Modulární simulátor tankových bitev

Michal Zapletal

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Michal Zapletal**
Osobní číslo:	**A19130**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Modulární simulátor tankových bitev**
Téma práce anglicky:	**Modular Tank Battle Simulator**

Zásady pro vypracování

1. Vypracujte literární rešerši na téma vývoje síťových počítačových her.
2. Vypracujte literární rešerši na téma vývoje moduláních simulátorů.
3. Navrhněte aplikaci pro simulaci tankových bitev, která bude umožňovat její rozšíření pomocí modulů.
4. Vytvořte modulární simulátor tankových bitev.
5. Vytvořte alespoň dva různé ukázkové moduly.
6. Simulátor, moduly a samotný vývoj dostatečně popište.

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

Seznam doporučené literatury:

1. MURRAY, Jeff W. C# game programming cookbook for Unity 3D. Boca Raton: CRC Press, Taylor & Francis Group, [2014], xvii, 440 s. ISBN 978-1-4665-8140-1.
2. OKITA, Alex. Learning C# programming with Unity 3D. Second edition. Boca Raton, FL: CRC Press, Taylor & Francis Group, [2020], xvii, 671 s. ISBN 978-1-138-33681-0.
3. HOLAN, Tomáš. Unity: první seznámení s tvorbou počítačových her. Praha: CZ.NIC, z.s.p.o., 2020, 172 s. CZ.NIC. ISBN 978-80-88168-57-7. Dostupné také z: https://knihy.nic.cz/files/edice/Unity_prvni_seznameni_s_tvorbou_pocitacovych_her.pdf
4. Unity User Manual 2020.3 (LTS) [online]. San Francisco: Unity Technologies, 2021 [cit. 2021-11-29]. Dostupné z: https://docs.unity3d.com/Manual/index.html
5. Getting Started with Netcode [online]. San Francisco: Unity Technologies, 2021 [cit. 2021-11-29]. Dostupné z: https://docs-multiplayer.unity3d.com/docs/getting-started/about/index.html

Vedoucí bakalářské práce: **Ing. Tomáš Vogeltanz, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **3. prosince 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

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

Cílem této práce je vytvořit modulární simulátor tankových bojů, který bude dostatečně jednoduchý, aby jej mohli využívat začínající vývojáři k vytvoření jejich vlastní první hry i bez předchozích znalostí programování, a nenáročný, aby byl přístupný i pro hráče/vývojáře, kteří nemají přístup k nejnovějším počítačům. K tomu je využit herní engine Unity, včetně rozsáhlé Unity Asset store a knihovny pro tvorbu multiplayerových her MLAPI, která bude zabezpečovat připojení přes síť LAN. Simulátor bude kromě jiného schopen simulovat fyziku pohybu tanků, umožňovat stavět vlastní tankové sestavy v herní garáži a bude obsahovat jednoduchý systém na vytváření dalších modulů, včetně ukázkového modulu, pomocí kterého bude jednoduché pochopit, jak systém funguje.

Klíčová slova: Unity, Simulátor, Tanky, MLAPI, LAN, Moduly, C#

ABSTRACT

The main purpose of this thesis is to create a modular tank battle simulator that will be simple enough so that inexperienced game developers can use it to create their own very first game, even without knowing any code language, and lightweight enough so that it can be run on older computers without issues. This is achieved by using the game engine Unity and its vast Unity Asset Store and MLAPI library, which will allow communication via Local Area Network. The Tank Simulator should be able to simulate tank movement physics, allow users to create their own tank assemblies in the in-game garage, and use a simple system to allow users to add their own tank modules thanks to the preview module, which will be simple enough so that users should easily understand how the whole system works.

Keywords: Unity, Simulator, Tanks, MLAPI, LAN, Modules, C#

Chtěl bych poděkovat především Ing. Tomáši Vogeltanzovi, Ph.D. za ochotu a úsilí, které vynaložil během konzultací této bakalářské práce.

Dále bych chtěl poděkovat tvůrcům modulů, které byly použity pro tvorbu této bakalářské práce a tvůrcům online návodů, které mi výrazně usnadnily a urychlily tvorbu práce.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Od roku 1958, kdy vznikla první videohra [1], se lidstvo snaží tuto formu krácení času pozdvihnout na co možná nejvyšší úroveň. Vytváří složitější a náročnější videohry, které mají za cíl uspokojit i ty nejnáročnější hráče, což především v dnešním světě není jednoduchá záležitost.

Podle amerického průzkumu [2] má každá skupina hráčů na videohry jiné požadavky. Někteří vyžadují kvalitní grafické zpracování, někteří požadují ráznou herní dynamiku, ale mezi hlavní a nejdůležitější aspekty hráči řadí jistou úroveň realismu a jsou to právě simulátory, které mají za úkol poskytnout hráčům co nejvyšší realističnost.

Cílem simulace [3] je zasadit hráče do umělého světa, který může do jisté míry ovlivnit svými rozhodnutími. Simulace se používají pro výuku, dynamické výpočty, ale i pro zabavení obyčejného člověka. Jejich oblíbenost u hráčů je dána především ponořením hráče do virtuálního světa, možností hráče interagovat s virtuálním světem kolem sebe a v neposlední řadě také touhou vyhrát, ať už poražením ostatních hráčů nebo splněním všech cílů, které simulátor hráči zadá.

Simulátory bývají zpravidla komplexní, a kromě zábavy nabízí hráčům jistou formu vzdělání, ať už formou simulace reálné fyziky, jako například u simulátoru Dirt Rally [4] nebo formou vzdělání v oblasti historie, jako například u simulátoru Hearts of Iron IV [5]. Oba zmíněné simulátory se kromě poskytování zábavy snaží rozšířit znalosti hráče.

K vytvoření úspěšného simulátoru bude zapotřebí do vývoje zakomponovat všechna tyto důležitá fakta. Simulátor musí být dostatečně realistický, aby vtáhnul uživatele do děje. Dále musí uživateli určit jasný cíl, který se bude uživatel pokoušet splnit a umožní mu hledat a využívat výhody, které mu nabídne okolní svět. Jako poslední bod by měl být simulátor poučný, aby uživatele obohatil o nové zkušenosti a informace.

I. TEORETICKÁ ČÁST

1 GAME ENGINE

Základem každé videohry je tzv. „Game Engine“ [6] nebo někdy též „Game Framework“, což je specializovaný software, který má za úkol zkombinovat všechny aspekty hry do jednoho hotového produktu. Název „Game Engine“ pochází z anglického „Engine“, což je výraz označující motor a Game Engine opravdu takovým motorem je, protože roztáčí a rozhýbává celou videohru.

Game Engine má spoustu funkcí, které zabezpečují správný chod videohry. Kromě jiného zabezpečuje načítání a umístění herních objektů, zprostředkovává herní fyziku, vykresluje grafiku, simuluje osvětlení a umožňuje připojení více počítačů k jedné instanci hry.

V dřívějších dobách si museli vývojáři vytvořit svůj vlastní Game Engine, aby mohli začít vytvářet samotnou videohru, což výrazně omezovalo rychlost vývoje takové hry. V dnešní době je naštěstí k dispozici hned několik dostupných Game Engine, které jsou spravovány a průběžně aktualizovány velkými společnostmi, a které je možné po získání příslušné licence využívat ke tvorbě vlastní hry.

1.1 Unity Game Engine

„Unity Game Engine“, někdy též jenom „Unity“ [7], je produktem firmy Unity Technologies, který vyšel v roce 2005, původně jako nástroj pro vývoj na operačním systému OS X, ale od té doby prošel mnoha vylepšeními a nyní je plně multiplatformní a podporuje mimo jiné platformy jako je Windows, Linux, Android nebo WebGL.

Unity nabízí několik licencí [7]. Základní licence je licence pro osobní použití. Tato licence umožňuje používat a vyvíjet na platformě Unity bezplatně, pokud jednotlivec nebo firma nepřekročí výdělek více než 100 000 dolarů za rok. Velice podobná je licence pro studenty, která navíc nabízí cloudovou diagnostiku v reálném čase a 5 sezení na hodinách Unity Teams Advanced. Unity dále nabízí i některé placené licence, ale při tvorbě této práce je využíváno pouze licence pro osobní použití.

1.1.1 Princip Unity

Základní jednotkou videohry v Unity je tzv. „Scéna“ [8]. Tyto scény obsahují herní objekty, které se opakovaně posunují a vykreslují pomocí kamery, která představuje pohled hráče na simulaci. K tomu, aby mohla kamera cokoliv vykreslit je zapotřebí světlo, podobně jako je tomu ve skutečném světě.

1.1.1.1 Skripty v Unity

K přecházení mezi scénami a vytvářením herní logiky slouží skripty, které se sepisují v jazyce C#. Ke tvorbě skriptů je ve výchozím nastavení pro Windows a macOS využíváno vývojového prostředí Microsoft Visual Studio [7], ale pokud nejsme s tímto vývojovým prostředím spokojeni, Unity nabízí alternativu v podobě Microsoft Visual Studio Code nebo JetBrains Ryder.

1.1.1.2 Vykreslování v Unity

K renderování grafiky lze využít několik „Render Pipelines“ [7]. Jako výchozí vykreslování se používá URP (Universal Render Pipeline), která je optimalizovaná tak, aby mohla běžet i na méně výkonných zařízeních, například na mobilních telefonech. Pro náročnější hry je k dispozici také HDRP (High Definition Render Pipeline), která nabízí podstatně kvalitnější vykreslování, ale vyžaduje silnější výpočetní techniku, nejlépe osobní počítač vybavený dedikovanou grafickou kartou. V případě potřeby Unity nabízí možnost používat vlastní vykreslování, které si může uživatel vytvořit sám nebo jej zakoupit na Unity Asset Store od jiného vývojáře.

1.1.1.3 Fyzika v unity

Pro usnadnění programování nabízí Unity systém simulace fyziky, který má za úkol řešit kolize mezi objekty, zrychlovat a zpomalovat objekty a celkově vytvářet dojem existence reálných fyzikálních sil, tak jak je známe z reálného světa [7]. Unity nabízí několik základních fyzikálních modulů, které si může uživatel zvolit podle typu hry. Jsou to objektově-orientované moduly pro 2D nebo 3D aplikace a datově-orientované, které slouží pro komplexní aplikace, které nabízí více možností. Datově-orientované systémy nejsou součástí základního balíčku Unity, ale dají se dodatečně stáhnout pomocí Unity Asset Store.

1.1.2 Prostředí Unity

Unity nabízí dedikovaný software, Unity3D Editor, který slouží jako vývojářské prostředí pro práci s Unity Game Engine. Prostředí se skládá z několika oken, která lze libovolně přesouvat, zavírat nebo otvírat dodatečná okna. Unity dokonce nabízí možnost si vytvořit vlastní nástroje a okna, která můžeme opakovaně používat během vývoje. Mezi některá důležitá okna patří například Hierarchie, Konzole, Inspektor a Zobrazení scény nebo hry.

1.1.2.1 Hierarchie

Okno hierarchie zobrazuje strukturu právě otevřené scény [6] a vazby, které mezi sebou objekty mají. Objekt v Unity, který se nachází na vrcholu hierarchie, a nemá rodiče, se nazývá „root“ [7]. Všechny ostatní objekty mají právě jednoho rodiče. Každý prvek navíc může mít libovolné množství potomků, které se pohybují spolu s rodičem. Okno hierarchie nám umožňuje jednoduše prohlížet hierarchii objektů a otvírat jednotlivé objekty v okně Inspektor.

1.1.2.2 Inspektor

Inspektor je důležité okno, které ukazuje jednotlivé komponenty zvoleného objektu [9]. Objekty samy o sobě nemají žádnou funkci a pracují jako zásobníky na komponenty [7]. Každý komponent dodává objektu určitou funkci, jako například zdroj světla, generátor dodatečné gravitace nebo kolizní box. Pomocí okna inspektoru lze objekty vypínat a zapínat, nastavovat jejich velikost a polohu ve světě, přidávat a odebírat komponenty nebo přidělovat komponentům „Tag“ nebo „Layer“.

1.1.2.3 Konzole

Pro výpis informací pro vývojáře slouží okno konzole, které generuje hlášení programu v podobě zpráv, varování a chyb. Volání těchto výpisů lze deklarovat v jednotlivých skriptech a slouží jako zpětná vazba vývojáři [9]. S jejich pomocí může vývojář rychle a snadně zjistit v jakém skriptu a na kterém řádku nastala chyba a jednoduše ji opravit.

1.1.2.4 Struktura projektu

K prohlížení jednotlivých souborů lze použít strukturu projektu, která umožňuje přesouvat, otevírat, vytvářet a mazat jednotlivé soubory uložené v adresářové struktuře projektu. Uživatel tak může provádět veškeré akce se soubory přímo v editoru, aniž by musel přepínat do okna prohlížeče disku. Právě struktura projektu je nejvíce užitečná v případě, že chce

uživatel přiřadit soubor k objektu nebo komponentu, kdy nemusí zadávat adresu souboru, ale stačí soubor přetáhnout ze struktury projektu pomocí funkce „drag and drop“.

1.1.2.5 Zobrazení scény

Pomocí okna pro zobrazení scény může vývojář provádět změny ve scéně v reálném čase, ať už se jedná o nastavení pozic objektů nebo přidávání, či odebírání objektů. Dále umožňuje uživateli zapnout, pozastavit nebo přeskakovat zobrazenou scénu, takže uživatel nemusí kompilovat projekt do spustitelného souboru pokaždé, když udělá nějakou menší změnu [7].

1.1.3 Práce v Unity

Kromě základních funkcí nabízí Unity arzenál pokročilých funkcí, které umožňují vývojáři řídit každý detail hry. Jako několik příkladů je potřeba zmínit Prefabs, Events a Asset Bundles.

1.1.3.1 Prefabs

V případě, že vývojář potřebuje znovupoužít jeden ze svých objektů, je možné jej uložit jako tzv. „Prefab“ [10]. Tyto objekty si udrží stejnou strukturu, včetně všech komponent, kterou měly ve scéně. Tento postup může být výhodný v případě, že má vývojář zájem mít svůj objekt více než jednou, protože je možné tyto objekty instancovat, tzn. Vytvářet za běhu programu. Taková instance se po vytvoření přidá do zobrazení scény, přičemž se k názvu instance přidá „(Clone)“, aby bylo vývojáři jasné, že byl objekt instancován. Tyto instance nemusí být vždy stejné.

V případě, že má vývojář zájem instancovat například stromy rostoucí v lese, bude pravděpodobně chtít, aby se od sebe částečně lišily a nevypadaly všechny identicky. K tomu může použít „Prefab Variants“ [7], které umožní vývojáři před vytvořit varianty již hotových Prefabs.

V případě potřeby může vývojář zanořovat jednotlivé Prefabs do sebe dle potřeby [7]. Jestliže chce vývojář například vytvořit postavu sedláka s kosou, vytvoří instanci sedláka, kterému přidělí jako nástroj instanci kosy.

1.1.3.2 Unity Events

V případě, že hra provádí činnost, která může trvat delší dobu a vývojář potřebuje navázat na provedení této činnosti, například načítání dat z databáze, může být výhodné použít Unity Events. Pomocí deklarace veřejné proměnné typu „UnityEvent“ [9] a následnému zavolání funkce „Invoke()“ lze nastavit, které funkce mají být spuštěny poté, co hra dokončí náročnou operaci.

1.1.3.3 Asset Bundles

Občas se může stát, že je potřeba skladovat, přenášet a načítat objekty po startu programu. V takovém případě lze použít „Unity Asset Bundles“, které slouží jako jakýsi archiv, ve kterém jsou uloženy Prefabs [7]. Tento adresář může obsahovat modely, textury, zvuky a dokonce i celé scény. Jednotlivé Asset Bundles na sebe mohou navzájem navazovat, v případě, že pro načtení prvku z jedné Asset Bundle je potřeba soubor ze druhé.

K jejich ukládání a načítání je potřeba sestavit skript, který funguje podobně jako serializování „.xml“ souborů. Napřed je potřeba najít soubory na disku, potom se soubory načtou do paměti pomocí metody LoadAllAssets() a vývojář může dále tyto materiály instancovat, podobně jako obyčejné Prefabs.

Dokumentace Unity doporučuje využívat Asset Bundles pro vytváření a přenášení DLC (DownLoadable Content) [7], ale v rámci tvorby modulárního simulátoru je tato mechanika nezbytná pro správné fungování modulů.

1.2 Alternativy k Unity Game Engine

Kromě Unity Game Engine se nabízí celá řada komerčně dostupných variant. Každý z těchto enginů má řadu výhod, ale i nevýhod oproti Unity Game Engine. Za zmínku zcela jistě stojí Unreal Engine [11], ale i řada dalších, méně známých enginů.

1.2.1 Unreal Engine

Vytvořen společností Epic Games Inc. v roce 1998, Unreal Engine je přímým konkurentem Unity Game Engine [12]. Oba enginy jsou si velice podobné, ale liší se několika zásadními vlastnostmi.

1.2.1.1 Skriptovací jazyk

Na rozdíl od Unity, které používá pro psaní pluginů a skriptů jazyk C#, skripty do Unreal jsou napsány v jazyce C++ a samotný kód enginu kombinuje „Blueprint“ [11], speciální jazyk, které používají produkty firmy Epic Games a jazyk C++, což může být odradit některé začínající programátory. Podle článku Hackr.io [12] je dokonce C# považován za vhodnější jako jazyk pro tvorbu her a v konečné fázi běží Unity rychleji než Unreal.

1.2.1.2 Přístupnost

Unity je považován za uživatelsky příjemnější a jednodušší na použití než Unreal [11], především díky jednoduššímu startu pro úplné nováčky. Tomu napomáhá velké množství návodů, které je možné najít na internetu od ostatních vývojářů nebo oficiálních Unity kurzů.

1.2.1.3 Vykreslování

Co se týče vykreslování, má Unreal nad Unity nespornou výhodu [12]. I když je Unity poměrně výkonná platforma, Unreal je proslulý svým bleskově rychlým vykreslováním [11] a nabízí spoustu efektů, které zaručí, že bude výsledná hra nádherně zpracovaná.

1.2.1.4 Přístup ke zdrojovému kódu frameworku

Unreal engine je „Source available“ [12], tudíž nabízí možnost do jisté míry upravit základní kód frameworku. Oproti tomu Unity nabízí možnost upravení samotného engine pouze po zakoupení příslušné licence.

1.2.1.5 Cena

Posledním velkým rozdílem mezi Unity a Unreal je cena. Zatímco vývoj jak na Unity, tak na Unreal je zdarma, pokud vývojáři stačí osobní licence, Unreal požaduje 5 % ze zisků [11], které bude výsledný produkt vynášet. Což může být v případě velice úspěšné hry podstatně více než jednorázové zakoupení pokročilé licence Unity.

1.2.1.6 Známé projekty

Mezi známé videohry, které byly vytvořeny na Unity Game Engine patří především post-apokalyptická hra o přežití „Rust“, vyvíjená studiem Facepunch a hardcore střílečka z první osoby, zasazená do oblasti zasažené civilní válkou „Escape from Tarkov“ od studia BattleState Games.

Unreal engine využívá například série „Unreal Tournament“, vyvinutá vývojáři Epic Games nebo simulátor moderního konfliktu „Squad“ od studia Offworld Industries.

1.2.1.7 Shrnutí

Unity je tedy zpravidla lepší volbou pro začínající vývojáře [11], kteří mají zájem vyvíjet v jednoduchém prostředí a rozsáhlou komunitou, která nabízí spoustu nástrojů, skriptů a herních objektů, jež lze stáhnout nebo zakoupit na online Asset store.

Oproti tomu Unreal nabízí propracovanější grafické zpracování, které i přesto, že je náročnější na výpočetní výkon, vypadá zpravidla lépe než podobně zpracovaná aplikace na Unity. Unreal je tudíž zpravidla lepší volbou pro větší týmy [11], pracující na velkých, komerčních hrách.

1.2.2 GoDot

GoDot je open-source gamee engine, vydaný v roce 2014. V současné době je to 4. nejoblíbenější game engine [13] a největší obliby se těší u začínajících vývojářů, podle kterých je jeednodušší začít pracovat v GoDotu než v Unity [13].

1.2.2.1 Skriptovací jazyk

Zatímco Unity využívá ve svých pluginech i skriptech jazyk C#, GoDot používá svůj nativní jazyk „GDScript“ [14], který vychází z Pythonu a podle zdroje WhimsyGames by jej měl profesionální vývojář pochopit za méně než jeden den [13]. Navíc GoDot nabízí možnost využívat skripty v jazyce Visual Script, C++ a C#, ale doporučuje se používat nativní jazyk.

1.2.2.2 Přístupnost

Na rozdíl od Unity, které se snaží zaměřit na všechny skupiny vývojářů, GoDot cílí především na začátečníky [14]. Oba zdroje se shodují [13] [14], že GoDot nabízí přehlednější a přístupnější uživatelské rozhraní.

1.2.2.3 Vykreslování

Vzhledem k tomu, že je GoDot poměrně mladý framework a jeho primárním cílem je spíše uvést vývojáře do světa tvorby videoher, nabízí GoDot pouze omezené grafické možnosti. I přes to, že je v tomto směru Unity podstatně výkonnější, pro projekty s nenáročnou grafikou je GoDot dostačující.

1.2.2.4 Cena

GoDot je dostupný zcela zdarma pod licencí MIT [14], takže nabízí podstatně větší svobodu než Unity. Po stažení GoDotu dokonce nemusí vývojář instalovat rozhraní, protože GoDot je k dispozici jako standalone spustitelný soubor [14]. Pro vývojáře, který má zájem si herní engine upravit podle své potřeby, je proto GoDot optimální.

1.2.2.5 Shrnutí

Pro začínajícího vývojáře je GoDot pravděpodobně lepší volbou [14], protože nabízí nástroje přímo cílené na nezkušeného uživatele. Pro vyvíjení většího projektu bude lepší Unity, které poskytuje více možností a stabilnější grafický aparát, za cenu případného zakoupení licence, pokud bude výsledný projekt výnosný.

2 KNIHOVNA PRO TVORBU MULTIPLAYEROVÝCH HER

I když Unity nabízí integrovanou knihovnu pro zprostředkování komunikace klientů se serverem „UNet“ (Unity Networking), není toto řešení příliš výkonné. Naštěstí Unity nabízí několik alternativ, které může vývojář použít pro vývoj online her.

2.1 Unity MLAPI

Akronym MLAPI znamená „Mid-Level Application Programming Interface“ a je to jakousi zlatou střední cestou mezi Low-Level API a High-Level API [15]. Zatímco se Low-Level API stará o serializaci dat, komunikace mezi klienty a servery a podobné, pro vývojáře náročné práce, High-Level API nabízí abstrakci nad těmito procesy a umožňuje vývojáři vytvářet jeho projekt. Bohužel tato výhoda má svou cenu a tou je podstatně vyšší náročnost High-Level API, která může zásadně zpomalit připojení. Jako řešením tohoto problému vzniklo MLAPI, které má za úkol vytěžit to nejlepší z obou světů a zajistit stabilní a rychlé připojení, zatímco usnadňuje práci vývojáře, který má pocit, jakoby pracoval s High-Level API.

2.1.1 UNet

Unity MLAPI neboli Unity Multiplayer Networking, je novým, oficiálním balíčkem Unity, který má za úkol zprostředkovávat připojení hry přes internet [16]. Pomalu, ale jistě začíná nahrazovat starý UNet, který byl donedávna oficiální multiplayer knihovnou Unity.

UNet se skládá ze dvou součástí, z Multiplayer Services, což jsou servery Unity, které se starají o úkoly jako například Matchmaking a HLAPI/LLAPI, což je systém, pomocí kterého vývojář píše svůj kód [17].

Systém UNet má spoustu omezení. Především HLAPI je považováno za nespolehlivé [17] a Multiplayer Services neumožňují hostovat hry jako autoritativní server [17]. Pokud má vývojář potřebu využít této funkce ve své hře, musí si celý systém naprogramovat sám.

Z těchto důvodů se firma Unity Technologies rozhodla, že celý systém UNet zavrhne a vytvoří nový, dokonalejší systém MLAPI, který právě vzniká.

2.1.2 Princip MLAPI

Aby mohla jakákoliv funkcionalita MLAPI správně fungovat, jsou zapotřebí dva základní koncepty. Je to komponenta „Network Object“ a třída „NetworkBehaviour“ [16]. Na tyto

koncepty dále navazují složitější systémy, jako jsou „Remote Procedure Calls“, „Network Manager“ nebo „Object Spawning Systém“.

2.1.2.1 Network Object

V případě, že je potřeba synchronizovat objekt přes síť, je potřeba napřed tento objekt správně označit, a to komponentou Network Object. Takto označený objekt bude po „Spawnu“ na serveru vytvořen i ostatním klientům [16] tak, aby měl každý klient svou verzi objektu. Každý takto vytvořený objekt má své jedinečné „NetworkId“, podle kterého jde jednoznačně poznat, o který prvek se jedná.

Každý připojený klient má své ClientId a může vlastnit libovolný počet objektů. Vyjímku tvoří tzv. „Player Object“, které představují přímo postavu klienta, kdy každý klient má právě jeden Player Object [16]. Pokud je potřeba pro klienta vytvořit nový Player Object, ze starého Player Objectu se stane obyčejný Network Object, který stále vlastní stejný klient.

Vlastnictví objektů se může dynamicky měnit dle potřeby a klienti můžou odevzdávat vlastnictví svých objektů serveru, který je může přerozdělovat jiným klientům [16].

2.1.2.2 NetworkBehaviour

Abstraktní třída NetworkBehaviour dědí ze třídy MonoBehaviour, která je součástí standartního Unity a měla by to být výchozí třída, ze které dědí všechny skripty, které mají co do činění s komunikací přes internet [16].

Třídy dědící z NetworkBehaviour mohou obsahovat Remote Procedure Calls a tzv. Network Variables, které se na rozdíl od obyčejných proměnných sdílí ostatním klientům.

Tyto proměnné mají dvě podoby. První z nich je Network List, který představuje rozšíření objektu List z .NET [16]. Druhým je Network Variable, který může obsahovat jednoduché datové typy jako je typ float nebo int [16]. U obou jde nastavit typ synchronizace podle toho, jestli vývojář chce synchronizovat klienta se serverem, server s klientem nebo obojí.

2.1.2.3 RPCs (Remote Procedure Calls)

Ke správnému synchronizování hry mezi klienty a server slouží RPCs. MLAPI rozlišuje dva základní typy RPCs a to Server RPC a ClientRPC.

2.1.2.3.1 Server RPC

Pokud chce klient provést akci, kterou je potřeba synchronizovat s ostatními klienty, neprovádí akci sám, ze své pozice, ale volá Server RPC [16], kterým požádá server o provedení akce ze strany serveru. Ke správnému fungování volání je zapotřebí pojmenovat funkci tak, aby její název končil „*ServerRpc()“, a aby jí byl přidělen atribut „[Server Rpc]“.

2.1.2.3.2 Client RPC

Opakem Server RPCs jsou Client RPCs, které na rozdíl od klienta volá server, který chce informovat ostatní klienty o změně na serveru [16]. Podobně jako ServerRPC je potřeba metodu správně nazvat. Její název musí končit „*ClientRpc()“ a musí mít nastaven atribut „[Client Rpc]“.

V případě, že je potřeba ze serveru zavolat jenom určitého klienta [16], je možnost využít parametru „ClientRpcSendParameters“, který specifikuje cílového klienta a odešle volání pouze jemu.

2.1.2.4 Network Manager

Network Manager je komponent, který má úkol starat se o veškeré nastavení, které má co do činění s komunikací přes síť [16]. Obsahuje seznamy registrovaných scén a objektů, které je dovoleno synchronizovat přes síť. Pomocí tohoto komponentu probíhá připojování a odpojování klientů od serveru.

2.1.2.5 Object Spawning

Pokud v Unity vytvoříme objekt pomocí výše zmíněné funkce Instantiate(), vznikne objekt, který bude existovat pouze na straně klienta, který jej vytvořil [16]. K tomu aby mohl objekt vzniknout globálně, je potřeba nad takto instanciovaným objektem zavolat funkci Spawn(), jejíž definici obsahuje komponent Network Object. V případě že objekt nemá komponent Network Object, nelze jej globálně vytvořit.

V případě, že je zapotřebí objekt přidělit jednomu z klientů, může být výhodné použít funkci SpawnWithOwnership(), kde musíme navíc určit NetworkId klienta, kterému má nově vytvořený objekt patřit.

2.2 Alternativy MLAPI

Kromě oficiálního MLAPI jsou dostupné další knihovny, vytvořen komunitními vývojáři. Tyto knihovny nejsou oficiálně podporovány, i když je jejich zpracování na podobné úrovni jako MLAPI [17].

2.2.1 Photon

Photon Engine je framework od firmy Exit Games, který slouží ke tvorbě multiplayerových her, který dokáže úzce spolupracovat s Unity [18]. Mezi produkty Photonu patří „Realtime Networking engine“, který má za cíl snížit odezvu během hraní online her, nebo Photon Unity Networking, zkráceně PUN, který výrazně zlepšuje zastaralý UNet.

I přestože je toto řešení efektivní, Photon nenabízí licenci zdarma a nejlevnější licence stojí 95 amerických dolarů na 5 let, přičemž je výrazně limitován objem dat, který může vývojář využít [18].

2.2.2 Mirror

Další neoficiální multiplayer knihovnou pro Unity je „Mirror“, který je primárně vytvořen pro malá vývojářská studia, především pro MMORPG hry. Na rozdíl od MLAPI, Mirror funguje jako High-Level API [19], díky čemu nabízí řadu výhod, ale i nevýhod.

Zatímco MLAPI se snaží najít střední cestu mezi náročností a výkonem, Mirror je optimalizován tak, aby umožnil vývojáři co nejjednodušší práci s knihovnou [19], což má za následek horší optimalizaci samotné hry a vyšší odezvu.

Mirror nabízí zajímavý pohled na problematiku serveru a klienta a tvrdí, že server a klient je jen jeden projekt [19] a pro oba je použit stejný kód. To podstatně snižuje velikost projektu a čas, který musí vývojář vynaložit, aby projekt sestavil.

2.2.3 Shrnutí

Pro větší studio, které již má základní finanční zázemí, se nabízí jako nejlepší možnost produkty Photon, které jsou výkonnější než ostatní alternativy, ale pro začínající vývojáře nemusí být tento framework nejvhodnější.

Mirror je oproti MLAPI možná jednodušší, ale nenabízí takové možnosti jako MLAPI, a především není oficiálně podporován Unity, což může mít vliv na budoucnost tohoto řešení,

především teď, kdy vzniká zcela nová a Unity podporovaná knihovna MLAPI, která je stejně jako Mirror přístupná vývojářům bezplatně.

3 EXISTUJÍCÍ TANKOVÉ SIMULÁTORY

V praxi existuje hned několik zavedených simulátorů, které jsou přístupné veřejnosti. Každý se liší zpracováním a funkcionalitou. Některé simulátory se striktně snaží dosáhnout co nejvyšší realističnosti, zatímco některé omezují realističnost pro jednodušší hratelnost. Každý simulátor proto má své výhody i nevýhody.

3.1 World of Tanks

Asi největším a nejznámějším tankovým simulátor současnosti je World od Tanks, vytvořené firmou WarGaming v roce 2010 [20].

3.1.1 Princip World of Tanks

World of Tanks kombinuje tankový simulátor s prvky MMORPG a tudíž není čistě realistický. Zatímco čisté simulátory mají za cíl umístit hráče coby člena tankové posádky do neohrabaného bojového vozidla, ve kterém je zapotřebí spolupráce s ostatními členy posádky, aby tank mohl plnit své bojové nasazení, World of Tank tento proces zjednodušuje tak, aby byla hra pro hráče pohodlnější a více přístupná [20].

3.1.1.1 Systém ovládání tanků

Zatímco reálná posádka tanku se skládá z několika různých vojáků, přičemž každý má svou funkci a své přidělené úlohy, tanky ve WoT řídí pouze jeden hráč. Ke zjednodušení tohoto procesu slouží několik funkcí.

3.1.1.1.1 Funkce řidiče

Řízení reálného tanku je poměrně komplikovaná záležitost. Samotné ovládací prvky se liší tank od tanku a posádka vycvičená pro provoz jednoho modelu tanku bude mít s největší pravděpodobností problém s obsluhou jiného modelu. Navíc samotné řízení je velice náročný proces, protože řidič tanku má zpravidla omezený úhel pohledu, kvůli pancéřování tanku a musí se často spoléhat na instrukce velitele, který má ze svého stanoviště lepší výhled.

Ke zjednodušení těchto problémů využívá WoT řadu mechanik, které mají za cíl tento proces zjednodušit. Asi nejdůležitější mechanika je pohled ze třetí osoby, pomocí kterého hráč sleduje své vozidlo pomocí otočné kamery, která rotuje kolem tanku [20]. Díky této kameře má hráč dobrý rozhled a je schopný vidět i do úhlů, které by zevnitř tanku byly slepé.

Dalším usnadněním pro hráče je automatická převodovka, která sama přepíná na vyšší stupeň, což umožňuje hráči ovládat pouze rychlost vozidla pomocí kláves W, dopředu a S, dozadu.

Neméně důležité je i přepínání hnacího ústrojí. Zatímco zatáčení v reálném tanku je poměrně složitý proces, který vyžaduje snížit rychlost pásu na straně, na kterou chce řidič zatočit, pomocí změti pedálů a pák, ve WoT je tento proces značně zjednodušen a k zatáčení slouží tlačítka A, doleva a D, doprava, která umožňují, že se celý tank dá pohodlně řídit jednou rukou pomocí 4 kláves na klávesnici [20].

3.1.1.1.2 Funkce střelce

Podobně jako funkce řidiče je i funkce střelce výrazně zjednodušená. Zatímco pro střelbu z děla je potřeba základní znalost balistiky, aby bylo možné zasahovat nepřátelská vozidla na větší vzdálenosti, ve WoT je namísto toho využito náhodného generátoru.

Každé dělo ve WoT má svou hodnotu počáteční úsťové rychlosti a koeficient přesnosti [20]. Některé typy munice dokáží hodnotu úsťové rychlosti snížit nebo zvýšit. Pro zaměřování slouží záměrný kruh, který představuje úroveň přesnosti děla a ukazuje hráči, kam může dělo vystřelit. Na rozdíl od pravého kanonu tudíž nemusí výstřel letět tam, kam kanon míří, ale může zasáhnout úplně jiné místo, než měl střelec v plánu.

Pro úspěšné zasažení nepřátelského tanku je tedy potřeba zmenšit záměrný kruh na co nejmenší průměr [20]. Toho lze docílit zastavením vozidla, namířením děla na cíl a čekáním po určitou dobu, dokud se kruh nezmenší na nejmenší možný průměr, což je doprovázeno zvukovým znamením, které má hráči oznámit, že je dělo připraveno k výstřelu.

3.1.1.1.3 Funkce velitele

Velitel reálného tanku má za úkol vést posádku, hledat a určovat cíle pro střelce, komunikovat s ostatními tanky a velením, určovat trasu pro řidiče a řešit krizové situace, jako je například zjišťování škod po zásahu tanku. Ve WoT jsou všechny tyto funkce vypuštěny a obstarává je uživatelské rozhraní [20].

Každý tank ve WoT má koeficient maskování, který určuje jak je náročné toto vozidlo detekovat. Tento koeficient se snižuje pokud je vozidlo v pohybu a pokud vozidlo vystřelí. Dále má každé vozidlo parametr dohledu, který určuje na jakou maximální vzdálenost může odhalit nepřátelská vozidla. V případě, že je vozidlo v dohledu nepřátelského vozidla a

koeficient maskování nestačí, aby porazil hodnotu nepřátelského dohledu, je vozidlo odhaleno pro celý nepřátelský tým.

3.1.1.2 Zásahové body

V reálném boji není jednoduché poznat, zda bylo nepřátelské vozidlo zničeno. Posádka obvykle zůstává ve vozidle do poslední chvíle, aby byla krytá před palbou ručních zbraní nepřítele, protože tanky jsou většinou odolné a dokáží odrazit i více kritických zásahů. Posádka zpravidla opouští tank, pokud je nepojízdný a není schopen střelby, ať už z důvodu nedostatku munice nebo poničení ovládacích prvků děla a pokud tank začne hořet a posádka jej není schopná zevnitř uhasit. V případě, že je zasažena munice tanku, může dojít k její explozi, která zpravidla zabije posádku a zásadním způsobem poničí tank.

World of Tanks řeší tuto problematiku uvedením tzv. „Zásahových bodů“, které určují výdrž vozidla. Pokud tyto zásahové body klesnou na nulu, je vozidlo zničeno a hráč nad ním ztrácí kontrolu [20]. V případě zasažení munice může dojít k jejímu zničení, což automaticky sníží počet zásahových bodů na nulu.

Každý zásah vozidla je napřed vyhodnocen počítačem, přičemž se porovnává síla pancíře na místě zásahu, síla průbojnosti střely a úhel, pod kterým byl pancíř zasažen. V případě, že je síla průbojnosti vyšší, pancíř tanku byl proražen a počítač vygeneruje počet bodů poškození, který je odečten od počtu zásahových bodů tanku.

3.1.2 Modularita World of Tanks

I když World of Tanks nenabízí možnost přidávat vlastní vozidla, umožňuje uživateli do jisté míry skládat tankové sestavy. Každý tank se skládá z podvozku, věže, děla, motoru a radiostanice, přičemž vzhled tanku upravují pouze změny věže a děla.

Tento systém má za úkol spíše prohloubit MMORPG aspekt simulátoru než nabídnout uživateli možnost si svůj tank upravit dle svých potřeb, protože většina modulů slouží jako přechodná výzbroj, která je následně nahrazena výkonnější výzbrojí.

3.1.3 Shrnutí

Jako videohra určená pro široké publikum je WoT chytře řešená a nabízí nenáročným uživatelům zábavné řešení tankového boje. Jako simulátor je WoT příliš zjednodušený a omezený nemožností přidávat vlastní moduly, tudíž pro vývojáře prakticky nepoužitelný.

3.2 WarThunder

Vytvořen v roce 2012 firmou Gaijin Entertainment, WarThunder je původně letecký simulátor, který byl v průběhu let rozšířen o tankové, námořní a vrtulníkové souboje [21]. Považuje se za přímého konkurenta World of Tanks, což bylo více než patrné na některých reklamách, které měly ponižovat jisté mechaniky hry World of Tanks.

3.2.1 World of Tanks vs. WarThunder

Podobně jako WoT je WT simulátor s prvky MMORPG, ale liší se především reálněji zpracovanými mechanikami a širší nabídkou možností.

3.2.1.1 Modularita

WT na rozdíl od WoT umožňuje uživatelům vytvářet a vkládat do hry vlastní moduly, ať už v podobě herních map nebo vlastních vozidel. K tomuto procesu slouží oficiální WarThunder SDK [21]. Bohužel, i když je tento systém v praxi už delší dobu, WarThunder stále nepodporuje připojení více hráčů k takto upraveným serverům, tudíž hráči vytvořené moduly jsou přístupné pouze pro jednoho hráče.

3.2.1.2 Zásahové body

WarThunder se pyšní systémem tankových součástek, které se od zásahových bodů World of Tanks zásadně liší [21]. Tanky ve WT nemají souhrnný počet zásahových bodů jako WoT, ale jsou rozděleny na tankové součástky, jako je motor, závěr kanonu, munice a podobně, a tankovou posádku. Tyto součástky mají své vlastní zásahové body a dynamicky omezují efektivitu tanku.

Pokud je tank zasažen a dojde k poškození převodovky a zabití velitele, tank ztrácí možnost pohybu a všechny moduly, které ovládá velitel, jako je lafeta protiletadlového kanonu, jsou nedostupné. Tank je zpravidla zničen, pokud v tanku nezůstane více než jeden člen posádky nebo pokud nedojde k výbuchu munice nebo paliva.

3.2.1.3 Systém střelby

Na rozdíl od WoT, WT má vlastní systém balistiky a je na hráči, aby počítal s náměrem kanonu. K tomu slouží dynamická optika, která ukazuje hráči balistiku nabitého náboje a naznačuje mu, jak moc by měl kanon nadmířit.

Střela vždy letí tam, kam je právě namířen kanon, takže pokud hráč rozjede své vozidlo po nerovném terénu, bude střelba velice nepřesná.

3.2.2 Shrnutí

WarThunder nabízí větší poměr realističnosti než World of Tanks, ale stále je zaměřený především na nenáročné hráče. Celý tank je ovladatelný jedním hráčem, a i když je obsluha tanku náročnější než ve World of Tanks, stále je celý systém příliš zjednodušený. WarThunder sice podporuje přidávání vlastních modulů, ale celý systém je silně limitovaný.

3.3 Post Scriptum

Post Scriptum, vytvořené společností Periscope Games v roce 2018 [22], nabízí odlišné řešení než dvě předchozí hry. Zatímco World of Tanks a WarThunder jsou navržené jako tankové simulátory, Post Scriptum je především simulátor boje pěchoty z dob druhé světové války, který je pouze doplněn o simulaci tankového boje. Z toho důvodu nabízí pouze omezené množství vozidel.

3.3.1 Dostupnost

Oproti WT a WoT vznikal Post Scriptum jako mod na simulátor Squad [22], který se postupem času rozrostl tak, že jej vývojáři mohli vydat jako samostatnou hru a na rozdíl od WT a WoT, není Post Scriptum k dispozici hráčům zadarmo. Aby hráč mohl využívat Post Scriptum, musí si simulátor zakoupit na platformě Steam.

3.3.2 Modularita

Nespornou výhodou Post Scriptum je PS Software Development Kit [22], díky kterému může komunita vytvářet vlastní herní modifikace, jako jsou herní mapy, vozidla nebo ruční zbraně. Tanky jsou podobně jako ve WarThunder kompletní celky a není možnost je upravovat jako ve World of Tanks.

3.3.3 Simulace tankových bojů

Co se týče simulace tankového boje, je Post Scriptum překvapivě nejzdařilejší ze všech zmíněných simulátorů. Hráči jsou rozděleni na jednotlivé členy posádky a celé vozidlo je podobně jako ve WT rozděleno na zničitelné součásti. Navíc simulátor nabízí podobně propracovaný systém pancíře jako dvě předchozí hry.

3.3.4 Shrnutí

Jako simulátor je Post Scriptum nejvíce realistický, ale má několik nevýhod, které je potřeba vyzdvihnout. Především není k dispozici zadarmo a začínající vývojář by musel zaplatit, aby dostal přístup ke hře a SDK. Další velkou nevýhodou je jeho primární funkce. Vzhledem k tomu, že je simulátor především navržen jako simulátor pěchoty, nabízí řadu funkcí, které zbytečně komplikují vytváření modulů tanků a pro nezkušeného vývojáře můžou působit nepřehledně.

II. PRAKTICKÁ ČÁST

4 POŽADAVKY NA SIMULÁTOR

Před zahájením programování bývá dobrým zvykem sepsat si požadavky na finální produkt.

4.1 Funkční požadavky

Funkční požadavky představují funkcionality, které by měla aplikace nabídnout uživateli.

4.1.1 Připojení více hráčů

Systém umožnuje více klientům přístup k serveru, ať už lokálně nebo přes síť. K tomu slouží knihovna MLAPI verze 0.1.0, která zastřešuje komunikaci mezi klienty a synchronizaci samotné hry.

4.1.2 Fyzika pohybu tanků

K simulaci fyziky pohybu tanků slouží systém PTS, dostupný z Asset Store, který slouží jako nástroj pro tvorbu tanků v Unity. Tento nástroj je vytvořený pro tvorbu her pro jednoho hráče a pro jeho správné fungování bude potřeba přepsat skripty tak, aby dědily ze třídy NetworkBehaviour, kterou definuje MLAPI.

4.1.3 Rozdělení vozidel na moduly

Každé vozidlo je vytvořeno z několika modulů, které se skládají do hotové sestavy až při připojení na server. Hlavním důvodem proč byla zvolena právě tato možnost, je umožnit hráčům, upravit si svá vozidla podle svých představ přímo v herní garáži. Vozidla se následně skládají pomocí tzv. „Presets“, což jsou serializované soubory .xml, které popisují, z jakých modulů má být finální vozidlo složeno.

4.1.4 Dynamické načítání modulů

Všechny moduly jsou dynamicky načítány během připojení hráčů na server. Tento systém umožňuje snazší přidávání dodatečných modulů, které si mohou hráči sami vytvářet.

4.1.5 Vytváření dodatečných modulů

Uživatelé simulátoru mohou vytvářet vlastní moduly vozidel s pomocí ukázkového modulu, pomocí kterého bude jednoduché vytvořit vlastní modul i bez znalosti Unity nebo jakéhokoliv programovacího jazyku, pomocí interaktivních skriptů systému PTS. Tyto skripty navíc umožňují nastavit vozidlům různé atributy, jako je váha vozidla nebo rychlost otáčení věže.

4.2 Nefunkční požadavky

Nefunkční požadavky představují vlastnosti simulátoru, které jsou důležité pro jeho správný chod a pro jednoduché využívání jeho uživateli.

4.2.1 Náročnost na hardware

Systém by měl běžet i na starších počítačích, které nemají dostatek výpočetního výkonu pro nejnovější hry. Dále by neměl příliš zatěžovat síťové připojení, aby nedocházelo k výpadkům mezi serverem a klienty.

4.2.2 Dostupnost

Simulátor by měl být dostupný uživatelům zadarmo, aby neodrazoval nezkušené vývojáře, kteří by jej mohli potencionálně využít jako platformu pro svou první vytvořenou hru. To znamená, že nesmí být postaven na systémech, jejichž využívání je dlouhodobě zpoplatněno.

4.2.3 Jednoduchost

Jako poslední důležitý bod je jednoduchost použití simulátoru, jak pro samotné hraní, tak pro vytváření dodatečných modulů. Uživateli nesmí přijít práce se simulátorem náročná a nepřehledná.

5 SYSTÉM PTS

Tento systém byl vytvořen uživatelem ChobiGlass a jeho cílem je vytvořit základ pro vývojáře, kteří vytváří vlastní hru, ve které simulují tankové boje. Nabízí systém pro jednoduchou tvorbu vlastních tanků, pomocí interaktivních editor skriptů a simuluje pohyb vozidel za pomocí fyzikálního aparátu Unity.

5.1 Rozdělení systému PTS na moduly

V PTS jsou tanky seskládány dohromady do jednoho velkého Prefab, který je následně dynamicky nainstanciován po zahájení hry. Tento systém je pro tuto práci nevhodný a bylo jej potřeba částečně přepracovat.

5.1.1 Modul korby

Modul korby obsahuje kompletní model vozidla, včetně podvozku a pásů, kromě věže, která je vytvořena jako samostatný modul. Obsahuje skripty pro ovládání vozidla a kamery, kterou lze rotovat kolem tanku nebo ji přichytit k jednomu z bodů na tanku, jako je pohled z průzoru řidiče nebo pohled z velitelské kupole.

Samotné pásy nabízí 3 možnosti simulace pohybu pásů a to buď staticky, za pomoci rotující textury, pomocí fyzikálního aparátu Unity nebo pomocí animace, která je vygenerována pomocí skriptu „Physics Track System“ a má za cíl vytvořit dojem kvalitně zpracované fyziky pásů, aniž by zatěžovala systém jako standartní fyzikální aparát Unity.

Všechny součásti tanků obsahují editor skripty, pomocí kterých jde nastavovat řadu atributů jednotlivých součástek, například počet pojezdových kol v podvozku, váha jednotlivé součástky, třecí síla článků pásu nebo maximální míra stlačení pružin podvozku.

5.1.2 Modul věže

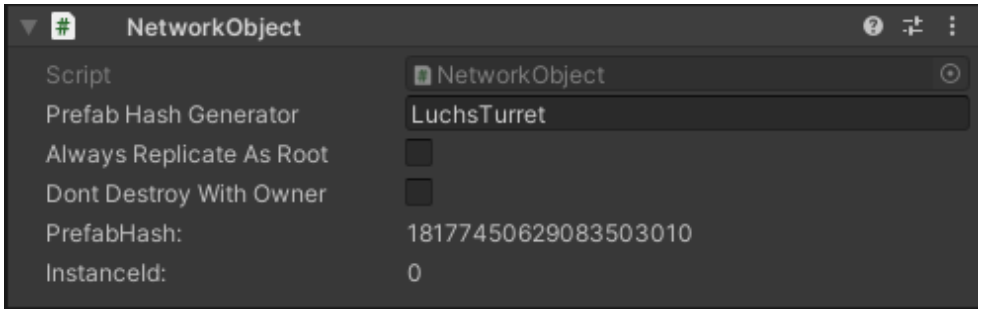
Druhým modulem, ze kterého se skládají tankové sestavy, je modul věže. Tento modul představuje kompletní tankovou věž, včetně výzbroje a obsahuje editor skripty, pomocí kterých je celý modul vytvořen. Dále obsahuje skripty pro ovládání věže a kanonu, a umožňuje uživateli nastavit řadu parametrů, jako je rychlost střelby děla, rychlost otáčení věže, maximální úhel sklopení děla nebo typ střely, která je generovaná při výstřelu.

5.2 Úprava pro MLAPI

Aby tento systém spolehlivě fungoval i pro hru více hráčů, bylo zapotřebí přepsat řadu skriptů, na kterých je celý systém postaven.

5.2.1 Network Object

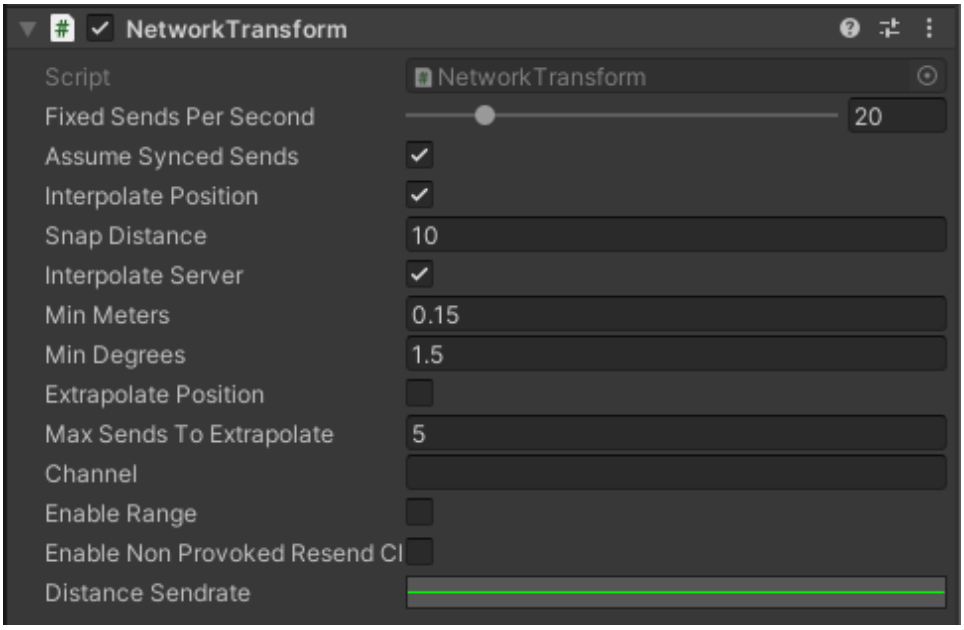
Základem každého modulu je komponent Network Object, který popisuje, jak má být objekt sdílen po síti. Obsahuje unikátní ID, podle kterého jde jednoznačně poznat o který objekt se jedná i na jiných klientech. Dále obsahuje ID svého vlastníka, který má pravomoce provádět operace s objektem, kterému přísluší Network Object. Tento vlastník jde dynamicky měnit a je možnost objekty předávat jiným klientům, pokud to situace vyžaduje. Velice důležitý je i PrefabHash, který je pro správnou funkčnost zapotřebí zaregistrovat do komponentu Network Manager, aby mohl být objekt synchronizován přes síť.



Obrázek 1. Ukázka komponentu NetworkObject

5.2.2 Network Transform

Komponent Network Transform má za úkol pravidelně odesílat polohu svého rodičovského objektu na server, který ji dále přeposílá ostatním klientům, aby byla poloha objektu pro všechny klienty stejná. Jedná se o jeden z prvků vyšší logiky, který MLAPI poskytuje pro uživatele, aby usnadnil programování pohybu objektů. V simulátoru obsahují komponent Network Transform objekty korby, věže a děla, tzn. všechny objekty, které se pohybují.



Obrázek 2. Ukázka komponentu NetworkTransform

5.2.3 Network Behaviour

Pro správnou funkci skriptu ve hře více hráčů je potřeba dědit od třídy Network Behaviour. Tato třída na rozdíl od Mono Behaviour, což je základní třída, ze které dědí skripty, které nekomunikují po síti, obsahuje řadu funkcí, které jsou nezbytné pro vytvoření synchronizované hry pro více hráčů.

Třídy zděděné od třídy Network Behaviour se odkazují na jejich příslušný Network Object, ze kterého zjišťují data, jako je ID vlastníka, a pokud tento objekt nenajdou, nemůžou dál pokračovat a vyhodí vývojáři chybovou hlášku. Dále mohou volat Remote Procedure Calls, kterými žádají server o provedení akce.

```
public class SetCamera : NetworkBehaviour
{
    public GameObject cameraPivot;
    void Start()
    {
        if (NetworkManager.Singleton.IsClient)
        {
            if (transform.parent.gameObject.GetComponent<NetworkObject>().IsOwner)
            {
                cameraPivot.SetActive(true);
            }
        }
    }
```

5.2.4 Remote Procedure Calls

Volání Server RPC umožňuje klientům vykonávat akce z pozice serveru, což je nezbytně nutné v případech, kdy je potřeba instancovat nové objekty, ať už tanky nebo vystřelené střely. Opakem Server RPC je Client RPC, pomocí kterého server posílá zprávu všem klientům. Toho je v simulátoru využíváno pro nastavování hierarchie modulů tanku.

```
[ClientRpc]
 void AssemblyClientRPC(ulong hullID, ulong turretID)
 {
     NetworkObject hullObj = NetworkSpawnManager.SpawnedObjects[hullID];
     NetworkObject turretObj = NetworkSpawnManager.SpawnedObjects[turretID];

     _turretInst = turretObj.gameObject;
     _hullInst = hullObj.gameObject;
     Transform mainbody = hullObj.transform.Find("MainBody");
     _turretInst.transform.SetParent(mainbody);
 }
```

5.2.5 Network Variable

Posledním důležitou funkcí MLAPI jsou Network Variables. Tyto proměnné jsou podobně jako Network Transform pravidelně synchronizovány a odesílají událost o změně ostatním klientům, kteří ji mohou odchytit a provést patřičné změny. V simulátoru jsou Network Variables využívány ve scéně Lobby a pro synchronizaci provizorních zásahových bodů tanků.

```
private NetworkList<LobbyPlayerState> lobbyPlayers = new NetworkList<LobbyPlay-
erState>();
```

6 ASSET BUNDLES – UKLÁDÁNÍ MODULŮ

K ukládání modulů slouží funkce Unity „Asset Bundles“. Jedná se o komprimované soubory, které slouží k přenosu dat, především modelů, textur, zvuků a dalších objemných dat.

6.1 Vytváření Asset Bundles

K vytváření Asset Bundles slouží v simulátoru oficiální editor skript CreateAssetBundles.cs, který přidává možnost do menu „Build AssetBundles“, která po spuštění projde všechny označené Prefabs a poskládá je do komprimovaných souborů, jež následně uloží na disk do složky Streaming Assets.

```
public class CreateAssetBundles
{
    [MenuItem("Assets/Build AssetBundles")]
    static void BuildAllAssetBundles()
    {
        string assetBundleDirectory = "Assets/AssetBundles";
        if (!Directory.Exists(assetBundleDirectory))
        {
            Directory.CreateDirectory(assetBundleDirectory);
        }
        BuildPipeline.BuildAssetBundles(assetBundleDirectory,
                                        BuildAssetBundleOptions.None,
                                        BuildTarget.StandaloneWindows);
    }
}
```

K označení Prefabs slouží dropdown menu v inspektoru, pomocí kterého lze specifikovat název AssetBundle, do které bude Prefab uložen, nebo AssetBundle Variant, kterou můžeme specifikovat různé varianty uvnitř jedné AssetBundle.

Obrázek 3. Označení Prefab jako AssetBundle

Pro správnou funkci v simulátoru musí všechny názvy AssetBundle, které obsahují modul korby, končit *Hull a všechny, které obsahují věž, končit *Turret, jinak jsou systémem při načítání vyřazeny jako neplatné a nedojde k jejich načtení.

6.2 Načítání AssetBundles

K převádění AssetBundles do Prefabs, které lze instancovat v simulátoru, slouží funkce AssetBundle.LoadFromFile() a LoadAllAssets(), které dekomprimují soubor AssetBundle, rozdělí ho na jednotlivé Prefabs, které načtou do paměti. Odkazy na tyto načtené Prefabs jsou následně uloženy do databáze, odkud na ně lze odkazovat při skládání modulů do tankových sestav.

```
string[] files = System.IO.Directory.GetFiles(Application.streamingAssetsPath,
"*.");
        foreach (string file in files)
        {
            var loadedBundle
            = AssetBundle.LoadFromFile(Path.Combine(Application.streamingAssetsPath,
file));

            if (loadedBundle == null)
            {
                Debug.Log("Failed to load AssetBundle!");
                return;
            }

            var assets = loadedBundle.LoadAllAssets<GameObject>();
            foreach (var asset in assets)
            {
                if (asset.name.EndsWith("Hull")) hulls.Add(asset);
                if (asset.name.EndsWith("Turret")) turrets.Add(asset);
            }
        }
```

7 PRESET SYSTÉM

K ukládání tankových sestav slouží tzv. Presets, které jsou serializovány v .xml souborech.

```
<?xml version="1.0"?>
<Preset xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <presetName>Luchsus</presetName>
  <hull>LuchsHull</hull>
  <turret>LuchsTurret</turret>
</Preset>
```

7.1 Třída Preset

K jednodušší práci s Presets byla vytvořena třída, která tyto objekty popisuje. Každý Preset má vlastní jméno, které si zvolí uživatel při ukládání sestavy, název modulu korby a název modulu věže. Všechny 3 proměnné jsou uloženy jako typ „string“.

```
public class Preset
{
    public string presetName;
    public string hull;
    public string turret;
}
```

7.2 Serializace

K serializaci Presets slouží funkce SavePreset(), která používá standartní .NET serializaci pomocí namespace System.Xml.Serialization. Nejprve je vytvořena instance Preset z uživatelem vyplněných údajů. Následně je vytvořena instance XmlSerializer a StreamWriter, a zavolána funkce Serialize(), která serializuje Preset do .xml souboru a uloží jej na disku do složky StreamingAssets/Presets. Po skončení procesu je StreamWriter zavřen.

```
Preset tonk = new Preset();
tonk.presetName = input;

tonk.hull = hullDropdown.options[hullDropdown.value].text;
if (tonk.hull == null) Debug.Log("No hull selected!");

tonk.turret = turretDropdown.options[turretDropdown.value].text;
if (selectedTurret == null) Debug.Log("No turret selected!");

XmlSerializer serializer = new XmlSerializer(typeof(Preset));
if (!Directory.Exists(Application.streamingAssetsPath + "/Presets/")) Di-
rectory.CreateDirectory(Application.streamingAssetsPath + "/Presets/");

StreamWriter writer = new StreamWriter(Application.streamingAssetsPath +
"/Presets/" + tonk.presetName + ".xml");

serializer.Serialize(writer.BaseStream, tonk);
writer.Close();
```

7.3 Deserializace

K deserializaci dochází při připojení klientů ke hře, kdy je zapotřebí vygenerovat seznam, ze kterého si bude klient vybírat sestavu, kterou chce poskládat pro danou hru. Systém prohledá složku StreamingAssets/Presets a pokusí se serializovat všechny .xml soubory, které ve složce najde. Platné Presets potom načte do seznamu, ze kterého si může uživatel vybrat svůj Preset.

```
Preset Deserialize(string path)
    {
        XmlSerializer serializer = new XmlSerializer(typeof(Preset));
        StreamReader reader = new StreamReader(path);
        Preset deserialized = (Preset)serializer.Deserialize(reader.BaseStream);
        reader.Close();
        return deserialized;
    }
```

7.4 Synchronizace Presets

Presets jsou uložené na lokálním disku každého klienta a během celého procesu nejsou nijak synchronizovány s ostatními. Každý klient odesílá serveru pouze název modulu korby a věže, takže každý klient může mít naprosto rozdílné Presets.

8 SCÉNY SIMULÁTORU

Simulátor je tvořen dohromady pěti scénami, přičemž každá má jinou funkci. K jejich vytváření bylo využito komunitního návodu MLAPI od uživatele DapperDino.

8.1 Hlavní menu

Scéna hlavního menu slouží pro jednodušší navigaci simulátorem. Nabízí hráči začít hrát, přejít do herní garáže, kde si může sestavit vlastní Presets, přistoupit k nastavení, které zatím není implementováno a ukončit program pomocí tlačítka Exit.

Obrázek 4. Ukázka pohledu v hlavním menu

O obsluhu tlačítek se stará objekt MenuController, vybaveným jednoduchými skripty, které přepínají scény po stisknutí příslušného tlačítka.

```
public class MenuController : MonoBehaviour
{
    public string _ConnectScene;
    public string _GarageScene;
    public void HostGameDialog()
    {
        SceneManager.LoadScene(_ConnectScene);
    }
    public void GarageDialog()
    {
        SceneManager.LoadScene(_GarageScene);
    }
    public void ExitGame()
    {
        Application.Quit();
    }
}
```

8.2 Garáž

Herní garáž slouží především k vytváření tankových sestav. Umožňuje uživateli přepínat mezi jednotlivými tankovými moduly pomocí dvou dropdown menu, které obsahují seznam načtených koreb a věží. K instanciování tankových modulů slouží objekt Preview, který pracuje spolu s objektem AssetDatabase.

Obrázek 5. Ukázka pohledu v herní garáži

8.2.1 AssetDatabase

V tomto objektu probíhá načítání modulů do paměti, pomocí skriptu Database, viz. Sekce 7.2, načítání AssetBundles, který obsahuje seznam načtených koreb a seznam načtených věží. Po načtení z AssetBundles se tyto listy naplní referencemi na načtené Prefabs, které jsou následně instanciovány pomocí skriptu Preview.

8.2.2 Preview

Tento skript má za úkol obsluhovat UI garáže, serializovat Presets do souborů a zobrazovat uživateli tank, sestavený ze součástek, vybraných uživatelem pomocí dvou Dropdown menu.

8.2.2.1 Zobrazení tankové sestavy

Pro zobrazení tanku uživateli je potřeba vybrat z databáze správné moduly, které jsou následně instanciovány. Jakmile dojde ke změně v jednom z Dropdown menu, je zavolána funkce Preview(), která načte nové moduly

```
public void Preview()
    {
        hullDropdown = GameObject.Find("HullDropdown").GetComponent<Dropdown>();
        turretDropdown = GameObject.Find("TurretDropdown").GetComponent<Dropdown>();

        if (instantiatedHull != null || instantiatedTurret != null) CleanInstan-
      ces();

        selectedHull = dbComponent.hulls.Find(x => x.name == hullDropdown.opti-
      ons[hullDropdown.value].text);
        if(selectedHull == null) Debug.Log("No hull selected!");

        selectedTurret = dbComponent.turrets.Find(x => x.name == turretDropdown.op-
      tions[turretDropdown.value].text);
        if (selectedTurret == null) Debug.Log("No turret selected!");

        NetworkTransform mainbodyNt = selectedHull.GetComponentInChildren<Network-
      Transform>();
        DestroyImmediate(mainbodyNt, true);

        NetworkTransform[] turretNts = selectedTurret.GetComponentsInChil-
      dren<NetworkTransform>();
        foreach (var nt in turretNts)
        {
            DestroyImmediate(nt, true);
        }

        instantiatedHull = Instantiate(selectedHull, new Vector3(0, 0, 0), Quater-
      nion.identity);

        Transform mainbody = instantiatedHull.transform.Find("MainBody");

        Transform turretGO = instantiatedHull.transform.Find("TurretMount");
        Vector3 turretMount = turretGO.position;
        instantiatedTurret = Instantiate(selectedTurret, turretMount, Quater-
      nion.identity, mainbody);
    }
```

8.2.3 Ukládání Presets

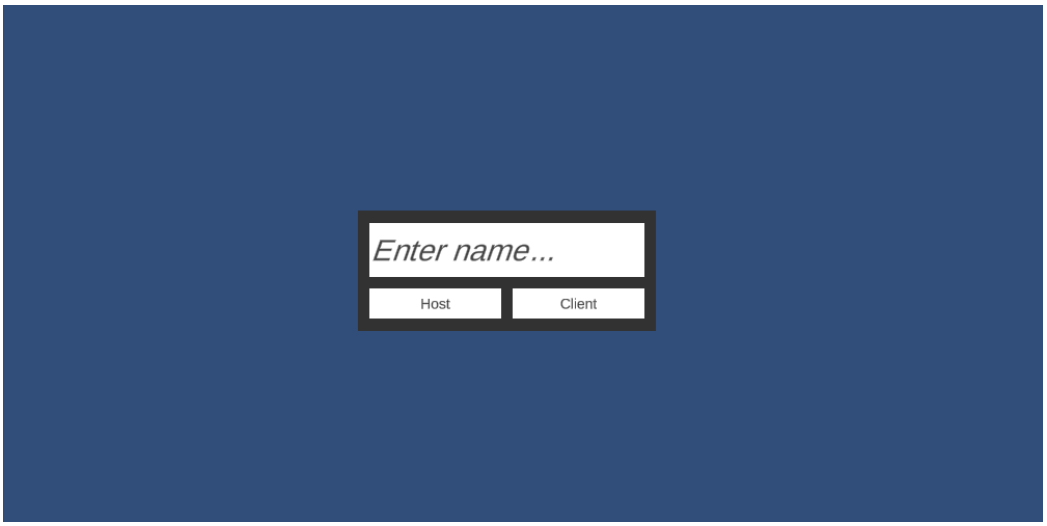
K uložení Preset dojde po stisknutí tlačítka Save Preset. Nejprve proběhne kontrola, jestli byl vložen text do pole Preset Name a pokud ano, proběhne serializace dat do souboru .xml a uživatel bude přesměrován zpět do hlavního menu.

```
public void SavePreset()
    {
        string input = GameObject.Find("PresetInputField").GetComponent<Input-
Field>().text;
        Debug.Log("Text in inputfield: " + input);
        if (input != null){
```

```
            Preset tonk = new Preset();
            tonk.presetName = input;

            tonk.hull = hullDropdown.options[hullDropdown.value].text;
            if (tonk.hull == null) Debug.Log("No hull selected!");

            tonk.turret = turretDropdown.options[turretDropdown.value].text;
            if (selectedTurret == null) Debug.Log("No turret selected!");

            XmlSerializer serializer = new XmlSerializer(typeof(Preset));
            if (!Directory.Exists(Application.streamingAssetsPath + "/Presets/"))
            Directory.CreateDirectory(Application.streamingAssetsPath + "/Pre-
            sets/");

            StreamWriter writer = new StreamWriter(Application.streamingAssetsPath +
            "/Presets/" + tonk.presetName + ".xml");
            serializer.Serialize(writer.BaseStream, tonk);
            writer.Close();
        }
        SceneManager.LoadScene(_MainMenuScene);
    }
```

8.3 Scéna Connect

Mezi hlavním menu a herním Lobby je zasazena scéna ConnectionScene, která představuje výběr serveru. Pro zjednodušení nabízí pouze možnost zahájit hru jako host nebo připojit se ke hře jako klient. Dále se dotazuje hráče na uživatelské jméno, pod kterým bude vyobrazen v Lobby.



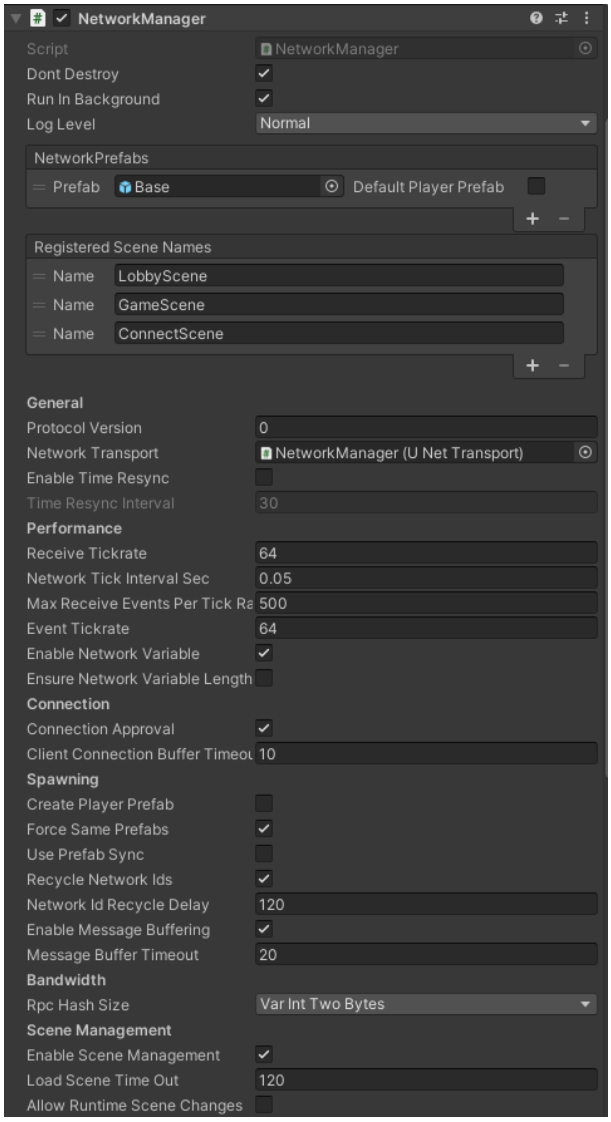
Obrázek 6. Ukázka pohledu ve scéně Connect

8.3.1 AssetDatabase

Podobně jako v garáži, je potřeba inicializovat databázi tankových modulů ještě, než dojde k připojení na server.

8.3.2 NetworkManager

NetworkManager je důležitým objektem, který má za úkol zprostředkovávat komunikaci mezi serverem a klienty. Obsahuje stejnojmenný komponent, který je součástí MLAPI a usnadňuje uživateli přístup k nižším aspektům síťové komunikace. V simulátoru jsou všechny tyto proměnné nastaveny na výchozí hodnotu.



Obrázek 7. Ukázka komponentu NetworkManager

Dále obsahuje seznam registrovaných NetworkPrefabs. Pouze registrované objekty mohou být sdíleny klientům a je tudíž potřeba do tohoto listu přidat všechny moduly, které budou v simulátoru vytvořeny. K tomu slouží skript Register Hashes.

8.3.2.1 Register Hashes

Tento skript je spuštěn poté, co se dokončí inicializace databáze a dojde k provedení události IsDbInitialized(), která oznámí ostatním objektům, že byly všechny moduly úspěšně načteny a mohou k ní přistoupit další skripty.

```
public class RegisterHashes : NetworkBehaviour
{
    private GameObject assetDb;
    public Database dbComponent;
    public NetworkManager nManager;

    public void GetAssetDb()
    {
        {
              nManager = GameObject.Find("NetworkManager").GetComponent<NetworkMa-
              nager>();
            assetDb = GameObject.Find("AssetDatabase");
            dbComponent = (Database)assetDb.GetComponent("Database");

            foreach(var hull in dbComponent.hulls)
            {
                NetworkPrefab np = new NetworkPrefab();
                np.PlayerPrefab = false;
                np.Prefab = hull;
                nManager.NetworkConfig.NetworkPrefabs.Add(np);
            }

            foreach (var turret in dbComponent.turrets)
            {
                NetworkPrefab np = new NetworkPrefab();
                np.PlayerPrefab = false;
                np.Prefab = turret;
                nManager.NetworkConfig.NetworkPrefabs.Add(np);
            }
        }
    }
}
```

8.3.3 NetPortals

NetPortals mají za cíl oddělit logiku serveru od logiky klientů, jsou proto rozděleny na 3 skripty, GameNetPortal, ClientGameNetPortal a ServerGameNetPortal. Objekt NetPortals je nastaven jako DontDestroyOnLoad objekt, tudíž může přecházet mezi povolenými scénami, aniž by došlo k jeho zničení tak, jako to nastává u ostatních Unity objektů při změně scény. Všechny 3 skripty jsou inicializovány jako třídy Singleton.

8.3.3.1 GameNetPortal

Jako výchozí třída, na kterou navazují zbyvající NetPortals, zastřešuje logiku společnou pro server i pro klienty. Obsahuje metody pro zahájení hostování hry, a obsluhuje připojování a odpojování klientů.

8.3.3.2 ClientGameNetPortal

Obsahuje logiku klienta a zastřešuje připojení k serveru jako klient, odpojování klienta od serveru a reakci klienta na změnu scény.

8.3.3.3 ServerGameNetPortal

Definuje maximální počet hráčů připojených k jednomu serveru, pracuje s daty, které klienti odesílají serveru, organizuje přepínání scén a kontroluje klienta po připojení na server pomocí metody ApprovalCheck().

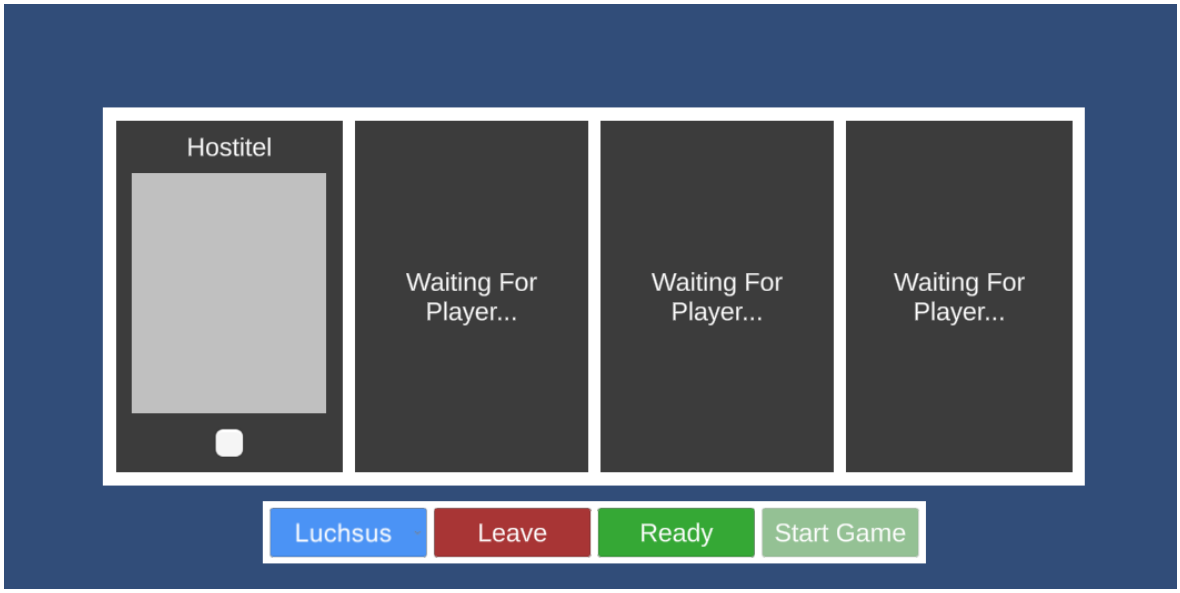
8.3.3.3.1 ApprovalCheck

Tato metoda má za úkol zkontrolovat, jestli je možné umožnit přístup klienta k serveru. Kontroluje se, zda je v lobby místo a jestli hra ještě nezačala. O výsledku informuje klienta pomocí hodnoty ConnectStatus, která popisuje, jak připojení proběhlo.

```
private void ApprovalCheck(byte[] connectionData, ulong clientId, NetworkMa-
nager.ConnectionApprovedDelegate callback)
        {
            if (connectionData.Length > MaxConnectionPayload)
            {
                callback(false, 0, false, null, null);
                return;
            }

            string payload = Encoding.UTF8.GetString(connectionData);
      var connectionPayload = JsonUtility.FromJson<ConnectionPayload>(payload);

            ConnectStatus gameReturnStatus = ConnectStatus.Success;
            if (gameInProgress)
            {
                gameReturnStatus = ConnectStatus.GameInProgress;
            }
            else if (clientData.Count >= maxPlayers)
            {
                gameReturnStatus = ConnectStatus.ServerFull;
            }
            if (gameReturnStatus == ConnectStatus.Success)
            {
                clientSceneMap[clientId] = connectionPayload.clientScene;
                clientIdToGuid[clientId] = connectionPayload.clientGUID;
               clientData[connectionPayload.clientGUID] = new PlayerData(connecti-
              onPayload.playerName, clientId);
            }

            callback(false, 0, true, null, null);
            gameNetPortal.ServerToClientConnectResult(clientId, gameReturnStatus);

            if (gameReturnStatus != ConnectStatus.Success)
            {
                StartCoroutine(WaitToDisconnectClient(clientId, gameReturnStatus));
            }
        }
```

8.4 Lobby

Po úspěšném připojení na server jsou hráči přesunuti do lobby, kde se vyčkává na potvrzení připravenosti od všech hráčů a následně na hostitele, který začíná hru. Každý hráč má přidělenu jednu kartu, která obsahuje jeho uživatelské jméno a ukazatel, jestli je hráč připraven ke hře. Klienti mají k dispozici tlačítko pro změnu připravenosti, tlačítko pro odpojení od serveru a tlačítko pro výběr tankové sestavy, kterou chtějí, aby jim server po zahájení hry sestavil. Hostitel má navíc k dispozici tlačítko pro zahájení hry, které nejde stisknout, pokud v lobby nejsou alespoň 2 hráči anebo pokud někdo z hráčů není připraven.



Obrázek 8. Ukázka pohledu v lobby

Celé lobby je řízeno skriptem Lobby UI, který je dále vybaven komponentem Network Object, který je zapotřebí, aby mohl skript Lobby UI využívat funkcí MLAPI.

8.4.1 Lobby UI

Úkolem tohoto skriptu je řídit přechod mezi scénou lobby a scénou samotné hry. Lobby UI volá procedury ServerRpc, což správně není možné, protože vlastníkem LobbyUI není žádný z klientů, ale samotný server. K tomu aby volání mohlo proběhnout je zapotřebí nastavit atribut RequireOwnership, který je ve výchozím nastavení true, na false, což umožní odesílat volání i přes objekt, který klient nevlastní.

```
[ServerRpc(RequireOwnership = false)]
        private void ToggleReadyServerRpc(ServerRpcParams serverRpcParams = default)
        {
            for (int i = 0; i < lobbyPlayers.Count; i++)
            {
             if (lobbyPlayers[i].ClientId == serverRpcParams.Receive.SenderClientId)
```

```
                {
                    lobbyPlayers[i] = new LobbyPlayerState(
                        lobbyPlayers[i].ClientId,
                        lobbyPlayers[i].PlayerName,
                        !lobbyPlayers[i].IsReady
                    );
                }
            }
        }
```

8.5 Scéna hry

I přesto, že v herní scéně probíhá samotná hra, je tato scéna téměř prázdná. Obsahuje pouze herní plochu, vytvořenou pomocí nástroje MapMagic, která představuje hornatou krajinu, na které je sváděn herní boj, a objekt RoundSystem, který má za úkol vytvořit objekt pro každého hráče, který bude následně pro hráče sestavovat tankovou sestavu.

8.5.1 RoundSystem

Podobně jako Lobby UI má RoundSystem přidělený komponent NetworkObject, aby mohl volat ServerRpc a stejně tak musí být každé toto volání voláno s atributem RequireOwnership = true. Obsahuje dvě metody, které mají za úkol vytvořit pro klienta Prefab s názvem Base, který začne skládat hráčův tank. Pro každého hráče je vybrána jedna počáteční lokace ze seznamu tak, aby se vozidla hráčů nevytvořila na jednom místě.

```
[ServerRpc(RequireOwnership = false)]
    void ClientIsReadyServerRpc(ServerRpcParams serverRpcParams = default)
    {
        if (!loadingClients.Contains(serverRpcParams.Receive.SenderClientId)) {
return; }

        SpawnPlayer(serverRpcParams.Receive.SenderClientId);
        loadingClients.Remove(serverRpcParams.Receive.SenderClientId);
    }

    void SpawnPlayer(ulong clientId)
    {
        int spawnPointindex = Random.Range(0, remainingSpawnPoints.Count);
        Transform spawnPoint = remainingSpawnPoints[spawnPointindex];
        remainingSpawnPoints.RemoveAt(spawnPointindex);
       GameObject playerInstance = Instantiate(playerPrefab, spawnPoint.position,
      spawnPoint.rotation);
        playerInstance.GetComponent<NetworkObject>().SpawnAsPlayerObject(clientId,
null, true);
    }
```

8.6 Prefab Base

Úkolem tohoto Prefab je seskládat tank z modulů, které mu poskytne AssetDatabase. K tomu slouží skript Assembly, který má za úkol najít databázi, vyhledat z ní moduly, které obsahuje Preset zvolený v Lobby a zavolat ServerRpc, která součásti tanku nainstancuje a zavolá všechny klienty pomocí ClientRpc, ve kterém nastaví vozidlu správnou hierarchii, tzn. přidělí instanci věže jako potomka objektu MainBody, který se nachází v instanci korby.

```
[ServerRpc]
    public void AssemblyServerRpc(ulong netID, string hull, string turret)
    {
        Database dbComponent = (Database)assetDb.GetComponent("Database");

        hullPrefab = dbComponent.hulls.Find(x => x.name == hull);
        turretPrefab = dbComponent.turrets.Find(x => x.name == turret);

        GameObject hullInst = Instantiate(hullPrefab, transform.position, Quater-
       nion.identity, transform);
        hullInst.GetComponent<NetworkObject>().SpawnWithOwnership(netID);

        Transform turretGO = hullInst.transform.Find("TurretMount");
        Vector3 turretMount = turretGO.position;
        Transform mainbody = hullInst.transform.Find("MainBody");

        GameObject turretInst = Instantiate(turretPrefab, turretMount, Quater-
       nion.identity, mainbody);
        turretInst.GetComponent<NetworkObject>().SpawnWithOwnership(netID);

        AssemblyClientRPC(hullInst.GetComponent<NetworkObject>().NetworkObjectId,
       turretInst.GetComponent<NetworkObject>().NetworkObjectId);
    }

    [ClientRpc]
    void AssemblyClientRPC(ulong hullID, ulong turretID)
    {
        NetworkObject hullObj = NetworkSpawnManager.SpawnedObjects[hullID];
        NetworkObject turretObj = NetworkSpawnManager.SpawnedObjects[turretID];

        _turretInst = turretObj.gameObject;
        _hullInst = hullObj.gameObject;
        Transform mainbody = hullObj.transform.Find("MainBody");
        _turretInst.transform.SetParent(mainbody);

    }
```

9 TVORBA UKÁZKOVÉHO MODULU

Jako ukázkový modul simulátoru slouží model Německého Panzerspähwagen II Ausf. L. Který byl vytvořen v programu Blender, nabarven v programu Adobe Substance Painter a nakonec implementován do Unity.

Vytváření těchto modulů je časově náročné a trvá přibližně kolem 80 hodin. Nejnáročnější částí modelování druhoválečných tanků je vyhledat k těmto starým vozidlům odpovídající dokumentaci. Většina internetových zdrojů se obvykle shodne na základní charakteristice vozidla a jeho přibližnému tvaru, ale je náročné sehnat přesnější specifikace.

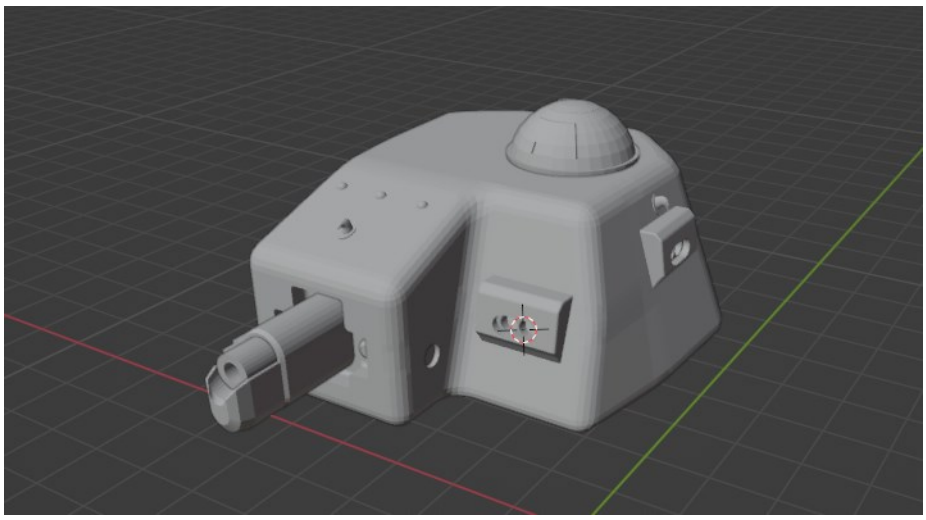
9.1 Tvorba 3D Modelu

Proces modelování začíná výběrem vhodných referenčních obrázků, které slouží jako vodící linky pro tvorbu modelu korby. Tato část je velice důležitá, aby mělo vozidlo správné proporce. Potom co jsou referenční obrázky slazeny na správné jednotné rozměry, začne proces modelování korby.

K tomu je nejlepší zezačátku využít modifikátoru Mirror, který zrcadlí model podle dané osy, v tomto případě je to osa Y. I když to není na první pohled zřejmé, většina tanků není symetrická. Je to dáno mnoha faktory, nejvýznamnějším je zpravidla rozložení posádky, kdy především malé, starší tanky mají posazenou věž na levé nebo pravé straně korby, aby uvolnili místo pracovišti řidiče.

Potom co je vymodelován hrubý tvar korby, začne tvorba podvozku. Tato část je velice časově náročná pro začínající grafiky, protože vyžaduje práci s modifikátory a neustále přeměřování vzdáleností mezi jednotlivými modely. Nejlepší způsob jak tuto část modelovat je skrýt si všechny ostatní prvky a pracovat pouze s jednotlivou součástkou, aby nedošlo k přehlcení obrazovky vodícími body ostatních součástek. Výhodou modelování podvozku je, že stačí vymodelovat každou součástku jednou, protože většina tanků využívá normovaných velikostí kol, aby se usnadnila jejich výroba. Nevýhodou tohoto procesu je, že grafik neví, jak na sebe součástky navazují, dokud nevytvoří kompletní podvozek a neposkládá součástky na jejich správnou pozici.

Když už je podvozek hotový, přichází čas na tvorbu věže, kdy hned narazíme na první problém. Je velice těžké zjistit, jak velký je prstenec věže, pomocí kterého drží věž na korbě, protože málokdy jde z nákresů na první pohled vidět. Věže jsou zpravidla osově souměrné, pokud zrovna nejde o francouzský tank.



Obrázek 9. Věž francouzského tanku

Jakmile je věž dokončena, přichází na řadu detaily, jako jsou světlomety, úložné boxy, průzory, periskopy, výfuk a podobně. Tyto detaily jsou na tanku velice důležité, protože tanky jsou poměrně rozměrné stroje a pokud na sobě tyto detaily nemají, vypadají příliš plasticky a nudně.

Modelování kromě jiného zahrnuje spoustu dalších úprav, které na první pohled nejsou zřejmé, jako je správné směrování normál ploch, zahlazení kulatých ploch, a především tvorba UV map. UV mapy jsou nezbytné pro vytváření textur a z většiny na nich závisí, jak bude textura vypadat. Cílem je zaplnit co největší plochu UV mapy, tak aby měl model detailní texturu, ale přitom textura nezabírala moc místa, což může mimo jiné nepříznivě ovlivnit rychlost načítání simulátoru.

9.2 Tvorba textur

Jak již bylo zmíněno, k vytvoření textur bylo využito programu Adobe Substance Painter, který na rozdíl od Blenderu není k dispozici zadarmo, ale je potřeba zakoupit licenci, což je poměrně pochopitelné, vzhledem k tomu, jak je to silný nástroj. Je to velice výkonný program, který nabízí spoustu před chystaných materiálů a generátorů, které výrazně urychlují práci.

Prvním krokem k vytvoření textury je naimportovat .fbx (Filmbox) model z Blenderu. Následně je potřeba vytvořit mapy, se kterými Substance Painter pracuje. Celý proces je automatický a vyžaduje minimum zásahů od grafika.

Substance Painter pracuje s pěti mapami. Barevnou (Color), Výškovou (Height), Jemnostní (Roughness), Metalickou (Metallic) a Normálovou (Normal). Postupným skládáním

materiálů, ať už před chystanými nebo uživatelem vytvořenými, se zaplní všech 5 map a vznikne textura, která bude následně exportována do .png (Portable Network Graphics) formátu. Substance Painter přímo nabízí textury exportovat do formátu Unity HD Render Pipeline, takže bez delšího nastavování vzniknou textury přímo uzpůsobené pro Unity Engine.

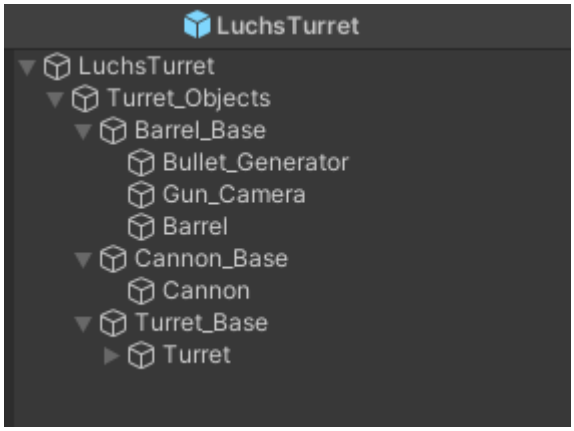
Obrázek 10. Textura věže tanku 40M Turán

9.3 Import do Unity

Pro vytvoření vlastního modulu je zapotřebí model ve formátu .fbx, rozdělený na jednotlivé součástky tanku. K zapotřebí bude model věže, lafety kanonu a hlavně kanonu pro vytvoření věže a model korby, článku pásů, pojezdového, hnacího a vodícího kola a vratné kladky, pokud ji tank má mít. Tanky využívají pro vyhodnocování nárazů a zásahů Mesh Colliders, které by měly být také vytvořeny v blenderu. Každý z těchto modulů by měl mít vytvořenou texturu, kterou je zapotřebí implementovat do Unity a vytvořit z ní Material, který bude přidělen modelu.

9.4 Vytváření modulu věže

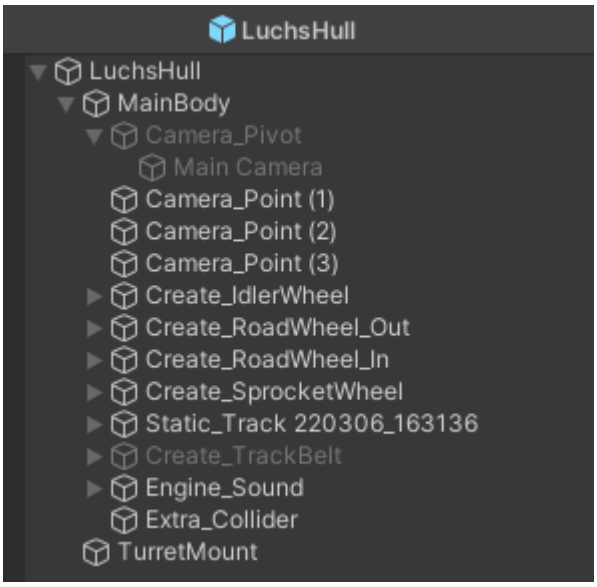
Modul věže je podstatně jednodušší než modul korby a je tudíž lepší volbou pro začínající vývojáře, protože má přehlednou hierarchii a bude mnohem rychleji hotový než složitý modul korby. Skládá se ze tří velkých objektů. Barrel_Base, který definuje hlaveň kanonu, včetně mechanik pro střelbu, Cannon_Base, který definuje lafetu děla, včetně mechanik pro vertikální pohyb děla a Turret_Base, který definuje samotnou věž a mechaniky otáčení věže.



Obrázek 11. Hierarchie modulu věže

9.5 Vytváření modulu korby

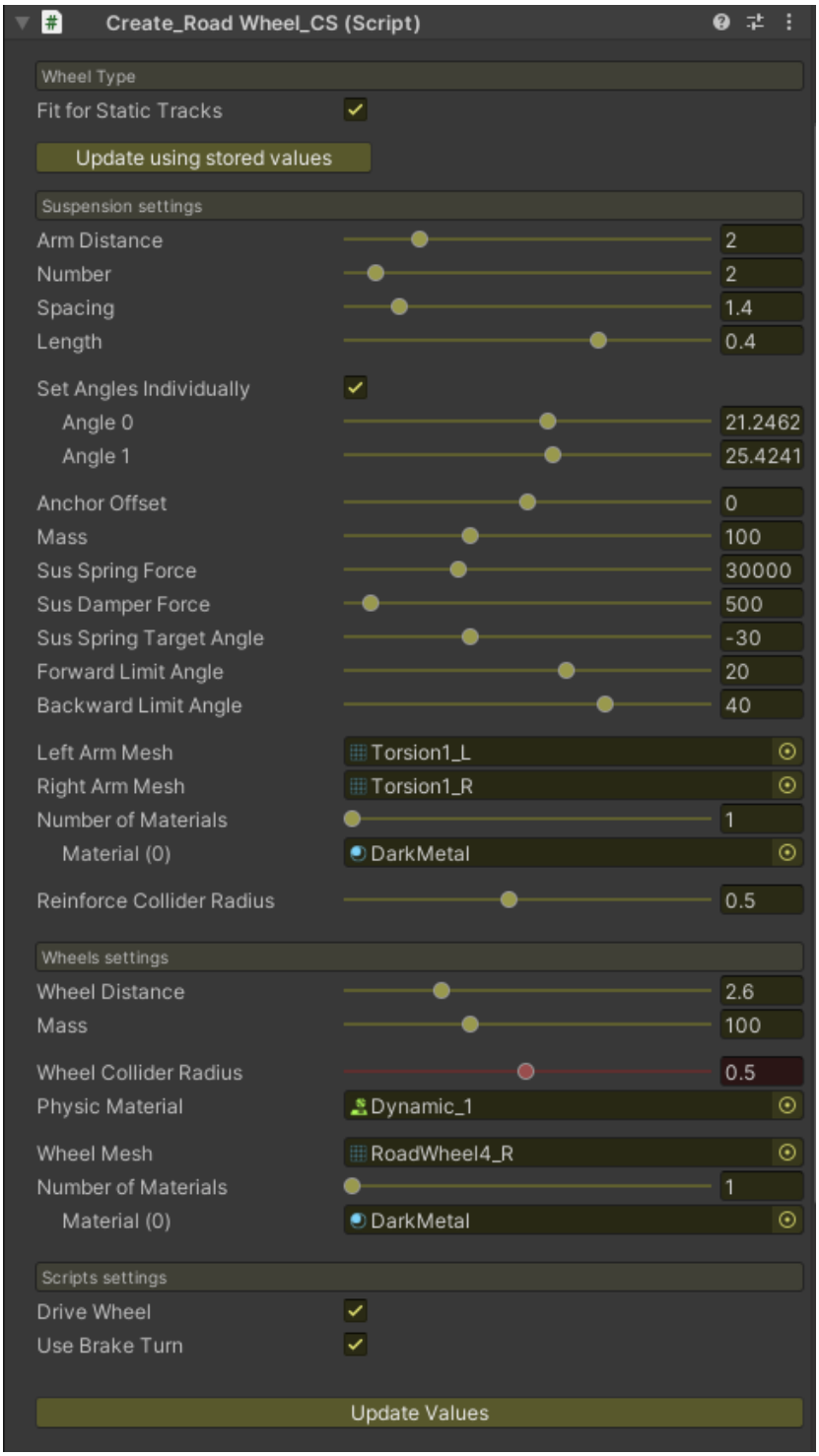
Hierarchie korby je složitější a skládá se z pěti základních editor skriptů, které popisují korbu tanku a jednotlivé díly podvozku, kamery, která rotuje kolem tanku nebo je přichycena na jeden z „Camera_Point“, zvuku motoru, dodatečného collideru, který zjišťuje, jestli bylo vozidlo převráceno a vodícího bodu „TurretMount“, který upřesňuje, na které místo bude k tanku připojen modul věže.



Obrázek 12. Hierarchie modulu korby

9.6 Ukázka editor skriptu

Součástí kompletního tanku je velké množství editor skriptů, které slouží k nastavení jednotlivých součástí vozidla. Všechny jsou podobně zpracovány a umožňují vývojáři nastavovat atributy součástky, aniž by měli jakoukoliv praxi v kódování a tvoření v Unity.



Obrázek 13. Ukázka editor skriptu pro tvorbu pojezdových kol

Skripty dynamicky upravují zobrazení modulu, takže si jej může vývojář kdykoliv v průběhu tvoření prohlédnout, aniž by musel přepínat scénu nebo zapínat program. Hodnoty atributů se nastavují pomocí slideru nebo číselné hodnoty a odkazy na model nebo materiál je zapotřebí přetáhnout ze složky v unity nebo použít „Select menu“ kliknutím na ikonu terčíku napravo od pole s referencí.

ZÁVĚR

V teoretické části byly porovnány typy herních frameworků a jako nejvhodnější byl zvolen game engine Unity, především pro své příjemné uživatelské prostředí a dostatek funkcionalit.

K Unity bylo dále potřeba zvolit networking knihovnu, která má zprostředkovávat komunikaci klientů se serverem a jako nejlepší byla zvolena oficiální knihovna MLAPI, především pro její dostupnost a oficiální status, který slibuje dostatek aktualizací a dlouhodobou podporu.

Poslední sekce teoretické části bylo porovnání existujících simulátorů, které již jsou k dispozici na trhu. Jako nejzajímavější simulátor se ukázal Post Scriptum, který je bohužel limitovaný poplatkem za užívání. Oproti dedikovaným tankovým simulátorům World of Tanks a WarThunder dokonce nabízel nejvíce funkcionalit, o které by mohl mít budoucí moder zájem.

Prvním krokem tvorby samotného simulátoru bylo upravit všechny moduly tak, aby všechny společně fungovaly. Poté bylo zapotřebí vytvořit několik dodatečných mechanik, jako dynamické načítání modulů a serailizace sestav.

Kromě samotné logiky aplikace bylo potřeba sestavit i přehledné uživatelské rozhraní, pomocí kterého lze simulátor používat. Systém byl rozdělen na jednotlivé scény, které mají své určené funkce a obsahují prvky UI, které komunikují s logikou aplikace.

Finálním krokem bylo vytvoření ukázkových modulů, na kterých lze představit funkce simulátoru a zároveň poslouží jako vzor, podle kterého lze jednoduše sestavovat dodatečné moduly.

Tvorba simulátoru byla více než náročná a za jeho úspěšným vytvořením stálo nespočet modulů a řada člověkohodin, které bylo potřeba vynaložit, aby spolu dokázaly jednotlivé moduly bez problémů pracovat.

Simulátor ve své současné podobě může sloužit jako dostatečný základ pro tvorbu vlastních modulů pro začínající vývojáře. V plánu je pokračovat ve vývoji a postupně přidávat dodatečné funkcionality, které by mohly přilákat více uživatelů.

SEZNAM POUŽITÉ LITERATURY

[1] October 1958: Physicist Invents First Video Game. *APS Physics* [online]. College Park: American Physical Society, 2008 [cit. 2022-05-04]. Dostupné z: https://www.aps.org/publications/apsnews/200810/physicshistory.cfm

[2] T A WOOD, Richard. The structural characteristics of video games: a psycho-structural analysis. *Pub Med* [online]. Rockville Pike: National Library of Medicine, 2004 [cit. 2022-05-04]. Dostupné z: https://pubmed.ncbi.nlm.nih.gov/15006163/

[3] Simulations. *Teaching UNSW* [online]. Sydney: Teaching at UNSW, 2018 [cit. 2022-05-04]. Dostupné z: https://www.teaching.unsw.edu.au/simulations

[4] Officiální stránka Dirt Rally. *Dirtgame* [online]. Southam: Codemasters, 2007 [cit. 2022-05-04]. Dostupné z: https://dirtgame.com/dirtrally/uk/about

[5] Officiální stránka HOI4. *Paradox Interactive* [online]. Stockholmu: Paradox Interactive, 2016 [cit. 2022-05-04]. Dostupné z: https://www.paradoxinteractive.com/games/hearts-of-iron-iv/about

[6] MARTIN, Jennifer. What is a Game Engine?. *University of Silicon Valley* [online]. San Jose: University of Silicon Valley, 2020 [cit. 2022-05-04]. Dostupné z: https://usv.edu/blog/what-is-a-game-engine/

[7] Unity User Manual 2020.3 (LTS) [online]. San Francisco: Unity Technologies, 2021 [cit. 2021-11-29]. Dostupné z: https://docs.unity3d.com/Manual/index.html

[8] HOLAN, Tomáš. Unity: první seznámení s tvorbou počítačových her. Praha: CZ.NIC, z.s.p.o., 2020, 172 s. CZ.NIC. ISBN 978-80-88168-57-7. Dostupné také z: https://knihy.nic.cz/files/edice/Unity_prvni_seznameni_s_tvorbou_pocitacovych_her.pdf

[9] MURRAY, Jeff W. C# game programming cookbook for Unity 3D. Boca Raton: CRC Press, Taylor & Francis Group, [2014], xvii, 440 s. ISBN 978-1-4665-8140-1.

[10] KITA, Alex. Learning C# programming with Unity 3D. Second edition. Boca Raton, FL: CRC Press, Taylor & Francis Group, [2020], xvii, 671 s. ISBN 978-1-138-33681-0.

[11] FOGLEMAN, Evelyn. Game engine comparison guide for 2021. *Evercast* [online]. Unknown: Evercast, 2021 [cit. 2022-05-04]. Dostupné z: https://www.evercast.us/blog/unity-vs-unreal-engine

[12] KAUR ARORA, Simran. Unity vs Unreal Engine: Which Game Engine Should You Choose?. *Hackr.io* [online]. Unknown: Hackr.io, 2021 [cit. 2022-05-04]. Dostupné z: https://hackr.io/blog/unity-vs-unreal-engine

[13] KLIUCH, Denys. Godot vs Unity: All You Need to Know. *Whimsy Games* [online]. Unknown: Whimsy Games, 2021 [cit. 2022-05-04]. Dostupné z: https://whimsygames.co/blog/godot-vs-unity-all-you-need-to-know/

[14] WIRTZ, Bryan. Unity vs. Godot: Performance, Community Support, Ease of Use, & Pricing. *Game Designing* [online]. Unknown: Game Designing, 2022 [cit. 2022-05-04]. Dostupné z: https://www.gamedesigning.org/engines/unity-vs-godot/

[15] BONZON, Tim. Unity MLAPI Multiplayer Tutorials – Complete Guide. GameDev Academy [online]. Unknown: Zenva, 2022 [cit. 2022-05-04]. Dostupné z: https://gamedevacademy.org/unity-mlapi-tutorial/

[16] Getting Started with Netcode [online]. San Francisco: Unity Technologies, 2021 [cit. 2021-11-29]. Dostupné z: https://docs-multiplayer.unity3d.com/docs/getting-started/about/index.html

[17] Multiplayer in Unity3D. *Medium* [online]. Unknown: Ben, 2018 [cit. 2022-05-04]. Dostupné z: https://medium.com/@codingpanda/multiplayer-in-unity3d-d292367ebc49

[18] WIRTZ, Bryan. Photon Engine For Multiplayer Games: Try it for Free (What You Need To Know). *Game Designing* [online]. Unknown: Game Designing, 2021 [cit. 2022-05-04]. Dostupné z: https://www.gamedesigning.org/engines/photon-multiplayer/

[19] Mirror Networking Documentation. *Mirror* [online]. Unknown: Mirror, 2022 [cit. 2022-05-04]. Dostupné z: https://mirror-networking.gitbook.io/docs/

[20] World of Tanks: Newcomer's Guide. *World of Tanks* [online]. Unknown: Wargaming, 2022 [cit. 2022-05-04]. Dostupné z: https://worldoftanks.eu/en/content/guide/newcomers-guide/

[21] About the game WarThunder. *WarThunder* [online]. Unknown: Gaijin Entertainment, 2022 [cit. 2022-05-04]. Dostupné z: https://warthunder.com/en/game/about/

[22] What is Post Scriptum?. *Post Scriptum game* [online]. Unknown: Periscope Games, 2022 [cit. 2022-05-04]. Dostupné z: http://postscriptumgame.com/the-game/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
DLC	Downloadable Content
HDRP	High Definition Render Pipeline
HLAPI	High-Level Application Programming Interface
LAN	Local Area Network
LLAPI	Low-Level Application Programming Interface
MIT	Massachussets Institue of Technology
MLAPI	Mid-Level Application Programming Interface
MMORPG	Massive Multiplayer Online Role-Playing Game
PTS	Physics Track System
PUN	Photon Unity Networking
RPC	Remote Procedure Call
SDK	Software Development Kit
UI	User Interface
UNet	Unity Networking
URP	Universal Render Pipeline
WoT	World of Tanks
WT	War Thunder

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

Příloha P1: DVD s elektronickou verzí bakalářské práce a zpracovanou praktickou částí

PŘÍLOHA P1: DVD S PRAKTICKOU ČÁSTÍ PRÁCE

Toto DVD obsahuje:

- Bakalářskou práci M3_Zapletal_BP_2022.pdf
- Build tankového simulátoru UTW_Build.zip
- Projekt simulátoru v Unity UTW_Unity.zip
- Modely a textury využité v simulátoru Assets.zip