TECHNICKÁ SPECIFIKA VÝVOJE MOBILNÍCH HER

Kamil Chytil

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Kamil Chytil**
Osobní číslo:	**A20269**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Technická specifika vývoje mobilních her**
Téma práce anglicky:	**Technical Specifics of Mobile Game Development**

Zásady pro vypracování

1. Vypracujte literární rešerši na zadané téma.
2. Porovnejte odlišnosti vývoje mobilních her od vývoje desktopových her.
3. Navrhněte ukázkovou hru, kde bude demonstrována optimalizace pro mobilní platformu.
4. Vytvořte navrženou hru.
5. Vhodně popište postup vývoje a samotnou hru.

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 bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- 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 – 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 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 …….

podpis studenta

ABSTRAKT

Tato bakalářská práce se zaměřuje na technická specifika vývoje mobilních her v prostředí Unity. Teoretická část práce poskytuje informace o Unity, rozdílech ve vývoji mezi desktopovými a mobilními zařízeními, optimalizaci hry a využití Unity Profileru pro měření výkonu aplikace. V práci byly vytvořeny dva prototypy mobilní hry. První prototyp obsahoval neoptimalizované metody, zatímco druhý prototyp obsahoval tyto metody optimalizované. Prostřednictvím testování a měření výkonu pomocí Unity profileru byl zjištěn rozdíl výkonu mezi oběma prototypy, přičemž druhý prototyp s optimalizovanými metodami dosáhl větší efektivity a lepšího výkonu.

Klíčová slova: technická specifika, vývoj mobilních her, optimalizace, UnityProfiler, výkon hry.

ABSTRACT

This bachelor's thesis focuses on the technical specifics of developing mobile games in the Unity environment. The theoretical part of the thesis provides information about Unity, differences in development between desktop and mobile devices, game optimization, and the use of Unity Profiler for performance measurement. Two prototypes of a mobile game were created in the thesis. The first prototype included non-optimized methods, while the second prototype featured these methods optimized. Through testing and performance measurement using Unity Profiler, a difference in performance between the two prototypes was identified, with the second prototype employing optimized methods achieving greater efficiency and improved performance.

Keywords: technical specifics, mobile game development, optimization, Unity Profiler, game performance.

Vedoucí mé bakalářské práce, Ing. Tomáš Vogeltanz, Ph.D. si zaslouží poděkování za vstřícnost, laskavost a snahu při konzultacích bakalářské práce a včasné odpovědi. Dále bych tímto poděkoval své rodině, které vděčím za úspěšné dokončení této práce.

Obsah

ÚVOD

Tato bakalářská práce se zaměřuje na technická specifika vývoje mobilních her a zkoumá, jakým způsobem ovlivňují výkon a optimalizaci her. Cílem práce je analyzovat a porovnat rozdíly ve vývoji her mezi desktopovými počítači a mobilními zařízeními a využít nástroje, jako je Unity Profiler, k optimalizaci a sledování výkonu her na obou platformách.

V teoretické části práce je podrobně popsána herní platforma Unity, která se stala jedním z nejpoužívanějších nástrojů pro vývoj her. Jsou zde vysvětleny základní koncepty a funkce Unity, které jsou nezbytné pro pochopení procesu vývoje mobilních her. Dále je zdůrazněna důležitost rozlišování mezi desktopovými a mobilními zařízeními při vývoji her a jsou popsány klíčové rozdíly a výzvy, které s tím souvisejí.

V praktické části práce je provedena simulace zatížení pomocí různých herních prvků, jako je Particle systém, 3D objekty, LOD a analýza neoptimalizovaného kódu. Tyto zátěže jsou následně optimalizovány a sledovány pomocí nástroje Unity Profiler, který umožňuje detailní analýzu výkonu her a identifikaci možných problémů. Testy jsou prováděny jak na desktopovém zařízení, tak na mobilním zařízení, aby bylo možné porovnat výsledky a zhodnotit rozdíly ve výkonu.

Cílem této bakalářské práce je poskytnout ucelený pohled na technické aspekty vývoje mobilních her a ukázat, jakým způsobem mohou být tyto hry optimalizovány pro dosažení co nejlepšího výkonu na obou platformách. Výsledky analýz a testů provedených pomocí Unity Profileru poskytnou cenné informace o zatížení různých herních prvků a neoptimalizovaného kódu na mobilních zařízeních a desktopových počítačích.

I. TEORETICKÁ ČÁST

1 UNITY

Unity je cross-platformový herní engine používaný pro vytváření interaktivních 2D a 3D her pro různé platformy, jako jsou počítače, mobilní zařízení, konzole a VR. Engine umožňuje vývojářům vytvářet hry pomocí intuitivního rozhraní, robustních editorových nástrojů a bohaté knihovny prostředků. To činí Unity silným a univerzálním enginem jak pro začátečníky, tak pro profesionály.

Jednou z nejvýraznějších funkcí Unity je jeho flexibilita. Engine umožňuje vývojářům používat různé programovací jazyky, jako je C#, JavaScript a Boo. Tato všestrannost činí Unity přístupným pro širokou škálu vývojářů s různými úrovněmi odbornosti.[1]

1.1 Vlastnosti Unity

Unity nabízí intuitivní rozhraní, které usnadňuje práci vývojářům. Rozhraní umožňuje vývojářům rychle a snadno vytvářet scény, přidávat objekty, upravovat vlastnosti a mnoho dalšího.[2]

Další rozšíření Unity je jeho Asset Store na kterém si vývojáři mohou nakupovat a prodávat prostředky pro vývoj her, jako jsou 3D modely, zvukové efekty, skripty a další zdroje, které lze použít k tvorbě her. Asset Store má tisíce položek a nabízí skvělý způsob, jak rychle a snadno přidat kvalitní prostředky do projektu. To umožňuje vývojářům rychleji vytvářet hry a šetřit čas a zdroje při tvorbě vlastního obsahu.[3]

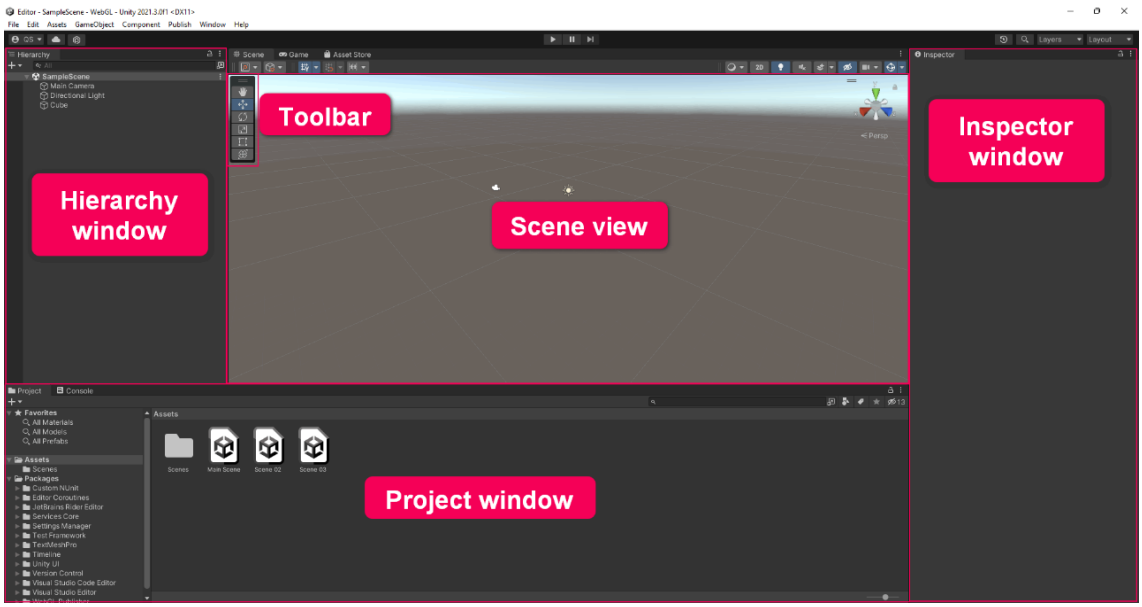
Vývojáři také mohou využít Unity Analytics, což je nástroj pro sledování a sběr dat o hráčských interakcích s hrou. Tento nástroj umožňuje vývojářům získávat užitečné informace o tom, jak hráči interagují s jejich hrou, jak dlouho hru hrají, kde se zasekli a jakou část hry opakovaně hrají. Tyto informace mohou pomoci vývojářům při vylepšování hry a vytváření lepšího hráčského zážitku.

Vývoj v oblasti VR/AR se stává stále populárnějším a Unity nabízí vynikající podporu pro tyto technologie. Díky speciálním funkcím a nástrojům navrženým pro VR/AR mohou vývojáři snadno vytvářet poutavé a interaktivní zážitky pro tyto nové technologie.[4]

Pro vývojáře je důležitá také silná a aktivní komunita, která se v okolí Unity vytvořila. Unity umožňuje vývojářům sdílet své zkušenosti, nápady a řešit společně problémy, což umožňuje rychlejší a efektivnější vývoj her. Komunita také vytváří rozsáhlou knihovnu tutoriálů a užitečných nástrojů, které mohou být k dispozici zdarma.

1.2 Unity Editor

Unity Editor je integrované vývojové prostředí pro vývoj her. Editor nabízí přívětivé rozhraní, které umožňuje snadné vytváření a editování herních objektů a scén. Uživatelé mohou používat rozhraní pro přidání a nastavení herních objektů, jako jsou postavy, objekty v prostředí, osvětlení a zvuky. Editor nabízí nástroje pro tvorbu herní logiky, vizuálního designu a dalších aspektů hry. Mezi klíčové prvky editoru patří Scene view a Game view, Hierarchy window, Project window, Inspector window a Toolbar.[5]



Obrázek 1: Unity Editor[6]

Scene view zobrazuje aktuální scénu, což je virtuální prostředí, ve kterém jsou umístěny všechny herní objekty. V Scene view lze herní objekty přidávat, mazat, přesouvat, měnit jejich velikost a rotaci a provádět mnoho dalších úprav. Scene view také umožňuje změnu kamery, osvětlení a dalších prvků scény.

Game view je okno, které zobrazuje aktuální stav hry, tak jak by ho viděl hráč. Umožňuje testovat hru v reálném čase a vidět ji, tak jak ji bude vidět hráč. V Game view slouží pro spouštění hry, testovat ji, sledovat výkon a další věci. Veškeré manipulace a úpravy s herními objekty v Scene view, se okamžitě promítají v Game view, takže lze okamžitě vidět, jaké jsou výsledky vašich úprav.

Hierarchy window je okno, ve kterém jsou zobrazeny všechny herní objekty v scéně a jejich hierarchie. Slouží k vytváření, přidávání, mazání a upravování herních objektů. Dálelze změnit hierarchii herních objektů a určit, které objekty jsou rodičovské a které jsou potomky. Hierarchy window pomáhá organizovat herní objekty a spravovat je.

Project window je okno, ve kterém jsou zobrazeny všechny soubory, které jsou součástí projektu. Project window umožňuje přidávat, mazat a upravovat soubory a složky v projektu, jako jsou scény, prefaby, modely, textury, zvuky a další. Project window také umožňuje filtrovat soubory a vyhledávat v nich.[7]

Inspector window umožňuje upravovat vlastnosti herních objektů a komponent. V Inspector window lze měnit různé vlastnosti, jako jsou velikost, barva, pozice, rotace, materiály, fyzikální vlastnosti a další. Také lze přidávat a odebírat komponenty, které ovlivňují chování herních objektů v hře. Všechny změny provedené v Inspector window se okamžitě promítají do Scene view.[8]

2 ROZDÍLNÝ VÝVOJ DESKTOPOVÝCH A MOBILNÍCH HER

Jedním z klíčových rozdílů mezi desktopovými a mobilními hrami jsou hardwarové požadavky. Desktopové počítače mají obvykle vyšší výpočetní výkon, paměť a grafickou kapacitu než mobilní zařízení. To umožňuje vývojářům vytvářet komplexnější a graficky náročnější hry s rozsáhlými světy, pokročilou fyzikou a dalšími efekty. Naopak, mobilní zařízení mají omezenější hardwarové možnosti, což může mít vliv na výkon hry a vyžaduje pečlivé optimalizace pro dosažení plynulého a stabilního chodu hry na různých mobilních zařízeních.[9]

Pro desktopové hry je optimalizace zpravidla zaměřena na výkon a grafiku. Vývojáři se snaží dosáhnout co nejvyššího počtu snímků za sekundu, aby hra byla plynulá a hráči měli dokonalý herní zážitek. Optimalizace grafiky může zahrnovat použití technik jako je level of detail nebo culling. Další optimalizační techniky mohou zahrnovat použití hardwarové akcelerace, optimalizace paměti, a optimalizace algoritmů pro fyziku, kolize, umělou inteligenci a další herní prvky.[10]

Pro mobilní hry je optimalizace zaměřena na výkon, ale také na spotřebu energie a paměťové nároky. Optimalizace výkonu může zahrnovat snížení počtu polygonů, použití optimalizovaných textur a shaderů, a minimalizaci používání složitých fyzikálních simulací a jiných výpočetně náročných operací. Optimalizace spotřeby energie může zahrnovat omezení výpočetních operací ve chvílích, kdy je zařízení v režimu úspory energie nebo kdy je baterie téměř vybitá. Optimalizace paměťových nároků může zahrnovat minimalizaci velikosti textur, modelů a zvukových souborů, aby se snížila paměťová zátěž zařízení.

Mobilní hry často využívají dotykové obrazovky a další senzory, jako jsou akcelerometry nebo gyroskopy, což vyžaduje speciální návrh ovládacích prvků, které jsou vhodné pro dotykové rozhraní. Vývojáři musí pečlivě navrhnout ovládání, které je snadno ovladatelné na malém displeji a umožňuje hráčům pohodlně ovládat hru.

Velikost obrazovky mobilního zařízení je obvykle omezenější než u desktopu, což vyžaduje pečlivé plánování a návrh herního rozhraní tak, aby bylo optimalizováno pro malý displej. Grafický design, rozložení ovládacích prvků, typografie a další vizuální prvky musí být pečlivě navrženy tak, aby byly snadno čitelné a použitelné na mobilním zařízení.

Důležité je taky zohlednit různé způsoby, jakými hráči používají mobilní zařízení. Mobilní hry jsou často hrané na cestách, což znamená, že hráči mohou být více rozptýlení nebo hrát v krátkých intervalech. Proto je důležité, aby hra byla navržena s ohledem na tyto aspekty,

například s možností rychlého ukládání a načítání hry, jednoduchým a intuitivním ovládáním, a zábavným herním zážitkem, který je přizpůsobený hráčům, kteří hrají na mobilním zařízení.[11]

Distribuce her je také odlišná pro desktopové a mobilní platformy. Desktopové hry jsou obvykle distribuovány prostřednictvím digitálních distribučních platforem, jako je Steam nebo Epic Games Store, nebo jsou prodávány na fyzických médiích. Na druhou stranu jsou mobilní hry distribuovány prostřednictvím mobilních aplikací, které jsou ke stažení z oficiálních mobilních aplikacích obchodů, jako je Google Play Store pro Android nebo App Store pro iOS.[12]

Při distribuci mobilních her je důležité brát v úvahu různé omezení a politiky platforem, jako je například schválení a certifikace her před jejich uvedením na trh, dodržování pravidel pro ochranu soukromí a osobních údajů hráčů, a případně i nastavení cen a monetizace her. Distribuce her na mobilních platformách také zahrnuje optimalizaci her pro různé zařízení a operační systémy, aby byla hra dostupná pro co nejširší spektrum mobilních zařízení.[13]

Obrázek 2: Mobilní verze PUBG[14]

Obrázek 3: Desktopová verze PUBG[15]

3 ROZDÍL VÝVOJE HER MEZI UNITY A UNREALEM

Jednou z nevýhod Unity je to, že nemá takové výkonnostní možnosti jako Unreal Engine. To může být problém, pokud potřebujete vytvořit hru s velkým množstvím grafických prvků nebo s velkou mapou. Další nevýhodou Unity je, že neobsahuje podporu pro vytváření her s velkým rozsahem, jako jsou open-world hry.[16]

Unreal Engine je herní engine vyvinutý společností Epic Games. Je často používán pro vývoj her AAA a dalších velkých projektů. Unreal Engine obsahuje mnoho pokročilých nástrojů pro tvorbu her, včetně fyzikálního enginu, vizuálního editoru scény, animací a post-processingu.

Unreal Engine také obsahuje mnoho funkcí pro vytváření her s velkým rozsahem, jako jsou open-world hry. To vše je umožněno díky jeho výkonnému enginu, který umožňuje vývojářům vytvářet hry s velkým množstvím grafických prvků a složitými interakcemi.[17]

Hardwarové požadavky

Unreal Engine má velkou náročnost na hardwarové požadavky. Unreal Engine je navržen tak, aby využíval moderní hardwarové technologie, což může být problém pro menší studia s omezeným rozpočtem. Vývojový tým potřebuje silný počítač, který dokáže zvládnout náročné procesy jako je kompilace kódu a renderování grafiky. Další nevýhodou Unreal Enginu je jeho složitost a s tím související obtížnost učení. Unreal Engine nabízí mnoho funkcí a nástrojů, ale jejich použití a porozumění jim může být pro začátečníky obtížné.

Unity má poměrně nízké hardwarové požadavky a lze jej používat na široké škále zařízení, od mobilních telefonů a tabletů až po herní konzole a počítače. To umožňuje vývojářům testovat a vyvíjet hry na různých platformách bez nutnosti vlastnit speciální hardware pro každou z nich. Další výhodou Unity je jeho snadné ovládání a přátelské uživatelské rozhraní. To znamená, že se rychle naučíte pracovat s enginem a vytvářet hry bez velkého úsilí.

Vývojové nástroje a rozhraní

Unity má intuitivní uživatelské rozhraní a je relativně snadný na naučení. Poskytuje vizuální nástroje pro tvorbu her, jako jsou editory pro scény, animace, osvětlení a vizuální efekty. Kromě toho nabízí velké množství assetů, knihoven a pluginů, které usnadňují a urychlují vývoj.

Unreal Engine má také velmi intuitivní rozhraní a je dobře dokumentován. Je zaměřený více na programování a jeho vizuální nástroje jsou méně uživatelsky přívětivé. Na druhé straně

Unreal Engine nabízí pokročilé vizuální funkce, jako jsou nástroje pro tvorbu fyzikálních simulací a propracovaný systém osvětlení.

Podpora platforem

Unity podporuje širokou škálu platforem, včetně PC, Mac, iOS, Android, Xbox, PlayStation, Nintendo Switch a dalších. Unity je navržen tak, aby umožňoval snadný přenos mezi různými platformami a minimalizoval počet potřebných úprav pro každou platformu.

Unreal Engine podporuje většinu populárních platforem, včetně PC, Mac, iOS, Android, Xbox, PlayStation, Nintendo Switch a dalších. Podpora platformy Unreal Engine zahrnuje také VR/AR. Unreal Engine poskytuje podporu pro různé renderovací enginy, jako jsou DirectX, OpenGL a Vulkan.

Uživatelská přívětivost

Unity je často považován za nejvíce přístupný herní engine. Jedním z důvodů je to, že má poměrně jednoduché uživatelské rozhraní, které je snadno ovladatelné. Navíc má velké množství dokumentace a tutoriálů na internetu, což znamená, že i lidé bez zkušeností s vývojem her se mohou rychle naučit, jak s Unity pracovat.

Unreal Engine je považován za složitější na naučení než Unity. Unreal má sice nástroje pro vývoj her s velkým rozsahem, ale právě kvůli svému pokročilému uživatelskému rozhraní může být složitější na ovládání pro ty, kteří nemají zkušenosti s vývojem her.[18]

4 OPTIMALIZACE HRY

Optimalizace hry je klíčovým aspektem vývoje her, neboť umožňuje zlepšit výkon, kompatibilitu a celkovou kvalitu hry. Optimalizace se zaměřuje na zlepšení efektivity použití výpočetních a grafických zdrojů, minimalizaci nároků na hardware a zajištění co nejplynulejšího herního zážitku pro hráče. Důvody pro optimalizaci hry jsou mnohé, od dosažení vyššího výkonu a stability hry, kompatibility s různými zařízeními, snížení spotřeby zdrojů až po zajištění široké dostupnosti hry na různých platformách.[19]

4.1 Optimalizace hry po Kódové stránce

Optimalizace hry po kódové stránce je klíčovým prvkem vývoje her, protože nejenže zlepšuje výkon hry a zkracuje dobu načítání, ale také zvyšuje přehlednost a přístupnost kódu. Efektivní kód výrazně snižuje nároky na hardware, což znamená, že hra bude moci běžet plynule na větším počtu různých zařízení.[20]

Asynchronní operace jsou užitečné pro optimalizaci hry, zejména pokud se jedná o operace, které mohou být časově náročné a mohou způsobit snížení výkonu hry.

Volba správných datových struktur pro ukládání a manipulaci s daty může ovlivnit výkon hry. Například použití "List" může být rychlejší než "Array" pro časté změny velikosti kolekce, zatímco "Dictionary" může být efektivnější pro rychlé vyhledávání hodnot.

Efektivní algoritmy jsou dalším důležitým prvkem efektivního kódu. Například binární vyhledávání může být rychlejší než lineární vyhledávání v seřazených polích. QuickSort je účinným řadícím algoritmem, který může být použit pro rychlé řazení dat.

Minimalizace počet volání funkcí je také důležitou součástí efektivního kódu. Zbytečné volání funkcí může mít negativní vliv na výkon hry, proto je důležité optimalizovat kód tak, aby byl co nejefektivnější. Lze použít různé techniky, jako jsou memoizace, odstranění zbytečných proměnných a použití lokálních proměnných místo globálních.

Pooling je proces opakovaného používání stejné instance objektu namísto vytváření nových. Při používání pooling se vytváří určitý počet instancí objektu v předem určeném okamžiku a jsou uloženy v paměti. Tyto instance jsou pak použity v průběhu hry, kdy jsou potřebné, místo aby se vytvářely nové instance. Tento přístup umožňuje snížit spotřebu systémových prostředků a zvýšit výkon hry.

4.2 Optimalizace ve Scéně

Animace lze optimalizovat několika způsoby. Jedním způsobem je snížení počtu klíčových snímků animace, což může snížit spotřebu výpočetního výkonu. Další možností je použití animací s menším rozlišením, což může také snížit spotřebu výpočetního výkonu. Dále je možné použít animační nadstavby jako Animator Controller, které umožňují snadné řízení animací v hře.[21]

LOD umožňuje snížit detail 3D modelů v závislosti na vzdálenosti od kamery. Tím se snižuje počet polygonů, které musí být vykresleny, a tím i spotřeba systémových prostředků. Pokud je objekt daleko od kamery, není nutné vykreslovat vysoko-detailní model, ale stačí vykreslit méně detailní verzi objektu. Tento přístup může být použit i pro další prvky hry, jako jsou stromy, tráva nebo budovy.

Culling umožňuje vyloučit objekty, které nejsou viditelné z kamery, přesněji nejsou vykresleny, což snižuje spotřebu systémových prostředků. Existují různé techniky cullingu, jako je frustum culling, occlusion culling nebo view frustum culling. Tyto techniky umožňují vyloučit objekty, které nejsou viditelné pro hráče, což výrazně snižuje spotřebu systémových prostředků.

Snížení počtu herních objektů v scéně může výrazně zlepšit výkon hry. Čím více objektů je v scéně, tím více systémových prostředků je potřeba na jejich správné vykreslení a aktualizaci. Proto je důležité minimalizovat počet objektů v scéně a používat mnohoúrovňové struktury. Jedním z příkladů může být použití objektu jako kontejneru, který obsahuje další objekty. Tím se minimalizuje počet herních objektů v scéně a zlepšuje se výkon hry.[22]

Postprocessing efekty nebo částicové efekty mohou být velmi náročné pro výkon hry. Pokusit se minimalizovat jejich počet a používat efekty s menší kvalitou. Nejlépe používat efekty pouze v klíčových místech hry, kde mají největší vizuální dopad. Použití efektů s menší kvalitou, jako jsou efekty s menším počtem částic nebo s menšími rozlišeními, může výrazně snížit spotřebu systémových prostředků.

Textury mohou mít velký vliv na výkon hry. Používejte textury s menším rozlišením a správným formátem, jako jsou komprimované formáty jako DXT nebo ASTC. Tyto formáty umožňují snížení velikosti textur a tím i spotřeby systémových prostředků. Pokud používáte textury s vysokým rozlišením, jako jsou textury s rozlišením 4K, může to výrazně zpomalit výkon hry.

Objekty, které se v průběhu hry nemění, jako jsou terény, stěny nebo jiné statické objekty, je nutné nastavit je jako statické objekty. Tím se zrychlí proces vykreslování, protože Unity může předpřipravit statické objekty před spuštěním hry.

Světla jsou náročné na výkon, a proto je důležité optimalizovat jejich použití. Použít pouze nezbytně nutná světla a optimalizujte jejich parametry, jako je dosah, úhel, barva a intenzita. Můžete také využít statické světelné mapy, které předpočítají osvětlení scény a umožní rychlejší vykreslování.[23]

5 UNITY PROFILER

Unity Profiler je výkonný nástroj, který umožňuje vývojářům analyzovat výkon své hry a identifikovat oblasti, které mohou být optimalizovány. Pro použití Unity Profileru je nutné spustit hru v editoru Unity a následně spustit Profiler v záložce Window. Profiler zobrazí různé informace o výkonu hry, jako jsou využití CPU, GPU a paměti. Tyto informace jsou zobrazeny v grafu, který lze přizpůsobit a přiblížit. Profiler také umožňuje vývojářům sledovat, které scény a herní objekty mají nejvyšší nároky na výkon.

Jednou z nejdůležitějších funkcí Unity Profileru je možnost profilování skriptů. Profiler zobrazuje seznam všech skriptů a jejich časovou náročnost. Tento seznam lze seřadit podle času potřebného k vykonání skriptu a lze tak snadno identifikovat skripty, které mají největší vliv na výkon hry.

Profiler také umožňuje vývojářům sledovat, jaký vliv mají různé funkce hry na výkon, jako jsou fyzikální výpočty, animace, osvětlení a efekty. Tuto informaci lze použít k identifikaci oblastí, které mohou být optimalizovány.

Jednou z nejlepších funkcí Unity Profileru je možnost generování zpráv, které umožňují vývojářům snadno identifikovat problémy s výkonem hry. Tyto zprávy mohou být odeslány vývojářům pomocí e-mailu nebo uloženy jako soubor. Vývojáři mohou také uložit profily pro pozdější analýzu.

5.1 CPU Usage Profiler module

Slouží k analýze využití CPU při běhu hry. Pomocí tohoto modulu lze identifikovat neefektivní části kódu a optimalizovat je pro zlepšení výkonu hry.

Modul zobrazuje informace o čase, který jednotlivé části kódu trvají. Tyto informace jsou zobrazeny v tzv. Call Hierarchy, kde jsou metody seřazeny podle toho, kolik času zabírají. U každé metody jsou zobrazeny další informace, jako například počet volání, počet alokovaných objektů, alokovaná paměť a další.[24]

Pro efektivní použití CPU Usage Profileru je nutné mít dobře porozumět kódu a algoritmům použitým v hře. Je důležité se zaměřit na části kódu, které zabírají nejvíce času a optimalizovat je pro snížení náročnosti na CPU. Je také důležité analyzovat vliv různých nastavení a filtrů na výsledky analýzy.

Others, zahrnuje všechny ostatní procesy a funkce, které se nevejdou do ostatních kategorií. Tyto procesy mohou být například vstupní systém, správa zdrojů nebo zvukový systém.

Rendering, obsahuje všechny procesy související s vykreslováním 3D grafiky. Patří sem například výpočty stínů, osvětlení, textury a geometrie.

Scripts, tato kategorie obsahuje všechny procesy související s prováděním skriptů, jako jsou skripty pro chování objektů nebo skripty pro UI. Tato kategorie může být velmi náročná, pokud se ve hře používají složité skripty.

Physics, zahrnuje všechny procesy související s výpočty fyzikálních interakcí v herním světě. Patří sem například výpočty kolizí, gravitace a síly.

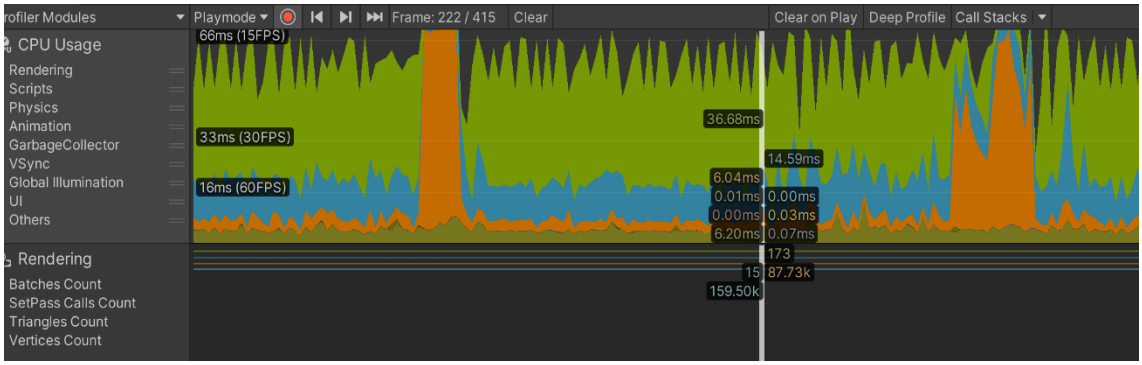
Animation, tato kategorie zahrnuje všechny procesy související s animacemi herních objektů. Patří sem například výpočty pozic, rotací a škálování objektů.

Garbage Collector. tato kategorie pokrývá všechny procesy související s čištěním paměti od nepoužívaných objektů. Garbage Collector je důležitý pro udržení výkonu hry, protože nečištění nepoužívané paměti by mohlo vést ke snížení výkonu a pádu hry.

VSync, tato kategorie zahrnuje všechny procesy související s vertikální synchronizací (VSync), což je technologie, která synchronizuje počet snímků vykreslených na obrazovku s obnovovací frekvencí monitoru. To může mít vliv na výkon hry, protože pokud se hra snaží vykreslit více snímků, než je obnovovací frekvence monitoru, může to vést k neplynulému obrazu.

Global Illumination zahrnuje všechny procesy související s globálním osvětlením. Globální osvětlení je technologie, která umožňuje vypočítat, jak se světlo šíří a odráží v herním prostředí.

UI obsahuje využití CPU při vykreslování uživatelského rozhraní (UI) hry. UI je kritickou součástí hry, protože poskytuje uživatelům přístup k různým funkcím a informacím. Například informace o časové náročnosti vykreslování GUI prvků, jako jsou tlačítka, ikony, okna dialogu.



Obrázek 4: CPU Usage Profiler module[25]

5.2 Asset Loading Profiler module

Umožňuje sledovat načítání a zpracování assetů, jako jsou textury, mesh objekty, zvukové efekty, skripty a další. Pomocí tohoto modulu jde detekovat pomalé a výkonnostně náročné assety, které mohou mít negativní dopad na výkon hry.

Other Reads, tato metrika zahrnuje všechna ostatní čtení assetů, které nejsou klasifikovány jako textury, virtuální textury, mesh nebo zvuková data. Například to může být čtení konfiguračních souborů nebo shaderů.[26]

Texture Reads, sleduje počet čtení textur během načítání hry. Textury jsou důležitou součástí grafického enginu a často se používají k vytváření materiálů a dekorativních prvků.

Virtual Texture Reads, sleduje počet čtení virtuálních textur během načítání hry. Virtuální textury jsou relativně novou technologií a umožňují renderování velkých textur s nízkým výkonovým nárokem.

Mesh Reads, sleduje počet čtení meshů během načítání hry. Mesh je základním stavebním kamenem 3D modelů a používá se k vytváření herních objektů.

Audio Reads, tato metrika sleduje počet čtení zvukových dat během načítání hry. Zvuky jsou důležitou součástí hry a používají se k vytváření atmosféry a interakce s herními objekty.

Scripting Reads, sleduje počet čtení skriptů během načítání hry. Skripty jsou často používány pro herní logiku a umožňují programátorům vytvářet vlastní funkce a chování herních objektů.

Entities Reads, sleduje počet čtení herních objektů během načítání hry. Herní objekty jsou základním prvkem herního světa a zahrnují například postavy, vozidla, budovy a další prvky.

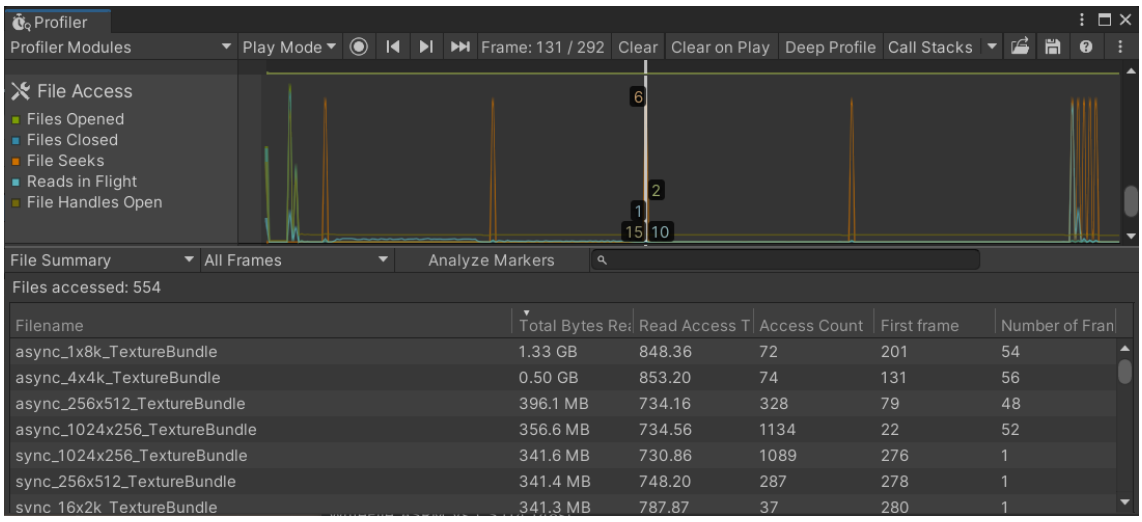
5.3 File Access Profiler module

File Access Profiler module (modul pro sledování přístupu k souborům) je nástroj, který umožňuje sledovat a analyzovat přístupy k souborům v aplikaci. Tento modul může být užitečný pro vývojáře, kteří potřebují optimalizovat výkon aplikace nebo vyhledat chyby a problémy s načítáním souborů.[27]

Files Opened, počet souborů, které byly otevřeny v průběhu sledování. Files Closed, počet souborů, které byly uzavřeny v průběhu sledování. File Seeks, počet operací přesunu ukazatele na jiné místo v souboru (tzv. seek) v průběhu sledování. Tyto operace se používají k čtení nebo zápisu souboru od určité pozice.

Reads in Flight, počet čtení ze souboru, která byla spuštěna, ale ještě nebyla dokončena v průběhu sledování. Tato metrika pomáhá identifikovat situace, kdy aplikace musí čekat na dokončení čtení ze souboru, což může zpomalit výkon aplikace.

File Handles Open, počet otevřených souborových handle v průběhu sledování. Handle je interní identifikátor, který používají operační systémy k identifikaci souborů, které jsou otevřeny v aplikaci.



Obrázek 5: Asset Loading Profiler module[28]

5.4 Audio Profiler modul

Audio Profiler module je součástí Unity Profileru a slouží k analýze využití zvukových prvků v hře. Tento modul umožňuje vývojářům identifikovat a vyřešit případné problémy týkající se zvuků, jako jsou například neefektivní způsoby načítání zvukových souborů, neoptimalizované efekty zvuků nebo nadměrné využití paměti pro zvukové soubory.

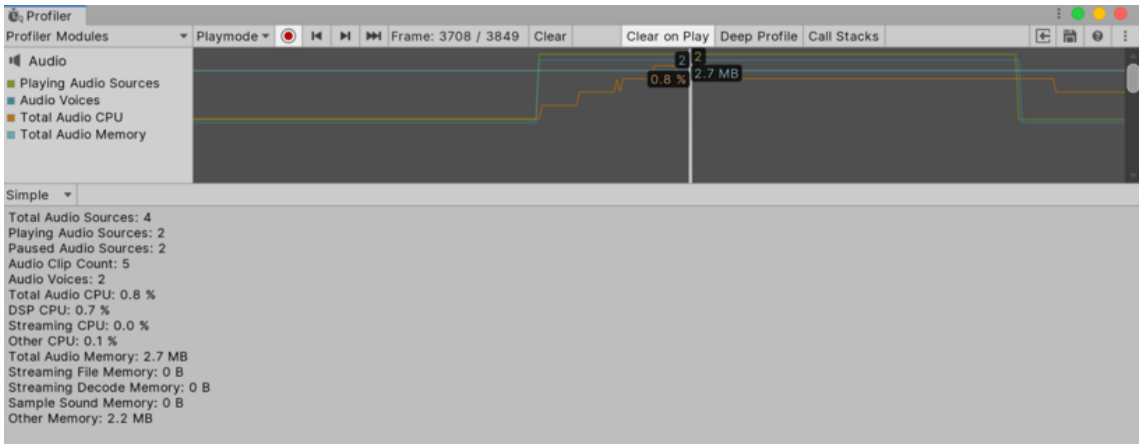
Dále umožňuje tento modul analyzovat využití zvukových prvků v čase, což pomáhá vývojářům identifikovat případné problémy týkající se zvuků. Tento modul také umožňuje generování zvukových událostí, což může pomoci při testování a ladění hry.

Playing Audio Sources, zobrazuje počet aktivních Audio Sources v aplikaci. Audio Source je komponenta, která umožňuje přehrávání zvukových efektů v aplikaci. Sledování této metriky umožňuje vývojářům identifikovat problémy s příliš mnoha aktivními Audio Sources v aplikaci, což může způsobit výrazný nárůst spotřeby procesoru.[29]

Audio Voices, ukazuje počet aktivních Audio Voices v aplikaci. Audio Voice je instance, která přehrává zvuk z Audio Source. Vývojáři mohou sledovat tuto metriku k identifikaci problémů s příliš mnoha aktivními Audio Voices v aplikaci, což může způsobit přetížení zvukového systému a narušení kvality zvuku.

Total Audio CPU, tato metrika zobrazuje celkovou spotřebu procesoru v aplikaci v důsledku zpracování zvukových efektů a přehrávání zvukových souborů. Vývojáři mohou sledovat tuto metriku k identifikaci problémů s příliš vysokou spotřebou procesoru v rámci zvukového systému.

Total Audio Memory, tato metrika ukazuje celkovou spotřebu paměti v aplikaci v důsledku načtení a uchování zvukových souborů a efektů. Vývojáři mohou sledovat tuto metriku k identifikaci problémů s příliš vysokou spotřebou paměti vzhledem k zvukového systému.



Obrázek 6: Audio Profiler modul[30]

5.5 Global Illumination Profiler

Global Illumination Profile je modul v rámci Unity Profileru, který umožňuje sledovat výkon a náročnost Global Illumination v hře. GI se zabývá výpočtem a aplikací osvětlení v

herních scénách, což může být náročné na výkon. GI Profiler umožňuje vývojářům monitorovat a optimalizovat využití GI v hře.

Umožňuje sledovat využití GI v různých scénách a zjistit, které scény jsou nejvíce náročné na výkon. To může pomoci vývojářům optimalizovat hru tak, aby v různých scénách dosahovala co nejlepšího výkonu.[31]

Light Probe sleduje výkon výpočtu osvětlení, které je vykresleno pomocí Light Probe. Light Probe jsou umístěny v různých oblastech herního světa a slouží k získání informací o okolním osvětlení, které pak mohou být použity pro dynamické osvětlení objektů. Tato metrika pomáhá vývojářům odhalit možné výkonnostní problémy spojené s výpočtem osvětlení pomocí Light Probe.

Setup se zaměřuje na dobu potřebnou pro nastavení Global Illumination v herním prostředí. To může zahrnovat například načtení dat z předchozích herních sezení, přepočítání osvětlení pro nové objekty nebo načtení dat z diskového úložiště. Tato metrika pomáhá vývojářům identifikovat oblasti, které mohou být optimalizovány a zrychleny.

Environment tato metrika sleduje výkon související s nastavením a vykreslením herního prostředí. Zahrnuje to například výpočet osvětlení v nejbližším okolí herního objektu a výpočet osvětlení z oblohy. Tato metrika pomáhá vývojářům identifikovat možné problémy s výkonem v oblasti herního prostředí.

Input Lighting ukazuje čas, který trvá na načtení vstupního osvětlení. To zahrnuje všechny světelné prvky a světelné mapy, které jsou součástí scény a používají se pro výpočet globálního osvětlení.

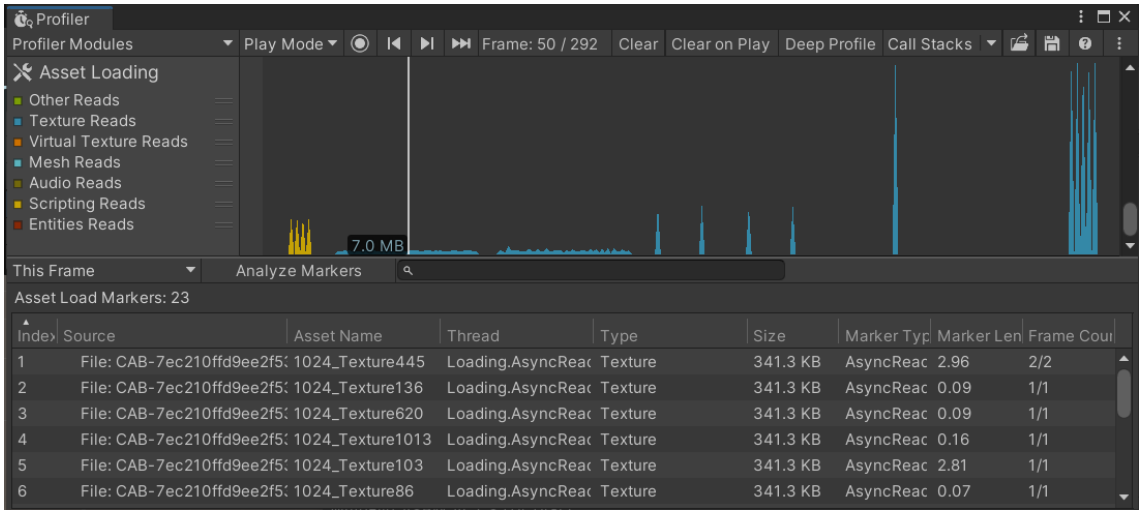
Systems tento parametr sleduje čas, který trvá na inicializaci a spuštění samotného globálního osvětlovacího systému. To zahrnuje všechny procesy, které jsou nutné k výpočtu osvětlení, včetně výpočtu světelných prvků, sběru dat a zpracování vstupního osvětlení.

Solve Tasks, tato metrika sleduje čas, který je potřebný na vyřešení jednotlivých úloh při výpočtu globálního osvětlení. Tyto úlohy mohou zahrnovat výpočet světelných map, výpočet osvětlení pro jednotlivé objekty v scéně a další úkoly spojené s výpočtem osvětlení.

Dynamic Objects, ukazuje, kolik času trvá na výpočet osvětlení pro dynamické objekty v scéně. Dynamické objekty jsou ty, které se mohou pohybovat nebo měnit své vlastnosti během hry, jako například postavy, vozidla apod. Čím více dynamických objektů v scéně, tím déle může trvat výpočet osvětlení pro tyto objekty.

Other Commands, zahrnuje čas, který trvá na zpracování jiných příkazů, které nebyly zahrnuty v předchozích kategoriích. Mezi tyto příkazy patří například příkazy na aktualizaci světla a stínu, příkazy na vykreslování GUI, příkazy na animaci a další.

Block Command Write, tato metrika zaznamenává čas, který trvá na zápis bloku příkazů pro Global Illumination do GPU. To může být pomalý proces, pokud se na GPU zapisuje velké množství příkazů najednou, což může způsobit zpoždění v rámci celé aplikace. V tomto případě může být potřeba optimalizovat proces zápisu bloku příkazů, aby se snížil vliv na výkon aplikace.



Obrázek 7: Global Illumination Profiler[32]

5.6 GPU Usage Profiler module

Umožňuje sledovat využití grafické karty v průběhu běhu hry a získat informace o tom, jaké procesy a funkce nejvíce ovlivňují výkon hry. To je užitečné pro optimalizaci grafického výkonu a dosažení co nejvyššího počtu snímků za sekundu.

Opaque, zobrazuje procentuální využití grafického procesoru pro vykreslování neprůhledných objektů. V praxi se jedná například o stěny budov, kameny na zemi nebo konkrétní objekty jako auta a postavy.[33]

Transparent, ukazuje procentuální využití GPU pro vykreslování průhledných primitiv, tedy objektů, které neblokují viditelnost objektů za nimi a jsou viditelné i přes ně. V praxi se jedná například o sklo, vodu, kouř a podobně.

Shadows/Depth, tato metrika ukazuje procentuální využití GPU pro vykreslování stínů a hloubkových map. Stíny jsou generovány v závislosti na umístění a orientaci světla

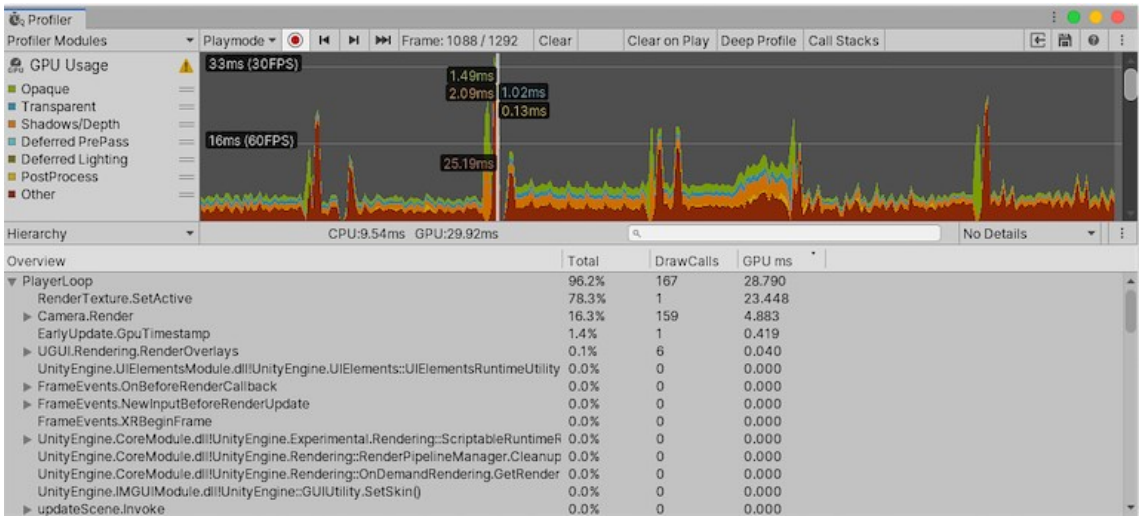
a objektů v prostoru. Hloubkové mapy slouží k určení, jak daleko od kamery se nachází jednotlivé objekty, což je důležité pro korektní vykreslování průhledných objektů.

Deferred PrePass, tato hodnota zobrazuje procentuální využití GPU pro vykreslování prvního průchodu při použití metody deferred shading. Tento průchod slouží k vypočtení normál, hloubky a dalších dat, které jsou důležité pro správné osvětlení scény v další fázi.

Deferred Lighting, tato metrika sleduje procentuální využití GPU pro vykreslování druhého průchodu při použití metody deferred shading. V této fázi jsou vypočteny osvětlovací efekty a aplikovány na jednotlivé objekty v závislosti na jejich vlastnostech (např. materiál, textura apod.).

PostProcess, znázorňuje procentuální využití GPU pro aplikaci různých postprocessing efektů, jako jsou například bloom, motion blur, depth of field a další.

Other, tato metrika zahrnuje všechny ostatní operace, které nebyly zahrnuty v předchozích kategoriích. Jedná se například o vykreslování GUI, textů, částicových efektů a další.



Obrázek 8: GPU Usage Profiler module[34]

5.7 Memory Profiler modul

Memory Profiler modul je nástroj určený k monitorování spotřeby paměti v herních aplikacích. Jeho hlavním účelem je identifikovat případy neefektivního používání paměti, jako jsou úniky paměti, fragmentace paměti nebo přebytečné alokace, které mohou vést k pádu hry nebo zpomalení výkonu.

Total Used Memory, tato metrika ukazuje celkové množství paměti, které je v současné době používáno pro běh hry. Zahrnuje paměť používanou pro všechny aspekty hry, včetně grafiky, zvuků, skriptů.[35]

Texture Memory, sleduje, kolik paměti je použito pro textury, které jsou v současné době načteny do paměti. Textury jsou obvykle největší spotřebitelé paměti v herním enginu a mohou být velmi náročné pro grafické výpočty.

Mesh Memory, tato metrika ukazuje, kolik paměti je použito pro zobrazování herních objektů. Mesh je základním stavebním kamenem pro většinu 3D modelů v herních enginu. Větší množství meshů v hře obvykle znamená větší spotřebu paměti.

Material Count, tato metrika ukazuje, kolik materiálů je použito v hře. Materiály definují, jak bude herní objekt vykreslen. Větší množství materiálů může vést k větší spotřebě paměti.

Object Count, ukazuje celkový počet vytvořených objektů v aplikaci v průběhu času. To zahrnuje všechny typy objektů, jako jsou herní objekty, UI prvky, efekty, zvukové efekty a další.

GC Used Memory, zobrazuje, kolik paměti je v současné době používáno pro garbage collector. GC je mechanismus, který pomáhá uvolňovat paměť, která již není používána herní aplikací.

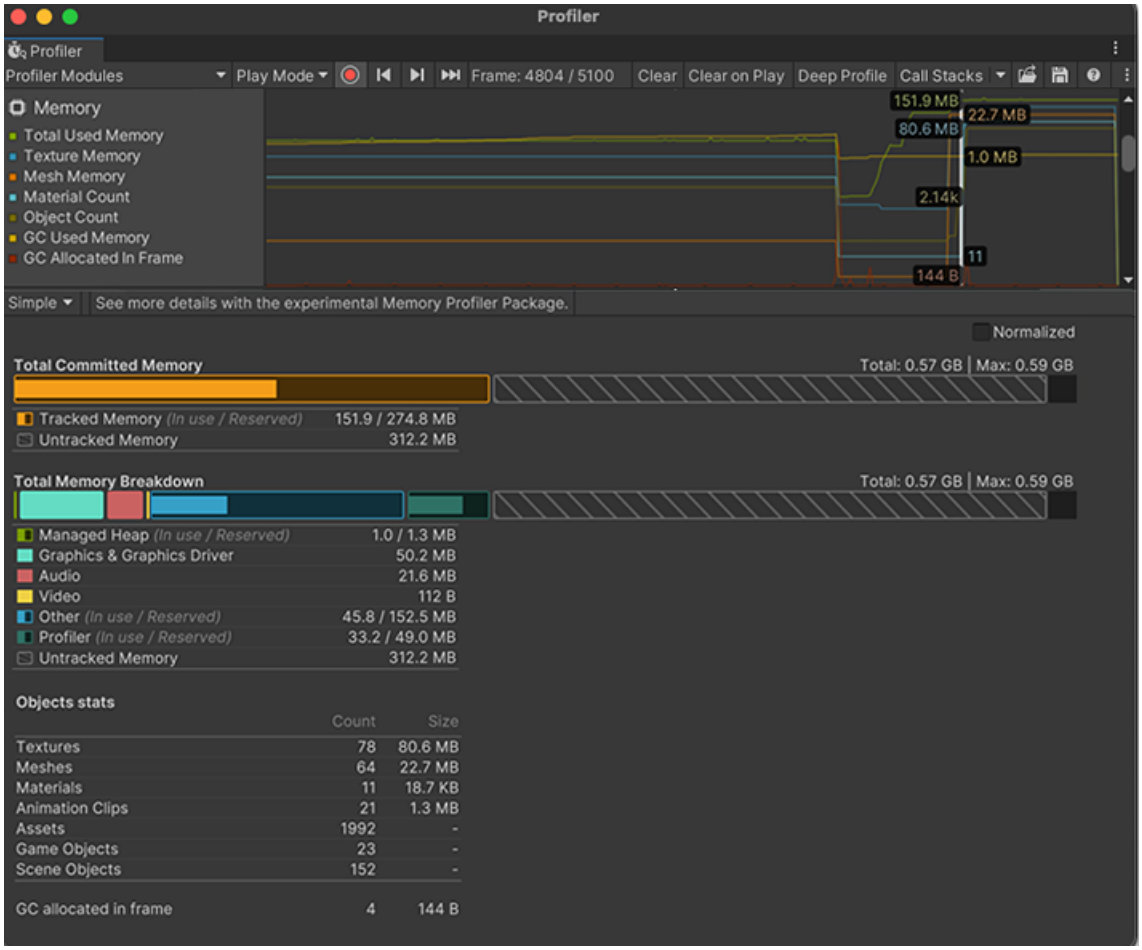
GC Allocated in Frame, tato metrika vyobrazuje, kolik paměti bylo alokováno v rámci jediného snímku herní aplikace. Větší množství alokované paměti v jednom snímku může způsobit výrazné zpomalení hry, pokud se garbage collector rozhodne provést velké množství čištění paměti.

Total Committed Memory, představuje celkové množství paměti, které bylo aplikací přiděleno, včetně spravované a nativní paměti. Tato hodnota poskytuje představu o maximálním množství paměti, které bylo aplikací v průběhu profilování kdykoliv alokováno. Jedná se o užitečnou metriku pro monitorování celkového využití paměti aplikací a identifikaci potenciálních úniků paměti nebo nadměrného využívání paměti.

Total Memory Breakdown, reprezentuje celkové množství paměti, které aplikace aktuálně používá. Tato metrika zobrazuje podrobné rozdělení používané paměti na jednotlivé kategorie, jako jsou textury, mesh objekty, materiály a další. Tento rozklad paměti poskytuje užitečnou informaci pro optimalizaci použití paměti v aplikaci, protože umožňuje identifikovat, které kategorie paměti jsou nejvíce náročné a případně optimalizovat použití paměti

v těchto oblastech. Celkový rozklad paměti může také pomoci s detekcí potenciálních problémů s pamětí, jako jsou úniky paměti nebo nadměrné využití paměti.

Sledování objemu paměti použitého jednotlivými herními prvky, umožňuje sledovat, jakou část paměti využívají jednotlivé herní prvky, jako jsou například modely postav, textury, zvuky nebo UI prvky. Tato informace umožňuje identifikovat prvky, které využívají nejvíce paměti a optimalizovat je pro lepší výkon herní aplikace.



Obrázek 9: Memory Profiler modul[36]

5.8 Modul Physics Profiler

Modul Physics Profiler je nezbytným nástrojem pro vývojáře her, kteří chtějí zajistit, aby fyzikální simulace v jejich hře běžely hladce a efektivně. Tento modul poskytuje řadu informací a metrik souvisejících s fyzikálními výpočty, umožňuje vývojářům optimalizovat výkon hry a zlepšit zážitek hráčů.[37]

Dalším důležitým aspektem modulu Physics Profiler je detekce kolizí. Tento modul může sledovat počet kontrol kolizí v reálném čase a identifikovat jakékoli problémy, které mohou

způsobovat výkonové problémy. Může se jednat o problémy, jako jsou příliš složité kolizní meshe nebo neefektivní kolizní algoritmy.

Kromě detekce kolizí poskytuje modul Physics Profiler také informace o fyzikálních efektech, jako jsou ragdoll fyzika a částicové systémy. Tyto efekty mohou být velmi výpočetně náročné, proto je sledování jejich výkonu klíčové pro zajištění, že nezpůsobují žádné zpomalení nebo výkonové problémy.

Physics Used Memory, tato metrika ukazuje, kolik paměti používá fyzikální simulace ve vaší aplikaci. To může být užitečné pro optimalizaci výkonu, zejména pokud má vaše aplikace velké množství fyzikálních objektů. Čím větší je tato hodnota, tím větší bude zátěž na paměťový subsystém vašeho počítače.

Active Dynamic Bodies, sleduje počet dynamických těles v aktuální scéně, což jsou objekty, které se mohou pohybovat a interagovat s jinými tělesy v simulaci. Vysoký počet dynamických těles může mít negativní dopad na výkon simulace, zejména pokud jsou složitější nebo mají složité interakce.

Active Kinematic Bodies, ukazuje počet kinematických těles v aktuální scéně, což jsou objekty, které se mohou pohybovat, ale nepodléhají fyzikálním zákonům, jakými jsou gravitace nebo interakce s dalšími tělesy. Kinematická tělesa jsou užitečná pro simulování pohybu hráče nebo jiných ovládaných objektů. Podobně jako u dynamických těles, vysoký počet kinematických těles může mít negativní dopad na výkon simulace, zejména pokud mají složitější pohyby nebo interakce s jinými objekty.

Dynamic Bodies, zobrazuje je počet dynamických těles (Rigidbody), které jsou v aplikaci aktivní. Dynamická tělesa jsou tělesa, která jsou ovlivněna fyzikálními zákony a mohou se pohybovat v reakci na síly, které na ně působí. Tato metrika umožňuje sledovat počet aktivních těles v aplikaci a identifikovat případné problémy s vysokým počtem těles, které by mohly zpomalovat výkon aplikace.

Overlaps, poskytuje informace o počtu překryvů (kolizí) mezi různými objekty v aplikaci. Překryv znamená, že se dva objekty dotýkají a navzájem ovlivňují své pohyby. Tento ukazatel může pomoci v identifikaci potenciálních problémů s kolizemi, které mohou zpomalovat výkon aplikace.

Trigger Overlaps, tato metrika poskytuje informace o počtu překryvů (kolizí) mezi tělesy typu Trigger v aplikaci. Trigger tělesa jsou tělesa, která nereagují na fyzikální zákony,

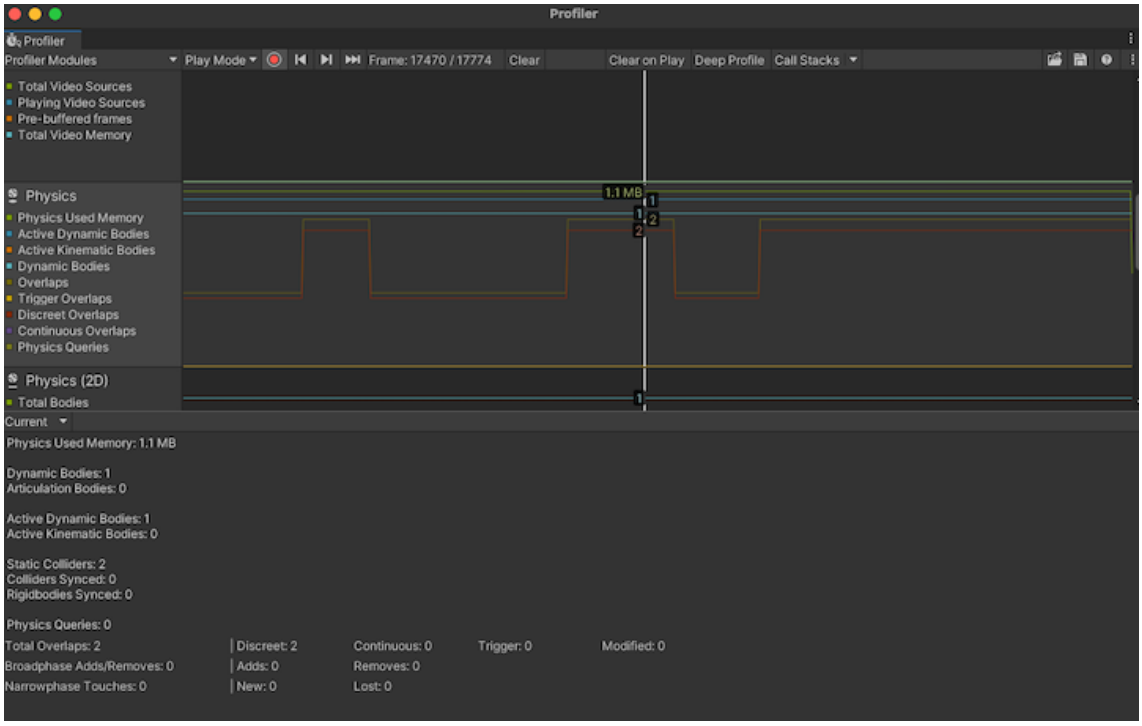
ale umožňují detekovat překryv s jinými tělesy. Tento ukazatel může pomoci v identifikaci případů, kdy jsou těla typu Trigger použita neefektivně nebo nevhodně v aplikaci.

Discrete Overlaps, zahrnuje počet překryvů mezi tělesy, které jsou detekovány diskrétně. Diskrétní detekce překryvů znamená, že překryvy jsou detekovány ve specifických časech, jako například během určitých fází fyzikálního výpočtu. Překryvy mohou být detekovány pomocí třídy Collider v Unity, která zajišťuje, že se dvě tělesa neprotínají.

Continuous Overlaps, tato metrika obsahuje počet překryvů mezi tělesy, které jsou detekovány kontinuálně. Kontinuální detekce překryvů znamená, že překryvy jsou detekovány v každém kroku výpočtu, čímž se zajišťuje, že tělesa se neprotínají. Překryvy mohou být detekovány pomocí třídy Rigidbody v Unity, která umožňuje tělesům interagovat s fyzikálním enginem a simulovat fyzikální chování.

Physics Queries, sleduje počet dotazů na fyzikální engine během profilování aplikace. Tyto dotazy zahrnují například kontrolu kolizí, detekci překryvů a výpočet síly a momentu. Počet dotazů na fyzikální engine může ovlivnit celkový výkon aplikace, zejména pokud je počet dotazů vysoký. Proto je důležité monitorovat tuto metriku a optimalizovat kód aplikace tak, aby minimalizoval počet dotazů na fyzikální engine.

Physics Profiler poskytuje informace o počtu a složitosti fyzikálních objektů ve světě hry. To může zahrnovat statické objekty, jako jsou stěny a podlahy, stejně jako dynamické objekty, jako jsou postavy a vozidla. Sledováním počtu a složitosti těchto objektů mohou vývojáři optimalizovat své fyzikální výpočty, aby zajistili, že běží efektivně a nezpůsobují žádné zpomalení nebo chyby.[38]



Obrázek 10: Modul Physics Profiler[39]

5.9 2D Physics Profileru Modul

Samotný modul 2D Physics Profileru je specializovaný nástroj pro monitorování fyzikálních výpočtů a simulací v 2D herních aplikacích. Jedná se o klíčový nástroj pro vývojáře, kteří se snaží zabezpečit hladký průběh fyzikálních simulací ve svých hrách a optimalizovat výkon herního enginu.

Modul poskytuje různé metriky a informace o fyzikálních výpočtech v aplikaci, včetně množství a složitosti objektů, které jsou používány v simulaci. To může zahrnovat statické objekty, jako jsou zdi a podlahy, ale také dynamické objekty, jako jsou postavy a vozidla. Tímto způsobem mohou vývojáři snadno optimalizovat fyzikální výpočty tak, aby byly spolehlivé a nedocházelo k výkonnostním problémům.

Total Bodies, zobrazuje celkový počet těles (body) v 2D fyzikálním světě. To zahrnuje všechna dynamická, kinematická a statická tělesa. Tato metrika je užitečná pro sledování celkového počtu těles v aplikaci a může pomoci identifikovat problémy s vysokým počtem těles.

Active Bodies, sleduje počet těles, která jsou aktivní v 2D fyzikálním světě. Aktivní tělesa jsou tělesa, která jsou součástí simulace a jsou zahrnuta do výpočtů fyzikálního enginu.

Tato metrika může pomoci identifikovat problémy s vysokým počtem těles, které se podílejí na fyzikální simulaci, což může vést k pomalému chodu aplikace.

Sleeping Bodies, tato metrika ukazuje počet těles, která jsou v klidovém stavu v 2D fyzikálním světě. Klidová tělesa jsou tělesa, která se nepohybují a neinteragují s ostatními tělesy. Fyzikální engine může umožnit těmto tělesům vstoupit do klidového stavu, aby se snížilo množství výpočetního výkonu potřebného pro simulaci fyziky. Tato metrika může pomoci identifikovat problémy s vysokým počtem těles, která jsou stále aktivní a nepotřebně zatěžují výkon aplikace.

Dynamic Bodies, dynamická tělesa reagují na fyzikální síly a jsou ovlivněna gravitací a srážkami s jinými tělesy. Jsou určena pro objekty, které mají hmotnost a mohou se pohybovat.

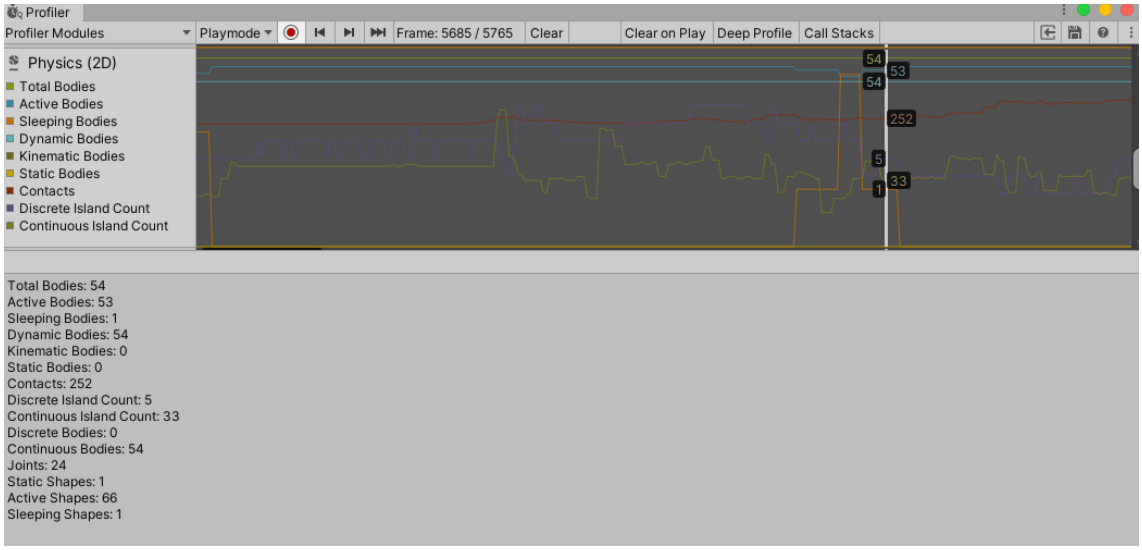
Kinematic Bodies, kinematická tělesa mají definovanou rychlost a směr pohybu, ale na rozdíl od dynamických těles, nejsou ovlivněna fyzikálními silami. Používají se pro objekty, které mají určený pohyb, ale nemají hmotnost.

Static Bodies: statická tělesa jsou nehybná a neovlivňují jiná tělesa, ale mohou být ovlivněna fyzikálními silami. Jsou určena pro objekty, které mají pevné umístění a nereagují na okolní interakce.

Contacts, ukazuje počet kontaktů mezi tělesy ve 2D fyzikálním světě. Kontakt se vytváří, když se dva objekty dotýkají. Vysoký počet kontaktů může způsobit zpomalení hry, protože musí být detekovány a vyřešeny kolize.

Discrete Island Count, v 2D fyzikálním světě mohou existovat samostatné izolované skupiny těles, které spolu neinteragují. Tyto skupiny se nazývají discrete islands. Metrika Discrete Island Count ukazuje, kolik takových skupin existuje v daném časovém okamžiku. Vysoký počet izolovaných skupin může být indikací neefektivního použití fyzikálního enginu.

Continuous Island Count, stejně jako u Discrete Island Count, tato metrika ukazuje počet izolovaných skupin těles. Rozdíl spočívá v tom, že Continuous Island Count sleduje skupiny, které spolu interagují nepřetržitě, například tělesa propojená pružinou. Vysoký počet takových skupin může způsobit zpomalení hry, protože musí být vypočítány a aktualizovány interakce mezi těmito skupinami.[40]



Obrázek 11: 2D Physics Profileru Modul[41]

5.10 Rendering Profiler module

Samotné vykreslování je klíčovým prvkem v herním enginu a má velký vliv na celkový výkon hry. Rendering Profiler module je nástroj, který umožňuje vývojářům sledovat výkon vykreslování v jejich hře a identifikovat možnosti optimalizace.

Jednou z nejdůležitějších funkcí Rendering Profileru je sledování počtu vykreslovaných objektů v různých situacích, například při zobrazování scény z různých úhlů nebo při použití různých grafických efektů. Tato informace umožňuje vývojářům identifikovat problémy s výkonem, jako jsou přetížení grafické karty nebo špatně optimalizované vykreslování objektů.

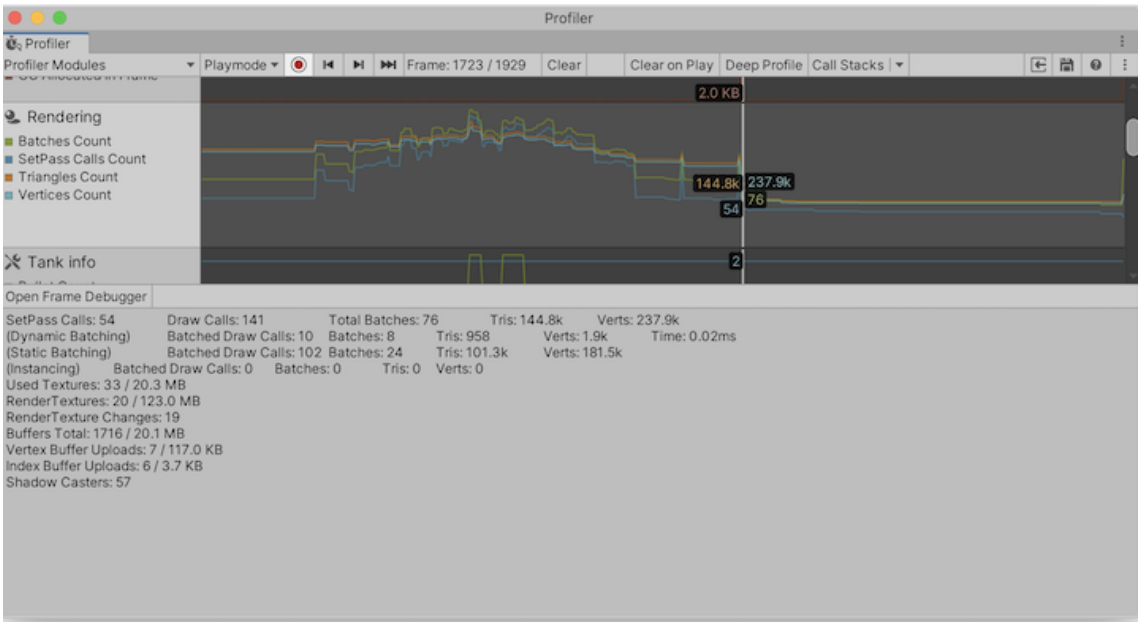
Rendering Profiler umožňuje vývojářům sledovat využití paměti grafickou kartou. To může být užitečné při ladění a optimalizaci grafických efektů a textur, aby se minimalizovala zátěž na grafické karty a zlepšila celková hratelnost hry.

Batches Count, počet samostatných skupin objektů, které musí být vykresleny zvlášť. Každá taková skupina vyžaduje jedno separátní volání, což může být poměrně náročné na výkon, zejména pokud je scéna složitá.

SetPass Calls Count, počet volání funkce SetPass, což je metoda, která se používá k nastavení aktuálního materiálu před vykreslením objektu. Každé volání SetPass může mít vliv na výkon, zejména pokud se provádí často.

Triangles Count, počet trojúhelníků, které jsou vykresleny v rámci framu. Trojúhelníky jsou základní geometrické prvky, které tvoří 3D modely. Vysoký počet trojúhelníků může mít negativní vliv na výkon, zejména pokud jsou trojúhelníky složité.

Vertices Count, počet vrcholů, ze kterých jsou vytvořeny trojúhelníky. Vrcholy jsou základní stavební bloky 3D modelů a vykreslování jich může být náročné na výkon. Vysoký počet vrcholů může mít negativní vliv na výkon, zejména pokud jsou vrcholy složité.[42]



Obrázek 12: Rendering Profiler module[43]

5.11 UI and UI Details Profiler

UI Profiler a UI Details Profiler jsou dva klíčové moduly, které pomáhají vývojářům her a aplikací optimalizovat uživatelské rozhraní (UI) a zlepšit zážitek uživatele. Tyto moduly poskytují detailní informace o výkonu UI, což umožňuje vývojářům identifikovat a odstranit potenciální problémy, jako jsou pomalé odezvy na uživatelské vstupy nebo zpoždění v zobrazení UI prvků.

UI Profiler umožňuje vývojářům sledovat celkový výkon UI, jako jsou doba odezvy, rychlost zpracování a frekvence snímání obrazovky. Tento modul také umožňuje vývojářům sledovat využití systémových prostředků, jako je CPU a GPU, a identifikovat případná úzká místa v UI procesu.[44]

UI Details Profiler umožňuje vývojářům prozkoumat detailněji jednotlivé prvky UI a jejich výkon. Tento modul zobrazuje informace o každém jednotlivém prvku UI, jako jsou doba

zpracování, velikost a umístění na obrazovce. Toto umožňuje vývojářům identifikovat jednotlivé prvky, které zpomalují celkový výkon UI a zlepšit jejich optimalizaci.

UI Profiler module

Layout, tato metrika měří čas potřebný k vypočtení a vykreslení layoutu (rozložení) UI elementů na obrazovce. Layout se obvykle provádí v situacích, kdy se změní velikost nebo pozice okna nebo kdy se přidají nebo odeberou prvky z UI. Vykreslení layoutu může být pomalé, pokud obsahuje složité prvky nebo pokud jsou prvky propojeny s mnoha dalšími prvky.

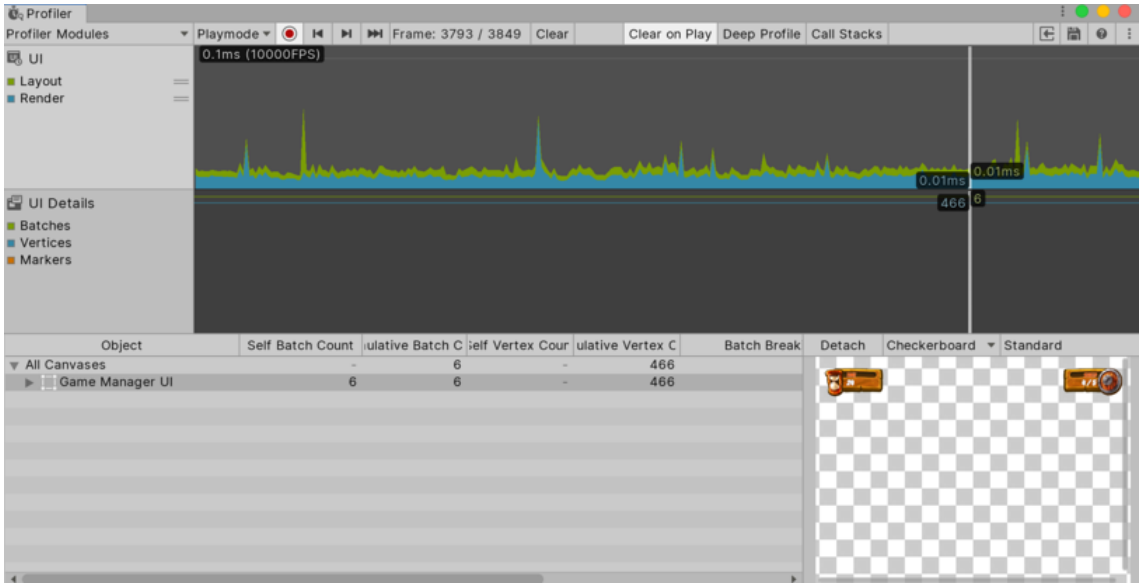
Render, měří čas potřebný k vykreslení UI na obrazovku. Vykreslení obvykle zahrnuje výpočet tvarů a velikostí jednotlivých prvků UI a jejich barev, textur a jiných vizuálních prvků. Vykreslování může být pomalé, pokud obsahuje složité prvky nebo pokud se provádí v každém snímku hry.

UI Details Profile module

Batches, zaznamenává počet renderovacích dávek (batches), které byly v průběhu profilování UI Details modulu vykresleny. Batches jsou soubory geometrických tvarů (např. obdélník, kruh), které se vykreslují najednou, aby se minimalizovala počet přepnutí mezi různými texturami a materiály.

Vertices, tato metrika zaznamenává počet vrcholů (vertices), které byly v průběhu profilování UI Details modulu vykresleny. Vrcholy jsou základní stavební kameny geometrických tvarů a každý vrchol obsahuje informace o poloze, normále, texturovacích souřadnicích atd.

Markers, zobrazuje počet značek (markers), které byly v průběhu profilování UI Details modulu vytvořeny. Značky slouží k označení určitých částí uživatelského rozhraní a mohou být použity k určení případných výkonnostních problémů.[45]



Obrázek 13: UI and UI Details Profiler[46]

5.12 Video Profiler module

Modul Video Profiler je nezbytným nástrojem pro vývojáře her k monitorování a optimalizaci výkonu přehrávání videa v jejich hře. Tento modul poskytuje různé informace a metriky související s přehráváním videa, umožňující vývojářům optimalizovat výkon své hry a zlepšit zážitek hráčů.

Jednou z klíčových funkcí modulu Video Profiler je schopnost sledovat výkon různých video kodeků a rozlišení. Tento modul může sledovat využití CPU a GPU během přehrávání videa a identifikovat jakékoliv problémy výkonu související s dekódováním nebo vykreslováním videa. Také umožňuje vývojářům testovat různé video kodeky a rozlišení, aby našli nejvhodnější konfiguraci pro potřeby své hry.

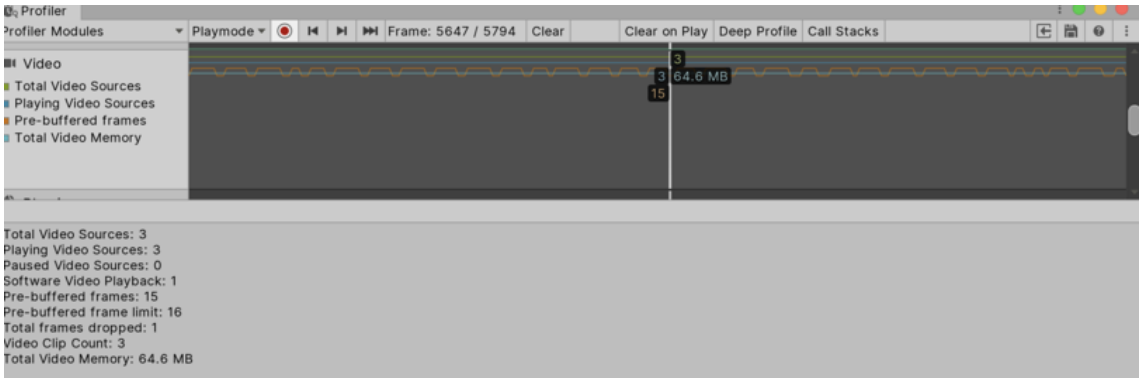
Dalším důležitým aspektem Video Profileru je schopnost sledovat kvalitu videa a identifikovat jakékoliv problémy související s artefakty komprese nebo jinými vizuálními defekty. Tento modul může analyzovat data videa fram po framu a detekovat jakékoliv problémy související s přesností barev, jasem, kontrastem nebo jinými vizuálními vlastnostmi. Identifikací a opravou těchto problémů mohou vývojáři zajistit, aby přehrávání videa v jejich hře bylo co nejkvalitnější.

Total Video Sources, sleduje celkový počet zdrojů videa v aplikaci. Pokud vaše aplikace obsahuje více videí, tak zde můžete sledovat, jak se počet zdrojů videa mění v průběhu času nebo v různých částech aplikace.

Playing Video Sources, tato metrika zobrazuje počet zdrojů videa, které právě hrají. Pokud vaše aplikace obsahuje více videí, tak tuto metriku můžete sledovat, aby zjistili, kolik videí se současně přehrává. Pokud je tato hodnota příliš vysoká, může to vést k problémům s výkonem.

Pre-buffered frames, ukazuje počet snímků videa, které jsou předem načteny do paměti. Tyto snímky jsou načteny předem, aby se předešlo zpožděním během přehrávání videa, kdy se musí načítat další snímky.

Total Video Memory, tato metrika ukazuje celkovou paměť použitou pro videa v aplikaci. Tuto metriku můžete sledovat, aby zjistili, jaký vliv má přehrávání videa na spotřebu paměti v aplikaci. Pokud je tato hodnota příliš vysoká, může to vést k problémům s výkonem.[47]



Obrázek 14: Video Profiler module[48]

5.13 Virtual Texturing Profiler

Virtual Texturing Profiler je výkonný nástroj pro vývojáře her, který umožňuje monitorovat a optimalizovat výkon virtuálního texturování ve hrách. Tento modul poskytuje různé metriky související s virtuálním texturováním, což umožňuje vývojářům optimalizovat výkon svých her a zlepšit herní zážitek hráčů.

Virtuální texturování je technika používaná v moderních herních enginách pro efektivní vykreslování textur s vysokým rozlišením s minimální paměťovou zátěží. Pracuje tím, že rozděluje velké textury na menší dlaždice a dynamicky je načítá a odstraňuje z paměti GPU podle potřeby. Tato technika může výrazně snížit využití paměti, ale pokud není správně implementována, může také způsobit výkonnostní problémy.

Modul umožňuje analýzu dat textury a detekci jakýchkoli problémů souvisejících s kompresními artefakty nebo jinými vizuálními defekty. Tím mohou vývojáři zajistit, že virtuální texturování jejich hry bude mít nejvyšší možnou kvalitu.

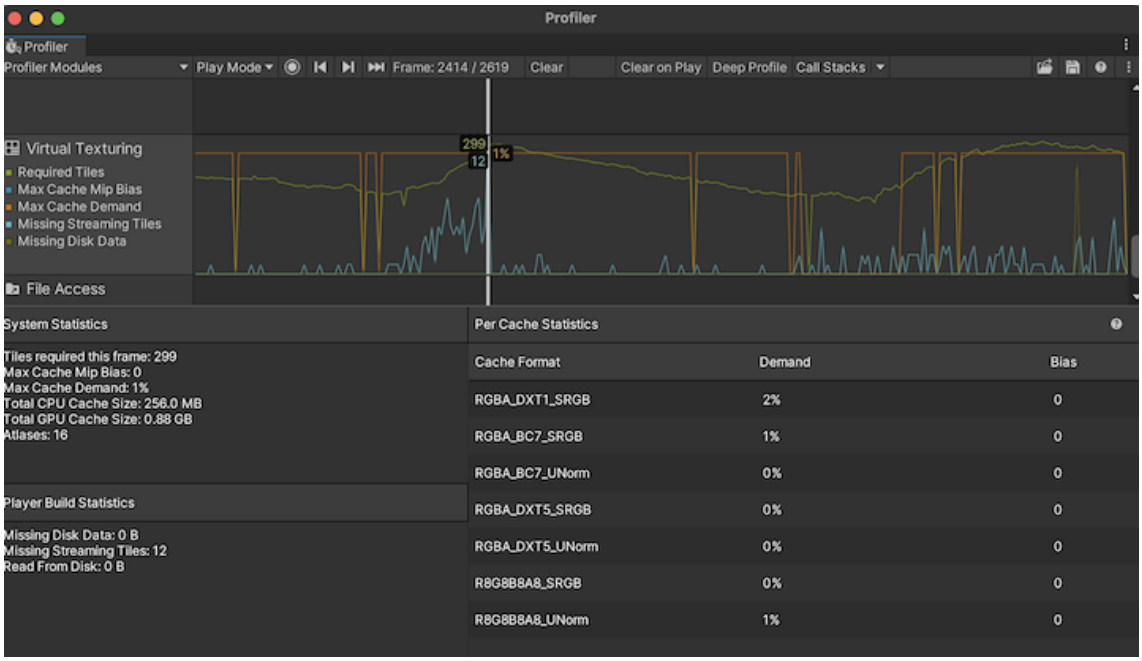
Required Tiles, požadované dlaždice jsou počet dlaždic, které jsou vyžadovány pro vykreslení aktuálních pohledů na scénu. Tato metrika ukazuje, jaké množství textur musí být načteno v paměti a jak velké bude zatížení disku.

Max Cache Mip Bias, virtuální texturování využívá mezipaměťování pro ukládání již načtených dlaždic. Tato metrika ukazuje maximální rozdíl mezi výchozím MIP level a požadovaným MIP level pro vykreslení dlaždice.

Max Cache Demand, ukazuje maximální počet dlaždic, které byly požadovány pro virtuální texturování v průběhu hry. Pokud se tato hodnota přibližuje k maximálnímu počtu dlaždic v mezipaměti, může to znamenat, že virtuální texturování je využíváno velmi intenzivně a může způsobit přetížení paměti.

Missing Streaming Tiles, sleduje počet dlaždic, které nebyly načteny včas pro vykreslení aktuálních pohledů na scénu. Pokud je tato hodnota vysoká, může to znamenat, že virtuální texturování má problém s načítáním textur a může způsobit zpoždění při renderování scény.

Missing Disk Data, tato metrika zobrazuje počet dlaždic, které nebyly nalezeny na disku. Pokud je tato hodnota vysoká, může to znamenat, že virtuální texturování má problém s načítáním textur a může způsobit zpoždění při renderování scény.[49]



Obrázek 15: Virtual Texturing Profiler[50]

II. PRAKTICKÁ ČÁST

6 POPIS MOBILNÍ HRY

Návrh

Tato mobilní hra je jednoduchá klikací hra, která umožňuje hráči získávat coiny prostřednictvím různých tlačítek a scén. Hráč má možnost přepínat mezi scénami. Hlavní scéna obsahuje primární tlačítko, které hráč používá k získávání coinů. Hlavní tlačítko lze vylepšovat, aby hráč mohl získávat více coinů snadněji a rychleji. Hra má možnost automatického kupování vylepšení nebo získávání gemů. Pomocí gemů může kupovat silnější vylepšení pro tlačítko. Má funkci ukládání dat, aby hráč mohl pokračovat po vypnutí. Umožňuje přizpůsobit orientaci na výšku nebo na šířku.

Požadavky

Hlavní scéna bude obsahovat primární tlačítko, které hráči umožňuje získávat coiny klikáním. Toto tlačítko bude možné vylepšovat, aby hráč mohl rychleji získávat více coinů.

Ve scéně Upgrades bude hráči k dispozici čtyři tlačítka. Dvě z těchto tlačítek slouží k vylepšení hlavního tlačítka, které kliknutím generuje více coinů. Zbývající dvě tlačítka slouží k vylepšení zisku coinů za sekundu, což umožňuje hráči získávat coiny i bez neustálého klikání.

V scéně Automation se bude nacházet čtyři tlačítka. Tato tlačítka slouží k automatickému vylepšování tlačítek ve scéně Upgrades. Automatizované vylepšení umožní hráči získávat coiny bez aktivního klikání.

Ve scéně Prestige hráč bude mít možnost získat prestige měnu po dosažení určitého množství coinů. Po kliknutí na tlačítko Prestige se hra resetuje a hráč začíná znovu. Získání prestige má výhodu, že hráč může vylepšit efektivitu kliknutí nebo zisku coinů za sekundu o 50 %.

Hra se bude automaticky ukládána každých 5 sekund. Tím se zajistí, že hráč bude mít uložen svůj aktuální stav a pokrok i po opuštění a načtení hry.

Hráč bude mít možnost volby mezi zobrazením hry na výšku nebo na šířku. Tím si každý hráč bude moci vybrat preferovaný způsob zobrazení a hrát pohodlněji podle svých individuálních preferencí.

Kromě výše uvedených požadavků bude vytvořena neoptimalizovaná verze hry, která bude sloužit k simulování zatížení hry s neoptimalizovaných metod jako je Particle systém, LOD, v kódu nebo snížení objektů. Také bude vytvořena optimalizovaná verze hry, kde budou tyto metody optimalizovány, aby byla dosažena plynulá a výkonná hra.

Implementace

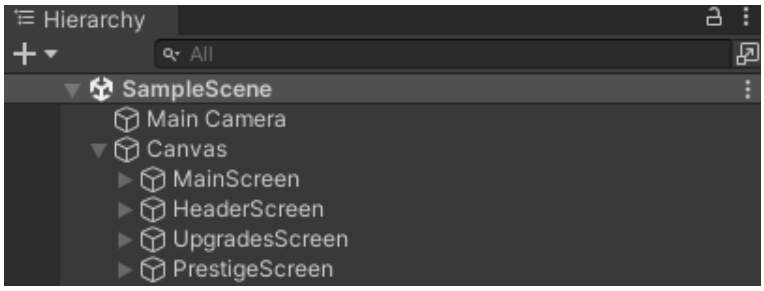
V hlavní scéně je umístěno primární tlačítko, které hráč používá k získávání coinů. Aby hráč mohl získávat více coinů mnohem snadněji a rychleji, je možné hlavní tlačítko vylepšovat. Ve scéně *Upgrades* se nachází čtyři tlačítka, která hráči umožňují vylepšit hlavní tlačítko. Dvě z těchto tlačítek slouží k zvýšení počtu coinů generovaných, kliknutím na hlavní tlačítko, zatímco zbývající dvě tlačítka zvyšují zisk coinů za sekundu, což hráči umožňuje získávat coiny i bez neustálého klikání. Další scéna s názvem *Automation* nabízí hráči možnost automatizovat proces vylepšování tlačítek ve scéně *Upgrades*. Čtyři tlačítka v této scéně slouží k automatickému vylepšování tlačítek a umožňují hráči získávat coiny bez aktivního klikání. Tato vylepšení jsou však nákladná, a proto se hráč bude muset potřeba šetřit a strategicky se rozhodovat, které automatizace si může dovolit. V poslední scéně s názvem *Prestige* hráč může získat prestižní měnu po dosažení určitého počtu coinů. Po kliknutí na tlačítko *Prestige* a získání prestiže se hra resetuje a hráč začíná znovu. Hráč však za získanou prestižní měnu může vylepšit efektivitu klikání o 50 % nebo zisk coinů za sekundu o 10 %. Pro zajištění pohodlného hraní je hra navržena tak, aby hráč měl možnost hrát na výšku i na šířku obrazovky, což jim umožňuje vybrat si nejpohodlnější variantu hraní. Hra se automaticky ukládá každých 5 sekund, aby hráč neztratil svá vylepšení a nasbírané coiny. Tím se zajišťuje, že hráč nebude muset začínat znovu od začátku a může pokračovat ve svém postupu.

7 POSTUP VYTVOŘENÍ

7.1 Příprava prostředí

Na začátku bylo nutné přepnout platformu, protože Unity automaticky vybere platformu pro počítač. Bylo tedy potřeba otevřít *Build Settings* ve záložce *File* a přepnout na Android. Po této konfiguraci bylo nutné počkat na dokončení nastavení prostředí. Poté jsem si stáhnul pomocný skript *BigDouble*, který slouží k převodu velmi velkých čísel na uživatelsky čitelný formát. Tento skript *BigDouble* jsem stáhl z repozitáře na Githubu.[51]

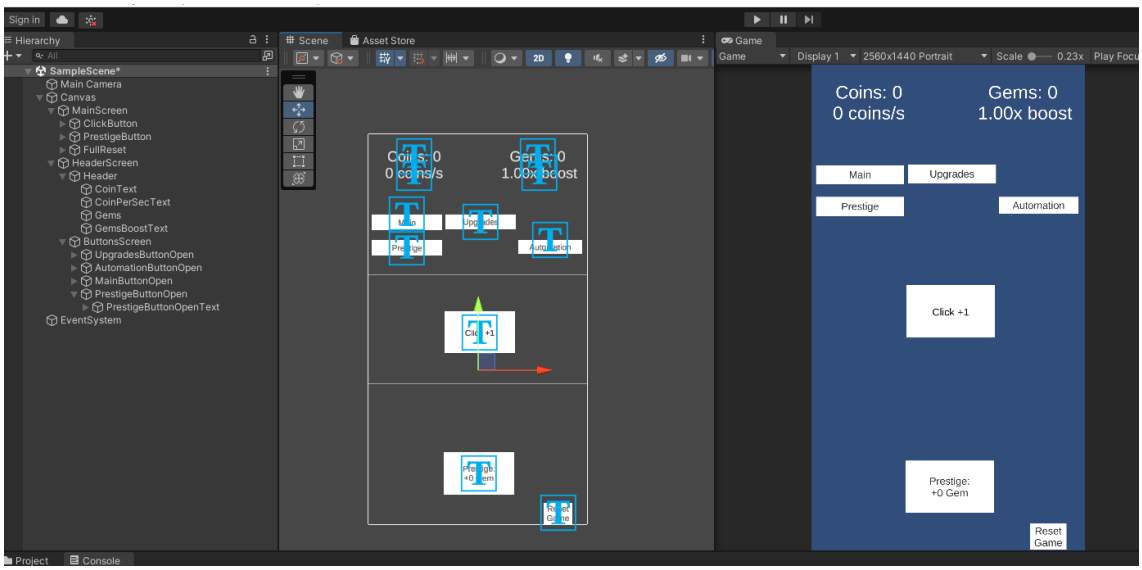
Jako první bylo nutné vytvořit Canvas, který byl umístěn nad ostatními objekty v hierarchii projektu. Vytvoření Canvasu bylo provedeno kliknutím pravým tlačítkem v okně pro zobrazení hierarchie a volbou možnosti UI, následovanou vytvořením Canvasu. Poté byly vytvořeny obrazovky, mezi kterými bude možné přepínat. V tomto případě se jednalo o obrazovky *MainScreen*, *HeaderScreen*, *UpgradeScreen*, *PrestgeScreen* a *AutomationScreen*. Tyto obrazovky byly vytvořeny stejným způsobem jako první Canvas, s tím rozdílem, že nebylo klikáno pravým tlačítkem myši v okně pro zobrazení hierarchie, ale přímo na Canvasu.



Obrázek 16: Příprava screeny v Hierarchy

Následně bylo nutné vytvořit hlavní tlačítko, pomocí kterého by hráč získával coins. Tento proces byl proveden kliknutím pravým tlačítkem na *MainScreen*, vybráním možnosti UI a následně možnosti *Buttons*. Tímto bylo vytvořeno tlačítko. Tento proces byl zopakován ještě dvakrát, čímž bylo získáno celkem tři tlačítka. Další dvě tlačítka byla použita později v *MainScreen*. Tato tři tlačítka byla pojmenována *ClickButton*, *PrestigeButton* a *FullReset*. Následně bylo pře-nášeno do *HeaderScreen*, kde bylo vytvořeno čtyři textová pole. Tato pole byla vytvořena kliknutím pravým tlačítkem UI a následně možnosti Text - *TextMashPro*. Textová pole byla pojmenována *CoinText*, *CoinPerSecText*, *Gems* a *GemBoostText*. Poté byl vytvořen *ButtonScreen* uvnitř *HeaderScreenu*. *ButtonScreen* sloužil k přepínání tlačítek v jednom screenu a usnadňoval jejich vyhledávání. Zde bylo vytvořeno čtyři tlačítka stejným způsobem jako předchozí. Tato tlačítka byla pojmenována *UpgradeButtonOpen*,

AutomationButtonOpen, *MainButtonOpen* a *PrestigeButtonOpen*. V scéně, kde se nacházela tlačítka a texty, byla jejich rozmístění provedeno dle vlastního uvážení.



Obrázek 17: Vytvoření tlačítek na ovládání scén

Nyní jsou připravena tlačítka a texty pro programování. V okně projektu je provedeno kliknutí pravým tlačítkem a vybrána možnost *Create*, a poté C# script. Tento script je pojmenován *GameManager* a je otevřen. Stejné kroky jsou opakovány pro druhý script, který je pojmenován *PlayerData*. V tomto scriptu budou uloženy veškeré nasbírané coiny, počet coinů za kliknutí a budoucí hodnoty.

Aktuálně jsou vytvořeny dva prázdné scripty, *GameManager* a *PlayerData*. Přechází se do scriptu *PlayerData* a vše je nutné smazat kromě using Systems. Vytvoří se *public class PlayerData*, která je *Serializable* field. Dále je provedeno nadefinování propojení s scriptem *GameManager,* pomocí zápisu public *GameManager* game. Posledním krokem je nadefinování coins a *coinsClickValue* jako *public BigDouble* a uložení těchto hodnot.

```
[Serializable]
Počet odkazů: 1
public class PlayerData
{
    public GameManager game;


    public BigDouble coins;
    public BigDouble coinsClickValue;
```

Obrázek 18: Třída Player na proměnné

Přechází se do scriptu *GameManager* a zde je nadefinován text, který se bude průběžně měnit a také tlačítka pro přepínání screenů. V scriptu nad metodou Start() je nutné napsat *TMP_Text*, který bude sloužit k zobrazení textu. Posledním krokem je propojení s *PlayerData* scriptem, aby bylo možné používat proměnné z tohoto scriptu. To je provedeno pomocí public *PlayerData* data.

```
public class GameManager : MonoBehaviour
{

    public PlayerData data;

    public TMP_Text coinText;
    public TMP_Text clickValueText;
    public TMP_Text coinsPerSecText;

    public TMP_Text coinLandScapeText;
    public TMP_Text clickValueLandScapeText;
    public TMP_Text coinsPerSecLandScapeText;
```

Obrázek 19: Definování proměnných pro text

Nyní se přechází do funkce Start(), která se spouští při zapnutí hry. V této funkci je nadefinováno, aby při zapnutí byly aktivovány pouze *HeaderGroup* a *mainMenuGroup*, aby nedošlo k aktivaci všech screenů najednou. To je provedeno pomocí použití SetActive(), který slouží k aktivaci vybraného screenu.

```
Zpráva Unity | Počet odkazů: 0
public void Start()
{


    mainMenuGroup.gameObject.SetActive(true);
    headerGroup.gameObject.SetActive(true);
    upgradesGroup.gameObject.SetActive(false);
    prestige.presige.gameObject.SetActive(false);



}
```

Obrázek 20: Zapnutí hlavní scény při startu

Nyní je nadefinováno hlavní tlačítko, které zvyšuje počet coinů při kliknutí. K tomu jsou nadefinovány dvě funkce, a to TotalClickValue() a Click(). Tyto funkce jsou umístěny pod funkcí Update(). Funkce TotalClickValue() slouží k výpočtu celkového počtu coinů za jedno kliknutí. Funkce Click() pak slouží k provedení kliknutí a přičtení počtu coinů za jedno kliknutí k celkovému počtu coinů.

```
private BigDouble TotalClickvalue()
{
    BigDouble temp = data.coinsClickValue;


    return temp;
}
Počet odkazů: 0
public void Click()
{
    data.coins += TotalClickvalue();
}
```

Obrázek 21: Funkce na výpočet coinů za kliknutí a samotné klinutí

Nyní je naprogramována funkce pro přepínání screenů, která umožňuje přecházet mezi jednotlivými screeny. Tato funkce se nazývá ChangeTabs() a přijímá vstupní parametr string id. V této funkci je použit switch, který jednoduše přepíná screeny na základě poskytnutého id. Každý case v switchi je pojmenován *Upgrades*, *Main*, *Prestige* a *Automation*. Pro každý case je použito již známé *SetActive*, které zapíná daný screen. Na konci každého case je také přidán příkaz *break*, aby se neprocházely další case. Kromě toho je také přidána funkce DisableAll(), která vypíná všechny screeny a není třeba je vypínat jednotlivě. Tato funkce je umístěna v ChangeTabs(), ale mimo příkaz *switch*.

```
public void ChangeTabs(string id)
{
    DisableAll();
    switch (id)
    {
        case "Upgrades":
            upgradesGroup.gameObject.SetActive(true);
            break;
        case "Main":
            mainMenuGroup.gameObject.SetActive(true);
            break;
        case "Prestige":
            prestige.presige.gameObject.SetActive(true);
            break;
        case "Automation":
            AutomationGroup.gameObject.SetActive(true);
            break;
    }


    void DisableAll()
    {
        mainMenuGroup.gameObject.SetActive(false);
        upgradesGroup.gameObject.SetActive(false);
        prestige.presige.gameObject.SetActive(false);
        AutomationGroup.gameObject.SetActive(false);
    }
```

Obrázek 22: Funkce pro tlačítka na přepínání a vypnutí veškerých scén

Poté bylo nutné přiřadit hodnotu proměnné coins k proměnné *coinText* a hodnotu proměnné *TotalCoinsPerSecond* k proměnné *coinsPerSecText*. Před tím však vytvoříme funkci NotationMethod(), která slouží k převodu čísel na zvolený formát.

```
public class Methods : MonoBehaviour
{


    Počet odkazů: 30
    public static string NotationMethod(BigDouble x, string y)
    {
        if (x >= 1000)
        {
            var exponent = (Floor(Log10(Abs(x))));
            var mantissa = (x / Pow(10, exponent));

            return mantissa.ToString("F2") + "e" + exponent;
        }
        return x.ToString(y);

    }


}
```

Obrázek 23: Funkce na převod coinů

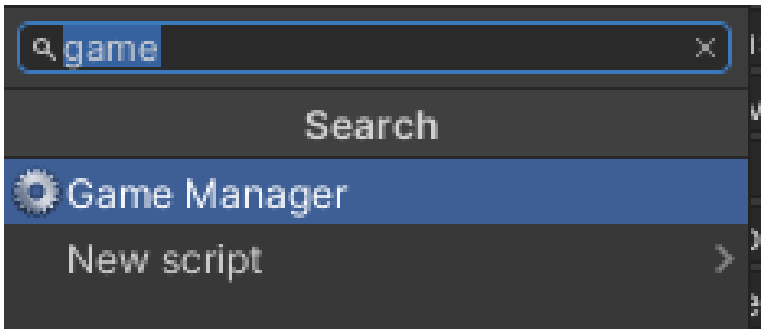
Funkce *NotationMethod* z obrázku 23 bude umístěna ve třídě *Methods*, kterou vytvoříme jako class. Funkce *NotationMethod* přijímá dva vstupní parametry: číslo *x* a řetězec *y*. Pokud je číslo *x* větší nebo rovno 1000, použije se exponentiální zápis. V tomto případě se nejprve určí exponent pomocí funkce *Floor*, která vrátí největší celé číslo menší nebo rovno zadanému argumentu. Poté se vypočítá mantisa, což je hodnota *x* dělená desetinou mocninou čísla 10 umocněného na exponent. Nakonec se vrátí řetězcová reprezentace mantisy s formátem *F2* (který značí zobrazení s dvěma desetinnými místy) a exponentu ve tvaru *e*. Jelikož se jedná o klikací hru, kde čísla mohou být velmi velká, což by pro uživatele mohlo být obtížně čitelné, proto je číslo převedeno na mnohem uživatelsky přívětivější formát.

Následně bylo nutné vytvořit část k zobrazení celkového počtu coinů a počtu coinů za sekundu. Toto je provedeno v části Update(), kde je využita proměnná coinsPerSecText a coinText.

```
coinsPerSecText.text = TotalCoinsPerSecond().ToString("F0") + " coins/s";
coinText.text = "Coins: " + Methods.NotationMethod(data.coins, "F0");
```

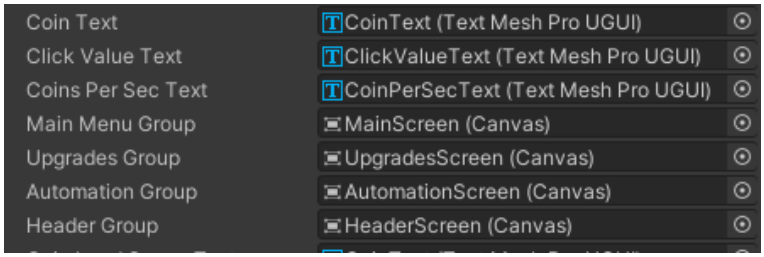
Obrázek 24: Přiřazení k textu hodnoty celkové coiny a coiny za sekundu

Scripty jsou uloženy a je nutné se vrátit zpět do editoru. Zde proběhne zkompilování *scriptů*. Po dokončení kompilace by v konzoli neměly být žádné chyby a může se přejít k přiřazování screenů, textů a tlačítek. To je provedeno vytvořením prázdného objektu v hlavním Canvasu. Následně kliknutím pravým na Canvas a výběrem možnosti *Create Empty*. Tento prázdný objekt je pojmenován *scripts*. V objektu *scripts* je vytvořen další *Create Empty*, který je pojmenován *GameManager*. Do tohoto objektu je přidán script *GameManager*. To je provedeno kliknutím na objekt *GameManager,* v pravém panelu *Inspector* a následně kliknutím na tlačítko *Add Component*. Zobrazí se seznam dostupných komponent a možnost vyhledávání podle názvu. Následně je napsán text *game* a vybrán *GameManager*.



Obrázek 25: Vybrání scriptu GameManager

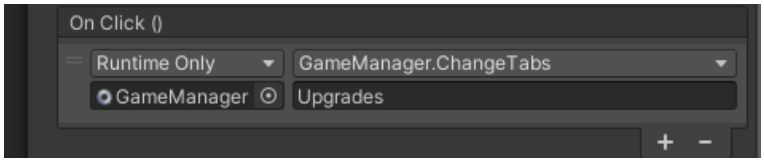
To způsobí, že se zobrazí seznam s instancemi. Následně je vybrán text s názvem *Coin Text* v Inspektoru a přiřazen ke *Coin Textu* v hierarchii. Stejně tak je přiřazován *Click Value Text* z Inspektoru k C*lickValueText* z hierarchie a pokračuje se v tomto procesu, dokud nejsou přiřazeny všechny texty a screeny.



Obrázek 26: Přiřazení textu z Hierarchie k proměnám scriptu

Pokud je vše přiřazeno, přejde se k přiřazení tlačítek. Klikne se na tlačítko *ClickButton* v hierarchii. V inspektoru se posune dolů k položce On Click(). Klikne se na tlačítko plus a zobrazí se možnost přidat objekt. Z hierarchie se vybere *GameManager* a přetáhne se do pole *None (Object)*, čímž se aktivuje pole *No Function*. Na toto tlačítko se klikne. Vybere se *GameManager* a zvolí se funkce Click(). Tyto kroky se opakují pro další tlačítka při změně screenů. Přiřazování tlačítek k funkcím se provádí opatrně, aby byla správně přiřazena

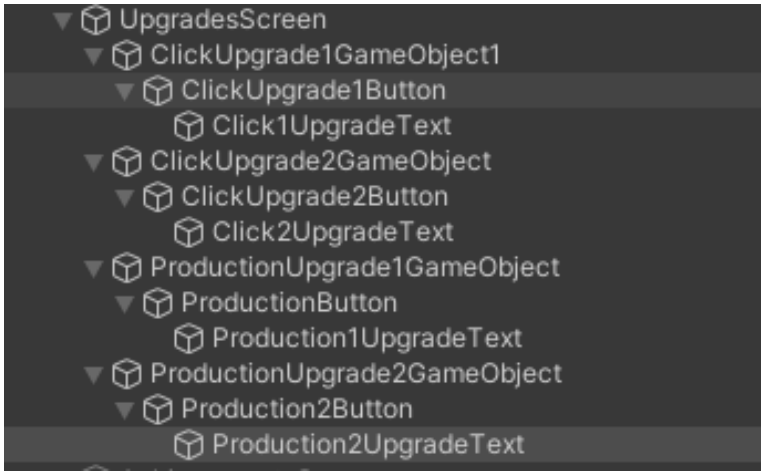
funkce k odpovídajícímu screenu. Při výběru funkce ChangeTabs() se zobrazí nová kolonka, která obsahuje vstupní řetězcový parametr *id*. Zde se napíše text pro screen, který má být ovládán daným tlačítkem.



Obrázek 27: Vybrání funkce na přepínání tlačítek

7.2 Upgrade Screen

Zde se bude vylepšovat naše hlavní tlačítko a generování coinů za sekundu. Vytvoří se čtyři tlačítka, dvě tlačítka slouží pro vylepšení hlavního tlačítka a další dvě pro generování coinů za sekundu. Začne se tím, že se vytvoří prázdný objekt v rámci *UpgradesScreen*. V tomto objektu se přidá čtyři tlačítka. Tento proces se zopakuje třikrát. Tlačítka se rozmístí a přejmenují podle obrázku 28.



Obrázek 28: Hierarchie tlačíteke v UpgradeScreenu

V projektovém okně je vytvořen další script s názvem *UpgradeManager*. Tento script slouží k nákupu vylepšení hlavního tlačítka a generování coinů za sekundu. Na začátku je nutné vytvořit proměnné pro *TMP_Text*, *GameObject* a samozřejmě propojení s *GameManagerem*.

```
public class UpgradeManager : MonoBehaviour
{
    public GameManager game;

    //Click Upgrades
    public GameObject clickUpgrade1;
    public GameObject clickUpgrade2;
    public TMP_Text clickUpgrade1Text;
    public TMP_Text clickUpgrade2Text;

    public BigDouble clickUpgrade1Cost;
    public BigDouble clickUpgrade2Cost;
    public BigDouble clickUpgrade1Power = 1;
    public BigDouble clickUpgrade2Power = 5;
    public BigDouble clickUpgrade1Level;
    public BigDouble clickUpgrade2Level;
    public BigDouble clickUpgrade1BaseCost = 10;
    public double clickUpgrade1BaseCostMult = 1.07;
    public BigDouble clickUpgrade2BaseCost = 25;
    public double clickUpgrade2BaseCostMult = 1.07;


    //Production Upgrades
    public GameObject productionUpgrade1;
    public GameObject productionUpgrade2;
    public TMP_Text productionUpgrade1Text;
    public TMP_Text productionUpgrade2Text;

    public BigDouble productionUpgrade1Cost;
    public BigDouble productionUpgrade2Cost;
    public BigDouble productionUpgrade1Level;
    public BigDouble productionUpgrade2Level;
    public BigDouble productionUpgrade1BaseCost = 25;
    public double productionUpgrade1BaseCostMult = 1.07;
    public BigDouble productionUpgrade2BaseCost = 250;
    public double productionUpgrade2BaseCostMult = 1.07;
```

```
public class UpgradeManager : MonoBehaviour
{
    public GameManager game;
    //Click
    public TMP_Text[] clickUpgradeText = new TMP_Text[2];

    public BigDouble[] clickUpgradeCost;
    public BigDouble[] clickUpgradePower;
    public BigDouble[] clickUpgradeLevels;
    public BigDouble[] clickUpgradeBaseCosts;
    public double[] clickUpgradeBaseCostsMults;
    //Production
    public TMP_Text[] productionUpgradeText = new TMP_Text[2];

    public BigDouble[] productionUpgradeCost;
    public BigDouble[] productionUpgradeLevels;
    public BigDouble[] productionUpgradeBaseCosts;
    public double[] productionUpgradeBaseCostsMults;
    Zpráva Unity | Počet odkazů: 0
    private void Start()
    {
        clickUpgradeCost = new BigDouble[2];

        clickUpgradeBaseCosts = new BigDouble[]{ 10,25 };
        clickUpgradePower = new BigDouble[] { 1, 5 };
        clickUpgradeBaseCostsMults = new [] { 1.07, 1.07 };
        clickUpgradeLevels = new BigDouble[2];


        productionUpgradeCost = new BigDouble[2];
        productionUpgradeLevels = new BigDouble[2];
        productionUpgradeBaseCosts = new BigDouble[] { 25, 250 };
        productionUpgradeBaseCostsMults = new[] { 1.07, 1.07 };


    }
```

Obrázek 29: Neoptimalizovaný (vlevo) a optimalizovaný (vpravo) kód

Rozdíl mezi levým a pravým kódem spočívá v použití pole pro jednodušší přiřazování proměnných a budoucích úprav.

Následně byly vytvořeny dvě pomocné funkce a to ArrayManager() a NonArrayManager(). Funkce ArrayManager() načítá hodnoty z datového pole *game.data* a ukládá je do odpovídajících proměnných *clickUpgrade1Level, clickUpgrade2Level, productionUpgrade1Level a productionUpgrade2Level*. Funkce NonArrayManager() provádí opačný proces a ukládá hodnoty proměnných *clickUpgrade1Level, clickUpgrade2Level, productionUpgrade1Level a productionUpgrade2Level* zpět do datového pole *game.data*.

```
Počet odkazů: 1
private void ArrayManager()
{
    var data = game.data;
    clickUpgrade1Level = data.clickUpgrade1Level;
    clickUpgrade2Level = data.clickUpgrade2Level;
    productionUpgrade1Level = data.productionUpgrade1Level;
    productionUpgrade2Level = data.productionUpgrade2Level;
}
Počet odkazů: 4
private void NonArrayManager()
{
    var data = game.data;
    data.clickUpgrade1Level = clickUpgrade1Level;
    data.clickUpgrade2Level = clickUpgrade2Level;

    data.productionUpgrade1Level = productionUpgrade1Level;
    data.productionUpgrade2Level = productionUpgrade2Level;
}
```

Obrázek 30: ArrayManager() a NonArraymanager() pro Upgrades

Dalším krokem bylo vytvoření funkce RunUpgrades(). Tato funkce provádí výpočty cen pro vylepšení klikání a produkce a tyto ceny jsou uloženy do odpovídajících proměnných: *clickUpgrade1Cost, clickUpgrade2Cost, productionUpgrade1Cost a productionUpgrade2Cost*. Ceny vylepšení jsou vypočítány jako základní cena vylepšení vynásobená mocninou koeficientu, který se zvyšuje s úrovní vylepšení. Pro výpočet nákladů na klikání jsou použity hodnoty z proměnných *clickUpgrade1BaseCost, clickUpgrade1BaseCostMult, clickUpgrade1Level, clickUpgrade2BaseCost* a *clickUpgrade2BaseCostMult*. Pro výpočet nákladů na produkci jsou použity hodnoty z proměnných *productionUpgrade1BaseCost*, *productionUpgrade1BaseCostMult*, *productionUpgrade1Level*, *productionUpgrade2BaseCost* a *productionUpgrade2BaseCostMult*.

```
public void RunUpgrades()
{
    var data = game.data;
    ArrayManager();
    clickUpgrade1Cost = clickUpgrade1BaseCost * Pow(clickUpgrade1BaseCostMult, data.clickUpgrade1Level);
    clickUpgrade2Cost = clickUpgrade2BaseCost * Pow(clickUpgrade2BaseCostMult, data.clickUpgrade2Level);
    productionUpgrade1Cost = productionUpgrade1BaseCost * Pow(productionUpgrade1BaseCostMult, data.productionUpgrade1Level);
    productionUpgrade2Cost = productionUpgrade2BaseCost * Pow(productionUpgrade2BaseCostMult, data.productionUpgrade2Level);
}
```

Obrázek 31: funkce RunUpgrades() s výpočtem ceny

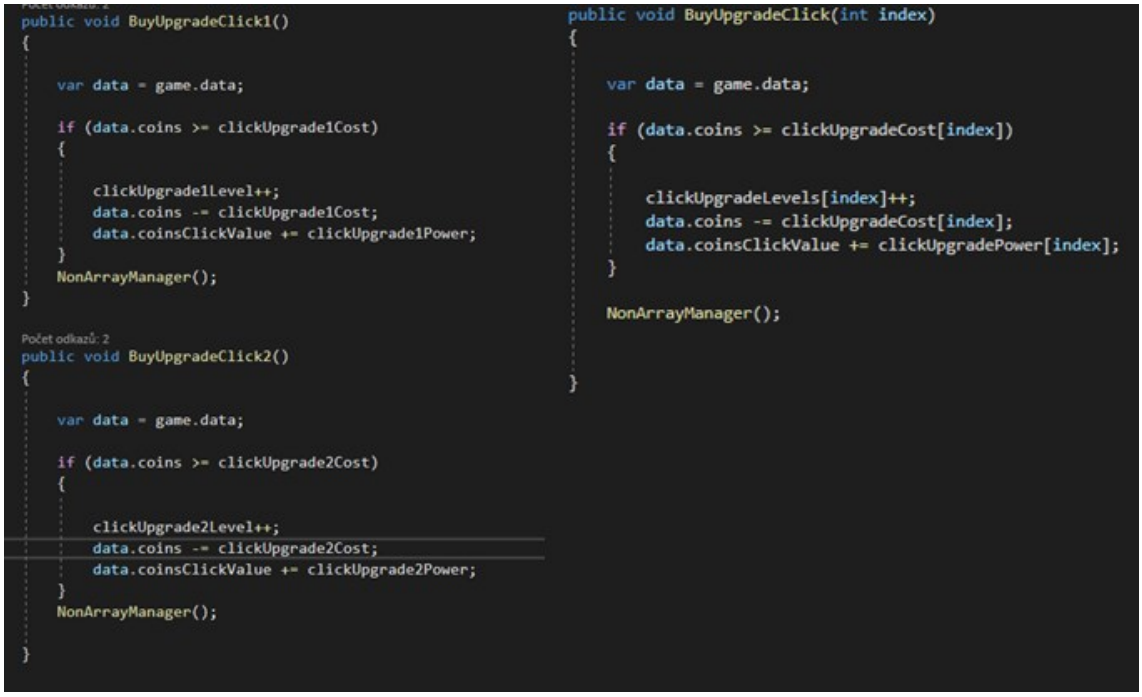
Bylo nutné vytvořit část, kde budou zobrazovány úrovně vylepšení, cena vylepšení a síla posílení. Pro tuto část byla použita metoda NotationMethod() a funkce RunUpgrades(). Funkce RunUpgrades() bude obsahovat logiku, která bude aktualizovat tyto proměnné.

```
public void RunUpgradesUI()
{
    var data = game.data;


        clickUpgrade1Text.text = $"Click Upgrade { 1 }\nCost: {Methods.NotationMethod(clickUpgrade1Cost, "F2")} " +
        $"coins\nPower: {clickUpgrade1Power} Click\nLevel: {Methods.NotationMethod(clickUpgrade1Level, "F2")}";
        clickUpgrade2Text.text = $"Click Upgrade { 2 }\nCost: {Methods.NotationMethod(clickUpgrade2Cost, "F2")} " +
        $"coins\nPower: {clickUpgrade2Power} Click\nLevel: {Methods.NotationMethod(clickUpgrade2Level, "F2")}";

        productionUpgrade1Text.text = $"Production Upgrade {1}\nCost: {Methods.NotationMethod(productionUpgrade1Cost, "F2")} " +
        $"coins\nPower: +{Methods.NotationMethod(game.data.prestigeUlevel2, "F2")} coins/s\nLevel: {Methods.NotationMethod(productionUpgrade1Level, "F2")}";
        productionUpgrade2Text.text = $"Production Upgrade {2}\nCost: {Methods.NotationMethod(productionUpgrade2Cost, "F2")} " +
        $"coins\nPower: +{Methods.NotationMethod(game.data.prestigeUlevel2, "F2")} coins/s\nLevel: {Methods.NotationMethod(productionUpgrade2Level, "F2")}";

}
public void RunUpgradesUI()
{
    var data = game.data;
    string GetUpgradeCost(int index,BigDouble[] upgrade)
    {
        return Methods.NotationMethod(upgrade[index], "F2");
    }
    string GetUpgradeLevel(int index,BigDouble[] upgradelevel)
    {
        return Methods.NotationMethod(upgradelevel[index], "F2");
    }
    for (var i = 0; i < 2; i++)
    {
        clickUpgradeText[i].text = $"Click Upgrade {i + 1}\nCost: {GetUpgradeCost(i, clickUpgradeCost)} coins\nPower:" +
        $" {clickUpgradePower[i]} Click\nLevel: {GetUpgradeLevel(i, clickUpgradeLevels)}";

        productionUpgradeText[i].text = $"Production Upgrade {i}\nCost: {GetUpgradeCost(i, productionUpgradeCost)} " +
        $"coins\nPower: +{Methods.NotationMethod(game.prestige.levels[1], "F2")} coins/s\nLevel: {GetUpgradeLevel(i, productionUpgradeLevels)}";
    }
}
```

Obrázek 32: Neoptimalizovaný (nahoře) a optimalizovaný kód s for cyklem (dole)

Rozdíl mezi kódy je ten, že druhý kód používá smyčku *for*, která prochází všechny upgrady a pomocí funkcí GetUpgradeCost() a GetUpgradeLevel() získává potřebné informace o nákladech a úrovních upgradů. Tyto informace jsou pak přiřazovány odpovídajícím textovým polím namísto explicitního aktualizování každého pole zvlášť.

Poté byly vytvořeny funkce pro vylepšení hlavního tlačítka. V těchto funkcích se kontroluje, zda hráč má dostatek coinů k zakoupení vylepšení. Pokud ano, provede se zvýšení úrovně vylepšení plus odečtení coinů a přidání navýšení počtu coinů za jedno kliknutí.



```
public void BuyUpgradeClick1()
{

    var data = game.data;

    if (data.coins >= clickUpgrade1Cost)
    {

        clickUpgrade1Level++;
        data.coins -= clickUpgrade1Cost;
        data.coinsClickValue += clickUpgrade1Power;
    }
    NonArrayManager();
}

Počet odkazů: 2
public void BuyUpgradeClick2()
{

    var data = game.data;

    if (data.coins >= clickUpgrade2Cost)
    {

        clickUpgrade2Level++;
        data.coins -= clickUpgrade2Cost;
        data.coinsClickValue += clickUpgrade2Power;
    }
    NonArrayManager();

}
```

```
public void BuyUpgradeClick(int index)
{

    var data = game.data;

    if (data.coins >= clickUpgradeCost[index])
    {

        clickUpgradeLevels[index]++;
        data.coins -= clickUpgradeCost[index];
        data.coinsClickValue += clickUpgradePower[index];
    }

    NonArrayManager();


}
```

Obrázek 33: optimalizace za pomocí indexu (vlevo před, vpravo po optimalizaci)

Rozdíl mezi těmito kódy spočívá v organizaci a struktuře. V pravém kódu se používá jedna univerzální metoda BuyUpgradeClick(), která přijímá index upgradu jako parametr. Naopak v levém kódu jsou použity samostatné metody pro každý upgrade kliknutí. Tímto způsobem je možné mít jednu metodu pro všechny upgrady kliknutí nebo oddělené metody pro každý konkrétní upgrade.

Poté byla vytvořena funkce pro nákup produkce coinů za sekundu, která je podobná nákupu vylepšení kliknutí. Tato funkce je identická s funkcí pro nákup vylepšení kliknutí, s výjimkou použitých proměnných.

Následně bylo nutné přidat úrovně vylepšení kliknutí a produkce do třídy *PlayerData*. Tyto úrovně jsou stejné pro tlačítko na vylepšení kliknutí a produkci, s výjimkou použitých proměnných.

```
public class PlayerData
{
    public GameManager game;


    public BigDouble coins;
    public BigDouble coinsClickValue;

    public BigDouble clickUpgrade1Level;

    public BigDouble clickUpgrade2Level;


    public BigDouble productionUpgrade1Level;

    public BigDouble productionUpgrade2Power;
    public BigDouble productionUpgrade2Level;
```

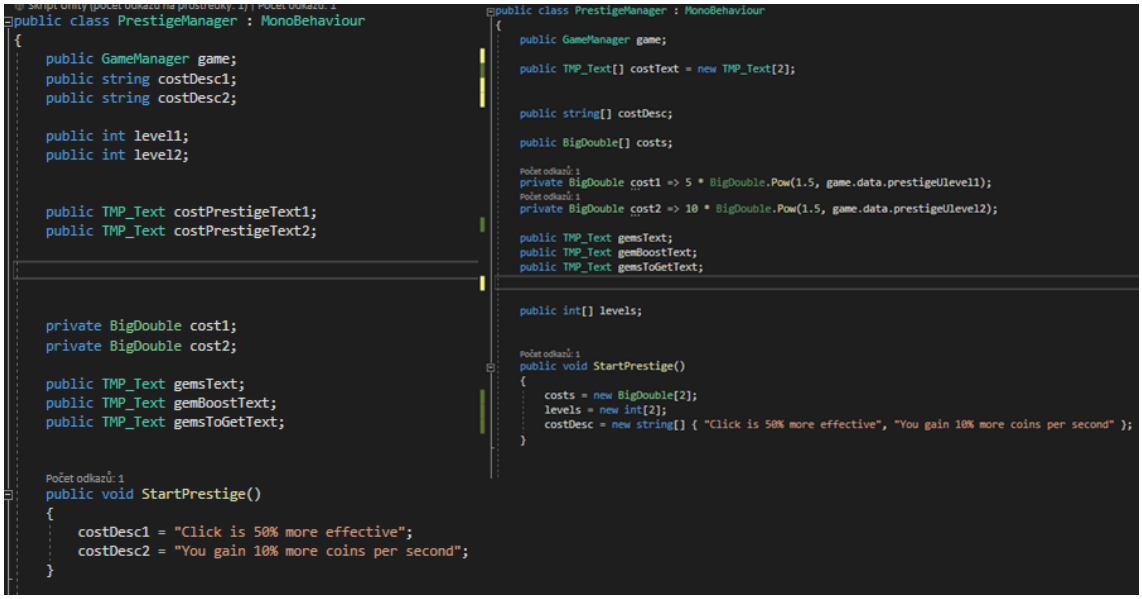
Obrázek 34: Class PlayerData s proměnnýma

V hierarchii projektu byl vytvořen nový prázdný objekt ve složce *Scripts* a byl přejmenován na *UpgradeManager*. K tomuto objektu byl přiřazen skript *UpgradeManager*. Následně byly přiřazeny textové proměnné v tomto skriptu k odpovídajícím textům ve hierarchii. Dále bylo nutné přiřadit funkce k tlačítkům. Bylo nutné vybrat tlačítka z *UpgradeScreen* ve hierarchii a v inspektoru jim byl přiřazen *UpgradeManager*. Poté pro vybraná tlačítka nutné vybrat buď upgrade kliknutí nebo upgrade produkce, stejně jako u předchozího *UpgradeManageru*.

7.3 Prestige

Tato část je zaměřena na upgrady, které ovlivňují zisk coinů z tlačítka a produkce za sekundu. Tato vylepšení jsou možná zakoupit za speciální měnu neboli gemy. Gemy lze získat za 1000 coinů za jeden gem. Po kliknutí na tlačítko *Prestige* dochází k získání coinů, avšak celá hra se resetuje, s výjimkou získaných gemů.

V hierarchii byla připravena scéna *PrestigeScreen* s tlačítky. Vytvořen byl prázdný objekt v hierarchii a přejmenován na *PrestigeManager*. V projektovém okně, kde již existují skripty jako *GameManager*, *UpgradeManager* a další, byl vytvořen skript *PrestigeManager* a přiřazen k objektu *PrestigeManager* v hierarchii. V tomto skriptu byly definovány proměnné pro cenu, level a texty upgradů *Prestige*. Nakonec byl ještě tento skript propojen se skriptem *GameManager*.



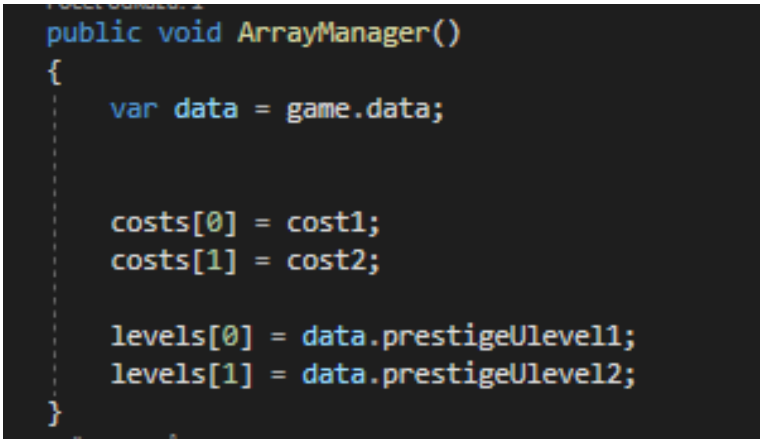
```
public class PrestigeManager : MonoBehaviour
{
    public GameManager game;
    public string costDesc1;
    public string costDesc2;

    public int level1;
    public int level2;


    public TMP_Text costPrestigeText1;
    public TMP_Text costPrestigeText2;



    private BigDouble cost1;
    private BigDouble cost2;

    public TMP_Text gemsText;
    public TMP_Text gemBoostText;
    public TMP_Text gemsToGetText;


    Počet odkazů: 1
    public void StartPrestige()
    {
        costDesc1 = "Click is 50% more effective";
        costDesc2 = "You gain 10% more coins per second";
    }
```

```
public class PrestigeManager : MonoBehaviour
{
    public GameManager game;

    public TMP_Text[] costText = new TMP_Text[2];


    public string[] costDesc;

    public BigDouble[] costs;

    Počet odkazů: 1
    private BigDouble cost1 => 5 * BigDouble.Pow(1.5, game.data.prestigeUlevel1);
    Počet odkazů: 1
    private BigDouble cost2 => 10 * BigDouble.Pow(1.5, game.data.prestigeUlevel2);

    public TMP_Text gemsText;
    public TMP_Text gemBoostText;
    public TMP_Text gemsToGetText;


    public int[] levels;


    Počet odkazů: 1
    public void StartPrestige()
    {
        costs = new BigDouble[2];
        levels = new int[2];
        costDesc = new string[] { "Click is 50% more effective", "You gain 10% more coins per second" };
    }
```

Obrázek 35: Levý kód neoptimalizovaný pravý optimalizovaný pomocí polí

Levý kód nepoužívá pole a jeho budoucí změny můžou být velmi složité. Zatím co pravý kód používá pole a díky tomu je možná jednoduchá škálovatelnost do budoucna ale je nutné použít identifikátor pro identifikování potřebné proměnné.

Poté byl v *PlayerData* přidán *prestige level*, kde se ukládají aktuální úrovně *prestige*. Poté byla v *PrestigeManageru* přiřazena cena a úroveň *prestige*.



```
public void ArrayManager()
{
    var data = game.data;


    costs[0] = cost1;
    costs[1] = cost2;

    levels[0] = data.prestigeUlevel1;
    levels[1] = data.prestigeUlevel2;
}
```

Obrázek 36: ArrayManager funkce

Funkce ArrayManager() má sloužit k přiřazení proměnných z PlayerData k proměnným, které budou provádět dané operace.

Následně bylo zapotřebí zajistit zobrazení textu obsahujícího cenu, level, počet gemů, a které hráč může získat. Zde byla znovu využita metoda NotationMethod() pro zobrazení objektů. Poté bylo nutné přidat funkci Run() do scriptu *GameManager* a funkci Update(), aby se textová pole průběžně aktualizovala.

```
public void Run()
{
    var data = game.data;
    cost1 = 5 * BigDouble.Pow(1.5, game.data.prestigeUlevel1);
    cost2 = 10 * BigDouble.Pow(1.5, game.data.prestigeUlevel2);

        costPrestigeText1.text = $"Level {game.data.prestigeUlevel1}\n  {costDesc1}\nCost: {cost1} Gems";
        costPrestigeText2.text = $"Level {game.data.prestigeUlevel2}\n  {costDesc2}\nCost: {cost2} Gems";


    data.gemsToGet = 100 * Sqrt(data.coins / 1e7);

    gemsText.text = "Games: " + Methods.NotationMethod(Floor(data.gems),"F2");


    if (game.mainMenuGroup.gameObject.activeSelf)
    {
        gemsToGetText.text = "Prestige:\n+" + Floor(data.gemsToGet).ToString("F0") + "Gems";
    }


}
```

```
public void Run()
{
    var data = game.data;
    ArrayManager();
    UI();
    data.gemsToGet = 100 * Sqrt(data.coins / 1e7);

    gemsText.text = "Games: " + Methods.NotationMethod(Floor(data.gems),"F2");
    gemsLandScapeText.text ="Games: " + Methods.NotationMethod(Floor(data.gems), "F2");



    if (game.mainMenuGroup.gameObject.activeSelf)
    {
        gemsToGetText.text = "Prestige:\n+" + Floor(data.gemsToGet).ToString("F0") + "Gems";
    }

    void UI()
    {

        for (var i = 0; i< costText.Length; i++)
        {
            costText[i].text = $"Level {levels[i]}\n  {costDesc[i]}\nCost: {costs[i]} Gems";
        }
    }
```

Obrázek 37: Vlevo neoptimalizovaný kód právo optimalizovaný kód

Hlavním rozdílem mezi těmito kódy je způsob aktualizace textových polí v metodě UI(). V pravý kódu je tato operace vykonávána prostřednictvím smyčky *for* v metodě UI(), která prochází pole *costText* a aktualizuje příslušná textová pole. V levém kódu jsou textová pole *costPrestigeText1* a *costPrestigeText2* aktualizována přímo v metodě Run().

Poté bylo nutné vytvořit funkce pro tlačítka ke kupování vylepšení a možnost získat gemy. Proto byly vytvořeny dvě funkce. První z nich je funkce EffectiveBuy(), která je volána, když hráč klikne na tlačítko pro nákup vylepšení *Effective*. Pokud hráč disponuje dostatečným počtem gemů, provede se odečtení ceny vylepšení od celkového počtu gemů, a zvýší se level vylepšení *effective*. Druhá funkce je CoinsPerSecondBuy(), která je volána, když hráč klikne na tlačítko pro nákup vylepšení coinspersecond. Opět zde platí, že pokud hráč má dostatek gemů, odečte se cena vylepšení od celkového počtu gemů a zvýší se level vylepšení coinspersecond.

```
public void EffectiveBuy()
{
    var datagame = game;
    if (datagame.data.gems < cost1) return;
    datagame.data.gems -= cost1;
    game.data.prestigeUlevel1++;
}
Počet odkazů: 0
public void CoinsPerSecondBuy()
{
    var datagame = game;
    if (datagame.data.gems < cost2) return;
    datagame.data.gems -= cost2;
    game.data.prestigeUlevel2++;
}
```

```
public void BuyUpgrade(int id)
{
    var data = game;

    switch(id)
    {
        case 0:
            Buy(ref data.data.prestigeUlevel1);
            break;
        case 1:
            Buy(ref data.data.prestigeUlevel2);
            break;
    }

    void Buy(ref int level)
    {
        if (data.data.gems < costs[id]) return;
            data.data.gems -= costs[id];
            level++;


    }
}
```

Obrázek 38: Vlevo neoptimalizovaný kód vpravo optimalizovaný kód

Pravý kód používá jednu metodu s přepínacím příkazem switch, který rozhoduje, které vylepšení se má provést na základě předaného id. levý kód používá dvě samostatné metody, přičemž každá metoda zajišťuje nákup jednoho konkrétního vylepšení. Pravý kód je také obecněji použitelný, protože umožňuje přidání dalších vylepšení bez nutnosti vytvářet nové metody.

V poslední části byla vytvořena funkce pro získání gemů a resetování hry. V této funkci jsou všechny hodnoty nastaveny na nulu a hlavní scéna je nastavena jako *Main*.

```
public void Prestige()
{

    var data = game.data;
    if (data.coins > 1000)
    {
        data.coins = 0;
        data.gems = 0;
        data.coinsClickValue = 1;
        data.clickUpgrade1Level = 0;
        data.clickUpgrade2Level = 0;
        data.productionUpgrade1Level = 0;
        data.productionUpgrade2Power = 5;
        data.productionUpgrade2Level = 0;

        game.ChangeTabs("Main");

        #region Prestige
        data.prestigeUlevel1 = 0;
        data.prestigeUlevel2 = 0;
        #endregion

        #region Autos
        data.autoLevel1 = 0;
        data.autoLevel2 = 0;
        data.autoLevel3 = 0;
        data.autoLevel4 = 0;
        #endregion
        data.gems += data.gemsToGet;

    }
}
```

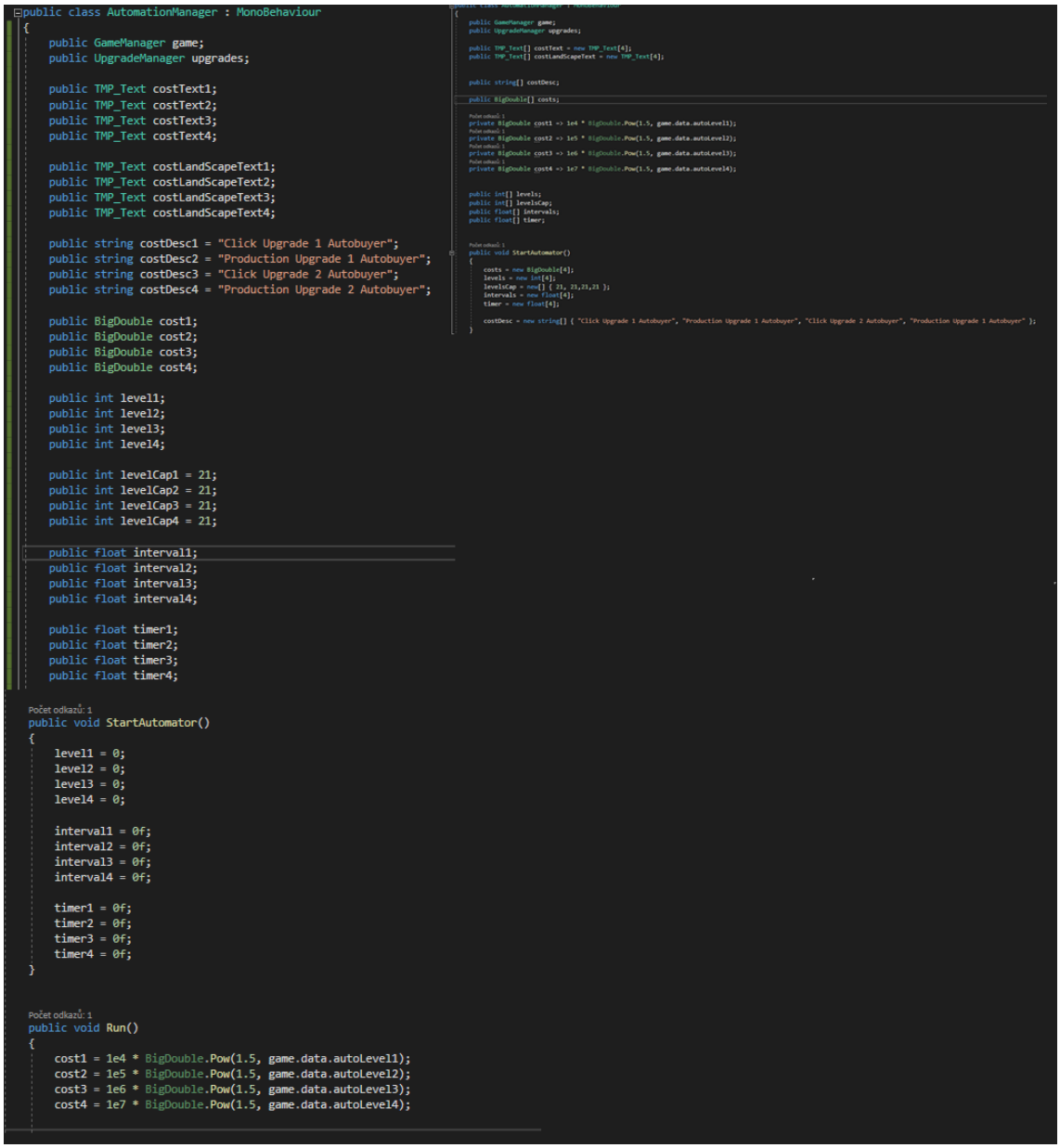
Obrázek 39: Pretige tlačítko pro získání gemů

Vše je uloženo a je nutné přejít do *PrestigeManageru* v *Hierarchii*, kde jsou přidány veškeré texty do *PrestigeManageru* v *Inspektoru*.

7.4 Automation

Zde bude probíhat automatické kupování vylepšení, aby hráč nemusel provádět klikání na vylepšení a mohl se plně soustředit na sbírání coinů.

Bylo nutné vytvořit další prázdný objekt v *Hierarchii* a přejmenovat ho na *AutomationManager*. V Projektovém okně jsem vytvořil skript *AutomationManager*. Zde byly opět definovány proměnné pro text, level, maximální level, timer a interval.



```
public class AutomationManager : MonoBehaviour
{
    public GameManager game;
    public UpgradeManager upgrades;

    public TMP_Text costText1;
    public TMP_Text costText2;
    public TMP_Text costText3;
    public TMP_Text costText4;

    public TMP_Text costLandScapeText1;
    public TMP_Text costLandScapeText2;
    public TMP_Text costLandScapeText3;
    public TMP_Text costLandScapeText4;

    public string costDesc1 = "Click Upgrade 1 Autobuyer";
    public string costDesc2 = "Production Upgrade 1 Autobuyer";
    public string costDesc3 = "Click Upgrade 2 Autobuyer";
    public string costDesc4 = "Production Upgrade 2 Autobuyer";

    public BigDouble cost1;
    public BigDouble cost2;
    public BigDouble cost3;
    public BigDouble cost4;

    public int level1;
    public int level2;
    public int level3;
    public int level4;

    public int levelCap1 = 21;
    public int levelCap2 = 21;
    public int levelCap3 = 21;
    public int levelCap4 = 21;

    public float interval1;
    public float interval2;
    public float interval3;
    public float interval4;

    public float timer1;
    public float timer2;
    public float timer3;
    public float timer4;

Počet odkazů: 1
public void StartAutomator()
{
    level1 = 0;
    level2 = 0;
    level3 = 0;
    level4 = 0;

    interval1 = 0f;
    interval2 = 0f;
    interval3 = 0f;
    interval4 = 0f;

    timer1 = 0f;
    timer2 = 0f;
    timer3 = 0f;
    timer4 = 0f;
}

Počet odkazů: 1
public void Run()
{
    cost1 = 1e4 * BigDouble.Pow(1.5, game.data.autoLevel1);
    cost2 = 1e5 * BigDouble.Pow(1.5, game.data.autoLevel2);
    cost3 = 1e6 * BigDouble.Pow(1.5, game.data.autoLevel3);
    cost4 = 1e7 * BigDouble.Pow(1.5, game.data.autoLevel4);
```

Obrázek 40: Levý kód neoptimalizovaná pravý optimalizovaný pomocí polí

V levém kódu jsou jednotlivé hodnoty pro každou úroveň nebo položku deklarovány jako samostatné proměnné. Jako je *costText1, costText2, costText3* a *costText4* jsou textová pole, *costDesc1, costDesc2, costDesc3, costDesc4* a další. Každá z těchto proměnných

je deklarována zvlášť. V prvém kódu jsou hodnoty pro jednotlivé úrovně nebo položky uloženy v polích. Například *costText* je pole obsahující textová pole, *costDesc* je pole obsahující textové popisy, costs je pole obsahující náklady, levels je pole obsahující úrovně a tak dále. Každý prvek pole odpovídá určité úrovni nebo položce. K přístupu k jednotlivým hodnotám se používá indexování pole.

Poté byl vytvořen *ArrayManager*. Nejprve byla vytvořena proměnná *data*, která přijímá hodnotu *game*. Následně byly přiřazeny hodnoty do pole *costs*. Konkrétně hodnoty *cost1, cost2, cost3* a *cost4* byly přiřazeny do odpovídajících indexů v poli. Poté byly přiřazeny hodnoty do pole *levels*. Hodnoty *autoLevel1, autoLevel2, autoLevel3* a *autoLevel4* z proměnné *data.data* byly přiřazeny do odpovídajících indexů v poli. Následují podmínky, které upravují hodnoty v poli *intervals* na základě úrovní v *data.data*. Pokud je hodnota některé z úrovní větší než 0, pak je vypočítána hodnota pro příslušný index v poli *intervals* na základě vzorce. Tato hodnota se odvíjí od úrovně a postupně se snižuje s každým navýšením úrovně.

```
Počet odkazů: 1
public void ArrayManager()
{
    var data = game;


    costs[0] = cost1;
    costs[1] = cost2;
    costs[2] = cost3;
    costs[3] = cost4;


    levels[0] = data.data.autoLevel1;
    levels[1] = data.data.autoLevel2;
    levels[2] = data.data.autoLevel3;
    levels[3] = data.data.autoLevel4;

    if (data.data.autoLevel1 > 0)
        intervals[0] = 10 - ((data.data.autoLevel1 - 1) * 0.5f);
    if (data.data.autoLevel2 > 0)
        intervals[1] = 10 - ((data.data.autoLevel2 - 1) * 0.5f);
    if (data.data.autoLevel3 > 0)
        intervals[2] = 10 - ((data.data.autoLevel3 - 1) * 0.5f);
    if (data.data.autoLevel4 > 0)
        intervals[3] = 10 - ((data.data.autoLevel4 - 1) * 0.5f);


}
```

Obrázek 41: ArrayManager pro automatizaci

Následně byl proveden samotný proces automatického klikání na tlačítka pro vylepšení hlavního tlačítka. Na začátku se kontroluje, zda je nějaké vylepšení zakoupené. Pokud ne, proces se ihned ukončí. V opačném případě se pokračuje dále. Poté se kontroluje, zda je dosažen maximální zakoupený level vylepšení. Pokud ne, pokračuje se v kupování až do vyčerpání coinů. Pokud jsou coiny vyčerpány, časový interval je uložen do timeru. Dále se kontroluje, zda uběhla potřebná doba od posledního intervalu. Pokud ano, provede se zakoupení

vylepšení a *timer* se nastaví na nulu. Nakonec jsou tyto funkce volány v RunAuto(), která je volána v Update() v *GameManageru*.

```
void RunAuto()
{
    AutoClick();
    AutoClick2();
    AutoPerSec1();
    AutoPerSec2();
    void AutoClick()
    {
        if (level1 <= 0) return;
        if (level1 != levelCap1)
        {
            timer1 += Time.deltaTime;
            if (!(timer1 >= interval1)) return;
            upgrades.BuyUpgradeClick1();
            timer1 = 0;
        }
        else
        {
            if (game.data.coins > upgrades.clickUpgrade1Cost)

                upgrades.BuyUpgradeClick1();
        }
    }
    void AutoPerSec1()
    {
        if (level2 <= 0) return;
        if (level2 != levelCap2)
        {
            timer2 += Time.deltaTime;
            if (!(timer2 >= interval2)) return;
            upgrades.BuyUpgradeProduction1();
            timer2 = 0;
        }
        else
        {
            if (game.data.coins > upgrades.productionUpgrade1Cost)

                upgrades.BuyUpgradeProduction1();
        }
    }
    void AutoClick2()
    {
        if (level3 <= 0) return;
        if (level3 != levelCap3)
        {
            timer3 += Time.deltaTime;
            if (!(timer3 >= interval3)) return;
            upgrades.BuyUpgradeClick2();
            timer2 = 0;
        }
        else
        {
            if (game.data.coins > upgrades.clickUpgrade2Cost)

                upgrades.BuyUpgradeClick2();
        }
    }
    void AutoPerSec2()
    {
```

```
void RunAuto()
{
    AutoC(0,0);
    AutoC(2,1);
    AutoP(1, 0);
    AutoP(3, 1);
    void AutoC(int id, int index)
    {
        if (levels[id] <= 0) return;
        if (levels[id] != levelsCap[id])
        {
            timer[id] += Time.deltaTime;
            if (!(timer[id] >= intervals[id])) return;
            upgrades.BuyUpgradeClick(index);
            timer[id] = 0;
        }
        else
        {
            if (game.data.coins > upgrades.clickUpgradeCost[index])

                upgrades.BuyUpgradeClick(index);
        }

    }

    void AutoP(int id, int index)
    {
        if (levels[id] > 0)
        {
            if (levels[id] != levelsCap[id])
            {
                timer[id] += Time.deltaTime;
                if (timer[id] >= intervals[id])
                {
                    upgrades.BuyUpgradeProduction(index);
                    timer[id] = 0;
                }
            }
            else
            {
                if (game.data.coins > upgrades.productionUpgradeCost[index])

                    upgrades.BuyUpgradeProduction(index);
            }
```

Obrázek 42: Proces automatického klikání, optimalizovaný (vlevo) a neoptimalizovaný (vpravo) kód

Rozdíl spočívá v parametrech funkcí. První kód přijímá parametry *id* a *index* pro identifikaci konkrétních upgradů, zatímco druhý kód používá specifické proměnné *level1, level2* a další jednotlivé proměnné pro identifikaci upgradů.

Poté byla provedena aktualizace textu po nákupu vylepšení, kde se zvyšuje cena a snižuje délka intervalu pro nákup vylepšení. Byla vytvořena funkce UI(), ve které je použita proměnná *costText*, do které je přiřazována cena vylepšení a interval pro další automatický nákup vylepšení.

```
public void Run()
{

    ArrayManager();
    UI();
    RunAuto();


    void UI()
    {

                costText1.text = $"{costDesc1}\nCost: {Methods.NotationMethod(cost1, "F2")} " +
            $"Coins\nInterval:{(level1 >= levelCap1 ? "Instant" : interval1.ToString("F1"))}";
                costText2.text = $"{costDesc2}\nCost: {Methods.NotationMethod(cost2, "F2")} " +
            $"Coins\nInterval:{(level2 >= levelCap2 ? "Instant" : interval2.ToString("F1"))}";
                costText3.text = $"{costDesc3}\nCost: {Methods.NotationMethod(cost3, "F2")} " +
            $"Coins\nInterval:{(level3 >= levelCap3 ? "Instant" : interval3.ToString("F1"))}";
                costText4.text = $"{costDesc4}\nCost: {Methods.NotationMethod(cost4, "F2")} " +
            $"Coins\nInterval:{(level4 >= levelCap4 ? "Instant" : interval4.ToString("F1"))}";



    }
public void Run()
{
    ArrayManager();
    UI();
    RunAuto();


    void UI()
    {

            for (var i = 0; i < costText.Length; i++)
            {
                costText[i].text = $"{costDesc[i]}\nCost: {Methods.NotationMethod(costs[i], "F2")} " +
                $"Coins\nInterval:{(levels[i] >= levelsCap[i] ? "Instant" : intervals[i].ToString("F1"))}";
            }



    }
```

Obrázek 43: první kód neoptimalizovaný druhý optimalizovaný

Rozdíl je že druhý kód využívá pole a cyklus *for* k aktualizaci textových polí najednou, zatímco první kód aktualizuje každé textové pole samostatně pomocí přiřazení hodnoty.

V poslední řadě byly provedeny samotné funkce pro tlačítka na nákup automatického kupování vylepšení. Bylo zkontrolováno, zda hráč má dostatek coinů na zakoupení vylepšení a zda je level menší než maximální level. Pokud jsou tyto podmínky splněny, provede se zakoupení vylepšení. To znamená, že se odečte cena potřebná k nákupu a zvýší se level automatizace.

```
public void BuyClickAutomation1()
{
    if (!(game.data.coins >= cost1 && level1 < levelCap1)) return;
    {
        game.data.coins -= cost1;
        game.data.autoLevel1++;
    }
}
Počet odkazů: 0
public void BuyClickAutomation2()
{
    if (!(game.data.coins >= cost2 && level2 < levelCap2)) return;
    {
        game.data.coins -= cost2;
        game.data.autoLevel2++;
    }
}
Počet odkazů: 0
public void BuyCoinPerSecAutomation1()
{
    if (!(game.data.coins >= cost3 && level3 < levelCap3)) return;
    {
        game.data.coins -= cost3;
        game.data.autoLevel3++;
    }
}
Počet odkazů: 0
public void BuyCoinPerSecAutomation2()
{
    if (!(game.data.coins >= cost4 && level4 < levelCap4)) return;
    {
        game.data.coins -= cost4;
        game.data.autoLevel4++;
    }
}
```

```
public void BuyUpgrade(int id)
{
    var data = game;

    switch (id)
    {
        case 0:
            Buy(ref data.data.autoLevel1);
            break;
        case 1:
            Buy(ref data.data.autoLevel2);
            break;
        case 2:
            Buy(ref data.data.autoLevel3);
            break;
        case 3:
            Buy(ref data.data.autoLevel4);
            break;
    }

    void Buy(ref int level)
    {
        if (!(data.data.coins >= costs[id] && level < levelsCap[id])) return;
        {
            data.data.coins -= costs[id];
            level++;
        }
    }
```

Obrázek 44: Neoptimalizovaný (vlevo) a optimalizovaný (vpravo) kód

Hlavním rozdílem mezi těmito kódy je způsob, jakým se řídí nákup upgradů. Pravý kód používá jednu univerzální metodu BuyUpgrade(), která přijímá id upgradu jako parametr a používá switch statement pro rozhodnutí, který upgrade se má koupit. Levý kód používá oddělené metody pro každý konkrétní upgrade.

Nyní bylo vše uloženo a skript byl přiřazen k objektu v Hierarchii. Poté byly v Inspektoru přiřazeny potřebné textové proměnné. Následně byla vybrána tlačítka a přiřazeny jim funkce pro zakoupení vylepšení z objektu *AutomationManager* v Hierarchii.

7.5 Schopnost orientace hry

Tato funkce byla přidána do mobilní hry z důvodu, že hráči mají různé preference ohledně toho, jak si přejí držet svá zařízení při hraní her. Někteří hráči preferují vertikální orientaci, zatímco jiní dávají přednost horizontální orientaci. Bylo důležité poskytnout hráčům flexibilitu a umožnit jim hrát hru ve způsobu, který je pro ně nejpříjemnější a nejpohodlnější.

Toho bylo dosaženo tím, že veškeré texty, které byly ve hře, bylo nutné přidat i pro šířkovou orientaci, známou také jako *LandScape*. Proto byly vytvořeny nové textové proměnné s příponou *LandScape*, například *coinLandScape*. Muselo být vyřešeno, kdy se hra přizpůsobí šířce a jak hra rozpozná, že hráč ji otočil. Toho bylo dosaženo přidáním jednoduché podmínky. Poté bylo uváženo, že hráč může hru otočit úmyslně nebo omylem. Hráč může otočit hru v *UpgradeScreenu* na šířku, a poté by se zobrazil zpět na *MainScreenu*. Proto bylo nutné

přidat v Hierarchii nové screeny, a to byly *LandScapeScreeny*. Toho bylo dosaženo tím, že všechny předchozí screeny byly zkopírovány. Následně bylo změněno rozlišení v *Scene window* na šířkové, aby se simulovalo otočení, a všechny nové screeny byly přizpůsobeny tomuto otočení. V posledním kroku bylo nutné přiřadit screeny k scriptu a otestovat, zda tyto screeny fungují správně včetně textu a tlačítek.

```
public void Update()
{

    if(Input.deviceOrientation == DeviceOrientation.LandscapeLeft)
    {
        if(mainMenuGroup.gameObject.activeSelf)
        {
            mainMenuGroupLandScape.gameObject.SetActive(true);
            mainMenuGroup.gameObject.SetActive(false);
        }
        if (upgradesGroup.gameObject.activeSelf)
        {
            upgradesGroupLandScape.gameObject.SetActive(true);
            upgradesGroup.gameObject.SetActive(false);
        }
        if (AutomationGroup.gameObject.activeSelf)
        {
           AutomationGroupLandScape.gameObject.SetActive(true);
            AutomationGroup.gameObject.SetActive(false);
        }

        if (headerGroup.gameObject.activeSelf)
        {
            headerGroupLandScape.gameObject.SetActive(true);
            headerGroup.gameObject.SetActive(false);
        }
        if (presigeGroup.gameObject.activeSelf)
        {
            presigeGroup.gameObject.SetActive(false);
            presigeGroupLandScape.gameObject.SetActive(true);
        }
    }

    if (Input.deviceOrientation == DeviceOrientation.Portrait)
    {
        if (mainMenuGroupLandScape.gameObject.activeSelf)
        {
            mainMenuGroupLandScape.gameObject.SetActive(false);
            mainMenuGroup.gameObject.SetActive(true);
        }
        if (upgradesGroupLandScape.gameObject.activeSelf)
        {
            upgradesGroupLandScape.gameObject.SetActive(false);
            upgradesGroup.gameObject.SetActive(true);
        }
        if (AutomationGroupLandScape.gameObject.activeSelf)
        {
            AutomationGroupLandScape.gameObject.SetActive(false);
            AutomationGroup.gameObject.SetActive(true);
        }

        if (headerGroupLandScape.gameObject.activeSelf)
        {
            headerGroupLandScape.gameObject.SetActive(false);
            headerGroup.gameObject.SetActive(true);
        }
        if (presigeGroupLandScape.gameObject.activeSelf)
        {
            presigeGroupLandScape.gameObject.SetActive(false);
            presigeGroup.gameObject.SetActive(true);
        }
    }
```

Obrázek 45: Aktualizace rozhraní

Tento kód provádí aktualizace rozhraní v závislosti na orientaci zařízení. Pokud je zařízení v režimu *Landscape*, provedou se určité akce jako je zapnutí *LandScape screnů*, a pokud je zařízení v režimu *Portrait*, provede se zapnutí *Portrait screeny*.

7.6 Save Systém

Na konec byl přidán systém pro ukládání a načítání dat, aby při opětovném zapnutí hry byly načteny uložené hodnoty. To bylo provedeno vytvořením dvou funkcí, Load() a Save(). V funkci Save() byly použity metody PlayerPrefs.SetString() nebo PlayerPrefs.SetInt(). Tyto metody jsou součástí Unity API a slouží k uložení řetězcových nebo celočíselných hodnot do herního úložiště. Mají dva parametry: klíč a hodnota. Klíč slouží k jednoznačnému identifikování uložené hodnoty a hodnota je řetězec nebo číslo, které je potřeba uložit.

V funkci Load() byly použity metody PlayerPrefs.GetString() a PlayerPrefs.GetInt(). Tyto metody také patří do Unity API a slouží k načítání řetězcových nebo celočíselných hodnot z herního úložiště. Mají dva parametry: klíč a výchozí hodnotu. Klíč slouží k identifikaci uložené hodnoty a výchozí hodnota je hodnota, která bude vrácena, pokud pod daným klíčem nebyla nalezena žádná hodnota.

```
public void Load()
{
    data.coins = Parse(PlayerPrefs.GetString("coins", "0"));
    data.gems = Parse(PlayerPrefs.GetString("gems", "0"));
    data.coinsClickValue = Parse(PlayerPrefs.GetString("coinsClickValue", "1"));
    data.clickUpgrade1Level = Parse(PlayerPrefs.GetString("clickUpgrade1Level", "0"));
    data.clickUpgrade2Level = Parse(PlayerPrefs.GetString("clickUpgrade2Level", "0"));
    data.productionUpgrade1Level = Parse(PlayerPrefs.GetString("productionUpgrade1Level", "0"));
    data.productionUpgrade2Power = Parse(PlayerPrefs.GetString("productionUpgrade2Power", "5"));
    data.productionUpgrade2Level = Parse(PlayerPrefs.GetString("productionUpgrade2Level", "0"));
    data.coinsCollected = Parse(PlayerPrefs.GetString("coinsCollected", "0"));
    data.prestigeUlevel1 = PlayerPrefs.GetInt("prestigeUlevel1", 0);
    data.prestigeUlevel2 = PlayerPrefs.GetInt("prestigeUlevel2", 0);
    data.autoLevel1 = PlayerPrefs.GetInt("autoLevel1", 0);
    data.autoLevel2 = PlayerPrefs.GetInt("autoLevel2", 0);
    data.autoLevel2 = PlayerPrefs.GetInt("autoLevel3", 0);
    data.autoLevel2 = PlayerPrefs.GetInt("autoLevel4", 0);

}
Počet odkazů: 1
public void Save()
{
    PlayerPrefs.SetString("coins", data.coins.ToString());
    PlayerPrefs.SetString("gems", data.gems.ToString());
    PlayerPrefs.SetString("coinsClickValue", data.coinsClickValue.ToString());
    PlayerPrefs.SetString("clickUpgrade1Level", data.clickUpgrade1Level.ToString());
    PlayerPrefs.SetString("clickUpgrade2Level", data.clickUpgrade2Level.ToString());
    PlayerPrefs.SetString("productionUpgrade1Level", data.productionUpgrade1Level.ToString());
    PlayerPrefs.SetString("productionUpgrade2Power", data.productionUpgrade2Power.ToString());
    PlayerPrefs.SetString("productionUpgrade2Level", data.productionUpgrade2Level.ToString());
    PlayerPrefs.SetString("coinsCollected", data.coinsCollected.ToString());
    PlayerPrefs.SetInt("prestigeUlevel1", data.prestigeUlevel1);
    PlayerPrefs.SetInt("prestigeUlevel2", data.prestigeUlevel2);
    PlayerPrefs.SetInt("autoLevel1", data.autoLevel1);
    PlayerPrefs.SetInt("autoLevel2", data.autoLevel2);
    PlayerPrefs.SetInt("autoLevel3", data.autoLevel3);
    PlayerPrefs.SetInt("autoLevel4", data.autoLevel4);
```

Obrázek 46: ukládání a načítání dat

Nyní stačí jen funkci Load() přidat do Start() v GameManageru a Save() do stejného scriptu ale do metody Update().

8 SIMULOVANÉ ZATÍŽENÍ

Průběhu vývoje hry byly implementovány různé prvky, které mohou významně ovlivnit výkon a odezvu hry na mobilních zařízeních. Mezi tyto prvky patří například Particle systém, který umožňuje vytváření a animaci efektů jako jsou ohně, exploze či kouř. Přestože Particle systém přináší vizuální efekty, je však známo, že jeho nesprávné použití a nadměrné množství částic může zatížit grafický procesor a snížit výkon hry.

Dalším prvkem, který byl použit, je velké množství objektů ve hře. Přidání většího počtu objektů může mít výrazný dopad na výkon hry. Správné zpracování a vykreslování těchto objektů je klíčové pro dosažení plynulého chodu hry.

Dále bylo upřednostňováno použití velkého LOD pro některé objekty ve hře. LOD umožňuje dynamicky upravovat podrobnost objektů. To umožňuje snížit počet vykreslovaných polygonů a tím optimalizovat výkon hry. Avšak nevhodně nastavené LOD úrovně nebo nedostatečná správa LOD systému mohou mít negativní dopad na výkon hry.

Kromě implementace výše zmíněných prvků byla provedena analýza výkonu hry na desktopovém a mobilním zařízení a byly porovnány výsledky. Cílem bylo zjistit, jak se hra chová a jaké jsou rozdíly ve výkonu mezi těmito dvěma platformami.

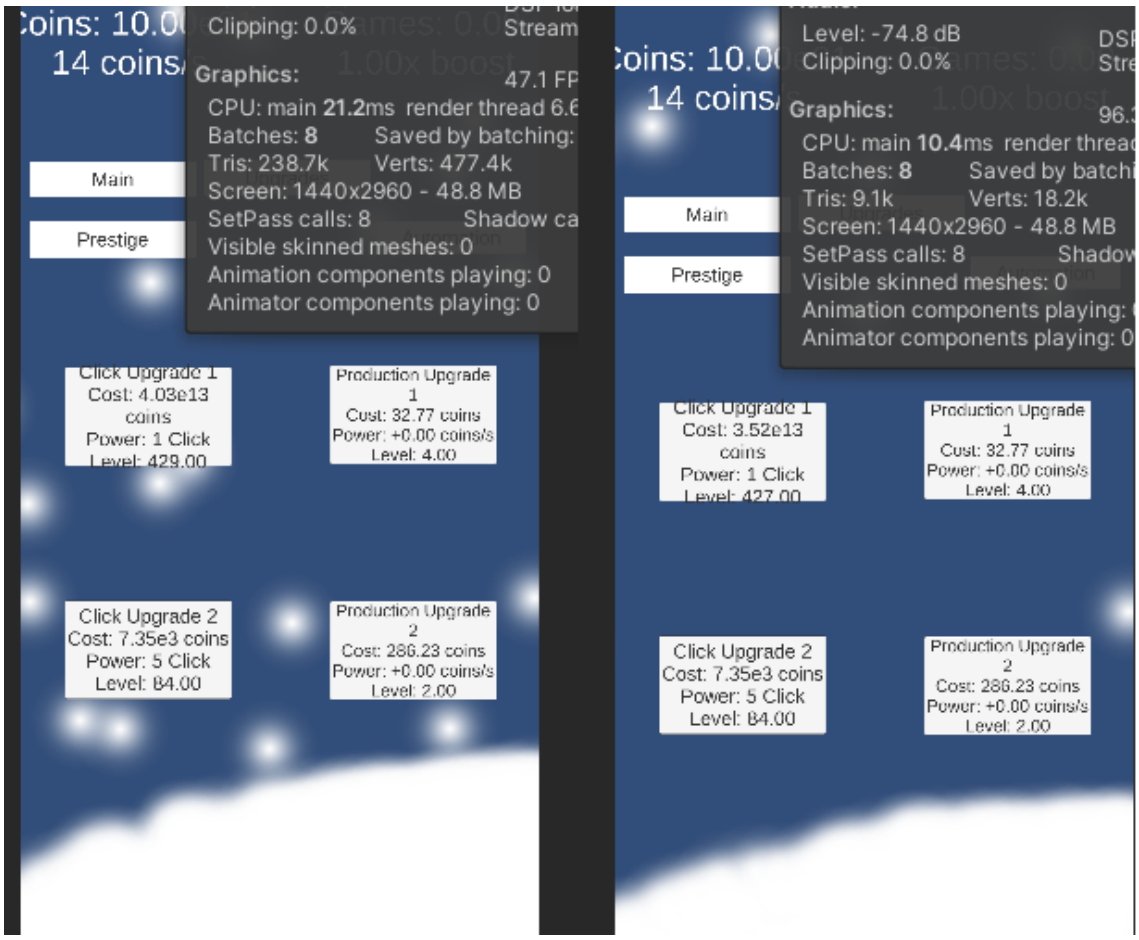
V neposlední řadě byla provedena analýza neoptimalizovaného kódu ve hře. Během tohoto procesu byly prozkoumány různé části kódu a hledány oblasti, které mohou zpomalovat hru a ovlivňovat její výkon. Neoptimalizovaný kód může zahrnovat neefektivní algoritmy, opakované výpočty, nevhodné použití paměti nebo nevhodné způsoby komunikace mezi herními prvky.

Editor loop je časový cyklus, který se spouští, když je hra spuštěna ve "Scéně" v Unity Editoru. Tento cyklus zahrnuje veškerý kód, který je spuštěn v Editoru, například aktualizace scény, manipulace s objekty, kontrola vstupů a další editorové funkce. Během Editor loopu se hra aktualizuje a vykresluje ve scéně Editoru, což umožňuje vývojářům vizuálně sledovat a upravovat hru přímo v Editoru.

Player loop je časový cyklus, který řídí chování hry během jejího běhu. Player loop obsahuje základní fáze, jako je vstup, aktualizace skriptů, fyzika, vykreslování a další. Během Player loopu se provádí všechny herní funkce, které jsou důležité pro běh hry, včetně aktualizace polohy a stavu objektů, ovládání uživatelského vstupu, detekce kolizí, vykreslování grafiky atd.

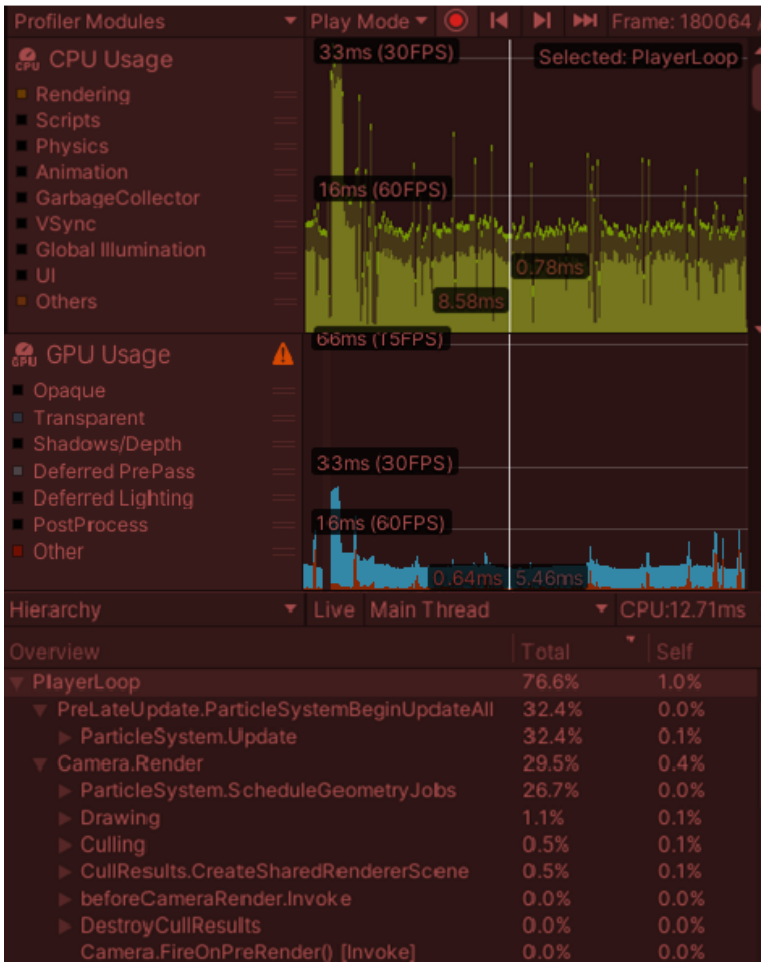
8.1 Particle Systém

Particle systém je důležitou součástí vizuálních efektů ve hrách a může přispět k atmosféře a realismu herního prostředí. Nicméně, neoptimalizované použití Particle systému může mít negativní dopad na výkon hry, zejména pokud je použit příliš vysoký počet objektů.



Obrázek 47: Levý neoptimalizovaný pravý optimalizovaný Particle systém

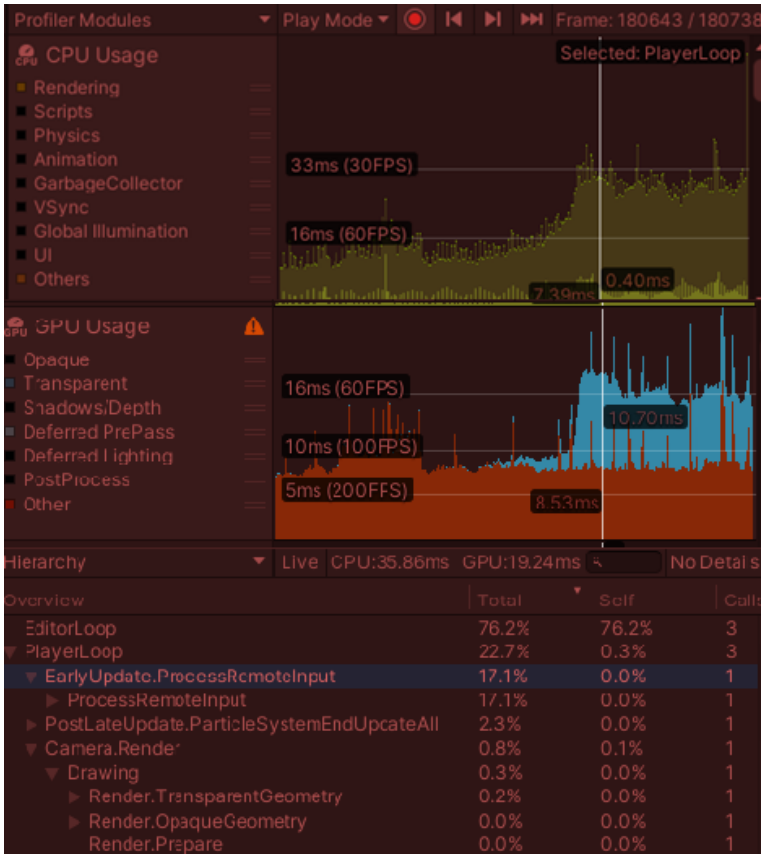
Na obrázku 47 lze vidět použití Particle systému. Kde levý je neoptimalizovaný Particle systém a obsahuje okolo 238 tisíc tris. Zatím co pravý optimalizovaný obsahuje pouze okolo 9 tisíc. Přičemž výsledem je z velké části totožný. Proto je dobré dbát na to kde a jak moc dané snížení provedeme.



Obrázek 48: Neoptimalizovaný particle systém testovaného na desktopovém zařízením

Obrázek 48 zobrazuje vytížení CPU a GPU u neoptimalizovaného Particle systému testovaného na desktopovém zařízením. Z obrázku 48 dále lze vyčíst že CPU a GPU zatížení je pod 16ms a 60 FPS. To znamená že hra se dostatečně rychle aktualizuje a hráč nezaznamená žádné zpomalení nebo zasekávání.

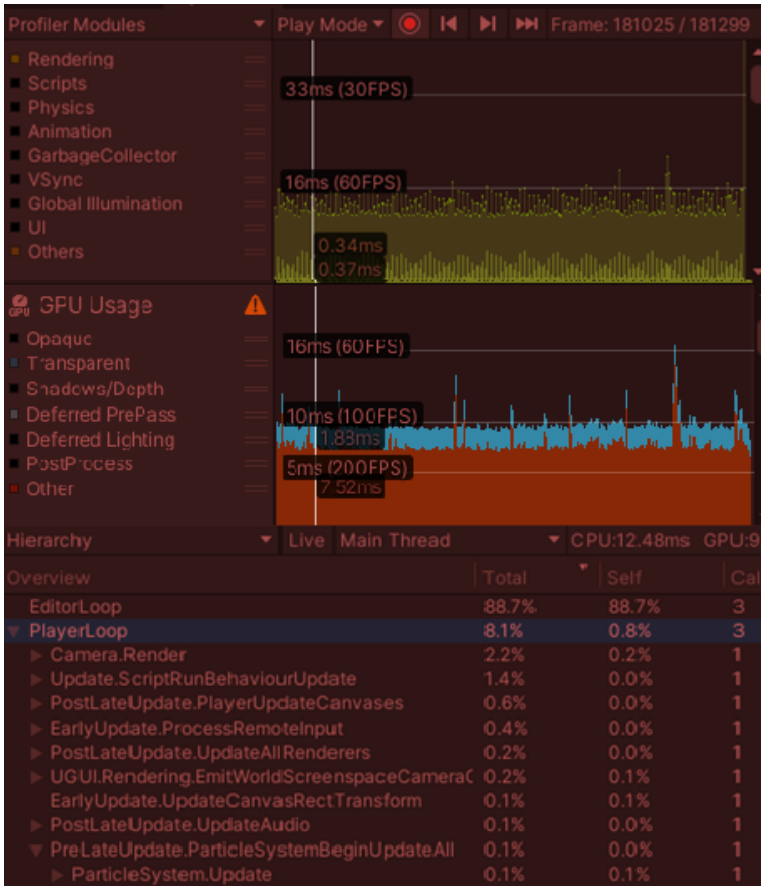
PlayerLoop je uvedeno s hodnotou 76,6 %. To znamená, že PlayerLoop část hry je hlavním přispěvatelem k celkovému času strávenému v CPU. Z těchto 76,6 % je 32,4 % přiděleno kódům ParticleSystémUpdate, ten je odpovědný za aktualizaci polohy, rychlosti, rotace, velikosti a dalších vlastností jednotlivých částic v systému a ParticleSystém.ScheduleGeometryJobs 26,7 % slouží k vizuálního vykreslování částic a zahrnuje informace o pozicích, normálách, indexech částic a dalších geometrických údajích.



Obrázek 49: Neoptimalizovaný particle systém testováno na mobilním zařízením

Obrázek 49 zobrazuje vytížení CPU a GPU u neoptimalizovaného Particle systému testovaného na mobilním zařízením a lze vyčíst že CPU zatížení je pod 33ms a 30 FPS. To znamená že hra se nedostatečně rychle aktualizuje a hráč může u určitý her zaznamenávat zpomalení nebo zasekávání. Dále jde vyčíst, že GPU je občas přes hodnotu 16ms(60FPS), to může způsobovat občasné trhání obrazu.

PlayerLoop je uvedeno s hodnotou 22,7 %. To znamená, že PlayerLoop část hry je hlavním přispěvatelem k celkovému času strávenému na CPU. Z těchto 22,7 % je 17,1 % přiděleno kódu ProcesingRemoteInput. To naznačuje, že část zdrojů CPU je využívána pro zpracování vstupů ze vzdálených zařízení nebo síťových komunikací.

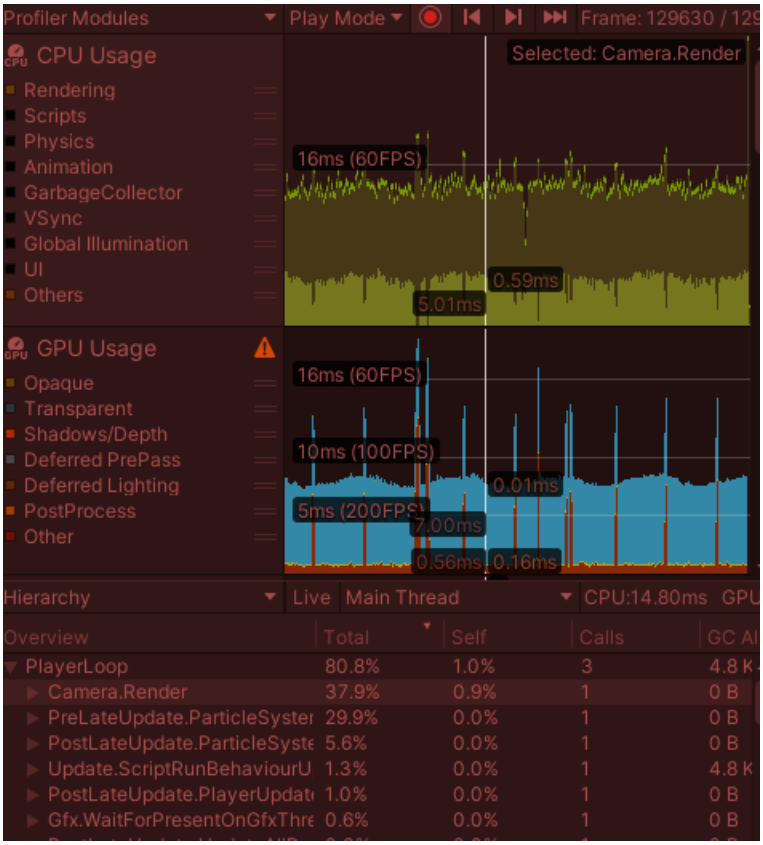


Obrázek 50: Optimalizovaný particle systém testováno na mobilním zařízením

Na obrázku 50 jsou vykresleny vytížení CPU a GPU s grafy u optimalizovaného Particle systému testovaného na mobilním zařízením. Lze vyčíst že CPU zatížení je pod 16ms a 60 FPS. To znamená že hra se dostatečně rychle aktualizuje a hráč nezaznamená žádné zpomalení nebo zasekávání. GPU je dokonce na tom ještě líp a je okolo 10ms(100FPS) při takové hodnotě nehrozí žádné trhání obrazu.

PlayerLoop je uvedeno s hodnotou 8,1 %. Z těchto 8,1 % je 2,2 % přiděleno kódu Camera.Render. To naznačuje, že část CPU výkonu je využívána pro vykreslování kamery, což zahrnuje proces renderování scény do obrazu z kamery.

EditorLoop je uvedeno s hodnotou 88,7 % to ukazuje, že většina času na CPU je využívána pro provádění editorových funkcí, manipulaci s objekty ve scéně a další související operace.



Obrázek 51: Optimalizovaný particle systém testováno na desktopovém zařízením

Na obrázku 51 jsou vykresleny vytížení CPU a GPU s grafy u optimalizovaného Particle systému testovaného na desktopovém zařízením. Dále lze z obrázku 51 vyčíst ze CPU zatížení je pod 16ms(60 FPS). To znamená že hra se dostatečně rychle aktualizuje a hráč neznamená žádné zpomalení nebo zasekávání. GPU je dokonce na tom ještě líp a je většinu času okolo 10ms(100FPS), i když jde vidět občasný nárust, ale naštěstí tento nárust nepřekročí hranici 16ms(60FPS).

PlayerLoop je uvedeno s hodnotou 80,8 %. Z toho je 37,9 % přiděleno kódu Camera.Render. Ten zahrnuje proces renderování scény z pohledu kamery, který se pak zobrazuje hráči. Další část je přidělena PreLateUpdateParticleSystem s 29,9 % přiděleno kódům ParticleSystémUpdate, ten je odpovědný za aktualizaci polohy, rychlosti, rotace, velikosti.

Tabulka 1: Srovnání optimalizované a neoptimalizovaného Particle systém

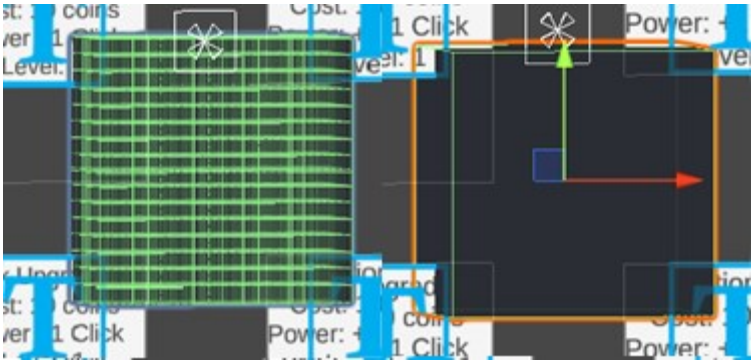
	CPU ms(FPS)	GPU ms(FPS)	Player-Loop %	Editor-Loop %
Neoptimalizovaný Particle systém – testováno na desktopovém zařízením	16(60)	16(60)	76,6	12,4
Optimalizovaný Particle systém – testováno na desktopovém zařízením	16(60)	10(100)	80,8	18,3
Neoptimalizovaný Particle systém – testováno na mobilním zařízením	33(30)	16(60)	22,7	76,2
Optimalizovaný Particle systém – testováno na mobilním zařízením	16 (60)	10(100)	8,1	88,7

V tabulce 1 jsou uvedeny hodnoty výkonu pro optimalizovaný a neoptimalizovaný Particle systém. Hodnoty CPU ms (FPS) udávají časovou odezvu systému v milisekundách a rychlost snímkování ve formátu FPS (snímky za sekundu) pro CPU. Hodnoty GPU ms (FPS) udávají časovou odezvu systému v milisekundách a rychlost snímkování ve formátu FPS pro GPU. PlayerLoop % a EditorLoop % vyjadřují procentuální zatížení systému pro daný Particle systém v Player Loop a Editor Loop.

Z výsledků lze vyčíst, že optimalizovaný Particle systém dosahuje nižšího zatížení CPU i GPU ve srovnání s neoptimalizovaným systémem. Na testovacím mobilním zařízení je optimalizovaný systém schopen udržet hru s plynulým snímkováním (60 FPS), zatímco neoptimalizovaný systém se zpomaluje na 30 FPS.

8.2 Velké množství 3D objektů

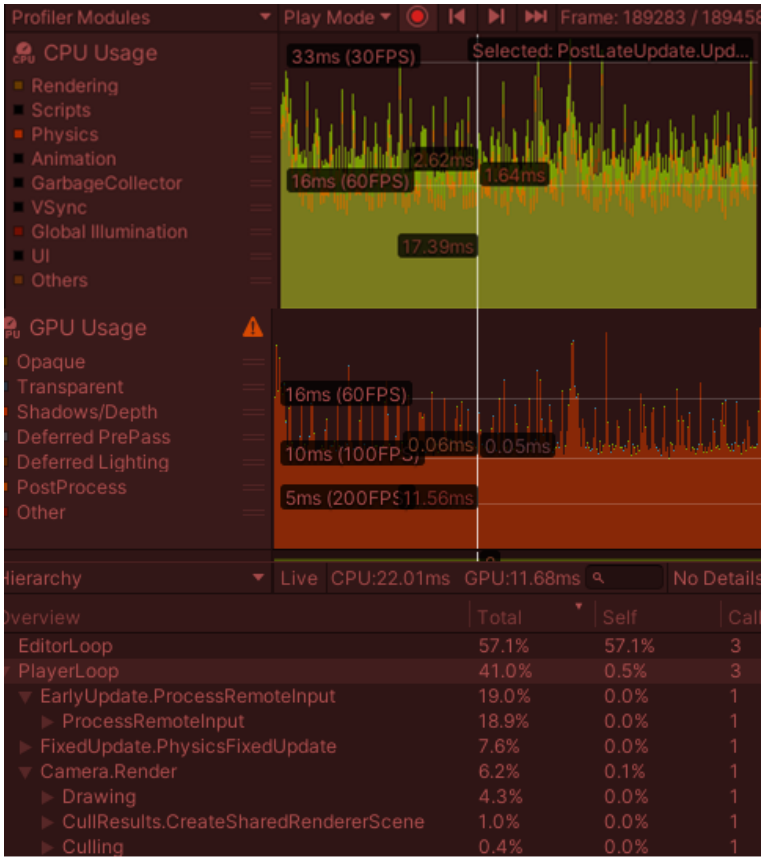
Optimalizace počtu 3D objektů je důležitým krokem při vývoji her, zejména pro mobilní zařízení, která mají omezené hardwarové možnosti. Snížení počtu objektů může pomoci zlepšit výkon hry a zkrátit dobu načítání scény.



Obrázek 52: Levý neoptimalizovaný pravý optimalizovaný 3D objekt

Na obrázku 52 vlevo je neoptimalizovaný 3D objekt, který je složen z mnoha dílků. Což může vést ke zvýšenému počtu nutných vykreslovacích operací. Při jeho použití jsem sledoval, jaký vliv má na výkon a odezvu hry. Výsledky ukázaly, že přítomnost tohoto neoptimalizovaného objektu měla negativní dopad na grafický procesor a snížila celkový výkon hry.

Na obrázku vpravo je zobrazen optimalizovaný 3D objekt. Tento objekt byl vytvořen tak, aby byl sestaven pouze z jednoho objektu místo mnoha dílků. Tímto způsobem se snížil počet objektů ve scéně. Při použití tohoto optimalizovaného objektu se opět provedlo měření a porovnání s předchozím neoptimalizovaným objektem.



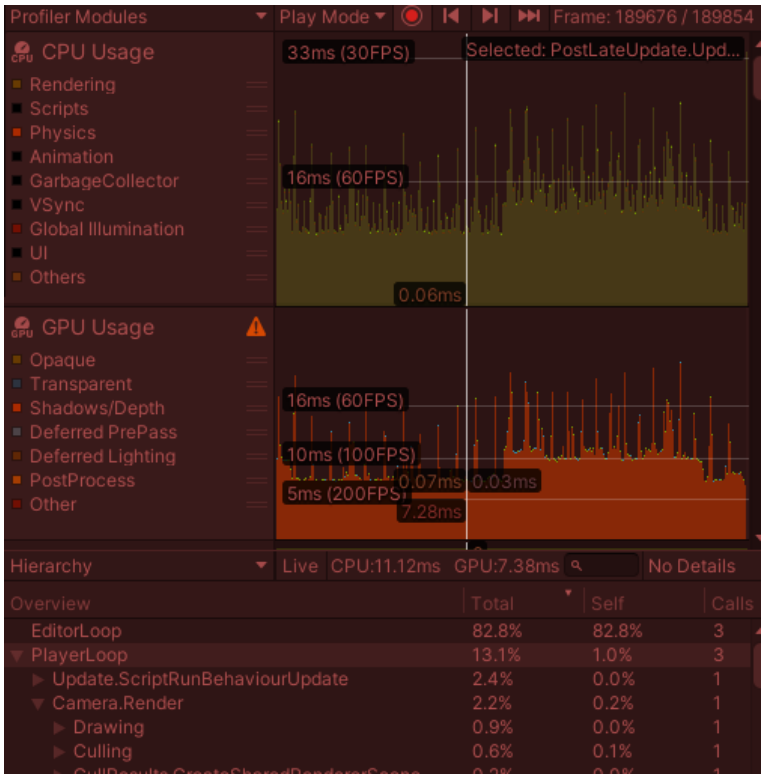
Obrázek 53: Neoptimalizovaný 3D objekt testovaného na mobilním zařízením

Obrázek 53 zobrazuje vytížení CPU a GPU u neoptimalizovaného 3D objektu testovaného na mobilním zařízením. Z obrázku 53 lze dále vyčíst, že CPU zatížení je nad 16ms(60 FPS) a pod 33ms(30FPS). To znamená, že hra se můžu potýkat s občasným zpomalení nebo zasekávání. GPU je dokonce na tom o něco líp a je většinu času okolo 16ms(60FPS), i když jde vidět občasný nárust, ale naštěstí tento nárust velmi zřídka překročí hranici 16ms(60FPS). To znamená, že hráč téměř nezaznamená zasekávání obrazu.

PlayerLoop je uvedeno s hodnotou 57,1 %. Z těchto 57,1 % je 41 % přiděleno kódu Early-Update.Pro-cesingRemoteInput. Tato hodnota využívána pro zpracování vstupů z různých zařízení nebo síťových komunikací. Dále 7,6 % je přiděleno kódu FixUpdate.PhysicsFi-xedUpdate. To znamená, že část CPU výkonu je využívána pro aktualizaci fyzikálního systému v herní scéně. Fyzikální výpočty a kolizní detekce jsou prováděny v rámci této části. Camera.Render má hodnotu 6,2 %.Tato část je zodpovědná za proces renderování scény do obrazu z kamery, který je pak zobrazen hráči.

EditorLoop představuje 57,1 % času stráveného na CPU. EditorLoop zahrnuje veškerý kód spouštěný v Unity Editoru, když je hra spuštěna ve scéně. V tomto případě 57,1 % ukazuje,

že většina času na CPU je využívána pro provádění editorových funkcí, manipulaci s objekty ve scéně a další související operace.

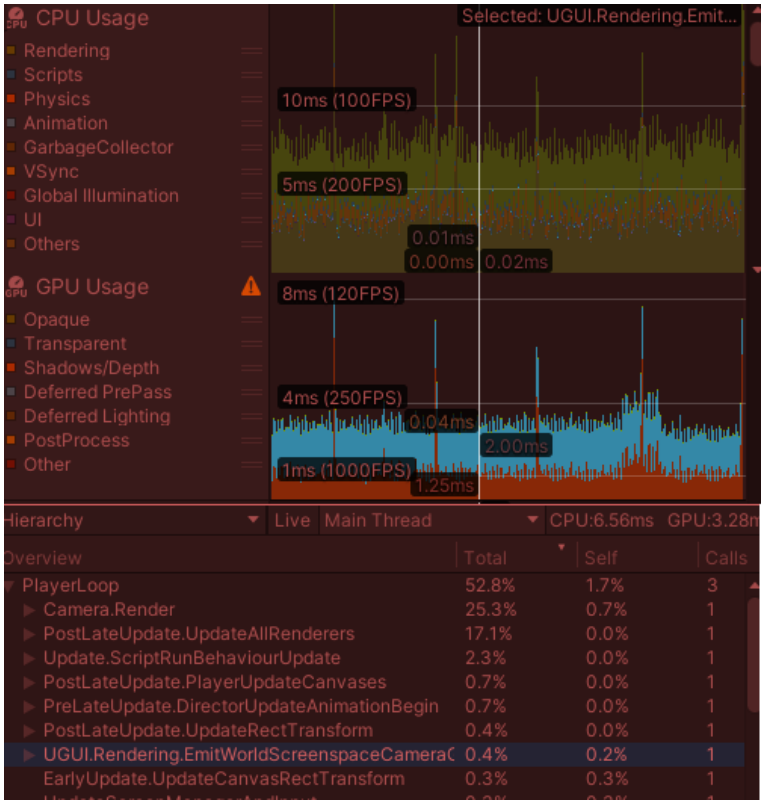


Obrázek 54: Optimalizovaný 3D objekt testovaného na mobilním zařízením

Na obrázku 54 jsou vykresleny vytížení CPU a GPU s grafy u optimalizovaného 3D objektu testovaného na mobilním zařízením. Dále lze vyčíst že CPU a GPU zatížení je pod 16ms(60 FPS) jen s občasnými výkyvy nad tuto hranici. To znamená že hra se po většinu času dostatečně rychle aktualizuje a hráč může zaznamenat občasné zpomalení nebo zasekávání. U GPU je nad 10ms (100 FPS) a pod 16ms(60 FPS) to umožnuje plynulí obraz bez jakýkoliv záseků.

PlayerLoop je uvedeno s hodnotou 13,1 %. Z těchto 13,1 % je 2,4 % přiděleno kódu Update.ScriptRunBehaviourUpdate. Tato část zahrnuje běh skriptů připojených k herním objektům, které provádějí logiku hry a aktualizace chování. Camera.Render má hodnotu 2,2 %. Tato část je zodpovědná za proces renderování scény do obrazu z kamery, který je pak zobrazen hráči.

EditorLoop je uvedeno s hodnotou 82,8 %. To ukazuje, že většina času na GPU je využívána pro provádění editorových funkcí, manipulaci s objekty ve scéně a další související operace.

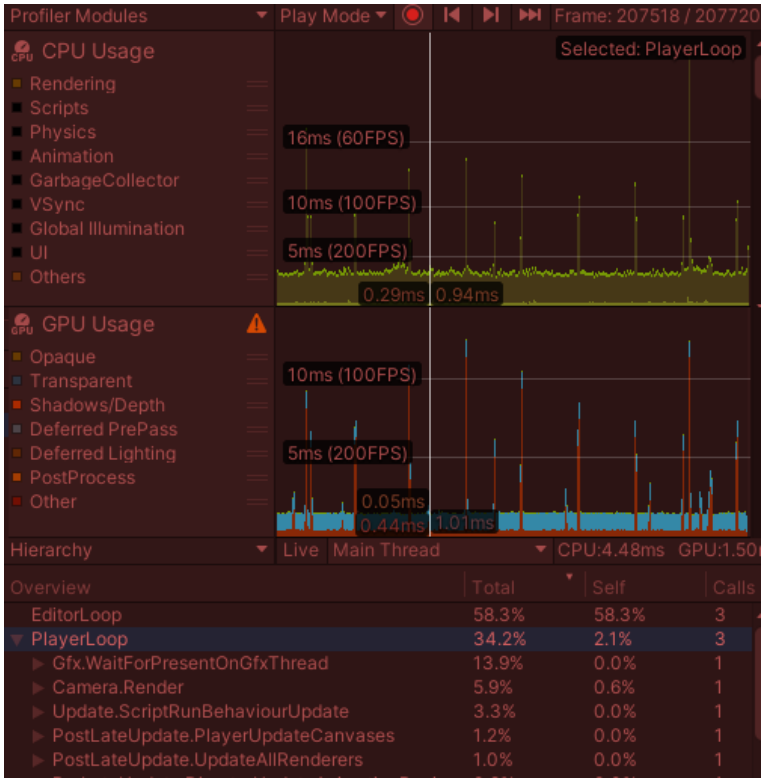


Obrázek 55: Neoptimalizovaný 3D objekt testovaného na desktopovém zařízením

Obrázek 55 zobrazuje vytížení CPU a GPU u neoptimalizovaného 3D objekt testovaného na desktopovém zařízením. Dále lze z obrázku 55 vyčíst že CPU zatížení je pod 10ms(100 FPS). To znamená že hra se dosta-tečně rychle aktualizuje a hráč nezaznamená žádné zpomalení nebo zasekávání. GPU je dokonce na tom ještě líp a je pod hodnotou 4ms(250FPS) při takové hodnotě nehrozí žádné trhání obrazu.

PlayerLoop je uvedeno s hodnotou 52,8 %. Z toho je 2,3 % přiděleno kódu Update.ScriptRunBehaviourUpdate. Tato část zahrnuje běh skript připojených k herním objektům, které provádějí logiku hry a aktualizace chování. PostLateUpdate.UpdateAllRenderers je uvedeno s hodnotou 17,1 %. Tato část zahrnuje aktualizaci všech renderovacích prvků ve scéně po vykreslení obrazu z kamer. Patří sem například aktualizace materiálů, osvětlení nebo stínování. Camera.Render má hodnotu 25,3 %, což znamená, že část CPU výkonu je využívána pro vykreslování kamery. Tato část je zodpovědná za proces renderování scény do obrazu z kamery, který je pak zobrazen hráči.

EditorLoop představuje 40 % času stráveného na CPU. EditorLoop zahrnuje veškerý kód spouštěný v Unity Editoru, když je hra spuštěna ve scéně. V tomto případě 82,8 % ukazuje, že většina času na CPU je využívána pro provádění editorových funkcí, manipulaci s objekty ve scéně a další související operace.



Obrázek 56: Optimalizovaný 3D objekt testovaného na desktopovém zařízením.

Obrázek 56 zobrazuje vytížení CPU a GPU s grafy u optimalizovaného 3D objektu testovaného na desktopovém zařízením. Z obrázku 56 dále lze vyčíst že CPU zatížení je pod 5ms(200 FPS) je občasné výkyvy k hranici 16ms (60FPS). To znamená že hra se dostatečně rychle aktualizuje a hráč nezaznamená žádné zpomalení nebo zasekávání. GPU je dokonce na tom ještě líp a je pod hodnotou 5ms(200FPS) jen občasné výkyvy k hranici 10ms (100FPS) při takové hodnotě nehrozí žádné trhání obrazu.

PlayerLoop je uvedeno s hodnotou 34,2 %. Z těchto 34,2 % je 13,9 % přiděleno kódu Gfx.WaitForPresentOnGfxThread. Tato hodnota znázorňuje čekání, které se obvykle provádí, aby se synchronizovaly grafické operace s ostatními částmi hry. Dále 3,3 % je přiděleno kódu Update.ScriptRunBehaviourUpdate. Tato část zahrnuje běh skriptů připojených k herním objektům, které provádějí logiku hry a aktualizace chování. Camera.Render má hodnotu 5,9 %. Tato část je zodpovědná za proces renderování scény do obrazu z kamery, který je pak zobrazen hráči.

EditorLoop je uvedeno s hodnotou 58,3 %. EditorLoop zahrnuje veškerý kód spouštěný v Unity Editoru, když je hra spuštěna ve scéně. V tomto případě 58,3 % ukazuje, že většina času na CPU je využívána pro provádění editorových funkcí, manipulaci s objekty ve scéně a další související operace.

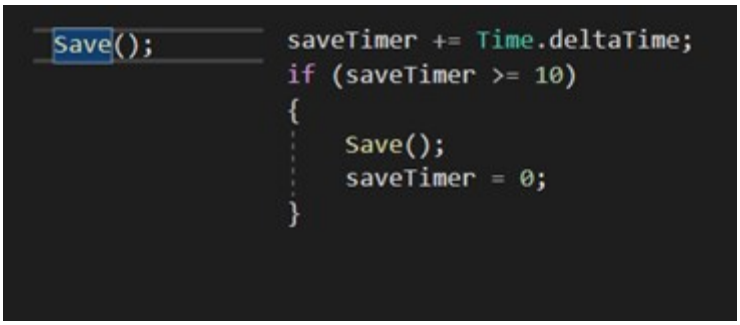
Tabulka 2: Srovnání optimalizované a neoptimalizovaného Particle systém

	CPU ms(FPS)	GPU ms(FPS)	Player-Loop %	Editor-Loop %
Neoptimalizovaný 3D objekt – testováno desktopovém zařízením	8(120)	4(250)	52,8	26,7
Optimalizovaný 3D objekt – testováno desktopovém zařízením	5(200)	1(1000)	34,2	58,3
Neoptimalizovaný 3D objekt – testováno na mobilním zařízením	24(45)	10(100)	41,0	57,1
Optimalizovaný 3D objekt – testováno na mobilním zařízením	13 (80)	5(200)	13,1	82,8

Tabulka 2 zobrazuje, že optimalizovaný 3D objekt dosahuje lepšího výkonu než neoptimalizovaný. Na mobilním zařízení je optimalizovaný objekt rychlejší, s CPU zpožděním 13 ms a GPU zpožděním 5 ms, což umožňuje dosáhnout vyšších FPS (80-200). Naopak, neoptimalizovaný objekt má vyšší zpoždění (24 ms CPU, 10 ms GPU) a nižší FPS (45-100).

Na desktopu je rozdíl mezi oběma verzemi menší. Optimalizovaný objekt dosahuje CPU zpoždění 5 ms a GPU zpoždění 1 ms, což umožňuje FPS až 200-1000. Neoptimalizovaný objekt má vyšší zpoždění (8 ms CPU, 4 ms GPU) a nižší FPS (120-250).

8.3 Neoptimalizovaný kód

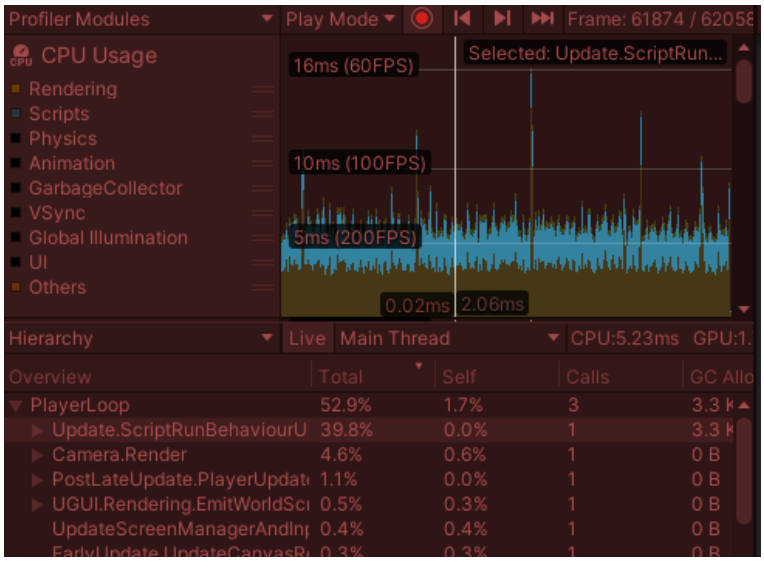


Obrázek 57: Levý neoptimalizovaný pravý optimalizovaný kód

V levém kódu je operace Save() volána přímo v každém snímku ve funkci Update(). To znamená, že Save() se spustí každý snímek, bez ohledu na časový interval. Tento přístup může vést k vyšší zátěži na systém, protože operace Save() se volá mnohem častěji.

V prvém kódu, který obsahuje časovač (saveTimer), je operace Save() volána pouze po uplynutí určité doby (10 sekund). To znamená, že Save() se spustí pouze jednou za určitý časový interval. Tímto způsobem se snižuje počet volání operace Save() a minimalizuje se tak zátěž na systém.

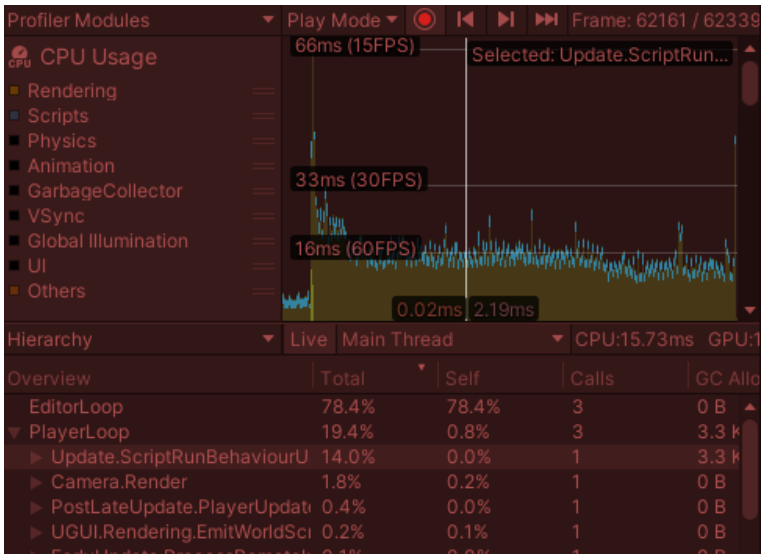
Optimalizace v tomto případě spočívá v použití prvního kódu s časovačem, kde se operace Save() volá pouze po uplynutí určitého časového intervalu. Tím se snižuje počet volání operace Save() a minimalizuje se zátěž na systém, což může přispět k lepšímu výkonu hry.



Obrázek 58: Neoptimalizovaný kód testovaného na desktopovém zařízením

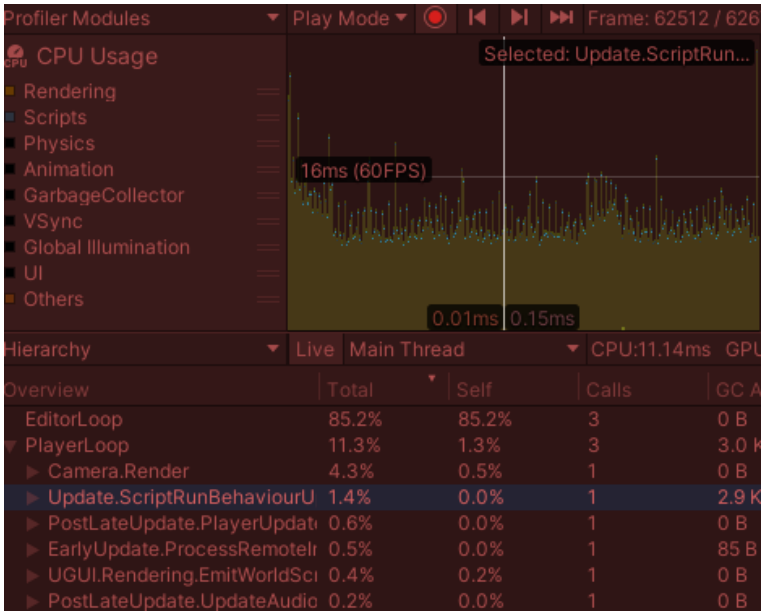
Obrázek 58 zobrazuje vytížení CPU a GPU s grafy u neoptimalizovaného kódu testovaného na desktopovém zařízením. Z obrázku 58 lze vyčíst ze CPU zatížení je pod 10ms(100 FPS). To znamená že hra se nedostatečně rychle aktualizuje a hráč nemůže zaznamenávat zpomalení nebo zasekávání. Lze si všimnou že modrá barva, která znázorňuje využití scriptu je ve větší míře. Oproti optimalizované části. PlayerLoop má hodnotu 52,9 a z toho je 39,8 % přiděleno Update.ScriptRunBehaviour. Tato část se týká provádění skriptů v rámci herních objektů. Skripty obsahují kód, který definuje chování objektů a provádí různé výpočty a operace. Další částí je Camera.Render, která má hodnotu 4,6 %. Tato hodnota se provádí různé operace související s vykreslováním, jako je sestavení seznamu viditelných objektů, nastavení transformací, shadery a další.

Na GPU toto zatížení nemělo žádný vliv z důvodu, že se jedná o script na ukládání většího počtu dat a nevykresluje žádné grafické objekty nebo textury.



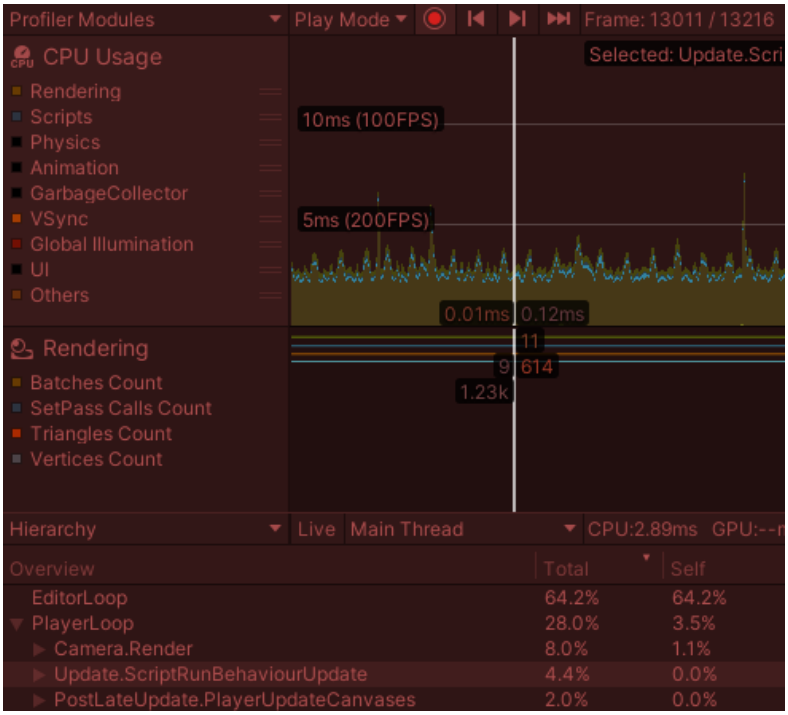
Obrázek 59: Neoptimalizovaný kód testováno na mobilním zařízením

Obrázek 59 zobrazuje vytížení CPU a GPU s grafy u neoptimalizovaného kódu testovaného na mobilním zařízením. Dále lze vyčíst ze CPU zatížení je pod 33ms(30 FPS) ale na hodnotou 16ms(60FPS). To znamená že hra se nedostatečně rychle aktualizuje a hráč může u určitý her zaznamenávat zpomalení nebo zasekávání. Lze si všimnou že modrá barva, která znázorňuje využití scriptu je ve větší míře. Oproti optimalizované části. PlayerLoop má hodnotu 19,4 a z toho je 14,0 % přiděleno Update.ScriptRunBehaviour. Tato část se týká provádění skriptů v rámci herních objektů. Skripty obsahují kód, který definuje chování objektů a provádí různé výpočty a operace. Další částí je Camera.Render, která má hodnotu 1,8 %. V této fázi se provádí různé operace související s vykreslováním, jako je sestavení seznamu viditelných objektů, nastavení transformací, shadery a další.



Obrázek 60: Optimalizovaný kód testováno na mobilním zařízením

Obrázek 60 zobrazuje vytížení CPU a GPU s grafy u optimalizovaného kódu testovaného na mobilním zařízením. Z obrázku lze vyčíst ze CPU zatížení je pod 16ms (60 FPS). To znamená že hra se dostatečně rychle aktualizuje a hráč může u určitý her zaznamenávat drobné zpomalení nebo zasekávání. Lze si všimnou že modrá barva, která znázorňuje využití scriptu je velmi malé. Oproti neoptimalizované části. PlayerLoop má hodnotu 11,3 % a z toho je 1,4 % přiděleno Update.ScriptRunBehaviour. Tato část se týká provádění skriptů v rámci herních objektů. Skripty obsahují kód, který definuje chování objektů a provádí různé výpočty a operace. Další částí je Camera.Render, která má hodnotu 4,3 %. Tato část zahrnuje proces renderování scény z pohledu kamery na CPU. V této fázi se provádí různé operace související s vykreslováním, jako je sestavení seznamu viditelných objektů, nastavení transformací, shadery.



Obrázek 61: Optimalizovaný kód testovaný na desktopovém zařízením.

Obrázek 61 zobrazuje vytížení CPU a GPU s grafy u optimalizovaného kódu testovaného na desktopovém zařízením. Z obrázku 61 lze vyčíst ze CPU zatížení je velmi pod 16ms(60 FPS). To znamená že hra se dostatečně rychle aktualizuje a hráč nemůže zaznamenávat zpomalení nebo zasekávání. Lze si všimnou že modrá barva, která znázorňuje využití scriptu je velmi malé. Oproti neoptimalizované části.

PlayerLoop má hodnotu 23,9 a z toho je 4,3 % přiděleno Update.ScriptRunBehaviour. Tato část se týká provádění skriptů v rámci herních objektů. Skripty obsahují kód, který definuje chování objektů a provádí různé výpočty a operace. Další částí je Camera.Render, která má hodnotu 6,0 %. Tato část zahrnuje proces renderování scény z pohledu kamery na CPU. V této fázi se provádí různé operace související s vykreslováním, jako je sestavení seznamu viditelných objektů, nastavení transformací, shadery a další.

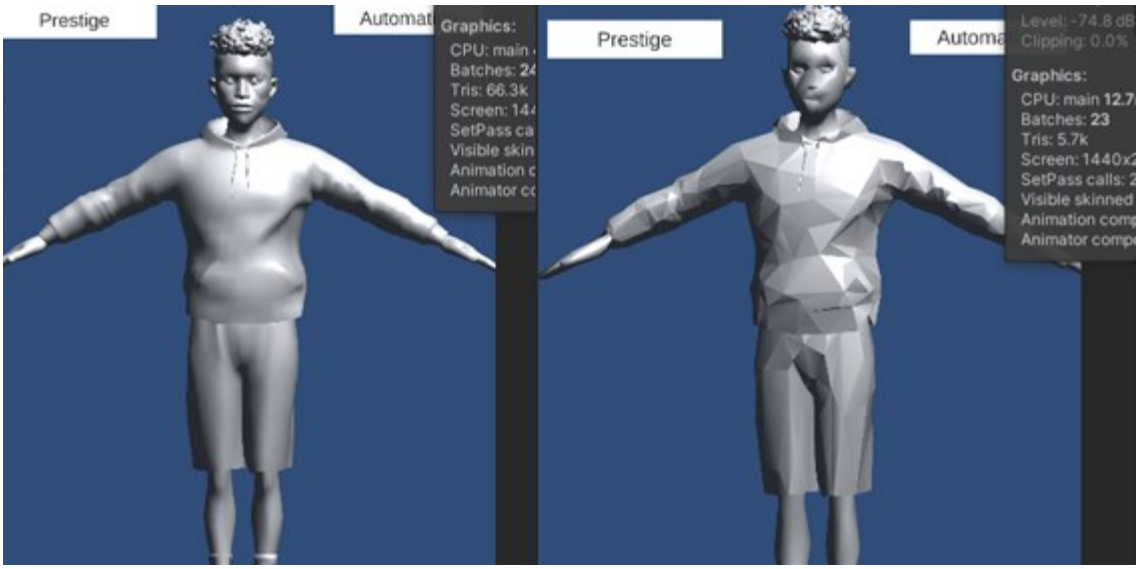
Tabulka 3: Srovnání optimalizované a neoptimalizovaného kódu.

	CPU ms(FPS)	Update Script vytížení %	Player-Loop %	Editor-Loop %
Neoptimalizovaný kód – testováno na desktopovém zařízením	5(200)	39,8	52,9	32,9
Optimalizovaný kód – testováno na desktopovém zařízením	4(250)	4,4	28,0	64,2
Neoptimalizovaný kód – testováno na mobilním zařízením	16(60)	14,0	19,4	78,4
Optimalizovaný kód – testováno na mobilním zařízením	13(80)	1,4	11,3	85,2

Výsledky z tabulky 3 naznačují, že na desktopovém zařízení optimalizovaný kód snižuje hodnotu CPU ms z 5 na 4 a vytížení Update Scriptu z 39,8 % na 4,4 %. Zatížení Player-Loopu je nižší u optimalizovaného kódu 28,0 % ve srovnání s neoptimalizovaným kódem 52,9 %. Na mobilním zařízení je rozdíl také znatelný, kdy optimalizovaný kód snižuje hodnotu CPU ms z 16 na 13 a vytížení update scriptu z 14,0 % na 1,4 %. Zatížení Player-Loopu je nižší u optimalizovaného kódu 11,3 % ve srovnání s neoptimalizovaným kódem 19,4 %. Celkově výsledky ukazují. Že optimalizovaný kód dosahuje lepšího výkonu a nižšího vytížení systému ve srovnání s neoptimalizovaným kódem

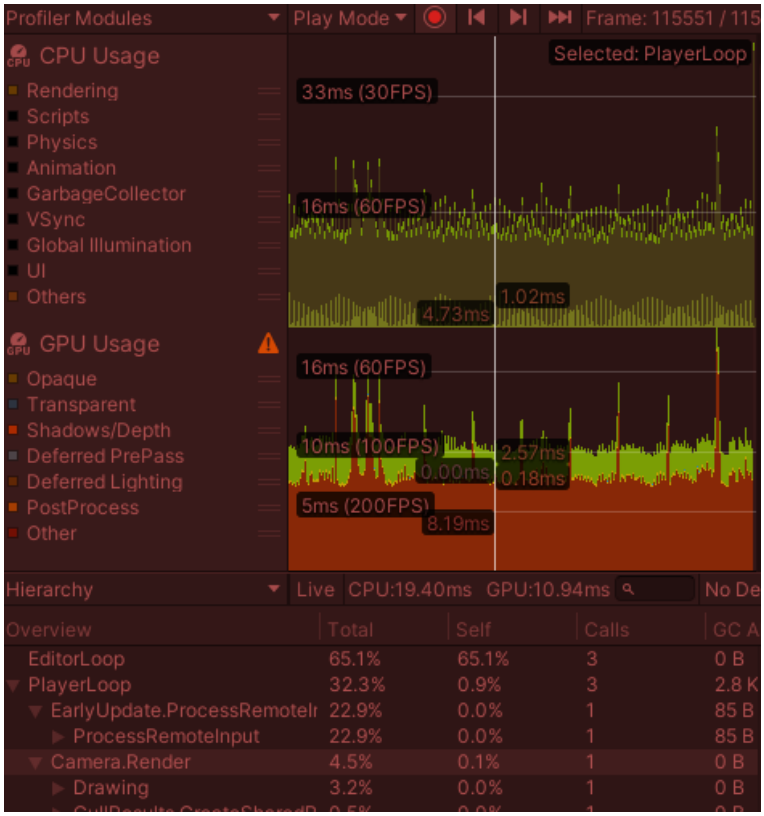
8.4 Zatížení LOD

Zde byl vybrán 3D objekt s více detaily, aby bylo možné více vizuálně zobrazit snížení LOD, přesněji charakter z internetu.[52] Tento charakter byl následně optimalizován pomocí snížení Tris.



Obrázek 62: Vlevo neoptimalizovaná charakter a vpravo optimalizovaný

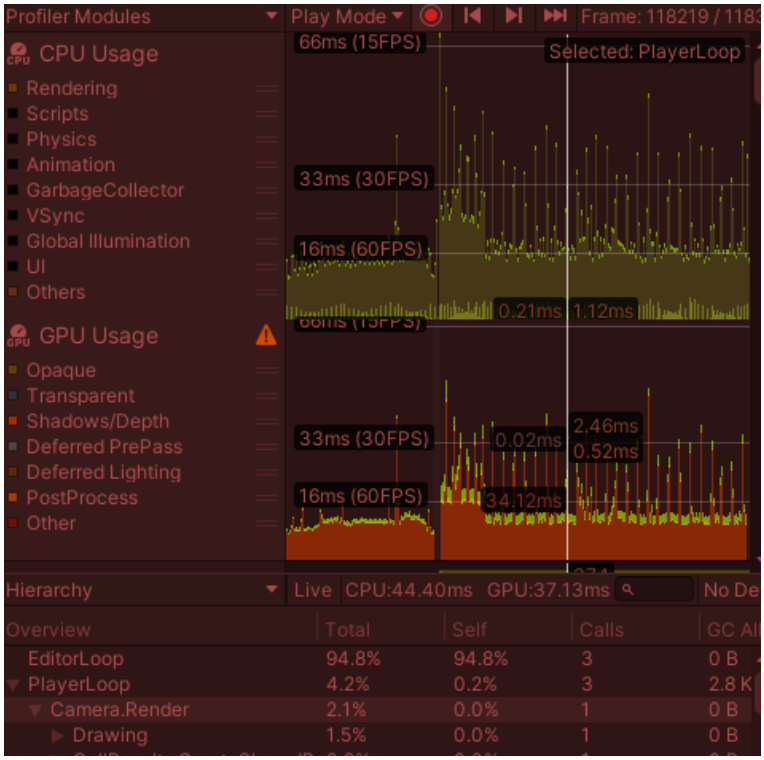
Na obrázku 62 jsi jde všimnou, že optimalizovaný LOD má své výhody i nevýhody. Mezi výhody patří menší zatížení hry a zlepšení výkonu. Naopak nevýhodou je, že některé charaktery při velkém snížení můžou vypadat nereálně a může to kazit hráčský dojem. Proto je dobré pamatovat, kdy, kde a jak moc se nám vyplatí danou optimalizaci provést.



Obrázek 63: Optimalizovaný charakter testován na mobilním zařízením

Obrázek 63 zobrazuje vytížení CPU a GPU s grafy u optimalizovaného charakteru testován na mobilním zařízením. Z obrázku 63 lze vyčíst ze CPU zatížení je okolo hranice 16ms(60 FPS) a jen malá část přesáhne tuto hranici. To znamená že hra občas nedostatečně rychle aktualizuje a hráč může u určitý her zaznamenávat zpomalení nebo zasekávání. Dále jde vyčíst, že GPU je občas okolo hodnoty 10ms(100FPS), to znamená, že hra nebude způsobovat trhání obrazu.

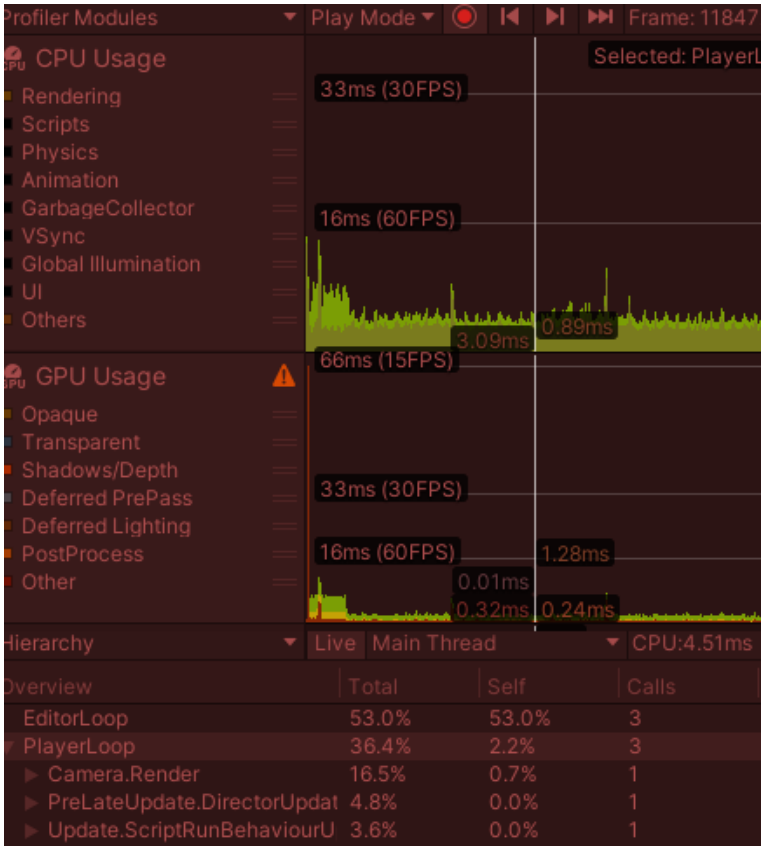
PlayerLoop má hodnotu 32,3 % a z této hodnoty je 22,9 % přiděleno kategorii EarlyUpdate.ProcessRemoteInput. Tato část zahrnuje zpracování vstupů na GPU, což může zahrnovat manipulaci s daty v souvislosti s ovládáním hry, síťovou komunikací nebo jinými vstupy. Další částí je Camera.Render, která má hodnotu 4,5 %. Zde se provádí výpočet shaderů, nasvícení, stínování a dalších operací souvisejících s vykreslováním objektů z pohledu kamery.



Obrázek 64: neoptimalizovaný charakter testován na mobilním zařízením

Obrázek 64 zobrazuje vytížení CPU a GPU s grafy u neoptimalizovaného charakteru testován na mobilním zařízením. Z obrázku 64 lze vyčíst ze CPU zatížení je okolo 24ms(45 FPS) a kolísavé navýšení, až k hranici 33ms (30FPS). To znamená že hra se nedostatečně rychle aktualizuje a hráč může u většiny her zaznamenávat zpomalení nebo zasekávání. Dále jde vyčíst, že GPU je občas přes hodnotu 16ms(60FPS) a to až 33ms(30FPS), to může způsobovat trhání obrazu, kterého si hráč můžeš všimnout.

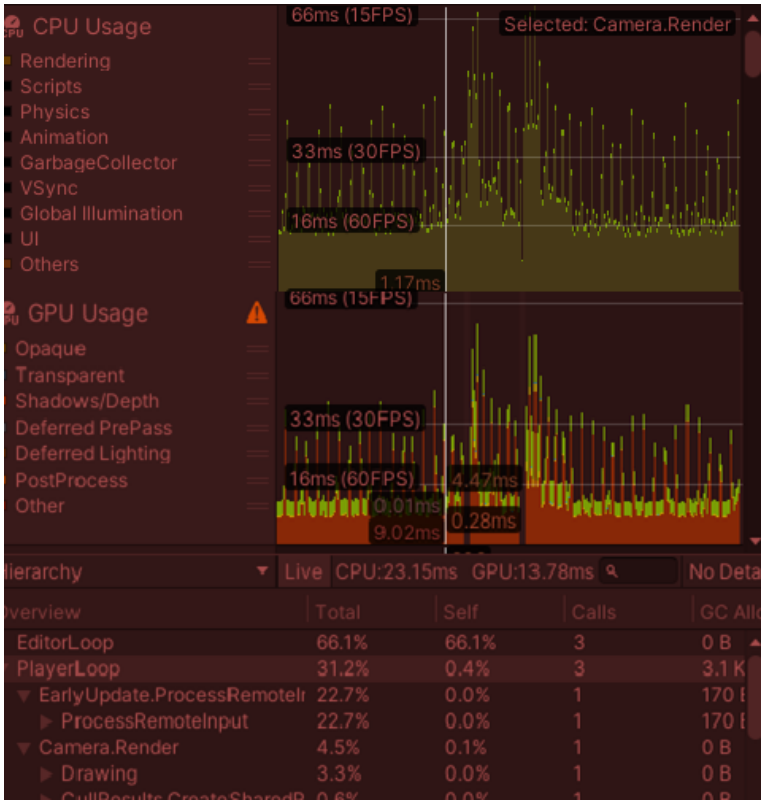
PlayerLoop má hodnotu 4,2 % a z této hodnoty je 2,1 % přiděleno Camera.Render. Tato část se týká renderování scény na GPU. Zde se provádí výpočet shaderů, nasvícení, stínování a dalších operací souvisejících s vykreslováním objektů z pohledu kamery.



Obrázek 65: Optimalizovaný charakter testován na desktopovém zařízením

Obrázek 65 zobrazuje vytížení CPU a GPU s grafy u optimalizovaného charakteru testován na desktopovém zařízením. Z obrázku 65 lze vyčíst ze CPU a GPU zatížení je velmi pod hranicí 16ms (60 FPS). To znamená že hra se dostatečně rychle aktualizuje a hráč nezaznamená žádné zpomalení nebo zasekávání obrazu.

PlayerLoop má hodnotu 36,4 % a z této hodnoty je 16,5 % přiděleno kategorii Camera.Render, která má hodnotu 16,5 %. Zde se provádí výpočet shaderů, nasvícení, stínování a dalších operací souvisejících s vykreslováním objektů z pohledu kamery. Update.ScriptRunBehaviourUpdate se zahrnuje běh skriptů připojených k herním objektům, které provádějí logiku hry a aktualizace chování. Hodnota 4,8 % je přiděleno kategorii PreLateUpdate.DirectirUpdateAnimationBegin. Tato část se týká přímé aktualizace animací před vykreslováním. V tomto kroku jsou prováděny výpočty a transformace související s animacemi herních objektů.



Obrázek 66: Neoptimalizovaný charakter testován na desktopovém zařízením

Obrázek 66 zobrazuje vytížení CPU a GPU s grafy u neoptimalizovaného charakteru testován na desktopovém zařízením. Dále lze obrázku 66 vyčíst ze CPU a GPU zatížení je u hranice 16ms(60 FPS) a oba občas přesahují hranice 33ms(30FPS) To může způsobovat, že hra se nedostatečně rychle aktualizuje a hráč může zaznamenávat zpomalení nebo zasekávání obrazu.

PlayerLoop má hodnotu 31,2 % a z této hodnoty je 22,7 % přiděleno kategorii EarlyUpdate.ProcessRemoteInput. Tato část zahrnuje zpracování vstupů, což může zahrnovat manipulaci s daty v souvislosti s ovládáním hry, síťovou komunikací nebo jinými vstupy. Další částí je Camera.Render, která má hodnotu 4,5 %. Tato část se týká renderování scény na GPU. Zde se provádí výpočet shaderů, nasvícení, stínování a dalších operací souvisejících s vykreslováním objektů z pohledu kamery.

Tabulka 4: Srovnání optimalizované a neoptimalizovaného LOD

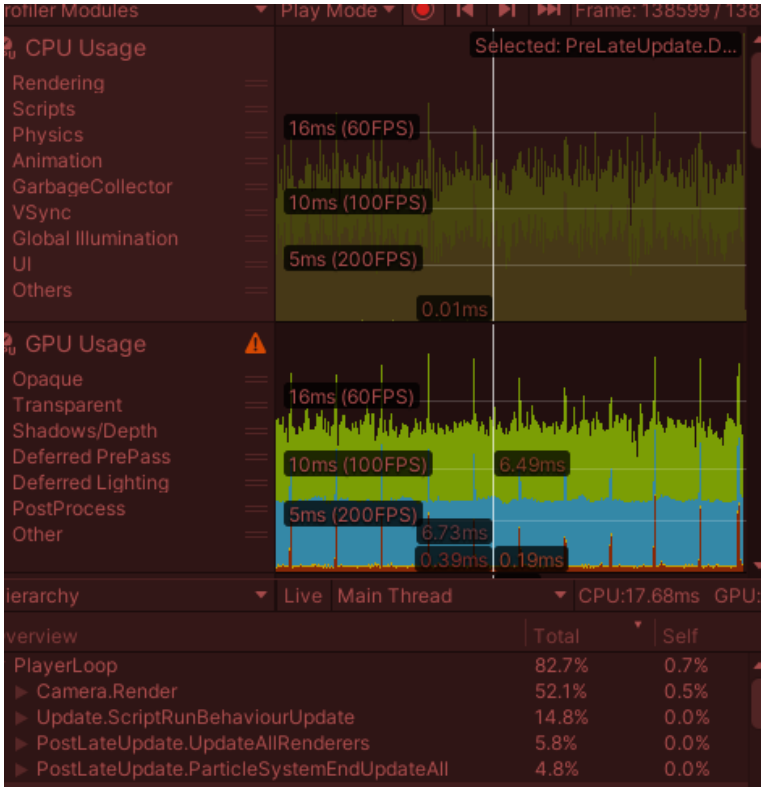
	CPU ms(FPS)	GPU ms(FPS)	Player-Loop %	Editor-Loop %
Neoptimalizovaný LOD – testováno na desktopovém zařízením	24(45)	24(45)	31,2	66,1
Optimalizovaný LOD – testováno na desktopovém zařízením	8(120)	8(120)	36,4	53
Neoptimalizovaný LOD – testováno na mobilním zařízením	24(45)	16(60)	32,3	65,1
Optimalizovaný LOD – testováno na mobilním zařízením	16 (60)	10(100)	4,2	94,8

Tabulka 4 prezentuje výsledky testů provedených na desktopovém a mobilním zařízení. Neoptimalizovaný LOD vykazuje vyšší hodnoty času zpracování CPU a GPU, nižší hodnoty snímkové frekvence (FPS) a vyšší zatížení Player-Loop a Editor-Loop. Naopak, optimalizovaný LOD dosahuje nižších hodnot času zpracování CPU a GPU, vyšších hodnot snímkové frekvence (FPS) a nižšího zatížení Player-Loop a Editor-Loop.

Tyto výsledky naznačují, že optimalizovaný LOD přináší zlepšení výkonu ve srovnání s neoptimalizovaným LOD. Jak u desktopového, tak i mobilního zařízení.

8.5 Všechny zatížení dohromady

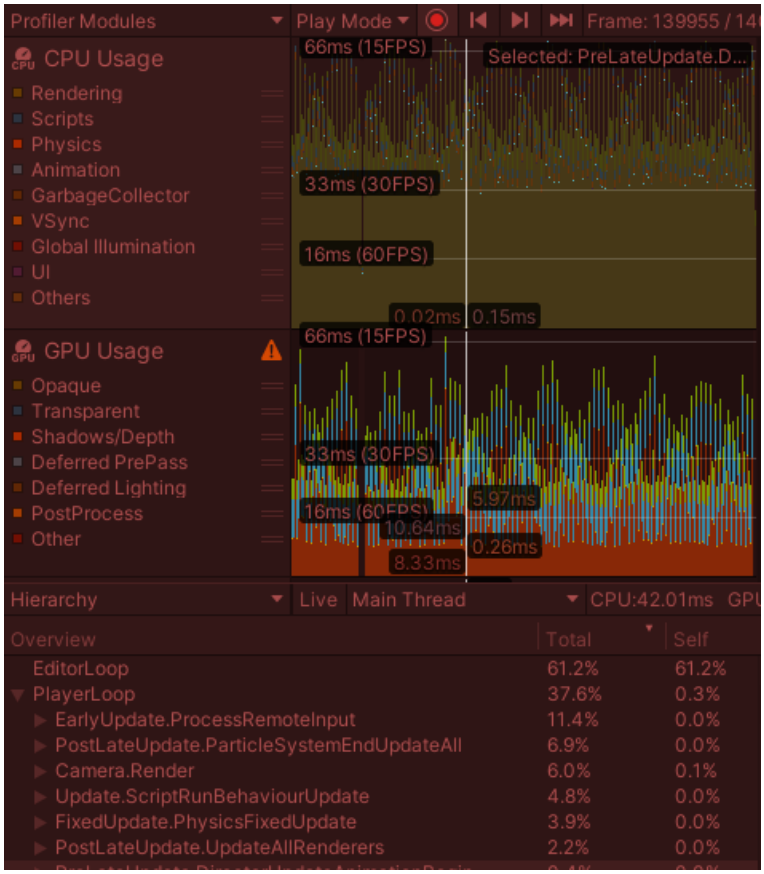
Zde jsem provedl zatížení se všemi předchozími metodami najednou a následnou jejich optimalizací.



Obrázek 67: Neoptimalizováno se všema metodama testováno na desktopovém zařízením

Obrázek 67 zobrazuje vytížení CPU a GPU s grafy při spuštění všech neoptimalizovaných metod testovaného na desktopovém zařízením. Z obrázku 67 lze vyčíst ze CPU a GPU zatížení je mezi hranicí 16ms(60 FPS) a 10ms (100FPS) a jen občasné přesáhnutí hranice 16ms (60FPS). To znamená že hra se dostatečně rychle aktualizuje a hráč může je za určitých podmínek postřehnout zpomalení nebo zasekávání obrazu.

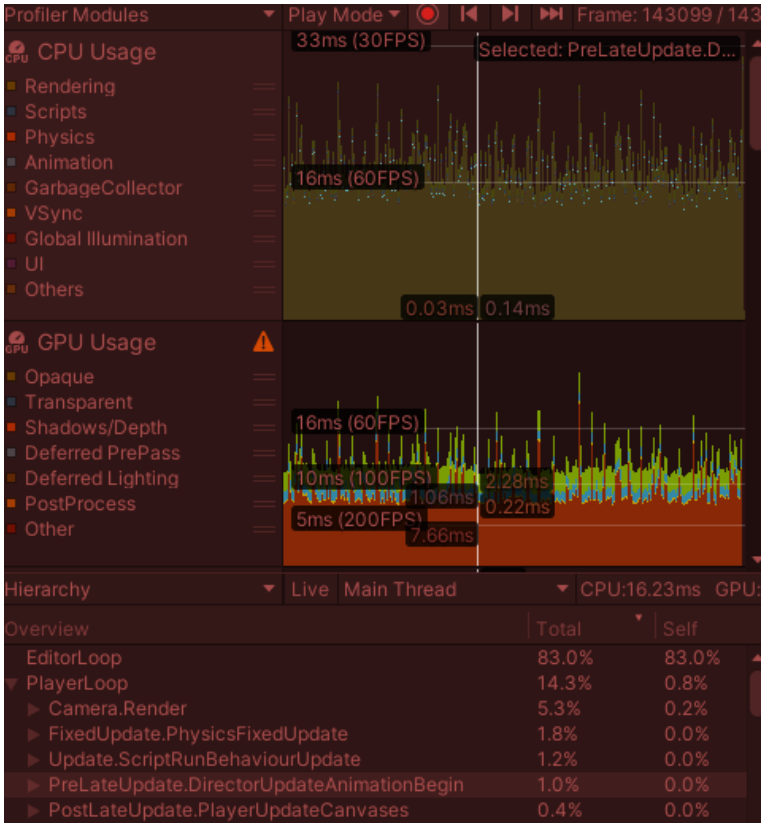
PlayerLoop, který představuje 82,7 % celkového zatížení. Z toho je 52,1 % připadá na Camera.Render, což naznačuje, že vykreslování scény na obrazovku je náročným úkonem. Dále ve frameworku PlayerLoop je také významná část zabírána funkcí Update.ScriptRunBehaviourUpdate, která je odpovědná za spouštění skriptů během každého snímku. Tato část představuje 14,8 % z celkového zatížení. Tímto je naznačeno, že provádění skriptů v každém snímku má také podstatný vliv na celkový výkon hry. Další důležitou částí v PlayerLoop je PostLateUpdate.UpdateAllRenderers, která zabírá 5,8 % zatížení. Tato funkce je zodpovědná za aktualizaci všech vykreslovačů (renderers) na konci snímku, což je důležitý proces pro správné vykreslování objektů ve scéně. Nakonec je tu ještě funkce PostLateUpdate.ParticleSystemEndUpdateAll, která představuje 4,8 % zatížení a je zodpovědná za dokončení aktualizace všech částicových systémů ve hře.



Obrázek 68: Neoptimalizováno se všema metodama testováno na mobilním zařízením

Obrázek 68 zobrazuje vytížení CPU a GPU s grafy při spuštění všech neoptimalizovaných metod testovaného na mobilním zařízením. Z obrázku 68 lze vyčíst ze CPU a GPU zatížení je mezi hranicí 66ms (15 FPS) a 33ms (30FPS). To určuje, že hra se nedostatečně rychle aktualizuje, a to může způsobit pravidelné zasekávání obrazu a zpomalování hry.

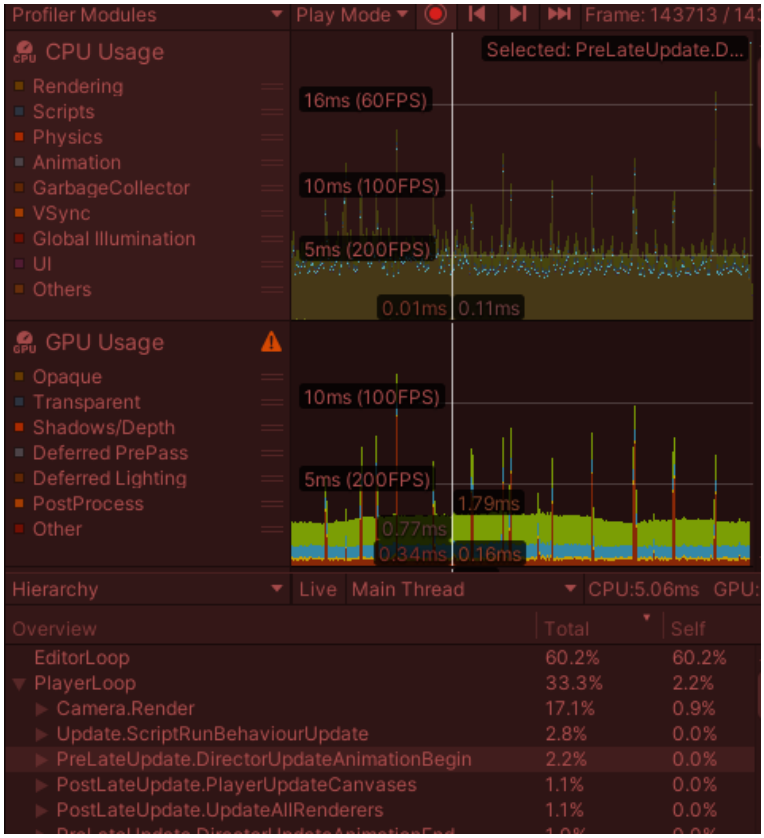
PlayerLoop, který představuje 37,6 % celkového zatížení. Z toho je 6,0 % připadá na Camera.Render, což naznačuje, že vykreslování scény na obrazovku je náročným úkonem. Dále je část zabírána funkcí Update.ScriptRunBehaviourUpdate, která je odpovědná za spouštění skriptů během každého snímku. Tato část představuje 4,8 % z celkového zatížení. Další naměřeným parametrem je PostLateUpdate.UpdateAllRenderers, která zabírá 2,2 % zatížení. Tato funkce je důležitá proces pro správné vykreslování objektů ve scéně. Následuje funkce PostLateUpdate.ParticleSystemEndUpdateAll, která představuje 6,9 % zatížení a je zodpovědná za dokončení aktualizace všech částicových systémů ve hře. Další zastoupenou funkcí je FixUpdate.PhysicsFixedUpdate, která představuje 3,9 % zatížení. Tato část se věnuje fyzikálním výpočtům a aktualizaci fyzikálního simulátoru. Poslední je EarlyUpdate.ProcessRemoteInput, která zabírá 11,4 % zatížení. Tato část se zaměřuje na zpracování vstupu a interakce s vzdálenými zařízeními.



Obrázek 69: Optimalizace všech metod testováno na mobilním zařízením

Obrázek 69 zobrazuje vytížení CPU a GPU s grafy při spuštění všech optimalizovaných metod testovaného na mobilním zařízením. Z obrázku 69 lze vyčíst, že CPU zatížení je mezi hranicí 16ms(60 FPS) a 33ms (60FPS). Může velmi pravidelně způsobovat neplynulí zážitek. U GPU je tato hodnota lepší, protože je mezi 10ms (100FPS) a 16ms (60FPS) bez větších zasekávání obrazu s grafickými prvky.

PlayerLoop, který představuje 14,3 % celkového zatížení. Z toho 5,3 % připadá na Camera.Render, což naznačuje, že vykreslování scény na obrazovku je značně náročným úkonem. Další významnou část zabírána funkcí Update.ScriptRunBehaviourUpdate, která je odpovědná za spouštění skriptů během každého snímku. Tato část představuje 1,2 % z celkového zatížení. Tímto je naznačeno, že provádění skriptů v každém snímku má také určitý vliv na celkový výkon hry. Následuje funkce FixUpdate.PhysicsFixedUpdate, která zabírá 1,8 % zatížení. Tato funkce se stará o aktualizaci fyzikálního systému ve hře, což je důležitý proces pro správné fungování kolizí a pohybu objektů ve scéně. Další část je PostLateUpdate.PlayerUpdateCanvases, která představuje 0,4 % zatížení. Tato část se věnuje aktualizaci a renderování všech canvasů ve hře. Poslední je PreLateUpdate.DirectorUpdateAnimationBegin, která představuje 1,0 % zatížení. Tato část se stará o aktualizaci animací ve scéně.



Obrázek 70: Optimalizace všech metod testovaného na desktopovém zařízením

Obrázek 70 zobrazuje vytížení CPU a GPU s grafy při spuštění všech optimalizovaných metod testovaného na desktopovém zařízením. Z obrázku 70 lze vyčíst ze CPU a GPU zatížení je pod hranicí 5ms(200 FPS) jen občasné přesáhnutí hranice 10ms (100FPS). To znamená že hra se dostatečně rychle aktualizuje a hráč může jen ve výjimečných situacích postřehnout zpomalení nebo zasekávání obrazu. Což je způsobené náhlým přesahem 10ms(100FPS).

PlayerLoop, který představuje 33,3 % celkového zatížení. Z toho 17,1 % připadá na Camera.Render, což naznačuje, že vykreslování scény na obrazovku je náročným úkonem. Další část zabírá funkce Update.ScriptRunBehaviourUpdate, která je odpovědná za spouštění skriptů během každého snímku. Tato část představuje 2,8 % z celkového zatížení. Tímto je naznačeno, že provádění skriptů v každém snímku má také podstatný vliv na celkový výkon hry. Následuje funkce PostLateUpdate.UpdateAllRenderers, která zabírá 1,1 % zatížení. Tato funkce se stará o aktualizaci všech renderers na konci snímku, což je důležitý proces pro správné vykreslování objektů ve scéně. Další část je PostLateUpdate.PlayerUpdateCanvases, která také zabírá 1,1 % zatížení. Tato část se věnuje aktualizaci a renderování všech canvasů ve hře. Poslední je PreLateUpdate.DirectorUpdateAnimationBegin, která představuje 2,2 % zatížení. Tato část se stará o aktualizaci animací ve scéně.

Tabulka 5: Srovnání optimalizované a neoptimalizovaného všechny metody

	CPU ms(FPS)	GPU ms(FPS)	Player-Loop %	Editor-Loop %
Neoptimalizovaný všechny metody – testováno na desktopovém zařízením	24(45)	16(60)	82,7	16,4
Optimalizovaný všechny metody – testováno na desktopovém zařízením	5(200)	5(200)	33,3	60,2
Neoptimalizovaný všechny metody – testováno na mobilním zařízením	66 (15)	33(30)	37,6	61,2
Optimalizovaný všechny metody – testováno na mobilním zařízením	16 (60)	10(100)	14,3	83,0

Tabulka 5 ukazuje, že optimalizované všechny metody dosahují nižšího vytížení CPU i GPU ve srovnání s neoptimalizovanými metodami. Při testování na mobilním zařízení je optimalizovaný kód vytížen 16 ms (60 FPS) CPU a 10 ms (100 FPS) GPU. V PlayerLoop má vytížení 14,3 % a v EditorLoop 83,0 %. I při testování na desktopovém zařízení je patrný rozdíl. Optimalizovaný kód má vytížení 5 ms (200 FPS) CPU a 5 ms (200 FPS) GPU. V PlayerLoop má vytížení 33,3 % a v EditorLoop 60,2 %. Naopak, neoptimalizovaný kód má vytížení 24 ms (45 FPS) CPU a 16 ms (60 FPS) GPU. V PlayerLoop má vytížení 82,7 % a v EditorLoop 16,4 %.

ZÁVĚR

Bakalářské práce poukazuje na význam optimalizace herního prostředí a ukazuje negativní dopad neoptimalizovaných částí, jako je nadměrné využití Particle systému, velké množství objektů a nedostatečné využití LOD. Analýza zatížení hry jasně ukazuje, že tyto neoptimalizační prvky mohou výrazně snižovat herní výkon a plynulost, což zhoršuje celkový uživatelský zážitek.

Optimalizace hry je nezbytnou součástí vývojářského procesu, a to zejména v případě, kdy se používají náročné herní prvky. Správné využití Particle systému, například omezení počtu částic nebo použití optimalizovaných shaderů, může výrazně snížit zatížení na grafické kartě. Implementace LOD modelů pro objekty umožňuje dynamické přizpůsobení detailu podle vzdálenosti od kamery, což snižuje nároky na grafiku a zvyšuje výkon hry.

Kromě toho je nezbytné věnovat pozornost optimalizaci herního kódu. Neoptimalizovaný kód, který obsahuje zbytečné smyčky, nepotřebné výpočty nebo neefektivní algoritmy, může zpomalovat běh hry a snižovat celkový výkon. Je důležité provádět průběžnou analýzu a ladění kódu, aby se minimalizovaly takové problémy a zlepšil se herní výkon.

SEZNAM POUŽITÉ LITERATURY

[1] Getting to Know the Unity Editor | Get Started with Unity | InformIT. InformIT: The Trusted Technology Source for IT Pros and Developers [online]. Copyright © 2023 Pearson Education, [cit. 12.05.2023]. Dostupné z: https://www.informit.com/articles/article.aspx?p=3129466&seqNum=2

[2] Unity - Developing Your First Game with Unity and C# | Microsoft Learn. [online]. Copyright © Microsoft 2023 [cit. 12.05.2023]. Dostupné z: https://learn.microsoft.com/en-us/archive/msdn-magazine/2014/august/unity-developing-your-first-game-with-unity-and-csharp

[3] Unity Game Engine Guide: How to Get Started with the Most Popular Game Engine Out There. [online]. Dostupné z: https://www.freecodecamp.org/news/unity-game-engine-guide-how-to-get-started-with-the-most-popular-game-engine-out-there/

[4] Android Authority: Tech Reviews, News, Buyer's Guides, Deals, How-To [online]. Dostupné z: https://www.androidauthority.com/what-is-unity-1131558

[5] Explore the Unity Editor - Unity Learn. Learn game development w/ Unity | Courses & tutorials in game design, VR, AR, & Real-time 3D | Unity Learn [online]. Copyright © 2023 Unity Technologies [cit. 12.05.2023]. Dostupné z: https://learn.unity.com/tutorial/explore-the-unity-editor-1#6273f00fedbc2a7f158cc1ee

[6] [online]. Copyright © [cit. 12.05.2023]. Dostupné z: https://connect-prd-cdn.unity.com/20220606/learn/images/7fabb375-5282-4852-9ecf-d8acc254052b_EditorExplore.png.2000x0x1.png

[7] DORAN, John P. Unity 2020 Mobile Game Development. 2. Birmingham, UK: Packt, 2020. ISBN 9781838987336.

[8] Beginner's Guide to Unity - Understanding Unity Editor and Its Interface. Circuit Stream | Online XR and Real-time 3D Courses [online]. Copyright © 2020 Circuit Stream. All rights reserved [cit. 12.05.2023]. Dostupné z: https://circuitstream.com/blog/beginners-guide-to-unity-understanding-unity-editor-and-its-interface

[9] Mobile Game Development: Differences, Challenges, New Solutions. 80 Level [online]. Copyright © [cit. 12.05.2023]. Dostupné z: https://80.lv/articles/mobile-game-development-differences-challenges-new-solutions/

[10] What are the differences between mobile and computer games?. Object moved [online]. Copyright © 2023 [cit. 12.05.2023]. Dostupné z: https://www.dotnek.com/Blog/Games/what-are-the-differences-between-mobile-and-c

[11] What are Diference Between Mobile and PC Game Development. Juego Studios: Game Development Studio | Game Design Studio [online]. Copyright © [cit. 12.05.2023]. Dostupné z: https://www.juegostudio.com/blog/difference-between-mobile-and-pc-game-development

[12] AVERSA, Dr. Davide a Chris DICKINSON. Unity Game Optimization. 3. Birmingham, UK: Packt, 2019. ISBN 9781838556518.

[13] PC Gaming Vs. Mobile Gaming: Which Is The Best For Gaming? | ONMO. ONMO - Just Beat It [online]. Copyright © Copyright ONMO [cit. 12.05.2023]. Dostupné z: https://www.onmo.com/trending/pc-gaming-vs-mobile-gaming-which-is-the-best-for-gaming;/

[14] [online]. Copyright ©S [cit. 12.05.2023]. Dostupné z: https://qph.cf2.quoracdn.net/main-qimg-d3cf98ee4a765f53e879470c2dde82e5

[15] [online]. Dostupné z: https://qph.cf2.quoracdn.net/main-qimg-e1e148ec9922ae4b371ca29de4cc50be-pjlq

[16] Unity vs Unreal: Which Game Engine Should You Choose?. Find the best online Programming courses and Tutorials - Hackr.io [online]. Dostupné z: https://hackr.io/blog/unity-vs-unreal-engine

[17] Unity vs Cryengine: 9 Point super comparison - VionixStudio. How to make Your own Game for free with Unity - VionixStudio [online]. Copyright © 2023 VionixStudio [cit. 12.05.2023]. Dostupné z: https://vionixstudio.com/2020/01/17/unity-vs-cryengine-game-engine-comparison/

[18] WELLS, Robert. Unity 2020 By Example. 3. Birmingham, UK: Packt, 2021. ISBN 9781800203389.

[19] Unity - Manual: Optimizing Scripts. [online]. Copyright © 2020 Unity Technologies. Publication Date [cit. 12.05.2023]. Dostupné z: https://docs.unity3d.com/2019.3/Documentation/Manual/MobileOptimizationPracticalScriptingOptimizations.html

[20] Optimize Your Games In Unity – The Ultimate Guide. Homepage - Anyone Can Learn To Make Games [online]. Copyright © 2023 [cit. 12.05.2023]. Dostupné z: https://awesometuts.com/blog/optimize-unity-game/#:~:text=Optimize%20Your%20Games%20In%20Unity%20%E2%80%93%20The%20Ultimate,...%203%20Be%20Careful%20With%20GameObject.Find%20Functions%20

[21] https://cgcookie.com/posts/maximizing-your-unity-games-performance

[22] How to optimize your Unity mobile game performance in four easy steps. AZUR GAMES - Play, Publish, Work, Grow with Us! [online]. Copyright © [cit. 12.05.2023]. Dostupné z: https://azurgames.com/blog/how-to-optimize-your-unity-mobile-game-performance-in-four-easy-steps/

[23] Optimizing Unity UI - Unity Learn. Learn game development w/ Unity | Courses & tutorials in game design, VR, AR, & Real-time 3D | Unity Learn [online]. Copyright © 2023 Unity Technologies [cit. 12.05.2023]. Dostupné z: https://learn.unity.com/tutorial/optimizing-unity-ui#

[24] Unity - Manual: CPU Usage Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/ProfilerCPU.html

[25] [online]. Copyright ©B [cit. 07.05.2023]. Dostupné z: https://docs.unity3d.com/uploads/Main/gi-chart.png

[26] Unity - Manual: Asset Loading Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/profiler-asset-loading-module.html
[27] Unity - Manual: File Access Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/profiler-file-access-module.html
[28] [online]. Dostupné z: https://docs.unity3d.com/uploads/Main/profiler-file-access.png
[29] Unity - Manual: Audio Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/ProfilerAudio.html
[30] [online]. Copyright © [cit. 07.05.2023]. Dostupné z: https://docs.unity3d.com/uploads/Main/audio-profiler-module.png
[31] Unity - Manual: Global Illumination Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/ProfilerGI.html
[32] [online]. Dostupné z: https://docs.unity3d.com/uploads/Main/profiler-asset-loading.png
[33] Unity - Manual: GPU Usage Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/ProfilerGPU.html
[34] [online]. Copyright © [cit. 07.05.2023]. Dostupné z: https://docs.unity3d.com/uploads/Main/gpu-profiler-module.png
[35] Unity - Manual: Memory Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/ProfilerMemory.html
[36] https://docs.unity3d.com/uploads/Main/profiler-memory-simple-view.png
[37] Unity - Manual: Physics Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/ProfilerPhysics.html
[38] BARON, David. Game Development Patterns with Unity 2021. 2. Birmingham, UK: Packt, 2021. ISBN 9781800200814.
[39] [online]. Dostupné z: https://docs.unity3d.com/uploads/Main/profiler-physics-module.png
[40] Unity - Manual: 2D Physics Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/profiler-2d-physics-profiler-module.html
[41] [online]. Dostupné z: https://docs.unity3d.com/uploads/Main/2d-physics-profiler-module.png
[42] Unity - Manual: Rendering Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/ProfilerRendering.html
[43] [online]. Copyright © [cit. 07.05.2023]. Dostupné z: https://docs.unity3d.com/uploads/Main/RenderProfiler.png
[44] SMITH, Matt a Shaun FERNS. Unity 2021 Cookbook. 4. Birmingham, UK: Packt, 2021. ISBN 9781839217616.
[45] Unity - Manual: UI and UI Details Profiler. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/ProfilerUI.html
[46] [online]. Dostupné z: https://docs.unity3d.com/uploads/Main/ui-profiler-module.png
[47] Unity - Manual: Video Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/profiler-video-profiler-module.html
[48] [online]. Dostupné z: https://docs.unity3d.com/uploads/Main/video-profiler-module.png
[49] Unity - Manual: Virtual Texturing Profiler module. [online]. Copyright © 2020 Unity Technologies [cit. 07.05.2023]. Dostupné z: https://docs.unity.cn/Manual/profiler-virtual-texturing-module.html
[50] [online]. Copyright ©S [cit. 07.05.2023]. Dostupné z: https://docs.unity3d.com/uploads/Main/profiler-virtual-texturing.png
[51] GitHub - Razenpok/BreakInfinity.cs: Double replacement for numbers that go over 1e308. GitHub: Let's build from here · GitHub [online]. Copyright © 2023 GitHub, Inc. [cit. 21.05.2023]. Dostupné z: https://github.com/Razenpok/BreakInfinity.cs
[52] Free Stylized Cartoon Boy Character Rigged 3D - TurboSquid 1979778. 3D Models for Professionals :: TurboSquid [online]. Copyright © TurboSquid 2023 [cit. 21.05.2023]. Dostupné z: https://www.turbosquid.com/3d-models/stylized-cartoon-boy-character-rigged-3d-1979778

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AR	Rozšířená realita
CPU	Centrální procesorová jednotka
FPS	Snímky za sekundu
GC	Garbage collector
GI	Global Illumination
GPU	Grafická procesorová jednotka
LOD	Úroveň detailu
PUBG	PlayerUnknown's Battlegrounds (herní název)
Tris	Počet trojúhelníků v 3D modelu
UI	Uživatelské rozhraní
VR	Virtuální realita
2D	Dvourozměrný
3D	Trojrozměrný

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: USB DISK, který obsahuje dva zazipované prototypy hry vytvořené v Unity. Jeden obsahuje neoptimalizované metody a druhý obsahuje tyto metody optimalizované.