7.1 _ready

Když jsou uzel a jeho potomci přidáni do scény, je v něm zavolána metoda `_ready`. Tato metoda je volána postupně ze zdola nahoru, čili nejprve pro potomky a poté v přidaném kořenovém uzlu. Pro uzel je to stěžejní metoda a pokud nějaký objekt obsahuje skript, je velmi pravděpodobné, že metodu `_ready` přepíše. Většinou obsahuje inicializační kód, protože rodič může plně nakládat se svými potomky, kteří již danou inicializací prošli. Dále je možné potomky adresovat a to buď relativně, nebo přímo, a tím pádem pracovat i s celým stromem scén.

3.7.2 _process a _physics_process

Metody `_process` a `_physics_process` slouží k provádění operací v časových intervalech. Pokud je nějaká metoda upravena, její kód je vždy zavolán v přesný čas enginem [23].

Metoda `_process` je jednoduše řečeno volána tak rychle, jak je to možné, a to na základě snímkové frekvence hry. Přepsaní bývá využíváno pro účely korigování a aktualizování animací, protože provedené změny budou viditelné co nejdříve a nejplynuleji.

Sesterská `_physics_process` je pak volána ve fixních krocích. V základním nastavení 60krát za sekundu. Uvnitř je možné programovat logiku, která musí být provedena v pevně daných časových intervalech, typicky, jak název napovídá, interakci s fyzikálním systémem.

Oběma metodám je předáván argument `delta`, který udává čas uběhlý od posledního snímku. Tento parametr je klíčový pro konzistentní chování obou metod, pokud by nebyl přítomen, metody by byly vázány na jejich frekvenci volání a mohlo by se například stát, že hra by plynula zrychleně nebo zpomaleně.

4 Grafická a zvuková stránka hry

Ačkoliv Godot poskytuje širokou paletu nástrojů, které pokrývají různé aspekty vývoje her. Vývoj her pouze v něm samotném by byl poměrně strohý, a to zejména v audiovizuálním aspektu. Vytváření a úprava textur nebo zvuků je nutná v jiných programech, přičemž vestavěné funkce umějí nakládat se všemi moderními formáty obrázků, jakožto textur, audio soubory pro efekty nebo hudební doprovod, dále pak importování 3D modelů z běžných modelovacích softwarů.

4.1 Prvky hry

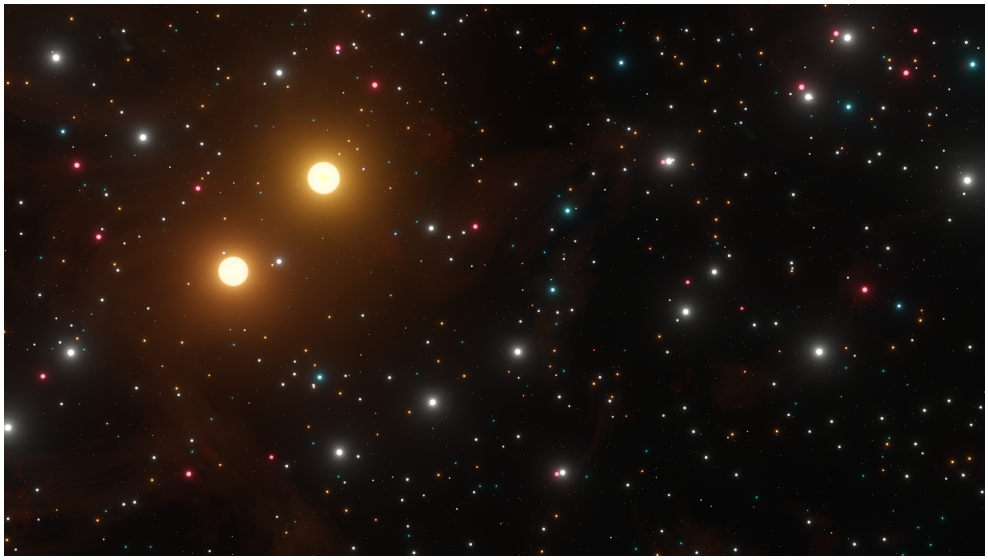
V případě AAA studií stojí za herními prvky taktéž **assety**, týmy grafiků, zvukařů a dalších specialistů. Jejich zastoupení ve studiu je sice poměrově menší než programátorská odnož studia, ale žádná hra by se bez nich neobešla. Nezávislí vývojáři mohou těžit z různých zdrojů určených pro assety, které buďto přímo podporují vývojáři herního enginu, Unity se svým Asset Store a Unreal Engine Marketplace, dnes přejmenovaný na Fab, nebo existují samostatné platformy jako Gamedevmarket.

Přístup k herním assetům je tak možný pro kohokoliv. V mnoha případech jsou assety dokonce dostupné zcela zdarma, pro potřeby inspirace, nebo jako cenově výhodné předpřipravené balíčky pro vytvoření jednoduchých her. Licence takto získaných produktů bývá často velmi benevolentní a poskytuje možnosti jakýchkoliv úprav. Zazmínku stojí itch.io, jedno z největších otevřených tržišť s videoherním obsahem. Mimo mnou amatérsky vytvořeného obsahu tato hra využívá volně dostupných prostředků stažených z tohoto serveru.

Audio bývá v mnoha hrách opomíjeno, což bylo v minulosti často způsobeno omezenou dostupností nebo nízkou kvalitou zvukových zdrojů. Dnešní trh je i zde s nabídkami velmi bohatý, s možností využití obrovských databází volně dostupných zvukových nahrávek. V mnoha případech jsou i dostupné zdarma a stačí k obohacení hry o zvukovou stránku. Zvukové efekty a ambiance byly převzaty primárně od publikovatelů Soniss a 99Sounds, jedněch z nejlepších poskytovatelů vysoce kvalitních audio souborů pro rozmanité projekty, přičemž celá řada souborů je nabízena zcela zdarma.

4.2 Vizuální koncept

Při vývoji hry je v raných fázích kladen důraz na vytvoření prvotních návrhů grafiky, označovaných jako **concept-art**. Concept-art určuje, jak bude finální hra vypadat, a prostřednictvím postupné iterace se vyvíjí v konečný vizuální styl hry. Jeden z prvních experimentů s herní grafikou, který sloužil jako inspirace při práci na projektu, je přiložen k náhledu zde: 4.



Obrázek 4: Příklad konceptuální grafiky

5 Obsah hry

Vytvořená hra je 2D top-down vesmírná arkáda, inspirována tituly Starsector a ΔV : Rings of Saturn. Cílem hry je vylepšit si svou loď a překonat všechny úrovně hry. V průběhu hry hráč postupně odemyká arzenál dostupných zbraní a předmětů, které je možné za minerály, získané z padlých nepřátel a vytěžené z asteroidů, koupit a následně namontovat na loď. Dále jsou postupně odemknuty dvě dovednosti, teleport a boost. Teleport slouží k okamžité transportaci na místo nedaleko hráče a vyhnutí se tak nepřátelským projektilům, nebo překonání nástrah. Boost slouží pro rychlejší pohyb po mapě.

Objekty v herním světě může hráč nejen ničit, ale i provádět interakci. Elementy, se kterými je možné provést interakci, buď přímo vystupují z herního světa svou texturou, jsou nasvíceny silným světlem, vrhajícím stín, nebo mají textový pop-up informující hráče o možné interakci a jejím účelu.

Celou hru hráč plní jednotlivé úkoly, **questy**, různých kategorií. Hlavní, neopakovatelné, kde každá úroveň má svůj hlavní quest, který je nutné splnit pro postup k dalšímu questu, nebo vedlejší, znovuopakovatelné, které začínají vždy při zahájení levelu, nebo jsou spustitelné interakcí v herním světě.

Další mechanikou je levelování hráče. Zabíjením nepřátel a úspěšným dokončováním úrovní je hráč odměňován zkušenostními body. Ty je možné v menu utratit za navýšení levelu na další stupeň. Se zvyšováním levelu hráč získá malý bonus ke svým statistikám: maximální počet životů, obrana, dávané poškození a rychlost, navíc je možné zvýšit úroveň zaměření nějaké specifické vlastnosti za dovednostní body, a tím ještě více zvýšit svou odolnost nebo ničivý potenciál. Každý level má vyšší nárok na počet zkušeností a při běžném průchodu hrou se hráč dosáhne levelu 5 až 6, přičemž může jednotlivé úrovně opakovat a tím navýšit level na stupeň 10, což je **level cap** hry [24, str. 57-59].

Postup hráče je vždy po úspěšném návratu z úrovně automaticky uložen. Minerály, zkušenosti a informace o splněných úkolech a úrovních jsou převedeny do uložené hry, ta je následně zapsána na disk. Tímto je možné pokračovat ve hraní za svůj charakter i při zavření a následném otevření hry. Pokud by hráč musel opustit rozehranou úroveň dříve, než by se dostal na její konec, ať už kvůli své smrti, nebo nutnosti vypnout program, může tak učinit. V takovém případě se ale získané suroviny k uložení nedostanou a jsou ztraceny.

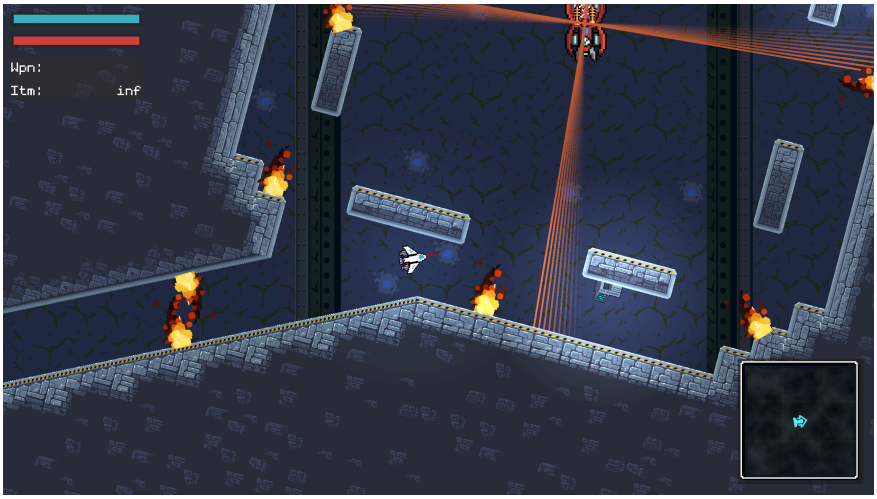
5.1 Nepřátelé

Žádná hra by se neobešla bez nějaké síly působící proti hráči. Zde je vydesignováno množství nepřátel od statických věží a malých dronů až po celé nepřátelské vesmírné flotily. Nepřátelé reagují v určitém okruhu na hráčův pohyb a pokud jej spatří, reagují na jeho přítomnost. Každý typ nepřítele má své chování, které určuje, jaká reakce bude zvolena.

5.2 Úrovně hry

Hra obsahuje 6 úrovní, přičemž první je pojata jako tutoriálová, seznamující hráče s ovládáním a herními mechanismy. Dále je ve hře **secret**, skrytá úroveň, ta je při běžném postupu zamčená a pro její odemknutí je nutné znovu splnit souboj ve třetí úrovni, který má snížený časový limit. Úrovně jsou pak buďto koncipované jako uzavřené úrovně s přesně danými nepřáteli, nebo miniaturní otevřené světy, které nemají pevně dané rozmístění nepřátel a objektů.

Třetí a šestá úroveň představují souboj proti **bossovi**, označení pro výjimečného nebo neobvykle silného nepřítele. V prvním souboji, k vidění na obrázku: 5, musí hráč prokázat znalosti hackovací minihry v časovém presu a za vyhýbání se hazardům rozmístěným po mapě. Druhý je zničení nepřátelské báze, která má navíc doprovod svých vesmírných lodí.



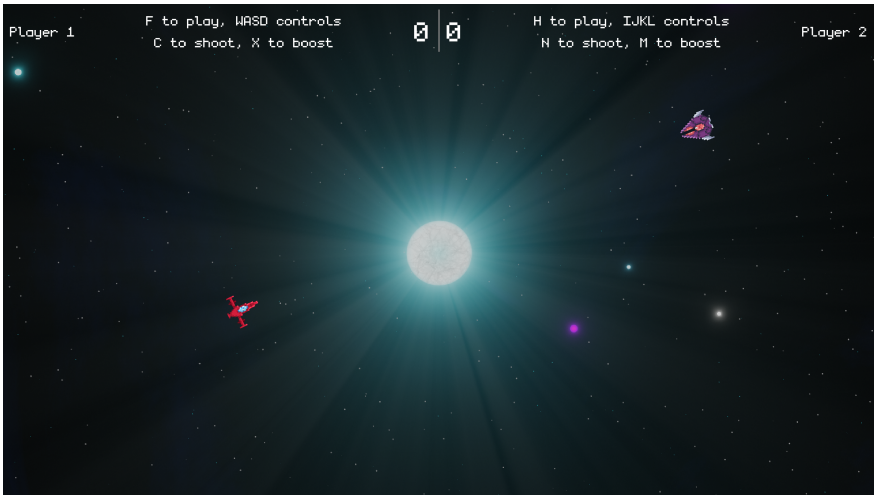
Obrázek 5: Ukázka ze třetí úrovně hry

5.3 Skrytá úroveň

Skryté úrovně vybočují z klasického postupu hrou a jejich umístění bývá maskováno za různé interakce a kódy. Nalezitelný obsah zde představuje imitace hry Spacewar!, zobrazena na obrázku: 6. Jakožto hra dvou hráčů má každý hráč přiřazenou svou část klávesnice. Hráčské lodě i projektily podléhají gravitačnímu působení slunce, které se nachází uprostřed obrazovky.

Pokud se některá z lodí přiblíží příliš blízko ke slunci, je okamžitě zničena. V případě, že hráč provede obnovení a na dané pozici se nachází nepřátelská loď, je proveden **telefrag**. Tento pojem, původem ze střílečky Quake, označuje zničení nepřítele tím, že je teleportován na stejnou pozici, kterou okupuje jiný objekt.

V horní části je zobrazeno skóre a ovládací schéma, to je oproti normální hře upravené na „Asteroids“ styl, nebo lépe řečeno „Spacewar!“ styl, protože právě od Spacewar! bylo ovládání v Asteroids převzato.

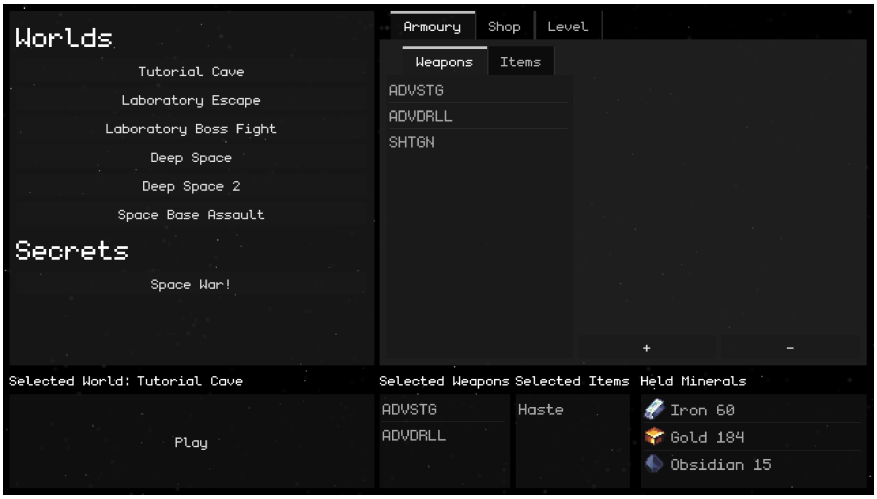


Obrázek 6: Screenshot skryté úrovně hry

5.4 Menu hry

Menu ve hře jsou dvě, první, Hlavní menu, k nahlédnutí zde: 7, slouží k zahájení nové hry, nebo k dostání se do selekčního menu. Druhé menu, „Selečkní“, viditelné na obrázku: 8, obsahuje seznam odemčených světů a možnost jejich výběru. Záložky pro nasazení zbraní a předmětů `Armoury`, nebo jejich nákup v `Shop` a `Level`, kde je možné utrácet zkušenosti pro navyšování levelu a přiřazovat dovednostní body. Dále jsou přítomny seznamy s právě nasazenými zbraněmi, předměty a nashromážděnými minerály. Všechny tyto prvky jsou pak uspořádány v co možná nejpřehlednější kompozici [25, str. 382-385].

Obrázek 7: Hlavní menu hry



Obrázek 8: Selečkní menu hry

5.5 Hackovací minihra

Minihry jsou malé interakce ve hrách sloužící k ozvláštnění hlavního průběhu hrou. Svou komplexností mohou být jednoduché, nevyžadující přílišnou pozornost, až po celé „hry ve hře“. Jedním z úsměvných příkladů může být Pac-Man, dříve celý arkádový kabinet, využitý jako minihra v načítací obrazovce hry Ridge Racer 6.

Zde byla vymyšlena kombinační hříčka, nazvaná hackování. Cílem je vytvořit kombinaci určeného čísla součtem dostupných číslic na hackovacím rozhraní. Hackování může mít různou obtížnost, počet pokusů a čas na dokončení. Obtížnost určuje, jestli jsou dostupné všechny číslice, nebo ne. Při normální obtížnosti je možné využít všechna čísla, v případě těžké jsou některá čísla zašedlá a nepoužitelná v kombinaci.

Obrázek 9: Okno hackovací minihry

5.6 Tištěný tutoriál

Pro ovládání hráčské lodi je vymezena skupina kláves W, A, S, D, sloužící pro pohyb nahoru, doleva, dolů, doprava, relativně k oknu hry. Klávesy E, Q jsou určeny pro rotaci lodi. Pohyb s hráčskou lodí je vždy postupný, ta podle právě zmáčknutých kláves zrychluje daným směrem, nebo po dané ose.

Interagování s herním světem je prováděno klávesou F. Pro automatickou rotaci lodi je možné zapnout rotaci za pohybem hráče stisknutím Z, nebo za pozicí myši pomocí T, opakované zmáčknutí klávesy danou funkci vypne. Další klávesy pro pohyb jsou ALT a mezerník, umožňující teleport a boost, pokud jsou odemknuté. Myš slouží k míření s právě nasazenou zbraní, přičemž z lodi je vykreslen paprsek, který znázorňuje, kam je právě zvolená zbraň natočena. Stiskem levého tlačítka je pak možné, za přítomnosti munice, střílet. Pro změnu nasazené zbraně je určeno kolečko myši. Předměty jsou vybírány klávesami J, L a následně aktivovány stiskem K.

Tabulátor vynese tabulku s informacemi o právě hrané úrovni, N, žurnál, který obsahuje aktivní a kompletované questy. Poslední využitá klávesa je M, která přepíná mezi mapou a hackovacím rozhraním.

Pro ovládání elementů mimo hráčskou loď, tj. menu a vyskakovací okna, slouží primárně myš, její levé tlačítko a kolečko. Pro návrat do předchozího menu, opuštění úrovně a ukončení hry slouží klávesa ESC.

S hackovací minihrou je možné interagovat dvojím způsobem, a to za pomocí hackovacího rozhraní, které automaticky nahradí minimapu, pokud je minihra zahájena, nebo numpadem na klávesnici a jeho třemi postranními klávesami, čili: mínus, plus a enter. Tlačítko - slouží k zrušení pokusu o hacknutí, tlačítko $<$ k vynulování právě vyťukané kombinace a $>$ k potvrzení kombinace. Obrázek: 9 zobrazuje příklad okna právě probíhající minihry, kde je nutné vytvořit finální součet 7 za pomocí některých z čísel: 1, 2, 4, 5 a 6.

Vzhledem k množství ovládacích prvků je k dispozici přiložená tabulka: 1 obsahující přehledně vypsané klávesy a jejich účel.

Klávesa	Funkce
	Akcelerace lodi nahoru
	Akcelerace lodi doleva
	Akcelerace lodi dolů
	Akcelerace lodi doprava
	Rotace doprava
	Rotace doleva
	Interakce
	Přepnutí automatické rotace za směrem pohybu
	Přepnutí automatické rotace za pozicí myši
	Teleport
	Boost
	Střelba
	Výběr předchozího předmětu
	Výběr následujícího předmětu
	Aktivace předmětu
	Informace o úrovni
	Zobrazení žurnálu
	Přepínání mezi minimapou a hackováním
	Návrat do menu nebo ukončení hry
	Hackovací minihra čísla 1 až 9
	Zrušení pokusu o hackování
	Vynulování právě vyťukané kombinace
	Potvrzení kombinace při hackování

Tabulka 1: Tabulka vysvětlující funkce kláves ve hře

5.7 Nerealizované nápady

Většina her má původní koncept vždy rozsáhlejší, než je proveditelné, a velká část obsahu je nakonec zredukována nebo zcela odstraněna během vývoje. Anglický výraz **Cutting Room Floor** označuje právě tento odstraněný obsah. Stejnojmenná stránka, která se zaměřuje na dokumentaci a archivaci tohoto obsahu, čítá přes 30 tisíc záznamů.

Myšlenka této hry byla poněkud obsáhlejší, přičemž původní návrh počítal s rozsáhlými vesmírnými prostředími, které by byly náhodně generované. Soubojový systém, který je ve finálním produktu zjednodušen na úroveň 2D střílečky, byl zamýšlen jako taktický a pomalý, inspirovaný hrami jako Mechwarrior, s více rušným herním rozhraním.

5.7.1 Ovládání lodě

Prvek hry, se kterým bylo experimentováno nejvíce, bylo ovládání lodi. Největší prioritou bylo vytvořit příjemný zážitek při jeho používání. A samotný název hry, „Space Flight“, v překladu „Vesmírný let“, je jedním z hlavních nositelů tohoto faktu.

V první iteraci bylo testováno ovládání založené na fyzikálních silách, od kterého muselo být upuštěno, vzhledem k náročnosti implementace kompletního fyzikálního systému a jeho následné integraci do hry. Další model využíval `RigidBody2D`, Godotí implementace fyzikálně akcelerovaných objektů, výsledné ovládání bylo ovšem velmi neohrabané, vzhledem k nutnosti pracovat někdy proti enginu samotnému.

Testování těchto možností tedy zabíralo nemalé nároky na čas, samotné prototypování systémů pak procházelo fázemi konceptu a ve druhém případě došlo i k pokusu o odladění systému, což se ukázalo být natolik složité, že byl tento modul jednoduše vyřazen [26, str. 13-17].

Ultimátně bylo zvoleno klasické schéma vycházející z tradičního ovládání 2D her. Z experimentování si ovšem zachovává váhu a jistou osobitost. Takže lodě ve hře mají různá zrychlení v závislosti na tom, kterým směrem se chtějí pohybovat a na jakou stranu jsou natočeny.

6 Programátorská příručka

Většina skriptů hry jsou poměrně jednoduché a krátké úseky kódu, nenáročné na pochopení své funkcionality. Globální pohled na věc a komunikace mezi objekty samotnými je ovšem již trochu složitější. Zde jsou představeny ty nejdůležitější použité skripty a jejich účel.

6.1 Herní entity

Všechny herní objekty, které se ve hře vyskytují, vycházejí z vestavěných tříd CharacterBody2D v případě pohyblivých a StaticBody2D v případě statických objektů, celkově nazvaných `Entit`. Entita vždy při vstupu do hry informuje EntityManager, který se stará o jejich agregaci a poskytuje informace o právě přítomných entitách ve scéně.

Hráč a pohybliví nepřátelé vycházejí ze stejného základu ThrusterEntity, který umožňuje pohyb při dodání uzlu Thrusters. Hráčův vstup je pak očekáván v metodě input a následně předán ke zpracování. Pro nepřátelský pohyb byl, vzhledem k jednoduchosti map, implementován vlastní systém několika spolupracujících uzlů. Zejména `AiMoveStateMachine` a `NavigationAgent`.

`AiMoveStateMachine`, jednoduchý stavový automat [19, str. 120], určuje, jakým způsobem a kam se chce daný objekt pohybovat. V závislosti na tom, jestli má cíl a vzdálenosti od něj, je určen aktuální stav, ve kterém se automat nachází. Pohyby mohou být jednoduché od místa k místu, nebo složitější, jako třeba kroužení kolem bodu. Při kombinaci různých stavů dochází k různému chování. Stavům je pak možné nastavovat odchylku a aktualizační čas a tím zajistit, že pohyb bude méně předvídatelný a méně strojový.

`NavigationAgent` se dle názvu stará o navigaci v prostředí. Objekt se může zeptat na to, jestli je ve směru dodaném `AiMoveStateMachine` nějaká překážka, a pokud ano, je navrhnut jiný směr, co nejpodobnější tomu původnímu, který se již kolizi vyhne.

Pro reakci na podněty v herním světě jsou pak tyto systémy doplněné o uzel `AiAgroManager`, ten se podle názvu stará o **agro**, označení pro systém zpracovávání agrese nehráčských charakterů ve hrách. Tento uzel čeká buď na signál od zdraví, které mu pošle zprávu s parametrem označujícím, kdo změnu způsobil, nebo na informaci z oblasti, ve které sleduje viditelnost nepřátelských objektů.

6.1.1 Komponenty entit

Každá entita může nést různé uzly, které fungují jako komponenty rozšiřující její funkčnost. Pokud například obsahuje uzel `Health`, získává schopnost sledovat svůj stav zdraví a být informována o jeho změnách, včetně identifikace, kdo nebo co změnu způsobilo.

Tento uzel úzce souvisí se systémem hurtboxů a hitboxů. `Hurtbox` představuje oblast, která může utrpět poškození, zatímco `Hitbox` je oblast, která poškození uděluje. Health pak může být dále rozšířen o `Shield`, který se po obdržení

poškození regeneruje, případně po úplném vyčerpání projít restartem a dočasně chránit zdraví před dalším poškozením.

Hráč také může nést uzel `Inventory`, který spravuje aktuálně nesené zbraně a předměty, získané suroviny během úrovně a přepsání dál v případě jejího úspěšného splnění.

Pokud však dané uzly nejsou přidány, nic se neděje a entita je schopna provádět svou základní funkcionalitu.

6.2 Zbraně

Pro zbraně a podobné třídy slouží jako základ třída `Fireable`. Z ní potom dědí všechny uzly, které mají něco společného se střelbou. Každá zbraň má své statistiky jako rychlost střelby, poškození, dostřel, rozptyl, dostupná munice a další. Fungování zbraní ve hrách obecně je dvou typů: hitscan nebo projektilové.

Hitscan zbraně `WeaponRaycast` fungují na principu, že výstřel je vystřelen okamžitě při zahájení palby a detekce zásahu probíhá okamžitě. Ze zbraně je skenována linie, která hledá kolizi s možným cílem. Pokud s ním koliduje, je hned aplikováno poškození.

Projektilové zbraně `WeaponProjectile` využívají projektilů, které dopravují poškození k cíli. Při střelbě je vytvořen nový projektil, ten je umístěn do herního světa na místo výstřelu a odtud sám putuje ve své dráze. Zde může být simulována například balistická křivka, protože stačí pouze aplikovat síly působící na projektil. Projektily samotné mohou detonovat při zničení nebo mít jiné vlastnosti.

Další potomci fireable jsou `Arsenal`, která při příkazu pro střelbu zavolá metody ve všech svých dětech, tím pádem je možné mít jednu entitu s více zbraněmi. Nebo `Gimbal`, který má na starost rotaci za sledovaným cílem; pokud není rotace k cíli dostatečná, `Gimbal` zabrání nevystřelu.

Ve výsledku je na arsenalu zapojeno několik různých gimbalů a na nich jednotlivé zbraně. Střelbu nebo změnu cíle stačí říct pouze Arsenalu a ten potom požadavek rozdistribuuje dál.

6.3 Questy

`QuestManager` obsahuje všechny aktivní a dokončené questy. Jakožto singleton je jej možné adresovat odkudkoliv, takže veškeré operace s questy jsou dělány skrz něj. `Quest` je pak třída spravující jeden úkol a postup v něm. Na něj navázaná třída `DestroyQuest` obsahuje navíc sledování herních entit a při jejich zničení, posun ke splnění questu. Questy mohou být velmi široce upravované a nesou kód, který specifikuje, jaké objekty a co se s nimi má stát.

6.4 Databáze

`Database` má za úkol poskytovat informace o neměnných textech a proměnných, nebo překládat jednotlivá slova. Interně je kód, názvy entit a zbraní vedený v `camel_case`, což není optimální pro uživatelsky přívětivý vzhled. Pro zobrazovaný text tedy platí, že je nejprve přeložen v databázi do své vizuálně přívětivé podoby a až poté vypsán.

Databáze je založená na `TransDB`, které drží `translation_dictionary`, což jsou slovníky obsahující klíčová slova, asociovaná s názvem, popisem a možnou ikonou. Jejich nádstavba `ListTransDB` přidává k těmto datům i doplňkové pole `sorted_translation_list`, které určuje pořadí, ve kterém mají být daná data zobrazována. Minerály a dostupné vybavení se vždy zobrazí ve stejném pořadí, nezávisle na tom, kdy byly obdrženy. Navíc může databáze přikládat i filtrační list, který určuje, která data se vypsat nemají a v jakých případech.

Objekty, které mají být zobrazovány, obsahují metodu `to_dictionary`, která vrátí slovník s jejich vlastnostmi. Například statistika o zbrani, která obsahuje i pro zobrazování nedůležité informace, je vyfiltrována, přeložena a až potom předána listu k vypsání.

Třída quest může být zcela oddělena od zobrazovací logiky, přičemž k zobrazení postupu stačí pouze předání potřebných informací. Překlad probíhá nezávisle a vyžaduje pouze dotaz na databázi s názvem questu. Tímto způsobem je možné oddělit logiku a prezentaci, čímž se zabrání jejich vzájemnému míchání v jednom objektu.

Proměnné v databázi pak slouží pro potřeby interního fungování hry. Například ceníky určující nákupní cenu jednotlivých zbraní, nebo škálovací tabulky, **scaling tables**, které na základě levelu entity určí, jakým koeficientem jsou vynásobeny její statistiky. Pro hráče jsou přítomny tabulky dvě. Jedna určuje hodnoty na základě levelu, druhá potom na základě vylepšení dovednosti, hodnoty jsou viditelné na tabulkách 2 a 3.

Level	**Zdraví**	**Obrana**	**Poškození**	**Rychlost**
1	1.15	1.1	1.04	1.04
2	1.25	1.15	1.08	1.10
3	1.40	1.22	1.12	1.16
4	1.60	1.30	1.16	1.20
5	1.75	1.40	1.20	1.25

Tabulka 2: Dovednostní škálovací tabulka vlastností hráče

Level	Zdraví	Obrana	Poškození	Rychlost
1	1.0	1.0	1.0	1.0
2	1.1	1.1	1.05	1.05
3	1.15	1.15	1.05	1.10
4	1.20	1.17	1.10	1.15
5	1.30	1.20	1.15	1.15
6	1.35	1.30	1.20	1.20
7	1.45	1.35	1.25	1.20
8	1.50	1.40	1.25	1.23
9	1.60	1.45	1.30	1.23
10	1.75	1.50	1.33	1.25

Tabulka 3: Levelová škálovací tabulka vlastností hráče

6.4.1 Zobrazovací okna

Zobrazování informací hráči je kritická část herního designu, kdy hlavní prvky je lépe reprezentovat třeba i duplicitně. Zdraví a štíty hráče jsou zobrazeny v levém horním rohu obrazovky a jasně viditelné díky svému zatmavenému pozadí. Zdraví je pak duplikováno i viditelným poškozením lodi, které se úměrně ke zdraví zvětšuje.

Pro zobrazování textu je využita zmíněná databáze a různé panely, prezentující přeložený text hráči. Listy `TranslationItemList` pak plně využívají slovníky, a podle klíčových slov vytřídí a zobrazí uživatelsky přívětivý výstup. Při náběhu myší na tlačítka nebo jednotlivé položky je pak dynamicky zobrazován převzatý delší popis jejich funkce.

6.5 Scény hry

Jednotlivé úrovně ve hře dědí ze třídy `WorldScene`, nebo v případě menu z jednodušší `GameScene`. Samotná úroveň je reprezentována uzlem typu `Node`, na který je navázán její obsah. Ostatní uzly vidí právě aktivní úroveň prostřednictvím vlastnosti stromu scén `current_scene`. V autoloadu `SceneManager` je aktivní scéna monitorována a pomocí metody `change_scene` lze změnit aktuální úroveň zadáním názvu nové scény v argumentu.

6.6 Ukládání hry

O uloženou pozici se stará `SaveManager`, herní data jsou při spuštění hry načtena ze souboru *savefile.json*, který se nachází ve složce `Appdata`. Pokud tento soubor neexistuje, je vytvořen nový s výchozí „čistou“ uloženou pozicí. Z načtených dat je následně vytvořen objekt `GameSave`. Tento objekt obsahuje struktury jako `SaveWorldsData`, `SavePlayData` a další.

Díky tomu, že uložená pozice je reprezentována jako třída v Godotu, jsou data přístupná jako vlastnosti, což umožňuje snadný a rychlý přístup k jednotlivým položkám. Změny těchto dat jsou navíc vysílány jako signály, které využívají ostatní herní systémy. Menu ve hře tedy nemusí přistupovat k datům periodicky, ale pouze v případě, že dojde k zachycení signálu o změně.

Tento přístup zároveň minimalizuje počet zápisů na disk; při čtení nebo změně jedné položky není nutné přepisovat celý soubor s uloženou hrou. Místo toho se mění pouze interní objekt `GameSave`, jehož obsah se automaticky propíše v celku na dlouhodobé uložiště při změně scény.

7 Vytváření obsahu

Všechny základní třídy obsahují kostru pro jakýkoliv obsah. Pro vytvoření nové zbraně stačí stvořit novou scénu, která má jakožto kořenový uzel požadovaný styl střelby a nastavit její parametry v editoru. V případě zbraně je navíc možné dodat i další uzly jako značku, odkud zbraň střílí, nebo přehrávače zvuků výstřelu, zahájení a ukončení nabíjení.

Pro vytvoření nového úkolu slouží zděděná kosterní funkcionalita, která zajišťuje základní strukturu a procesy. Stačí dopsat logiku jednotlivých fází v metodách `stage_script`. Každá metoda `stage_script` obsahuje podmínky pro postup dál a nastavuje sledování jejich splnění. Pokud hráč dokončil quest úspěšně, je mu připsána odměna ve formě surovin, nově odemknutých zbraní, lokací a dokonce i dalších questů.

Založení nové úrovně obnáší dodání nutných prvků, jako uživatelské rozhraní nebo kamera. Dále je nutné přidat možnost opuštění úrovně buďto přímo ve skriptu úrovně, nebo přidáním uzlu `WorldExitTrigger`, který se stará o vyslání signálu, jenž značí kompletaci úrovně, ze základu v kombinaci s interakcí.

Na konec jsou v databázi vytvořeny příslušné asociace: cena zbraně, její název a popis. Znění questu v plné délce, vysvětlení jednotlivých fází a odměn za splnění. Pro levely je zapsán jejich název a poznámka o jeho náplni.

Závěr

Výsledkem bakalářské práce je počítačová hra vytvořená v Godot Engine. Hra obsahuje mimo šesti úrovní pro jednoho hráče i bonusovou hru dvou hráčů ve formě klonu hry Spacewar!. Dále je zaznamenán její vývoj a úskalí, na která jsem v průběhu její tvorby narazil. Samotná hra i přes nedosáhnutí mé původní vize přináší příjemný herní zážitek a splňuje své primární zamýšlené cíle. Do budoucna je možné rozšíření přidáním chytřejších stavů nepřátel a prodloužením questové linky.

Conclusions

The result of this bachelor thesis is a video game created in Godot Engine. The game includes, in addition to six levels for one player, a bonus two-player game in the form of a Spacewar! clone. Additionally, the development process and challenges encountered throughout the creation are documented. Although the game did not fully achieve my original vision, it provides an enjoyable gaming experience and meets its primary intended goals. In the future, it is possible to expand the game by adding smarter enemy states and prolonging the questline.

A Obsah elektronických dat

doc

Adresář s textem práce v souboru *kidiplom.pdf* a kompresovaná složka *kidiplom.zip* obsahující zdrojové soubory nutné k vytvoření textu práce.

bin

Složka obsahující všechny soubory potřebné pro běh hry a *Readme* obsahující popis spuštění hry.

src

Kompletní projektová složka se zdrojovými soubory hry, včetně složky `license` obsahující informace o použitých licencovaných materiálech.

Literatura

[1] Brown, Stuart. *The First Video Game*. 2019. Ahoy, Youtube kanál. Dostupný z: ⟨https://www.youtube.com/watch?v=uHQ4WCU1WQc⟩.

[2] *Video Games Worldwide*. 2024. Dostupný z: ⟨https://www.statista.com/outlook/dmo/digital-media/video-games/worldwide⟩.

[3] *Cinema Worldwide*. 2024. Dostupný z: ⟨https://www.statista.com/outlook/amo/media/cinema/worldwide⟩.

[4] *Music Worldwide*. 2024. Dostupný z: ⟨https://www.statista.com/outlook/dmo/app/music/worldwide⟩.

[5] *Statistiky služby Steam a her*. 2024. Dostupný z: ⟨https://store.steampowered.com/stats/stats/⟩.

[6] Museum, Computer History. *Dan Edwards (left) and Peter Samson playing Spacewar! on the PDP-1 Type 30 display*. 1962. Dostupný z: ⟨https://www.computerhistory.org/pdp-1/a87ddd9510aeebf6485c47a35f8a26aa/⟩.

[7] *Frostbite*. 2024. Dostupný z: ⟨https://www.ea.com/frostbite⟩.

[8] *Source 2*. 2024. Dostupný z: ⟨https://developer.valvesoftware.com/wiki/Source_2⟩.

[9] *About Game Engines*. 2023. Dostupný z: ⟨https://www.remedygames.com/article/about-game-engines⟩.

[10] *Introducing Enfusion Engine*. 2021. Dostupný z: ⟨https://www.bohemia.net/blog/Introducing-Enfusion-Engine⟩.

[11] *New Witcher Saga Announced*. 2022. Dostupný z: ⟨https://www.cdprojekt.com/en/media/news/new-witcher-saga-announced-cd-projekt-red-begins-development-on-unreal-engine-5-as-part-of-a-strategic-partnership-with-epic-games/⟩.

[12] *Halo Studios: New Name, New Engine, New Games, New Philosophy*. 2024. Dostupný z: ⟨https://news.xbox.com/en-us/2024/10/06/halo-studios-unreal-engine-interview/⟩.

[13] *Godot Frequently asked questions*. 2024. Dostupný z: ⟨https://docs.godotengine.org/en/4.2/about/faq.html#why-does-godot-aim-to-keep-its-core-feature-set-small⟩.

[14] *Class Object*. 2024. Dostupný z: ⟨https://docs.godotengine.org/en/4.2/classes/class_object.html⟩.

[15] *Class Node*. 2024. Dostupný z: ⟨https://docs.godotengine.org/en/4.2/classes/class_node.html⟩.

[16] *Godot Engine Inheritance class tree*. 2024. Dostupný z: ⟨https://docs.godotengine.org/en/4.2/contributing/development/core_and_modules/inheritance_class_tree.html⟩.

[17] *Godot Scenes*. 2024. Dostupný z: ⟨https://docs.godotengine.org/en/4.2/getting_started/introduction/key_concepts_overview.html#scenes⟩.

[18] *Unity Prefabs*. 2024. Dostupný z: ⟨https://docs.unity3d.com/Manual/Prefabs.html⟩.

[19] Robert, Nystrom. *Game Programming Patterns*. First. 2014. 456 s. ISBN 978-0990582922.

[20] *Class SceneTree*. 2024. Dostupný z: ⟨https://docs.godotengine.org/en/4.2/classes/class_scenetree.html#class-scenetree⟩.

[21] Games, Monko. *what happens when you play skyrim with more that 60 fps (physics broken)*. 2019. Dostupný z: ⟨https://www.youtube.com/watch?v=ZFVtXjr-Nno⟩.

[22] *GDScript README*. 2024. Dostupný z: ⟨https://github.com/godotengine/godot/blob/master/modules/gdscript/README.md⟩.

[23] *Idle and Physics Processing*. 2024. Dostupný z: ⟨https://docs.godotengine.org/en/4.2/tutorials/scripting/idle_and_physics_processing.html⟩.

[24] Ian, Schreiber; Brenda, Romero. *Game Balance*. First. 2019. 806 s. ISBN 978-1498799577.

[25] Jesse, Schell. *The Art of Game Design: A Book of Lenses*. Third. 2019. 652 s. ISBN 978-1138632059.

[26] Adams, Ernest; Dormans, Joris. *Game Mechanics: Advanced Game Design*. 2012. 368 s.

[27] *Class Node2D*. 2024. Dostupný z: ⟨https://docs.godotengine.org/en/4.2/classes/class_node2d.html⟩.

[28] *The Cutting Room Floor*. 2024. Dostupný z: ⟨https://tcrf.net/The_Cutting_Room_Floor⟩.

Online zdroje byly přístupné k datu 5. ledna 2025.