Architektury síťové komunikace ve hrách pro více hráčů

Bc. Michael Pluskal

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Michael Pluskal**
Osobní číslo:	**A22595**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Architektury síťové komunikace ve hrách pro více hráčů**
Téma práce anglicky:	**Network Communication Architectures in Multiplayer Games**

Zásady pro vypracování

1. Vypracujte literární rešerši na zadané téma.
2. Analyzujte současné techniky a postupy při návrhu architektury síťové komunikace ve hrách pro více hráčů.
3. Proveďte rozbor hlavních síťových knihoven pro herní engine Unity.
4. Vyberte jednu z knihoven a navrhněte řešení architektury pro existující projekt UTW.
5. Implementujte řešení dle návrhu a proveďte jeho testování.
6. Implementaci a testování vhodně popište.

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

Seznam doporučené literatury:

1. MADHAV, Sanjay a Josh GLAZER. *Multiplayer Game Programming: Architecting Networked Games (Game Design)*. Addison-Wesley Professional, 2015. ISBN 978-0134034300.
2. CHIU, Chihming. *Massively Multiplayer Game Programming With Unity 3d and Mirror: The Ultimate Guide to Building and Hosting Your MMOGS*. Tellwell Talent, 2021. ISBN 978-0228844105.
3. UNITY TECHNOLOGIES. *Unity Multiplayer Networking* [online]. 2023 [cit. 2023-11-12]. Dostupné z: https://docs-multiplayer.unity3d.com/.
4. FIRSTGEARGAMES. *Fish-Net: Networking Evolved* [online]. 2023 [cit. 2023-11-12]. Dostupné z: https://fish-networking.gitbook.io/docs/.
5. MIRRORNETWORKING. *Mirror* [online]. 2023 [cit. 2023-11-12]. Dostupné z: https://mirror-networking.gitbook.io/docs/.

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

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

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

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové 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 diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou 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 diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové 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 diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Bc. Michael Pluskal
podpis studenta

ABSTRAKT

Tato diplomová práce se zaměřuje na analýzu současných technik a postupů při návrhu architektury síťové komunikace ve hrách pro více hráčů a seznamuje čtenáře s dostupnými technologiemi, které tuto komunikaci realizují. Teoretická část práce se zaměřuje na identifikaci klíčových výzev spojených se síťovou komunikací ve hrách pro více hráčů. Dále se tato část zabývá rozborem hlavních síťových knihoven pro herní engine Unity. V praktické části práce proběhne implementace architektury client-to-server na existující projekt UTW za pomoci knihovny Fish-Net: Networking Evolved. Je zde popsán návrh řešení, implementace, testování, i ukázky kódu.

Klíčová slova: UTW, síťová architektura, Unity, Fish-Net, tankový simulátor, multiplayer

ABSTRACT

This thesis focuses on the analysis of current techniques and practices in the design of network communication architecture in multiplayer games and introduces the reader to the available technologies that implement this communication. The theoretical part of the thesis focuses on identifying the key challenges associated with network communication in multiplayer games. Furthermore, this section analyzes the main network libraries for the Unity game engine. In the practical part of the thesis, the implementation of a client-to-server architecture on an existing UTW project using the Fish-Net: Networking Evolved library is discussed. The solution design, implementation, testing, and code samples are described.

Keywords: UTW, computer network architecture, Unity, Fish-Net, tank simulator, multiplayer

Touto cestou bych chtěl poděkovat Ing. Tomáši Vogeltanzovi, Ph.D. za pomoc, čas a cenné rady, které mi věnoval při tvorbě této diplomové práce.

Dále bych rád vyjádřil svou vděčnost uživatelům komunitního Discord serveru „FirstGear-Games / Fish-Networking“, kde jsem získal mnoho užitečných rad od samotných vývojářů této síťové knihovny.

A v neposlední řadě, děkuji mé rodině za jejich neustálou podporu a porozumění během mého studia a tvorby této práce.

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

OBSAH

ÚVOD

S rozvojem technologií a rostoucí popularitou počítačových her pro více hráčů se stává síťová komunikace nezbytným prvkem pro zajištění plynulého a zábavného herního zážitku. Tyto hry, které spojují hráče z celého světa ve společných virtuálních světech, vyžadují sofistikované architektury umožňující synchronizaci akcí hráčů, správu herního stavu a minimalizaci latence. S rostoucí popularitou však přibývají i problémy jako podvádění, které mohou narušit integritu her. Vývojáři se proto intenzivně věnují zdokonalování metod detekce a prevence podvodů, aby zajistili férové prostředí pro všechny hráče a udrželi integritu těchto online komunit.

Tato diplomová práce se soustředí na zkoumání a identifikaci klíčových výzev spojených s vývojem her pro více hráčů a následně popisuje návrh a implementaci síťové architektury v rámci herního enginu Unity.

Teoretická část práce se zaměřuje na literární rešerši, která poskytne přehled současných poznatků a technik v oblasti architektur síťové komunikace ve hrách pro více hráčů. Tato rešerše identifikuje klíčové koncepty, problémy a postupy spojené s návrhem a implementací síťové komunikace v herním prostředí. Zahrnuje i zhodnocení různých přístupů k řešení synchronizace herního stavu, optimalizaci síťového provozu a minimalizaci latence.

Druhá polovina teoretické části se věnuje rozboru hlavních síťových knihoven pro herní engine Unity. Jsou zde identifikovány klíčové vlastnosti jednotlivých knihoven, popis jejich nejdůležitějších komponent a funkcí i praktické ukázky kódu. Vzhledem ke stejnému zaměření knihoven, může čtenář porovnat jejich obdobné funkcionality a usoudit, která by pro potřeby jeho projektu byla nejvíce vyhovující.

Praktická část práce se zabývá výběrem jedné z knihoven a návrhem řešení architektury pro existující projekt UTW. Tento návrh zahrnuje specifikace projektu, UML zobrazení plánované funkčnosti a zhodnocení klíčových výzev a možných rizik.

Dále se praktická část zaměří na implementaci navrženého řešení a provedení jeho testování. Implementace bude prováděna v prostředí herního enginu Unity v programovacím jazyce C# za pomoci zvolené síťové knihovny. Součástí budou i ukázky kódu. Testování bude zahrnovat ověření funkčnosti a stability síťového řešení.

Práce také obsahuje dokumentaci vývoje, která detailně popisuje proces implementace, použité technologie a metody testování a zhodnocení výsledků.

I. TEORETICKÁ ČÁST

1 ARCHITEKTURA SÍŤOVÉ KOMUNIKACE VE HRÁCH PRO VÍCE HRÁČŮ

V dnešní éře počítačových her, kde interaktivní zážitky spojují hráče z celého světa, je klíčovým prvkem úspěšného herního designu a provozu síťová komunikace. Tato kapitola se věnuje detailnímu pohledu na techniky a postupy, které stojí za návrhem architektury síťové komunikace ve hrách pro více hráčů. Budou zde vysvětleny komplexní aspekty, včetně komunikačních protokolů, síťových topologií, klíčových výzev a bezpečnostních opatření.

Základním kamenem tohoto tématu jsou komunikační protokoly, konkrétně TCP a UDP. Tyto protokoly tvoří páteř síťové komunikace a jejich správný výběr může ovlivnit stabilitu a výkon herního prostředí. Dále budou zkoumány síťové topologie, jako jsou klasický Client-To-Server model, decentralizovaný P2P přístup a hybridní kombinace, které umožňují flexibilitu a optimalizaci pro různé herní scénáře.

Kapitola prozkoumává i klíčové výzvy, kterým čelí herní síťová komunikace. Latence, predikce pohybu, a synchronizace jsou klíčové faktory ovlivňující uživatelský zážitek a vyžadují pečlivou analýzu a řešení. Bezpečnost a ochrana proti podvádění (anti-cheat mechanismy) jsou rovněž nezbytnými součástmi, aby se zajistila spravedlnost a integrita herního prostředí.

1.1 Komunikační protokoly

Komunikační protokol představuje systém pravidel, který umožňuje dvěma nebo více subjektům v rámci komunikačního systému výměnu informací. Tato sada pravidel může zahrnovat ověřování, detekci a následnou opravu chyb a signalizaci. Může také popisovat syntaxi, sémantiku a synchronizaci analogové a digitální komunikace. [1][2]

Komunikační protokoly jsou v telekomunikačních a jiných systémech velice důležité, jelikož zajišťují konzistenci a univerzálnost při odesílání a přijímání zpráv. Jsou implementovány jak hardwarově, tak softwarově. [1][2]

Tyto protokoly jsou koncepčně rozřazeny pomocí modelu OSI či TCP/IP do specifických síťových vrstev. Cílem těchto modelů je vytvořit kontext, na němž by se zakládaly komunikační architektury mezi různými systémy. Konkrétně TCP a UDP, které budou rozebírány na dalších odstavcích jsou řazeny do 4. vrstvy, tzn. transportní. [2][3][4]

1.1.1 TCP

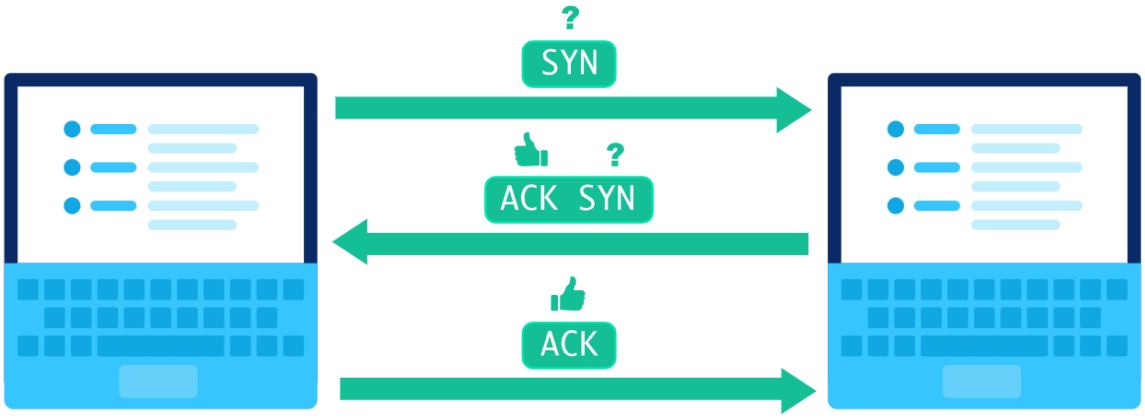
TCP, zkratka z anglického Transmission Control Protocol, představuje dnes pravděpodobně nejvytíženější protokol transportní vrstvy. Stará se o zabezpečení kontroly úspěšnosti přenosu, včetně opakovaného odesílání chybějících nebo poškozených částí, a optimalizuje rychlost přenosu podle aktuálních podmínek v síti. [3][4][5][6][7]

Jak už bylo řečeno protokol TCP je vysoce populární v datovém přenosu. To je do značné míry způsobeno tím, že ačkoliv je obvykle pomalejší než UDP, nabízí vysokou spolehlivost a schopnost korekce chyb. [8]

Funguje zde princip „třícestného podání rukou“, což je třístupňový proces, který vytváří spojení mezi zařízením a serverem. Po dokončení tohoto tříkrokového procesu je vytvořeno nepřetržité spojení, zahájí se přenos datových paketů přes internet, ty jsou doručeny v neporušeném stavu a je potvrzeno doručení. [3][9][10]

Zde je popsán způsob fungování protokolu TCP: [3][9][10]

1. Klientské zařízení, které iniciuje přenos dat, pošle serveru sekvenční číslo (SYN). To sděluje serveru číslo, kterým má přenos datového paketu začít.
2. Server potvrdí přijetí klientského SYN a odešle své vlastní číslo SYN. Tento krok se často označuje jako SYN-ACK (potvrzení SYN).
3. Klient poté potvrdí (ACK) serveru SYN-ACK, čímž je zformováno přímé spojení a začne přenos dat.



Obrázek 1. Třístupňové ověření u protokolu TCP [10]

Spojení mezi odesílatelem a příjemcem je udržováno až do úspěšného přenosu. Při každém odeslání datového paketu je vyžadováno potvrzení od příjemce. Pokud tedy není potvrzení přijato, data jsou odeslána znovu. Pokud je potvrzena chyba, je chybný paket zahozen a odesílatel doručí nový. Odeslání dat může zabránit také silný provoz nebo jiné problémy. V takovém případě je přenos odložen (aniž by došlo k přerušení spojení). [3][9]

Výhodou tohoto protokolu je tedy vysoká spolehlivost, ta však přichází za cenu pomalejší rychlosti. Tím se stává ideální volbou např. pro:

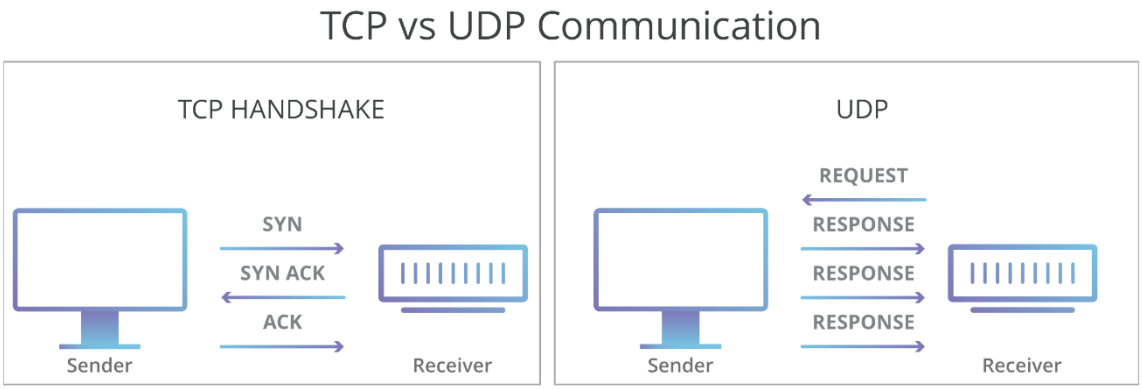
- Načítání webových stránek
- Posílání textu (email)
- Transferu souborů mezi lokálním a cloudovým úložištěm
- Služby vzdálené plochy
- Transakčně orientované aplikace

V těchto případech nelze chybu tolerovat a zároveň zde není kladen velký důraz na rychlý čas přenosu. Pokud je však za potřebí rychlejšího přenosu dat s menší spolehlivostí, přichází na řadu protokol UDP.

1.1.2 UDP

Protokol UDP, zkratka z anglického User Datagram Protocol, funguje na bázi okamžitého odesílání dat příjemci, který podal požadavek na přenos, dokud není přenos dokončen nebo ukončen. Protokol UDP, který se někdy nazývá „fire-and-forget“ (vystřel a zapomeň), odesílá data příjemci bez konkrétního pořadí, aniž by potvrzoval doručení nebo kontroloval, zda pakety dorazily tak, jak bylo zamýšleno. Tedy zatímco protokol TCP před odesláním dat navazuje formální spojení prostřednictvím dohody „handshake“. UDP urychluje přenos dat tím, že odesílá pakety, aniž by s příjemcem uzavíral jakoukoli dohodu. Pak je na příjemci, aby si s daty poradil. [3][9][11][12]

Tyto rozdíly přinášejí určité výhody. Protože UDP nevyžaduje „handshake“ ani kontrolu, zda data přicházejí správně, je schopen přenášet data mnohem rychleji než TCP. Tato rychlost však přináší kompromisy. Pokud se datagram UDP při přenosu ztratí, nebude znovu odeslán. Aplikace, které používají UDP, proto musí být schopny tolerovat chyby, ztráty a duplicity. [3][9][12]



Obrázek 2. Porovnání metod přenosu dat mezi protokoly TCP a UDP [12]

Protokol UDP doručuje data rychle a nezpomaluje se ani se nevrací zpět, aby obnovil ztracená data. To z něj činí ideální protokol pro nepřetržité doručování dat nebo vysílání, například pro: [9]

- Online hraní
- Videohovory
- Párování serverů s IP adresami.
- VoIP

1.1.3 Porovnání TCP a UDP v online hrách

V případě této práce se zaměřujeme především na online hraní videoher. V situaci, kdy uživatel má nekvalitní připojení nebo dochází ke ztrátě paketů z jakéhokoli jiného důvodu, může logika aplikace spolu s protokolem UDP zajistit, aby herní zážitek zůstal plynulý i pro ostatní uživatele.

TCP může působit naopak negativně na plynulost herního zážitku. Jeho pevnější kontrola nad přenosem dat a potvrzení doručení může způsobit zpoždění nebo vyvolat situace, kdy se hra musí zastavit a čekat na opětovné doručení ztracených dat.

Jak již bylo zmíněno, protokol UDP neposkytuje vestavěnou kontrolu integrity dat ani opakování přenosu ztracených paketů, což znamená, že může dojít ke ztrátě informací. V této situaci má klíčovou roli logika aplikace a síťový kód. Herní aplikace může implementovat strategie s cílem minimalizovat negativní vliv ztráty paketů na uživatelský zážitek, například prostřednictvím predikce pohybu a podobně.

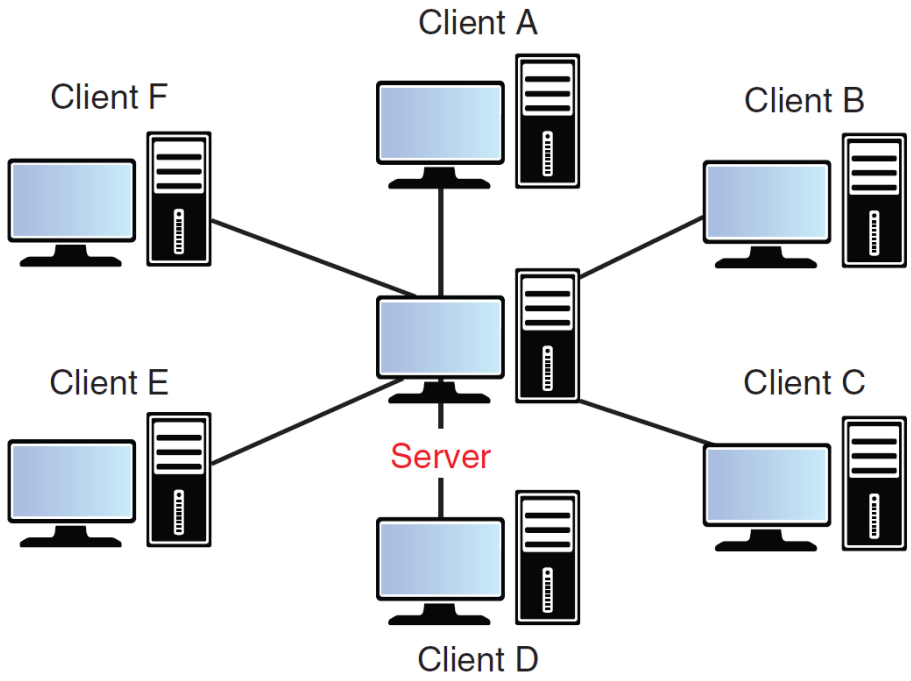
1.2 Síťové topologie

Topologie sítě určuje, jak jsou počítače v síti vzájemně propojeny. V kontextu hry topologie určuje, jak budou počítače účastnící se hry uspořádány, aby všichni hráči viděli aktuální verzi stavu hry. Stejně jako při rozhodování o síťovém protokolu existují kompromisy bez ohledu na zvolenou topologii. Tato část se zaměří na dva základní přístupy k organizaci síťového prostředí v herních systémech: Client-To-Server, P2P (peer-to-peer) a třetí méně využívaný hybridní přístup. [3]

1.2.1 Client-To-Server

Model Client-To-Server neboli architektura klient-server je distribuovaný aplikační rámec, který rozděluje úlohy mezi servery a klienty, kteří se buď nacházejí ve stejném systému, nebo komunikují prostřednictvím počítačové sítě nebo internetu. [3][13][14][15]

V topologii klient-server je jedna instance hry označena jako server a všechny ostatní instance jsou označeny jako klienti. Každý klient vždy komunikuje pouze se serverem, zatímco server je zodpovědný za komunikaci se všemi klienty. Požadavky klientů jsou organizovány a upřednostňovány v systému plánování, který pomáhá serverům zvládat případy, kdy v krátkém časovém úseku obdrží požadavky od mnoha různých klientů. [3][13]



Obrázek 3. Vizualizace topologie Client-To-Server [3]

1.2.1.1 Listen Server (server hostovaný klientem)

U tohoto typu serveru zastává jeden hráč roli hostitele a zároveň aktivně hraje, zatímco ostatní vzdálení členové skupiny se připojují k serveru. Tato konfigurace usnadňuje připojení všech účastníků do skupiny, nicméně nese několik nevýhod. Jednou z nich je pravděpodobná absence statické IP adresy, což může způsobit problémy s dostupností a bezpečností serveru. [16][17][18]

Mezi výhody tohoto přístupu patří:

- Nízké náklady ve srovnání s dedikovanými servery. Díky tomu, že tyto servery existují jako klienti, jsou zpravidla cenově dostupnější než vytvoření, konfigurace a placená údržba dedikovaného serveru. [16][17][19]
- Snadné nastavení. Listen servery nevyžadují další vybavení ani infrastrukturu, protože existují jako klienti. V důsledku toho neexistuje žádná technická překážka pro vstup a tato prostředí můžete spustit téměř okamžitě. [16][17][19]

Nevýhodami pak můžou být:

- Nestabilní síť. Výkon herní sítě závisí na hostitelském hráči. Pokud je síťové připojení hostitele pomalé, nekonzistentní nebo nespolehlivé, ovlivňuje to herní zážitek všech zúčastněných hráčů. [17][19]
- Latence. Vzhledem k tomu, že jsou vzdálení hráči závislí na připojení hostitele a samotný hostitel nikoli, může se stát, že bude mít hostitel rychlejší responzivitu než vzdálení hráči. Vzdálení hráči mohou také pociťovat větší zpoždění než hostitel, což přispívá k neférovosti daného herního prostředí. [16][17][19]
- Hratelnost často závisí na hostitelském hráči. Může se stát, že jakmile hostitelský hráč odejde, skončí tím rozehraná session. [16][17]

1.2.1.2 Dedikovaný Server

Dedikovaný server reprezentuje samostatný hardware, který nepotřebuje být hostován na zařízení hráče. Operuje nezávisle na herním klientovi a obvykle slouží k provozu serveru, na nějž se mohou hráči připojovat a odpojovat, aniž by se server vypnul spolu s nimi. Dedikované servery mohou být konfigurovány pro různé operační systémy a jsou schopny běžet na cloudových serverech, k nimž se hráči mohou připojovat prostřednictvím pevné IP adresy. Tato infrastruktura poskytuje stabilní a spolehlivé prostředí pro online hry,

umožňující hladký a bezproblémový průběh bez vlivu na výkon herních zařízení samotných hráčů. [16][18]

Výhody dedikovaného serveru mohou být:

- Žádný hráč nemá oproti ostatním žádnou výhodu, pokud není bráno v potaz umístění jednotlivých klientů. Všichni jsou ve stejném stavu a všichni sledují jeden autoritativní server, takže tato architektura je vhodná např. pro hry typu PvP. [16][20]
- Obvykle jsou využívány pronajaté pevné servery s adekvátně výkonným hardwarem a stabilní sítí pro operaci herních serverů. Tyto serverové stroje jsou navrženy tak, aby poskytovaly hráčům nepřetržitou herní relaci bez přerušení, a to díky svému stabilnímu výkonu a spolehlivé síťové infrastruktuře. [16][17]
- Prevence podvodů. Při hostování vlastních serverů lze zajistit, že na nich běží nepozměněné a legální verze her. Tím se zabezpečí, že všichni hráči zažívají konzistentní herní prostředí, které není ovlivněno subjektivními rozhodnutími správců. Tato praxe nejen umožňuje spravedlivé žebříčky a hodnocení, ale též poskytuje trvalý sled postupu hráčů založený na jejich herním výkonu. [3]

Mezi nevýhody poté patří:

- V případě, kdy hra zahrnuje sofistikované generování světa, objektů a komplexní fyzikální výpočty, vyžaduje každá aktivní relace významné zdroje, zejména co se týče paměti a procesorového výkonu. Nákup či pronájem těchto zdrojů může pro vývojáře představovat značné finanční zatížení, zejména s ohledem na potřeby správy a udržování dostatečně výkonných prostředků pro optimální provoz a uživatelský zážitek. [16][17][19]
- Dedikované servery nabízejí rozšířenou možnost kontroly nad nastavením a konfigurací, to s sebou však nese určitou míru odpovědnosti. Pro vývojáře, kteří si přejí nastavit dedikovaný server, je nezbytné disponovat specifickými technickými znalostmi, které umožní správné provedení postupů nezbytných pro efektivní provoz serveru. [17][19]
- Závislost na externích poskytovatelích. Přenesení provozu hry na servery společností jako Amazon či Microsoft znamená, že celkový osud vaší hry spočívá v rukou těchto firem. I přes existenci smluv o úrovni služeb, které zaručují minimální dobu dostupnosti, tato opatření poskytovatelů při výpadcích často nepostačují ke zklidnění plátců za hru, zejména pokud dojde k simultánnímu výpadku všech serverů. [3]

1.2.1.3 Praktický příklad postupu implementace ve hře

Na počáteční úrovni předpokládejme existenci hry pro jednoho hráče, v níž hráč přebírá roli jedné postavy. V rámci této hry je implementován proces generování nepřátelských entit, a také je zabezpečena fyzikální simulace objektů. [17]

Pro ilustrační účely je uvedeno, že hra musí podporovat režim dvou hráčů, kde hráči interagují se třemi nepřáteli. Jeden z možných přístupů spočívá v replikaci herního prostředí na oba počítače, duplikaci hráčských postav a implementaci programové logiky pro synchronizaci souřadnic postav mezi oběma zařízeními. Tímto způsobem jsou oba počítače schopny společně udržovat a aktualizovat stav hry tak, aby byl koherentní a odpovídal akcím obou hráčů. [17]

Vzniká však problém. První a druhý počítač simultánně generují nepřátelské entity, což má za následek celkový počet šesti nepřátel místo původně zamýšlených tří. Krom toho, že všechny entity jsou vytvářeny ve stejném bodě prostoru, dochází k jejich rozptylu do okolí v souladu s fyzikálními zákony hry, které každý počítač samostatně vypočítává. Objevuje se zde konflikt mezi „pravdivými“ stavy obou počítačů. Každý z nich generuje svůj vlastní virtuální svět a aplikuje svou vlastní fyziku na entity, což vede k divergenci ve vizualizovaném obrazu mezi jednotlivými počítači. [17]

Je tedy zaveden centralizovaný autoritativní zdroj pravdy, kde jediný počítač ponese zodpovědnost za výpočet fyzikálních aspektů hry a generování nepřátel. Druhý počítač pak slouží pouze jako pasivní entita, která opisuje a reprodukuje informace poskytnuté prvním počítačem. Jeho hlavním úkolem bude monitorovat a získávat souřadnice všech objektů, aby mohl efektivně vizualizovat herní scénu. Tato architektura přispívá ke specializaci úkolů mezi počítači, kde jeden slouží pro výpočetně náročné operace, zatímco druhý se soustředí na vizualizaci a prezentaci informací. [17]

V tomto uspořádání první počítač funguje jako tzv. listen server, zastávající roli centrálního zázemí. Tento server obstarává servisní úlohy pro ostatní hráče, prostřednictvím svých výpočetních prostředků. Aktivně předává aktuální stav hry, včetně souřadnic všech objektů a dalších relevantních informací. Ostatní počítače, nazývaní klienti, se specializují na opakované reprodukování herního stavu poskytovaného serverem, čímž se zajistí vizuální konzistence pro všechny hráče. Klienti poté interagují s herním světem posíláním zpětných informací o uživatelských akcích zpět na server (např. stisknutí kláves na klávesnici či myši), což server následně interpretuje a aktualizuje celkový herní stav. [17]

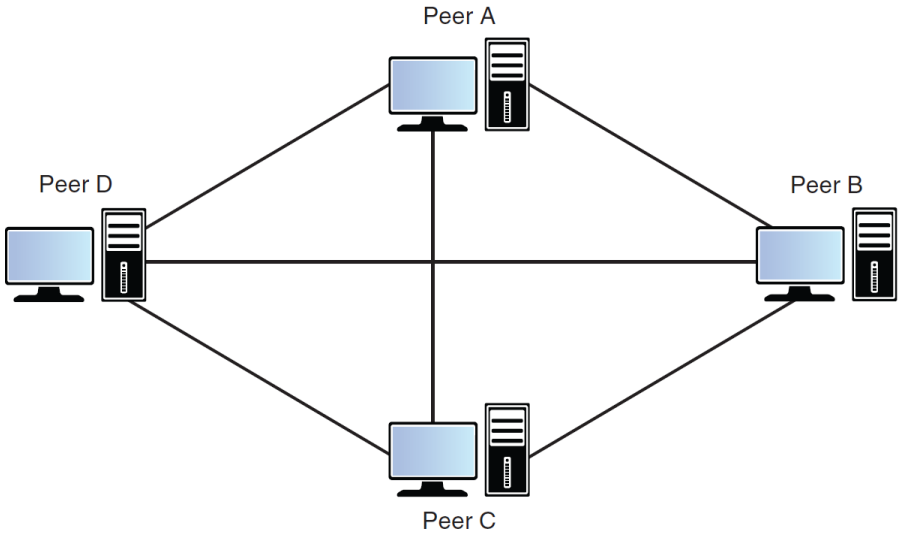
V důsledku implementace této struktury je hra, která původně existovala jako jednotný celek, nyní rozdělena na dvě základní části: aplikaci běžící na straně serveru a klientskou aplikaci. Počítač, který nese odpovědnost za provádění serverové aplikace, je označován jako hostitel. [17]

1.2.2 P2P (peer-to-peer)

Peer-to-peer (P2P) sítě jsou decentralizované, což znamená, že nemají žádný centrální server nebo orgán, který by je řídil. V tomto modelu mají všichni účastníci stejná práva a odpovědnosti. Uživatelé navzájem sdílejí zdroje a služby bez nutnosti spoléhání se na centrální řídicí bod. [3][19][21][22][23]

Tyto sítě představují samo organizující se systémy, které se dynamicky přizpůsobují a reorganizují v průběhu připojování a odpojování účastníků. Pokud by například došlo ke ztrátě komunikace s účastníkem dochází k dočasnému zastavení hry, obvykle po několik sekund, než je dotčený ze hry odstraněn. Jakmile se tak stane, zbývající účastníci mohou bezproblémově pokračovat v simulaci hry. [3][23]

V peer-to-peer je každý účastník přímo propojen s každým účastníkem. Tento přístup implikuje významný tok dat mezi klienty, kde počet spojení mezi nimi představuje kvadratickou funkci. S ohledem na n rovnocenných partnerů vyžaduje každý z nich O(n - 1) spojení, což vytváří $O(n^2)$ spojení v celé síti. To znamená, že požadavky na šířku pásma pro každého klienta rostou s připojením většího počtu účastníků ke hře. V kontrastu s modely typu klient-server jsou však požadavky na šířku pásma v peer-to-peer topologii symetrické, což znamená, že každý peer bude vyžadovat stejnou dostupnou šířku pásma jak pro odchozí, tak i pro příchozí datový tok. [3]



Obrázek 4. Vizualizace topologie Peer-To-Peer [3]

1.2.2.1 Výhody

- Malé náklady. Finanční úspory představují výhodu spojenou s absencí potřeby udržovat dedikovaný herní server, což tuto možnost činí atraktivní pro hry vyvíjené nezávislými vývojáři. [21][22][23]
- Přímá komunikace. Sítě peer-to-peer umožňují přímou interakci mezi rovnocennými partnery, což eliminuje potřebu prostředníků. Tato přímá komunikace výrazně zrychluje doručování obsahu, interaktivitu v reálném čase a efektivní distribuci dat. [19][23]
- Adaptivní škálovatelnost. Se zvyšujícím se počtem hráčů se peer-to-peer síť snadno přizpůsobuje prostřednictvím přidávání dalších připojení, což zajišťuje plynulost hry bez ohledu na počet účastníků. [21][23]

1.2.2.2 Nevýhody

- Bezpečnostní Aspekty. Vzhledem k tomu, že účastníci navzájem komunikují napřímo, je nezbytné zajistit pravost a integritu dat posílaných mezi účastníky. [19][22][23]
- Zvýšená složitost synchronizace. Synchronizace v P2P prostředí může být náročnější v porovnání s jinými modely, zejména kvůli potřebě koordinace a správy dat mezi rovnocennými partnery. Tato vyšší komplexita může vyžadovat sofistikovanější mechanismy a algoritmy pro udržení konsistence dat a spolehlivou koordinaci akcí mezi různými uzly v síti. [19][23]

- Závislost na stabilitě síťového připojení zařízení. Sítě P2P vyžadují kontinuální a spolehlivou konektivitu mezi jednotlivými uzly, což může být problematické v prostředí s nízkou stabilitou nebo v situacích, kdy dochází k častým výpadkům připojení. [19][22][23]
- V P2P sítích může dojít k situacím, kde někteří účastníci záměrně poskytují falešné nebo zmanipulované informace, což může mít negativní dopad na integritu herního zážitku. Bez centrální autority pro ověření totožnosti a integrity dat může být obtížné eliminovat a detekovat podvodné činnosti. [19][22]

1.2.2.3 Praktický příklad postupu implementace ve hře

V daném scénáři je realizována multiplayerová hra prostřednictvím neautoritativního P2P přístupu. Hra představuje arénu, kde každý hráč ovládá vesmírnou loď, která je zároveň schopna střílet.

Neautoritativní multiplayerová hra postrádá centralizovanou entitu, která by měla kontrolu nad stavem hry. Každý hráč je tak odpovědný za správu a sledování vlastního herního stavu a informování ostatních klientů o všech změnách a klíčových událostech. V důsledku toho hráč zaznamenává dva simultánní scénáře: [24]

- Pohyb vlastní lodě podle vlastních zadání.
- Simulaci všech ostatních lodí ovládaných soupeři.

Pohyb a akce lodi hráče jsou řízeny lokálními vstupy, což znamená, že stav hráče je aktualizován téměř okamžitě. Pro získání informací o pohybu všech ostatních lodí musí hráč přijmout síťové zprávy od každého soupeře, které poskytují aktuální polohu jejich lodi. Vzhledem k tomu, že přenos těchto zpráv po síti zabere určitý čas, nelze zaručit, že když hráč obdrží informaci o poloze soupeřovy lodi, lze tuto informaci považovat za aktuální, protože lodě mohou mezi časem odeslání a doručení zprávy změnit svou polohu. Tento jev je následně reflektován jako simulace herního stavu. [3][24]

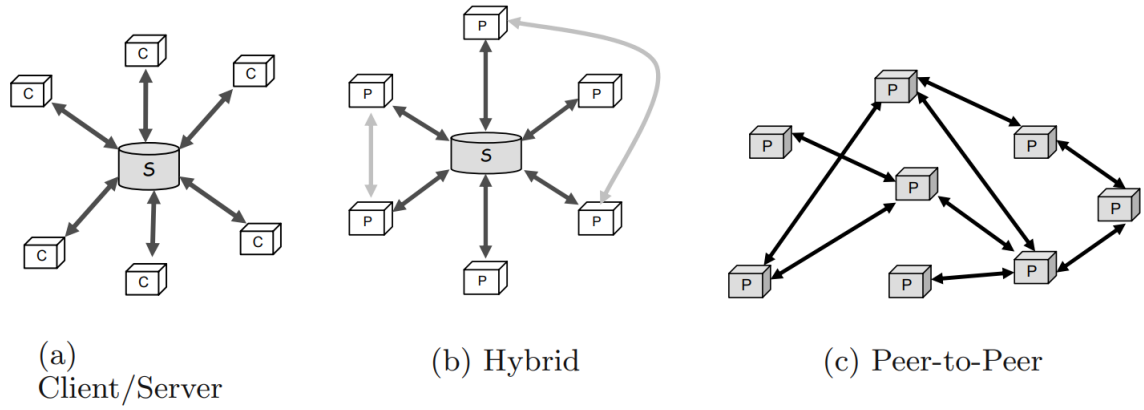
Pro dosažení přesné simulace je každý účastník odpovědný pouze za distribuci informací o vlastní lodi, nikoli o ostatních. To znamená, že v případě čtyř hráčů ve hře (označeni jako A, B a C) má hráč A jediné právo informovat o poloze své lodi, o střelbě, popř. o vlastním zásahu. Ostatním hráčům je doručena zpráva od hráče A, informující je o jeho akcích, na které následně reagují. Například, pokud střela od hráče A zasáhne loď C, pak C vysílá zprávu informující o svém zničení. [24]

V důsledku toho každý hráč vnímá všechny ostatní lodě (a jejich akce) v souladu s přijatými zprávami. Ideálně by v bezchybném světě bez síťových zpoždění byly zprávy přenášeny a doručovány okamžitě, což by zajišťovalo extrémně přesnou simulaci. [24]

Nicméně s nárůstem latence se přesnost simulace zhoršuje. Například hráč A vidí lokálně, že jeho střela zasáhla loď B, ale ve skutečnosti se tak nestane, protože hráč B je zpožděný v důsledku horšího připojení. Když B konečně obdrží zprávu od A o střele, je již na jiné pozici, což vede k nesouladu mezi zahlédnutým a reálným zásahem. [24]

1.2.3 Hybridní přístupy

Hybridní modely reprezentují symbiózu prvků peer-to-peer a klient-server architektur. Typickým příkladem hybridního modelu je existence centrálního serveru, který funguje jako prostředník usnadňující vzájemné propojení mezi peery. [25][26]



Obrázek 5. Srovnání struktur sítí [25]

1.2.3.1 Praktický příklad

Článek z roku 2015, který se věnuje problematice webové spolupráce při 3D modelování, navrhuje hybridní komunikační architekturu, která kombinuje prvky klient-server a peer-to-peer (P2P). Klientská část je odpovědná za vykreslování 3D scény a správu uživatelských interakcí. Současně zajišťuje hostování peer připojení, umožňující přenos aktualizací ostatním uzlům v síti. [26]

Centrální server plní roli spojovacího prvku mezi klientem a NoSQL databází. Tento server slouží k ukládání modifikací a správě přítomnosti uživatelů na 3D scéně. Kromě toho zabezpečuje automatické vytváření sítě P2P s plnou mesh topologií mezi klienty. [26]

Komunikace v rámci P2P sítě využívá technologii WebRTC, která umožňuje přenos informací přímo mezi webovými prohlížeči pomocí aktualizačních zpráv. K navázání komunikace mezi jednotlivými uzly P2P slouží signalizační server. [26]

Kvalitativní hodnocení provedených experimentů poskytlo celkově přesvědčivé výsledky, avšak byly identifikovány oblasti, kde je možné dosáhnout zlepšení. Při importu modelu byly zaznamenány problémy s latencí na přijímačích klientských vrstevníků, což se projevovalo například ve formě zamrzání kamery. S ohledem na redukci latence, zejména ve scénách s větším rozsahem, je uvažováno o implementaci progresivního vykreslování a optimalizovanějším využití sítě peer-to-peer pro přenos modelu s využitím seed peerů v rámci topologie částečné sítě. [26]

1.3 Klíčové výzvy

Herní prostředí pro více hráčů reprezentuje dynamický rámec, v němž se objevují komplexní výzvy v oblasti síťové komunikace. V této části práce budou důkladně analyzována klíčová témata týkající se architektur síťové komunikace ve hrách pro více hráčů, s výrazným důrazem na tři základní oblasti: Latence, Predikce a Synchronizace.

1.3.1 Latence

Pojednává-li se o „zpoždění“ či latenci ve světě her, referuje se na celkový čas, který je potřebný pro přenos informací z herního zařízení na server a následně zpět. Pro hráče se vysoká latence projevuje znatelným časovým zpožděním, například v situaci, kdy zadají příkaz a musí čekat několik sekund, než se příslušná akce promítne na obrazovce. Časový interval mezi vstupem hráče a reakcí serveru na tento vstup je v herním kontextu označován jako „ping“, nebo RTT. [27][28][29]

- V případě vysoké latence dochází k patrným prodlevám při přenosu informací a může se dokonce objevit „ztráta paketů“, kdy data na své cestě zaniknou. [27][28]
- Nízká latence ve hrách značí, že přenos informací probíhá takříkajíc v reálném čase, bez znatelného zpomalení. [27][28]

Představme si scénář, v němž se uživatel účastní hry typu FPS a identifikuje protivníka. Avšak kvůli několika milisekundovému zpoždění, které uživatele odděluje od momentu zamíření a provedení výstřelu, se protivník již stihl přesunout na jinou pozici. Tato situace samozřejmě indukuje frustraci daného uživatele, o to více pokud se jedná o kompetitivní prostředí. I když se latence a ping mohou jevit pouze jako metriky herního výkonu,

ve skutečnosti mohou být klíčovými faktory, které ovlivňují úspěch nebo nezdary ve hře. [27]

1.3.1.1 Další faktory ovlivňující latenci

Krom termínů jako ping, latence a ztráta paketů existuje několik dalších konceptů, které jsou klíčové pro pochopení, proč reakce hry nemusí být tak rychlá, jak by si uživatel přál. [28]

- Šířka pásma (Bandwidth). V podstatě se jedná o prostor, který je k dispozici pro zařízení v síti. Mnozí uživatelé si mohli například všimnout, že je obtížnější hrát, když jeden ze spolubydlících telefonuje přes Zoom a v obývacím pokoji je streamován film. Je to proto, že šířka pásma je omezená. Větší šířka pásma znamená, že se může připojit více zařízení najednou bez přerušení. Šířka pásma je ovlivněna jednak tím, jakou disponuje velikostí, a jednak tím, kolik lidí současně přistupuje k této šířce pásma. [28][30][31]
- Propustnost (Throughput). Termín „propustnost“ označuje objem dat, které mohou projít sítí ve stejném časovém okamžiku. Tato metrika je obvykle měřena v bitech za sekundu (bps) nebo v bajtech za sekundu (Bps). Analogií k tomuto pojmu by mohl být dav lidí, pokoušející se projít jedním vchodem. Rozšíření tohoto vstupu, tj. snížení hustoty zařízení, která se snaží získat přístup, vede k rychlejšímu průchodu dat. [28][32][33]

1.3.1.2 Význam vysoké latence

Neexistuje žádná univerzální definice toho, kdy je čekací doba ve hrách považována za příliš vysokou. Nicméně vášniví uživatelé tvrdí, že jakékoli zpoždění je nepřijatelné, neboť i krátký časový rozdíl může rozhodovat o vítězství či porážce. [27][28]

Tabulka 1. prezentuje data, jež jsou výrazně subjektivního charakteru, avšak byla systematicky získána z různorodých a vzájemně konzistentních zdrojů. Tato data zahrnují informace z článků věnovaných danému tématu a diskuzí na internetu.

Tabulka 1. Tabulka rozsahů vnímaní latence [29][34][35][36][37]

Rozsah pingu	Kvalita pingu	Vliv na herní zážitek
0–20 ms	Skvělá	Zpoždění nebo závady během hraní by měly být minimální; vysoce kvalitní vizuální efekty.
20–50 ms	Dobrá	Většinou plynulý a responsivní herní zážitek
50–100 ms	Slušná	Nejběžnější rozsah pro hráče; v závislosti na hře a nastavení zařízení může docházet k občasnému zpoždění.
100–150 ms	Špatná	Při hraní her pravděpodobně dochází k mnoha zpožděním a prodlevám.
150 ms a více	Nelze hrát	Rozsáhlé zpoždění odezvy; znatelné zpoždění, které způsobí nekvalitní herní zážitek.

1.3.1.3 Příčiny vysoké latence ve hrách

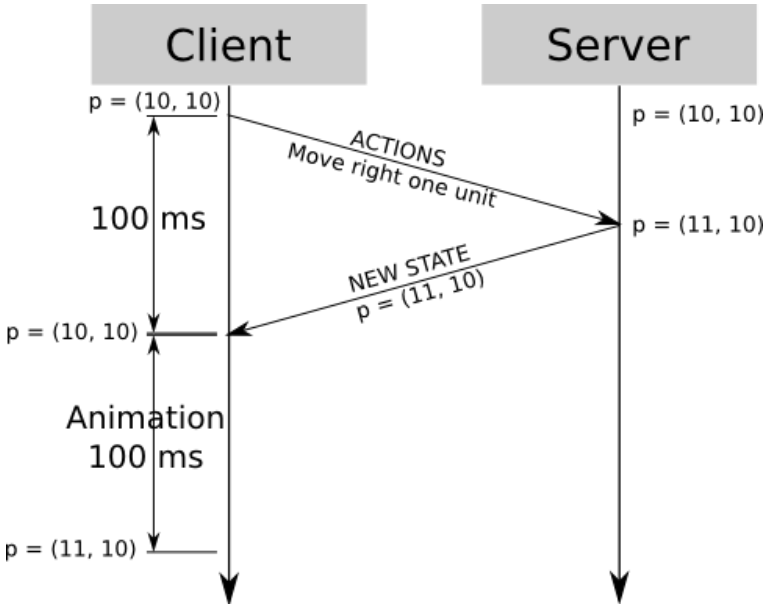
Mezi příčiny vysoké latence ve hrách mohou patřit následující:

- Hardwarové parametry serveru, zejména v případě méně výkonných nebo sdílených strojů, vždy ovlivní rychlost zpracování a renderování herních dat v porovnání s herním serverem disponujícím nízkou latencí. [27]
    - Poloha serveru také hraje klíčovou roli, neboť čím větší je geografická vzdálenost mezi serverem a hráčem, tím delší je doba, kterou informace potřebují k překonání této vzdálenosti. Při průchodu paketu internetovými sítěmi vzniká obvykle několik přechodů, které nadále prodlužují dobu přenosu dat. [27][28][29][38]
- Hardwarová konfigurace uživatele může ovlivnit plynulost vysokorychlostního hraní. Například zastaralý router může být zdrojem zpomalení. [27][28][29][38]
- Geografická poloha uživatele může významně ovlivnit plynulý herní zážitek. Čím větší je vzdálenost mezi serverem a místem, kde uživatel používá svá zařízení, tím vyšší je pravděpodobnost výskytu prodlev. Pro uživatele v Severní Americe je vhodné zajistit, aby byli připojeni k serveru umístěnému v jejich geografické oblasti, a vyvarovat se připojení k evropským serverům. [27][28][29][38]

1.3.2 Predikce

1.3.2.1 Autoritativní server a „hloupý“ klient

Tim Sweeney kdysi na téma topologie sítě klient-server napsal: *„The server is the man!“*. Jeho výrok reflektoval skutečnost, že v daném síťovém uspořádání je server jediným zařízením, které disponuje skutečným a korektním stavem hry. Toto stanovisko odpovídá tradičním požadavkům na bezpečnost klient-server architektur odolných vůči podvodům: server je jedinou entitou, která provádí simulaci, na níž záleží. To implikuje, že vždy existuje určité zpoždění mezi časem, kdy se událost stane, a časem, kdy hráč může vizuálně pozorovat výsledek této události. [3][39][40]



Obrázek 6. Základní schéma RTT u modelu Client-Server (bez predikce) [41]

Naivní implementace uvedeného schématu vykazuje významné zpoždění mezi uživatelskými příkazy a tím co se danému uživateli zobrazuje na monitoru. Toto zpoždění vzniká kvůli tomu, že uživatelský vstup musí projít cestou na server, kde je zpracován a nový stav hry je vypočítán a následně je aktualizovaný stav hry přenesen zpět ke klientovi. [3][40][41][42][43]

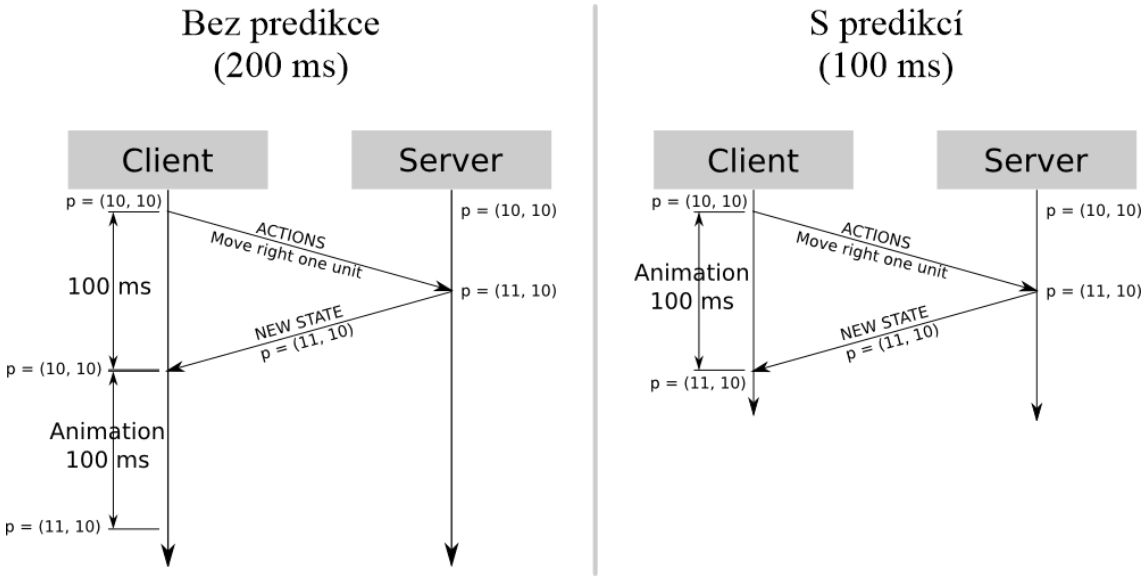
V kontextu síťového prostředí, jako je internet, kde latence může dosahovat desítek nebo stovek milisekund, se může stát, že v lepším případě hra reaguje s omezeným úspěchem, a v horším případě může být úplně nehratelná. [3][40][41][42][43]

1.3.2.2 Predikce na straně klienta

I přes existenci podvodných hráčů, herní server převažující část času zpracovává legitimní požadavky. V důsledku toho většina přijatých uživatelských vstupů odpovídá validním operacím a aktualizuje stav hry v souladu s očekáváním. Pokud se například postava hráče nachází na souřadnicích (10, 10) a je stisknuta klávesa pro pohyb do strany, předpokládaný výsledek bude pozice (11, 10). [40][41]

Tohoto principu můžeme využít v náš prospěch. Pokud je herní svět dostatečně deterministický, což znamená, že při daném herním stavu a konkrétní sadě vstupů lze přesně předpovědět výsledek, máme možnost odeslat vstupy na server a okamžitě je zpracovat na straně, nikoli už hloupého, klienta. Tím pádem můžeme předem odhadnout, jaký bude herní stav po provedení vstupů na serveru, což eliminuje prodlevu mezi odesláním vstupu a vizuálním vyobrazením jeho účinku. Tato předpověď je většinou přesná, takže po obdržení aktualizovaného stavu hry od serveru nedochází k žádným viditelným nesouladům. [3][40][41][43][44]

V případě zpoždění ve výši 100 ms a délky animace přechodu postavy mezi políčky trvající 100 ms by v naivní implementaci celý proces trval 200 ms. Díky predikci však dochází ke zkrácení doby na 100 ms, protože klient nezávisle na serveru zahajuje pohyb a potvrzení od serveru obdrží, jakmile dorazí na novou pozici. [40][41]



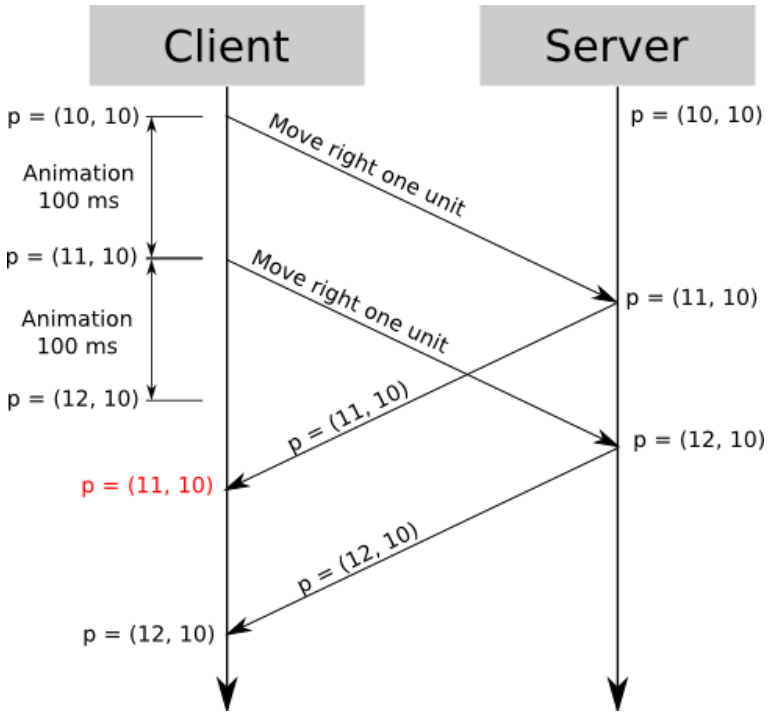
Obrázek 7. Schématické porovnaní RTT bez predikce a s predikcí [41]

Jinými slovy v současném stavu nenastává žádná prodleva mezi akcemi hráče a vizuálními výsledky na obrazovce. Přitom je stále zachována autoritativnost serveru. V případě, že by podvodný klient odesílal neplatné vstupy, mohl by na své obrazovce zobrazovat libovolné výsledky, nicméně by to nemělo žádný vliv na stav serveru, který je zaznamenán a viditelný ostatními hráči. [40][41][44]

1.3.3 Synchronizace

V předchozí sekci byl prezentován příklad, v němž nebylo bráno v úvahu žádné zpoždění při přenosu informací z klienta na server a zpět.

V případě upraveného scénáře se zpožděním 250 ms vůči serveru a délkou animace přechodu mezi čtverci trvající 100 ms předpokládejme, že hráč stiskne pravou klávesu dvakrát za sebou, snažíc se tak přesunout o dvě čtverce doprava. [40][41]



Obrázek 8. Problém s predikcí při opožděné komunikaci [41]

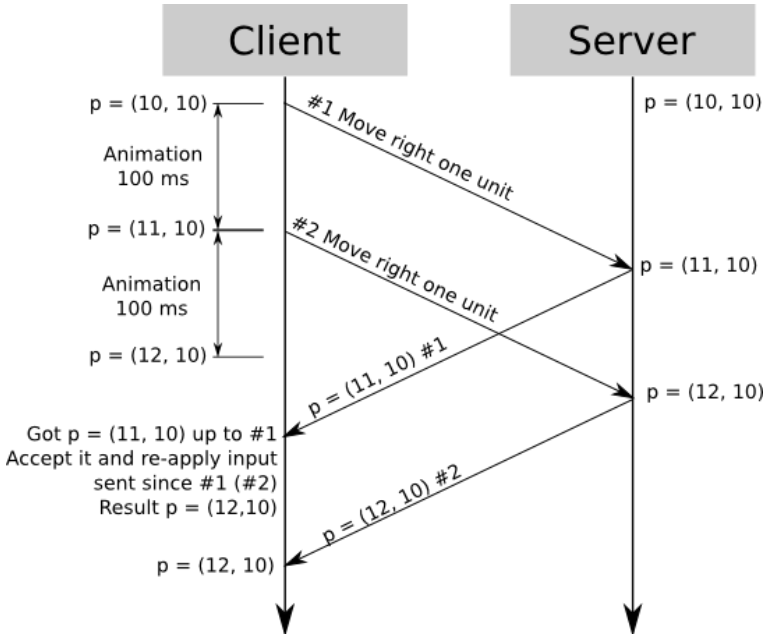
V čase t = 250 ms se objevuje zajímavý problém, kdy nový herní stav dorazí. Předpokládaný stav na straně klienta je x = 12, zatímco server hlásí, že nový stav hry je x = 11. V souladu s autoritativností serveru je klient nucen přesunout postavu zpět do stavu x = 11. Následně, v čase t = 350 ms, přichází nový stav serveru, který indikuje x = 12, což způsobuje další posun postavy, tentokrát dopředu. [40][41]

Z perspektivy hráče, který dvakrát stiskl klávesu se šipkou doprava, se zdá, že postava se posunula o dvě políčka doprava, zůstala tam 50 ms, skočila o jedno políčko doleva,

setrvala tam 100 ms a nakonec skočila o jedno políčko doprava. Tato situace je evidentně nežádoucí a může vytvářet nerealistické a frustrující herní zážitky. [40][41]

Klíčovým prvkem při řešení tohoto problému je uvědomit si, že klient vnímá herní svět v přítomném čase, avšak aktualizace, které obdrží ze serveru, jsou ve skutečnosti stavem hry z minulosti. V okamžiku, kdy server odesílá aktualizovaný stav hry, ještě nezpracoval všechny příkazy odeslané klientem. [40][41]

Nicméně existuje elegantní způsob, jak tuto situaci řešit. Klient připojí ke každému svému požadavku pořadové číslo; v tomto příkladu je první stisk klávesy požadavek č. 1 a druhý stisk klávesy je požadavek č. 2. Pro třetí stisk klávesy by to byl požadavek č. 3. Když pak server odpoví, uvede pořadové číslo posledního přijatého vstupu, který zpracoval [40][41][44]



Obrázek 9. Predikce na straně klienta + Sladění se serverem [41]

V čase t = 250 server sděluje, „Na základě informací, které jsem měl k dispozici do okamžiku vašeho požadavku č. 1, je vaše aktuální pozice x = 11.“ S ohledem na autoritativnost serveru je klient nucen nastavit pozici postavy na x = 11. Klient si udržuje kopii odeslaných požadavků a na základě nového herního stavu pochopí, že server již zpracoval požadavek č. 1. Tuto kopii může zahodit, ví však také, že server ještě nevyřídil požadavek č. 2. Klient znovu využívá predikce na straně klienta a vypočítá „aktuální“ stav hry na základě poslední autoritativní informace od serveru (x = 11) + vstupy, které server ještě neschválil.

V tomto případě vstup č. 2, „přesunout se doprava“. Konečný výsledek je x = 12, což odpovídá očekávané korektní hodnotě. [40][41][44]

Aby byl příklad dokončen, v čase t = 350 přijde od serveru nový herní stav s informací „x = 12, poslední zpracovaný požadavek = #2“. V tomto okamžiku klient zahodí všechny vstupy od #2 a později, aktualizuje stav na x = 12. Neexistuje žádný nezpracovaný vstup, který by bylo možné přehrát, a tím je zpracování ukončeno s korektním výsledkem. [40][41]

1.3.4 Interpolace

Skákavé aktualizace pozic, které jsou vyvolané sporadickými aktualizacemi stavu serveru, může vytvářet nepříjemný dojem ze hraní. Jedním z přístupů k odstranění tohoto problému je implementace interpolace na straně klienta. [3][40][45]

Při využití interpolace na straně klienta není objekt automaticky teleportován na své nové pozice, které jsou přijaty ze serveru. Místo toho klient plynule interpuluje stav objektu v průběhu času, vždy když obdrží nový stav. [3][45]

1.3.5 Kompenzace zpoždění (Lag Compensation)

V rámci aplikace metod diskutovaných v předchozích částech může hra efektivně zajistit rychlou odezvu pro hráče i při mírném zpoždění. Nicméně, situace, které vyžadují časovou a prostorovou náročnost, jako například střelba do hlavy nepřítele ve hře typu FPS, mohou představovat problematický scénář. [3][40][43][46]

Vzhledem k architektuře klient-server se stává, že při zamíření na místo, kde se nachází hlava nepřítele, klient sleduje události např. 100 ms pozadu. Tento efekt lze přirovnat k situaci, kdy hráč operuje ve vesmíru, kde rychlost světla je neobyčejně pomalá. V praxi to znamená, že hráč míří na minulou pozici nepřítele, ale v okamžiku výstřelu je tato pozice již překonána. [40][46]

Řešení, které se stalo populárním díky jeho použití v herním engine Source od společnosti Valve a výrazně přispělo k přesnosti střelby ve hrách, jako je Counter-Strike, vychází z principu známého jako „lag compensation“ (kompenzace zpoždění). Tato technika spočívá v tom, že stav na serveru je přepočítán tak, aby odpovídal přesnému stavu, který hráč viděl v okamžiku stisku spouště a výstřelu. [3][40][42][43][46]

Esencí tohoto přístupu je, že pokud hráč vnímal, že perfektně zamířil, server zajistí, aby jeho výstřel byl v souladu s touto percepcí ve 100 % případů. Tímto způsobem je sníženo riziko nesouladu mezi tím, co hráč viděl a jak byla jeho akce interpretována serverem, což vede k výraznému zlepšení přesnosti střelby a celkového uživatelského zážitku. [3][40][42][43][46]

Jednotlivé kroky při kompenzaci zpoždění: [40][43][46]

1. Po provedení výstřelu klient odesílá na server kompletní událost obsahující exaktní časové razítko (timestamp) výstřelu a přesné zaměření zbraně.
2. Zde nastává klíčový krok. Vzhledem k tomu, že server disponuje veškerými vstupními údaji s časovými razítky, má schopnost autoritativně rekonstruovat herní svět v libovolném okamžiku v minulosti. Konkrétně může přesně reprodukovat stav herního světa tak, jak ho viděl každý klient v daném časovém okamžiku.
3. To tedy implikuje, že server má přesnou informaci o tom, co bylo na zaměřovači střelcovi zbraně v okamžiku výstřelu. I když se jedná o minulou pozici hlavy cíle, server má schopnost rozpoznat, že se jednalo o aktuální pozici hlavy v kontextu střelcovi přítomnosti.
4. Server vyhodnotí výstřel v daném časovém okamžiku a provádí aktualizaci u všech klientů.

Tento postup však nese i své nevýhody. Kvůli posunutí času serveru o hodnotu závislou na latenci mezi serverem a klientem se mohou dít nepředvídatelné a frustrující situace obětím výstřelů. Hráč A by mohl být přesvědčen, že se bezpečně schovává za rohem a uniká hráči B. Nicméně, pokud má hráč B výrazné zpoždění ve svém síťovém připojení, jeho pohled na herní svět může být zpožděný např. o 300 ms ve srovnání s pohledem hráče A. To znamená, že na obrazovce hráče B se hráč A ještě nemusel skrýt za roh. Pokud hráč B provede střelbu v souladu s jeho aktuálním pohledem, server mu může připsat zásah a upozornit hráče A, že byl zasažen, i když si hráč A myslel, že je v bezpečí za rohem. [3][40][42][46]

Vývoj her vždy zahrnuje kompromisy, toto řešení je sice poněkud nespravedlivé, avšak představuje nejvyváženější kompromis pro všechny zainteresované strany. Alternativou by byl mnohem nepříjemnější scénář, kdy by hráči míjeli střely, které jsou nepopiratelné zásahy. [3][46]

1.4 Zabezpečení serveru

Servery hrají klíčovou roli při zpracování a ukládání dat. Ochrana serverů před vnějšími hrozbami pomocí serverových bezpečnostních opatření je zásadní pro zachování: [47]

- Integrity
  - Bezpečnostní opatření serveru je implementováno s cílem udržet integritu dat uložených na serveru tím, že zabraňuje jejich zkreslení či změnu. [47]
- Dostupnosti
  - Bezpečnostní opatření přispívají k udržení neustálé dostupnosti serveru a jeho služeb pro oprávněné uživatele. [47]
- Důvěrnosti
  - Ochrana citlivých dat uložených na serveru zabraňuje neoprávněným uživatelům používat data ke škodlivým účelům. [47]
- Reputace
  - Narušení zabezpečení může mít za následek ztrátu dat nebo přerušení poskytování služeb. Tyto incidenty mohou poškodit pověst společnosti. [47]

Klíčovým aspektem zabezpečení síťových her je ochrana herního serveru před potenciálními útoky. Tato opatření mají zvláštní význam u her se sdíleným virtuálním světem, který je centralizován na serverech. Nicméně i u každého jiného herního serveru je důležité brát v úvahu možnost útoku. Je nezbytné počítat s různými typy útoků a zajistit, že jsou přijata opatření k ochraně serveru. Důležité je také mít připraveny plány pro nepředvídatelné situace v případě, že ke zranitelnosti serveru dojde. [3]

1.4.1 Distributed Denial-of-Service Attack (DDoS)

Distribuovaný útok typu DDoS má za cíl zahlcovat server nepřetržitým tokem požadavků, které není schopen úspěšně zpracovat. Tím dochází ke ztížení nebo úplné nedostupnosti serveru pro legitimní uživatele, což v konečném důsledku narušuje jeho použitelnost. Mechanismus tohoto útoku spočívá v nadměrném objemu příchozích dat, která buď přetěžují síťové připojení serveru, nebo využívají tolik výpočetního výkonu, že server není schopen účinně zpracovávat aktuální požadavky. Téměř každá významná síťová hra nebo online služba pro hráče byla alespoň jednou v historii postižena útokem DDoS. [3][48][49][50]

Tento typ útoku obvykle řídí útočník, který ovládá botnet. Botnet představuje síť kompromitovaných a infikovaných zařízení, která mohou být na dálku ovládána nekalými

aktéry. Díky masivní škále je extrémně obtížné filtrovat požadavky těchto škodlivých zařízení, protože mohou generovat zdánlivě legitimní provoz, který je však zneužíván pro účely útoku. [49]

Existují tři hlavní typy DDoS útoků:

1. Útok DDoS na aplikační vrstvě
    a. Spočívá v zaplavování serveru řadou legitimně vypadajících požadavků s cílem vyčerpání jeho kapacity a narušení schopnosti reagovat na žádosti. [49][50]
2. Protokolový útok
    a. Tyto útoky jsou také známy jako útoky na vyčerpání stavu, které se snaží způsobit narušení služby přetížením systémových zdrojů zneužíváním firewallu nebo vyvažovačů zátěže. [49][50]
3. Útok objemu
    a. Možná nejběžnější útok díky své jednoduchosti spočívá v pokusu útočníka jednoduše zahltit šířku pásma serveru pomocí botnetu nebo nějaké formy zesilovače, aby vytvořil masivní objem provozu. [49][50]

V případě, že pro provoz herních serverů využívá vývojář vlastní hardware, může být obtížné a náročné účinně snižovat dopady útoků typu DDoS. Tento proces vyžaduje úzkou spolupráci s poskytovatelem internetových služeb a může zahrnovat i aktualizaci hardware a rozložení zátěže mezi různé servery. [3]

Naopak, využití cloudových hostingových řešení, přenáší část odpovědnosti za prevenci útoků typu DDoS na poskytovatele cloudu. Hlavní cloudové hostingové platformy jsou vybaveny určitou úrovní ochrany před útoky DDoS, a existují i specializované cloudové služby pro zmírnění těchto útoků, které lze implementovat. Přesto by mělo být chápáno, že i přes tyto opatření nelze předpokládat absolutní eliminaci potenciálních útoků typu DDoS. Je proto rozumné věnovat čas plánování a testování různých strategií na zmírnění těchto hrozeb. [3]

Potenciální opatření k ochraně před útoky typu DDoS:

- Adaptivní sledování hrozeb v reálném čase
    - Monitorování síťových protokolů může efektivně identifikovat potenciální hrozby prostřednictvím analýzy vzorců síťového provozu. [50]

  - Tato praxe zahrnuje pozornost k detekci nárůstu provozu, sledování jakýchkoli neobvyklých aktivit a adaptaci obran proti anomálním či škodlivým požadavkům, protokolům a blokování IP adres. [50]
- Ukládání do mezipaměti
  - Mezipaměť (cache) funguje tak, že ukládá duplicity vyžadovaného obsahu, čímž umožňuje původním serverům zpracovávat nižší počet požadavků. Implementace sítě pro doručování obsahu (CDN), která ukládá zdroje do mezipaměti, má potenciál snížit zátěž na servery a ztížit jejich přetížení jak legitimními, tak potenciálně škodlivými požadavky. [50]
- Omezení rychlosti
  - Omezení rychlosti představuje opatření, které reguluje objem síťového provozu v daném časovém rámci a efektivně brání zahlcení webových serverů prostřednictvím požadavků z konkrétních IP adres. Tato technika se často využívá k prevenci útoků typu DDoS, které využívají botnety k přetížení koncového bodu nadměrným objemem požadavků v krátkém časovém úseku. [49][50]
- Rozšíření sítě Anycast
  - Síť Anycast přispívá k rozšíření plochy sítě, což umožňuje efektivněji absorbovat náhlé nárůsty objemu provozu. Tím, že rozptyluje provoz mezi více distribuovaných serverů, přispívá k prevenci výpadků a zajišťuje robustnější infrastrukturu schopnou lépe zvládat zvýšený zátěžový provoz. [50]
  - Po odfiltrování části útočného provozu pomocí ostatních nástrojů pro zmírnění DDoS služba Anycast distribuuje zbývající útočný provoz mezi více datových center. Tímto způsobem předejde přetížení jediného místa požadavky. V případě, že kapacita sítě Anycast převyšuje objem útočného provozu, dochází k efektivnímu zmírnění útoku. [49][50][51]

1.4.2 Odposlech paketů (Packet Sniffing)

V běžném síťovém provozu jsou pakety směrovány od svého zdroje k cílové IP adrese přes několik různých bodů. Směrovače na této trase jsou povinni minimálně analyzovat informace v hlavičkách paketů, aby mohly určit optimální směr pro předání paketu. Nicméně vzhledem k otevřené povaze přenášených dat nebrání nic tomu, aby jakýkoliv aktér na trase nemohl zhlédnout obsah konkrétního paketu. Analyzování obsahu paketů na trase může být

prováděno z mnoha různých důvodů, včetně pokusů o krádež přihlašovacích údajů nebo podvádění v síťových hrách. [3][52]

1.4.2.1 Man-in-the-Middle

Při útoku typu man-in-the-middle počítač umístěný někde na trase od zdroje k cíli odposlouchává pakety bez vědomí zdrojového a cílového počítače. V praxi může k útoku dojít několika různými způsoby. Jedním ze způsobů je zachytávání a čtení paketů odcházejících z jakéhokoli počítače, který používá nezabezpečenou nebo veřejnou síť Wi-Fi. [3][52]

Obecným přístupem k boji proti MITM je šifrování všech přenášených dat. V případě síťové hry je třeba před zavedením jakéhokoli šifrovacího systému zvážit, zda daná hra obsahuje citlivá data, která je třeba šifrovat. Pokud hra obsahuje mikrotransakce, při nichž si hráč může zakoupit herní předměty, je bezpodmínečně nutné šifrovat veškerá data související s nákupy. Pokud jsou ukládány nebo jen zpracovány informace o kreditních kartách, může být zákonným požadavkem standard PCI DSS. Nicméně i v případě, že nedochází k nákupům ve hře, by každá hra, ve které se hráč přihlašuje k účtu, který ukládá postup, například MOBA nebo MMO, měla šifrovat data související s procesem přihlášení. V obou těchto případech existuje pro potencionálního útočníka finanční motivace ke krádeži informací – ať už kreditní karty, nebo přihlašovacích údajů. [3][53]

Na druhou stranu, pokud hra přenáší pouze replikační data, která jsou potřebná pro synchronizaci mezi klienty, nebo podobné informace, nezáleží na tom, zda tato data zachytí osoba se špatným úmyslem. V takovém případě mohou být data nešifrovaná, což nebude představovat významný problém. [3]

1.4.2.2 Odposlouchávání paketů na klientském zařízení

V oblasti her, které přenášejí citlivá data, je důležité brát v potaz hrozbu MITM útoků. Nicméně každá síťová hra může být ohrožena možností, že klientský počítač záměrně z paketů těží i informace, které by neměl. Šifrování dat představuje jistou překážku, avšak ne zcela spolehlivé opatření. [3]

Tento aspekt je podložen skutečností, že spustitelnou hru na jakékoli platformě je možné nějakým způsobem rozebrat a analyzovat, byť to může být komplikovaný a časově náročný proces. Uvnitř spustitelného souboru musí existovat kód, který popisuje postup dešifrování dat přijímaných spustitelným souborem. Z čehož vyplívá, že se útočník může

naučit přenášená data dešifrovat. Jakmile je tedy dešifrovací schéma odhaleno, paketová data mohou být čtena, jako by nebyla zašifrována. [3]

Reverzní inženýrství dešifrovacího kódu a odhalení soukromého klíče uloženého v klientovi vyžaduje značný čas a úsilí ze strany potenciálních podvodníků. [3]

Jedním ze způsobů, jak tuto práci zkomplikovat, je pravidelná změna šifrovacích klíčů a paměťových offsetů těchto klíčů. To vyžaduje, aby kdokoli, kdo se pokouší podvádět, opakoval proces reverzního inženýrství při každé aktualizaci hry. Stejně tak, pravidelná změna pořadí a formátu paketů hry, činí podvody, které jsou na těchto datech závislé, nefunkční. To nutí podvodníky vynakládat čas na osvojení nového formátu a opětovné vytvoření podvodných nástrojů. [3]

Stojí za to zvážit, čeho přesně chce hráč sniffováním paketů dosáhnout. Podvodný hráč na klientském zařízení se obvykle snaží využít informačního podvodu, což znamená, že se snaží získat informace, které by neměl znát. Běžným způsobem, jak v tomto případě zabránit podvádění, je omezit množství informací přenášených na jednotlivé klientské počítače. Ve hře typu klient-server je například možné, aby server omezil množství dat, která posílá každému klientovi. Předpokládejme, že síťová hra podporuje nepozorovaný pohyb hráčů ve skrytém režimu. Pokud by server stále odesílal informace o postavě ve skrytém režimu, pak by hráč mohl z paketů absolutně určit polohu těchto skrytých hráčů. Na druhou stranu, pokud se zasílání aktualizací pro pozici pozastaví, zatímco je postava ve skrytém režimu, nebude mít podvodný klient možnost zjistit aktuální pozici postavy. [3]

Tento přístup lze snáze implementovat v topologii klient-server než v topologii peer-to-peer, kde musí všechna důležitá data být odeslána každému partnerovi. Hry s peer-to-peer topologií musí využívat alternativní přístupy k potlačení podvodů. [3]

1.4.3 Škodlivá data

Je důležité počítat s tím, že škodlivý uživatel může usilovat odeslat na server nekorektní nebo nevhodné pakety. Existuje mnoho důvodů pro takové chování, ačkoliv jedním z nejjednodušších může být snaha uživatele server shodit. Zákeřnější útočník se může pokusit způsobit spuštění škodlivého kódu na serveru prostřednictvím technik, jako jsou přetečení vyrovnávací paměti paketu nebo podobné útoky. [3][54]

1.4.4 Validace vstupu

Validace vstupů usiluje o zajištění toho, že žádný hráč neprovádí neplatnou akci. Implementace validace vstupů se zjednodušeně opírá o základní premisu, že hra by neměla slepě provádět akci z paketu poslaného přes síť. Místo toho by měla být akce nejprve ověřena, aby se zajistilo, že je v daném čase platná. [3][49]

Například předpokládejme, že přes síť přijde paket s požadavkem, aby Hráč A vystřelil ze své zbraně. Server by jakožto autoritativní entita nikdy neměl předpokládat, že je platné, aby Hráč A vystřelil. Nejprve by mělo být potvrzeno, že Hráč A má zbraň, zbraň obsahuje náboje atd. Pokud některá z těchto podmínek není splněna, požadavek na výstřel by měl být odmítnut. [3]

Dále by mělo být potvrzeno, že při přijímání akce pro Hráče A jsou tyto akce skutečně odeslány od klienta, který je odpovědný za Hráče A. Například u hry typu klient-server je každá adresa klienta spojena s klientní proxy. Když jsou tedy přijímány pohyby přes síť, server umožňuje aplikaci těchto pohybů pouze na odpovídající proxy. [3]

V případě zjištění neplatných akcí by se zdálo vhodné potrestat provinilého hráče. Nicméně by měla být zohledněna možnost, že neplatný vstup vznikl náhodnou anomálií, například vlivem zpoždění nebo ztráty paketů. Uvažujme například situaci, kde hráči mohou ve hře odesílat kouzla. Předpokládejme, že je také možné, aby hráči „umlčeli“ ostatní hráče, což znamená, že po dobu trvání ticha nemohou odesílat kouzla. Představme si, že hráč A je momentálně umlčen a server mu tedy pošle aktualizační paket. Před tím, než hráč A obdrží paket o umlčení, odešle akci odeslání kouzla. Hráč A by tak mohl odeslat neplatnou akci, avšak ne kvůli nekalým důvodům. Z tohoto důvodu by bylo chybné potrestat hráče A. Objektivně lze říci, že konzervativnější přístup spočívající v jednoduchém odmítnutí paketu by byl lepším postupem, protože neplatný vstup by mohl být legitimním důsledkem nečekaných situací. [3]

1.4.5 Typy podvodů v online hrách

Podvádění v online hrách představuje porušování pravidel nebo herních mechanismů s úmyslem získat neférovou výhodu nad ostatními hráči, často prostřednictvím použití softwaru třetích stran. [3][49][55][56]

Mezi časté formy podvádění v online hrách patří:

- Boti
    - Usnadňují hráčům míření na ostatní hráče ve hrách typu FPS. [3][49][55][56]
    - V MMO hrách jsou boti využíváni k automatizaci činností, jako je zvyšování úrovně postavy nebo získávání herního bohatství, dokonce i v případech, kdy jsou hráči offline nebo pryč od svého počítače. [3][57]
- Mimosmyslové vnímání
    - Umožňují hráčům vidět skrz fyzické překážky či jiné obstrukce jako např. mlha. To jim následně umožní snadno identifikovat své protivníky. [3][49][55][56]
    - Příklad lze popsat na hře typu RTS a modelu peer-to-peer. Zde má každý peer kompletní informace o tom kde se nacházejí všechny jednotky ve hře uloženy v paměti. To znamená, že efekt „fog of war“, nutný pro férovost zápasu, je implementován pouze v lokálním spustitelném souboru. Odstranění této herní mechaniky je možné pouze napsáním podvodného programu. Po odstranění FOW dostává hráč nespravedlivou výhodu, protože může předvídat a reagovat na akce soupeře bez potřeby průzkumu nebo přímého vizuálního kontaktu. Obtížnost jeho odhalení spočívá v tom, že neexistuje mnoho způsobů, jak mohou ostatní hráči detekovat tento typ podvodu. Zde pomáhá až softwarová detekce podvodů, které bude věnována další kapitola. [3][55]
- Podvod související s mobilitou herní postavy
    - Může se jednat o rychlejší pohyb, teleportaci atd. [49][56][58]

2 SÍŤOVÉ KNIHOVNY PRO HERNÍ ENGINE UNITY

2.1 Netcode for GameObjects (NGO)

Netcode for GameObjects představuje vysokoúrovňovou síťovou knihovnu, která byla vyvinuta za účelem usnadnit synchronizaci scén a dat herních objektů (GameObjects) mezi různými klienty a platformami pomocí autoritativních modelů pro klienta nebo server. [59][60][61]

„Díky NGO se mohou vývojáři zaměřit na tvorbu hry místo manipulace s nízko úrovňovými protokoly a síťovými rámci.“ [59]

Dále je možné využít službu Relay od Unity Gaming Services, která představuje cenově výhodnou doprovodnou službu peer-to-peer, umožňující škálovat playtesty a vytvářet hry pro více hráčů bez potřeby investovat do dedicated hostingu. [60]

Netcode for GameObjects podporuje verzi Unity 2021.3 a pozdější a lze jej provozovat na následujících platformách: [59]

- Windows, macOS, and Linux
- iOS and Android
- Platformy XR, běžící na Windows, Android, and iOS operačních systémech
- Konzolové platformy jako PlayStation, Xbox, nebo Nintendo Switch
- WebGL

NGO je nejvhodnější pro hry menšího rozsahu, které se soustředí spíše na spolupráci hráčů (co-op) a doporučený počet současných hrajících uživatelů je 16. [61]

2.1.1 Komponenty

2.1.1.1 NetworkObject

Pro sdílení síťového stavu herního objektu mezi všemi hráči je nutné, aby tento GameObject obsahoval komponentu NetworkObject. Tato komponenta úzce spolupracuje s dalšími síťovými komponentami, jako je například NetworkTransform nebo NetworkBehaviour, které přesně specifikují, jaké informace mají být synchronizovány. [59]

Během procesu inicializace objektu NetworkObject slouží hodnota NetworkObject.GlobalObjectIdHash jako primární identifikátor síťového prefabu, který klienti používají za účelem vytvoření lokálního klonu. [59]

Po lokálním vytvoření je každému objektu NetworkObject přidělen NetworkObjectId, sloužící k propojení objektů NetworkObject v rámci síťového prostředí. [59]

Příklad: Klient A vytvoří síťový prefab s názvem „Enemy“. Objektu je přidělen NetworkObjectId 103. Klient B následně obdrží informaci o vytvořeném objektu „Enemy“ s NetworkObjectId 103. Oba klienti tak vědí, že se jedná o stejný objekt. [59]

Ve výchozím nastavení NetworkObjects vlastní server. Nicméně připojení a autorizovaní klienti mohou NetworkObjects vlastnit také, a to pomocí metody SpawnWithOwnership. Netcode for GameObjects je server-autoritativní, což znamená, že pouze server má oprávnění k vytváření (spawning) a odstraňování (despawning) síťových objektů. [59]

Výchozí metoda NetworkObject.Spawn předpokládá vlastnictví na straně serveru:

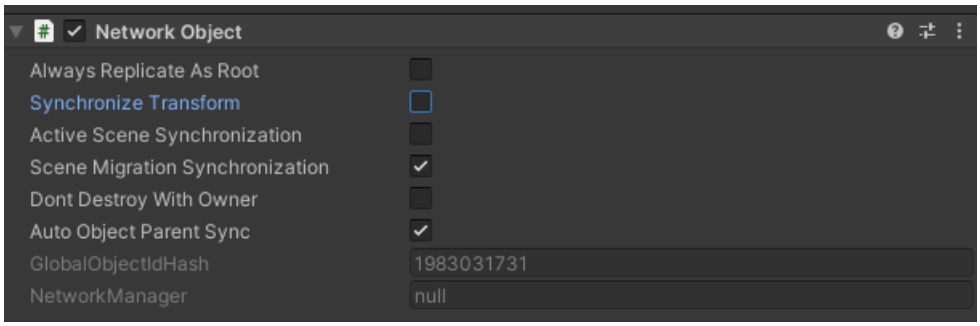
```
GetComponent<NetworkObject>().Spawn();
```

Pro vytvoření NetworkObject s vlastnictvím:

```
GetComponent<NetworkObject>().SpawnWithOwnership(clientId);
```

Pro následnou úpravu vlastnictví nebo jeho odebrání (návrat vlastnictví serveru) u daného NetworkObject:

```
GetComponent<NetworkObject>().ChangeOwnership(clientId);
    GetComponent<NetworkObject>().RemoveOwnership();
```



Obrázek 10. Komponenta NetworkObject uvnitř editoru Unity [59]

Běžně se při zničení vlastníka síťového objektu automaticky zničí i samotný NetworkObject. Pokud to však není požadované chování může se využít vlastnost NetworkObject.DontDestroyWithOwner. Nastavením této vlastnosti na hodnotu true je zajištěno, že vlastněný NetworkObject nebude spolu s vlastníkem zničen. [59]

Na obrázku 10. lze vidět vypnutí možnosti pro synchronizaci transformace. To může přijít v hod v určitých případech kdy je potřeba NetworkObject pro účely, které nevyžadují synchronizaci jeho transformace (umístění a rotace). Tento případ může nastat u NetworkObject, který slouží pouze pro správu herního stavu a jehož synchronizace transformace během hry by byla zbytečná. [59]

Při vytvoření se objekty GameObject umisťují do aktuálně aktivní scény. Může však nastat situace, kdy je zapotřebí změnit aktivní scénu a zároveň zajistit automatický přesun konkrétních instancí NetworkObject do nově nastavené aktivní scény. Ačkoliv je možné vytvořit seznam instancí NetworkObject a naprogramovat jejich migraci do nové aktivní scény, tento přístup může být časově náročný a v komplexnějších projektech obtížně udržovatelný. Doporučený způsob, dle dokumentace NGO, jak docílit automatické migrace konkrétních síťových objektů při změně aktivní scény, spočívá v zapnutí vlastnosti Active Scene Synchronization. Tato vlastnost je u každého NetworkObjectu ve výchozím nastavení vypnutá. [59]

Stejně jako vlastnost NetworkObject.ActiveSceneSynchronization, vlastnost Scene Migration Synchronization zajišťuje automatickou synchronizaci instancí síťových objektů na straně klienta. Tato synchronizace probíhá v případě, kdy jsou objekty přesunuty do jiné scény pomocí funkce SceneManager.MoveGameObjectToScene na hostitelském počítači nebo serveru. Vhodnost použití této vlastnosti se projevuje v případech, kdy je nutný cílený přesun instancí síťových objektů do specifické scény, která momentálně není aktivní. [59]

2.1.1.2 NetworkBehaviour

NetworkBehaviour je abstraktní třída odvozená od MonoBehaviour, sloužící primárně k tvorbě vlastní síťové logiky a herních mechanismů. [59]

Pokud je třída za potřebí, přidává se jako rodič ke třídě, která ji využívá.

```
public class MyClass : NetworkBehaviour
```

Tato třída umožňuje využívat vzdálené volání procedur (RPC) a síťové proměnné (NetworkVariables). Při volání RPC metody nedochází k jejímu lokálnímu spuštění. Místo toho se odešle síťová zpráva obsahující volané parametry, síťové ID (networkId) přidruženého NetworkObject a index samotné NetworkBehaviour komponenty. Síťový objekt totiž

může mít přiřazeno více NetworkBehaviour komponent, přičemž index určuje, na kterou konkrétní komponentu je zpráva cílena. [59]

2.1.1.3 NetworkRigidbody

Netcode for GameObjects nabízí vestavěné řešení se server-autoritativní fyzikou, kdy je fyzikální simulace prováděna pouze na serveru. Pro aktivaci síťové fyziky je k hernímu objektu nutné přidat komponentu síťového pevného tělesa (NetworkRigidbody). [59]

NetworkRigidbody je komponenta, která nastavuje standardní pevné těleso (Rigidbody) herního objektu do kinematického režimu na všech klientech kromě toho, který má nad objektem autoritu. Autorita je určena komponentou síťové NetworkTransform, která musí být povinně připojena ke stejnému hernímu objektu jako NetworkRigidbody. [59]

Ať už je síťová transformace server-autoritativní (výchozí stav) nebo má autoritu vlastník, model autority komponenty NetworkRigidbody ji bude zrcadlit. [59]

Tímto způsobem běží simulace fyziky na autoritativní instanci a výsledné pozice se synchronizují na neautoritativních instancích, přičemž každé jejich pevné těleso zůstává v kinematickém režimu bez vzájemného ovlivňování. [59]

2.1.1.4 NetworkManager

NetworkManager je nepostradatelná komponenta frameworku Netcode for GameObjects. Můžeme si ji představit jako „centrální síťový uzel“ celého projektu, která v sobě sdružuje veškeré nastavení týkající se síťového chování hry [59]

NetworkManager také disponuje referencemi na následující podsystémy: [59]

- NetworkManager.PrefabHandler
    - Umožňuje optimalizovat výkonnost hry pomocí poolů síťových objektů. Díky tomu se při opakovaném načtení těchto objektů nemusí pokaždé vytvářet nový, ale pouze se použije již existující instance z poolu. NetworkPrefabHandler pomáhá s konfigurací a správou těchto poolů.
    - Dává možnost překrýt výchozí chování síťových prefabů. To umožňuje vývojářům definovat vlastní pravidla pro jejich vytváření a synchronizaci v síťovém prostředí.

- NetworkManager.SceneManager
    - Dovoluje načítat a uvolňovat scény na hostiteli a serveru. Tento proces je synchronizován s klienty pro zajištění konzistentního herního stavu napříč sítí.
    - Nabízí možnost registrace pro různé události spojené se scénami, jako je načtení scény, její uvolnění nebo změna aktivní scény. Vývojáři tak mohou reagovat na tyto události a provádět specifické akce, například migraci síťových objektů do nově načtené scény.
    - Poskytuje i další funkce pro správu scén, jako je například získávání informací o aktuálně načtených scénách a manipulaci s pořadím scén v projektu.
- NetworkManager.SpawnManager
    - Umožňuje vytvářet instance síťových objektů na serveru a následně je synchronizovat s klienty.
    - Tato vlastnost zodpovídá nejen za samotné vytváření, ale také za správu životního cyklu síťových objektů. To zahrnuje jejich destrukci a případnou opětovnou tvorbu z poolů objektů pro optimalizaci výkonu.
- NetworkManager.NetworkTimeSystem
    - NetworkTimeSystem tvoří společný časový rámec pro server a všechny připojené klienty. Tento čas sice nemusí odpovídat přesnému reálnému času, ale je konzistentní pro všechny účastníky.
- NetworkManager.NetworkTickSystem
    - Používá se pro úpravu frekvence aktualizace NetworkVariables.

NetworkManager také umožňuje vývojářům snadno spustit herní server a klienty. Nabízí k tomu tři metody: [59]

- StartServer() – slouží ke spuštění samostatného serveru.
- StartHost() – používá se ke spuštění serveru i klienta v rámci stejné instance Unity.
- StartClient() – spouští klienta, který se následně připojí k existujícímu serveru.

Při spouštění klienta v síťové hře vyvinuté v Unity využívá NetworkManager k připojení k serveru IP adresu a port nastavené v komponentě Transport. Tyto hodnoty se dají nastavit jak v editoru, tak i v kódu. [59]

```
NetworkManager.Singleton.GetComponent<UnityTransport>().SetConnectionData(
      "127.0.0.1",       // IP adresa ve formě textového řetězce
      (ushort)12345      // Port ve formě proměnné ushort
);
```
[59]

Odpojení řeší NetworkManager poměrně jednoduchým procesem. Nicméně je třeba mít na paměti jistá omezení. Všem třem režimům (klient, host, server) stačí pro odpojení volání metody NetworkManager.Shutdown. Tato metoda se postará o korektní ukončení síťového spojení a odpojení od serveru. [59]

Po odpojení pomocí NetworkManager.Shutdown již nebudou dostupné žádné síťové subsystémy, včetně NetworkSceneManager. To znamená, že po odpojení nelze provádět žádné akce související se síťovou synchronizací nebo správou síťových objektů. [59]

Po odpojení tedy musí převzít kontrolu ne-síťové komponenty Unity, např. UnityEngine.SceneManagement, který v tomto příkladě načte po odpojení klienta zpět do hlavního menu: [59]

```
public void Disconnect()
{
      NetworkManager.Singleton.Shutdown();

      // Navigace zpět do hlavního menu
      UnityEngine.SceneManagement.SceneManager.LoadScene("MainMenu");
}
```
[59]

V situacích, kdy je nutné odpojit specifického klienta bez přerušení chodu celého serveru, lze využít metodu NetworkManager.DisconnectClient. Tato metoda přijímá jediný parametr ve formě identifikátoru klienta, který má být odpojen.

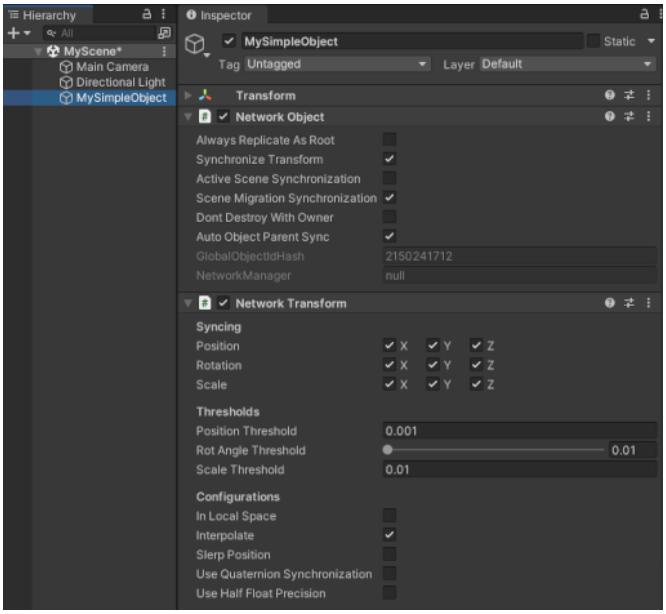
```
void DisconnectPlayer(NetworkObject player)
{
      NetworkManager.DisconnectClient(player.OwnerClientId);
}
```
[59]

2.1.1.5 NetworkTransform

Komponenta NetworkTransform představuje zásadní prvek pro synchronizaci polohy, rotace a měřítka herních objektů v rámci síťové hry. Tato komponenta je klíčová pro efektivní a spolehlivou replikaci transformačních dat mezi klienty a serverem. [59]

Dále NetworkTransform využívá techniky interpolace a extrapolace k plynulému zobrazování pohybu objektů i v případě zpoždění (latence) v síti. To zajišťuje, že pohyb objektů je pro všechny klienty plynulý a bez trhání. Podporuje také predikci pohybu, která umožňuje klientům předvídat budoucí polohu objektů na základě jejich minulého pohybu. To pomáhá snižovat dopady zpoždění a zlepšovat plynulost hry. [59]

NetworkTransform se snadno používá a integruje do herních objektů. Stačí přidat komponentu NetworkTransform k objektu, který už disponuje komponentou NetworkObject, a nastavit požadované parametry. [59]

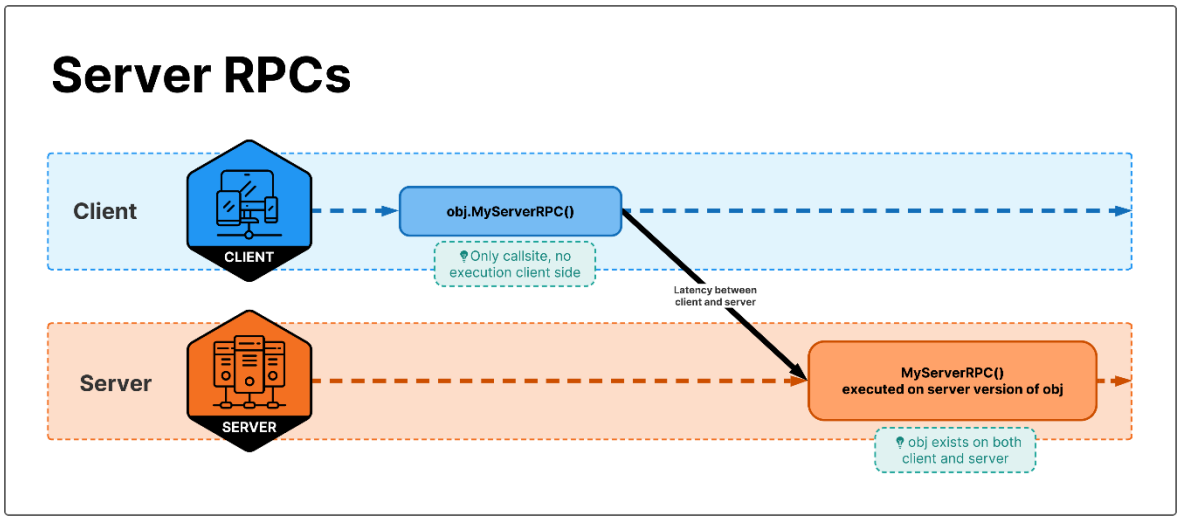


Obrázek 11. Komponenta NetworkTransform uvnitř GameObject [59]

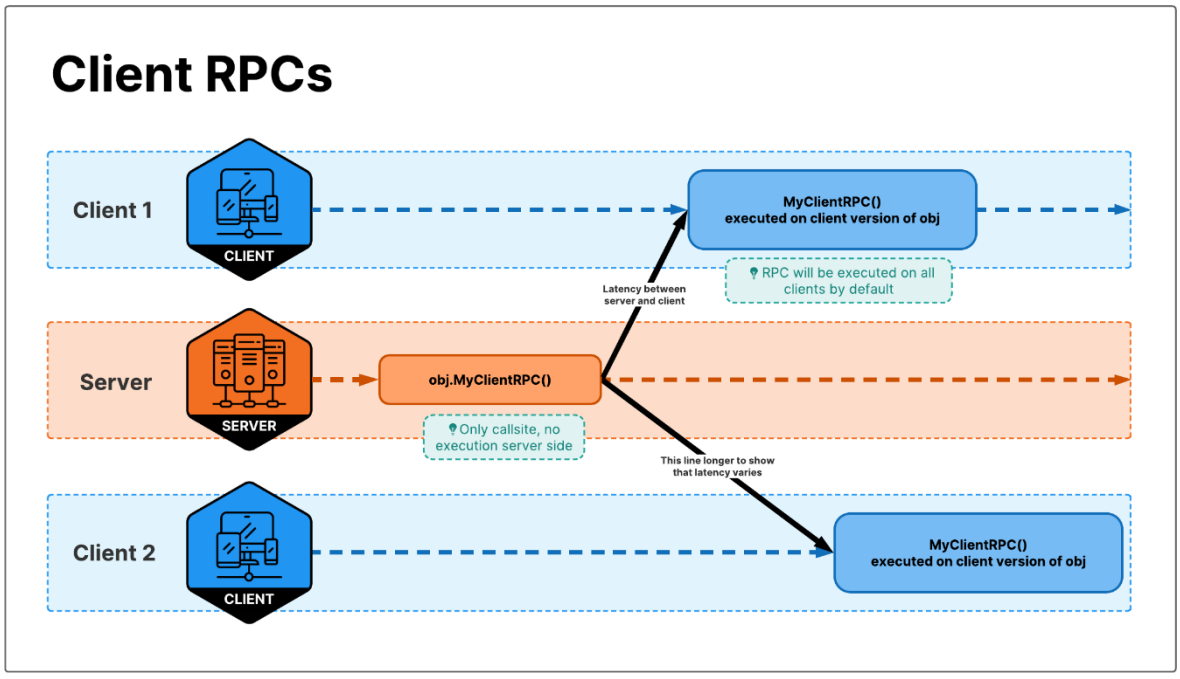
2.1.2 Vzdálené volání procedury

Jedná se o mechanismus, umožňující serveru nebo klientovi v rámci síťové hry spustit metodu na jiném klientovi, popř. serveru. [59]

RPC lze deklarovat označením metody atributem [Rpc] a zahrnutím přípony Rpc do názvu metody. RPC má několik možných cílů, které lze deklarovat jak v době běhu, tak v době kompilace, ale v atributu [Rpc] musí být vždy zadán výchozí cíl. [59]



Obrázek 12. Grafické znázornění RPC vykonávaného na serveru [59]



Obrázek 13. Grafické znázornění RPC vykonávaného na klientech [59]

2.1.2.1 Deklarace

Například vytvoření RPC, které má být prováděno na serveru, by se deklarovalo následovně:

```
[Rpc(SendTo.Server)]
public void PingRpc(int pingCount) { /* ... */ }
```
[59]

Pokud by bylo zapotřebí vykonat RPC na všech klientech:

```
[Rpc(SendTo.Server)]
public void PingRpc(int pingCount)
{
	// Server -> Klienti, jelikož PongRpc posílá všem kromě serveru
	// (SendTo.NotServer)
	PongRpc(pingCount, "PONG!");
}

[Rpc(SendTo.NotServer)]
void PongRpc(int pingCount, string message)
{
	// Něco vykonej
}

void Update()
{
	if (IsClient && Input.GetKeyDown(KeyCode.P))
	{
		// Klient -> Server (SendTo.Server)
		PingRpc();
	}
}
```
[59]

Lze samozřejmě cílit i na konkrétního klienta za použiti RpcParameters a SendTo.SpecifiedInParams:

```
[Rpc(SendTo.Server)]
public void PingRpc(int pingCount, RpcParams rpcParams)
{
	PongRpc(
	pingCount,
	"PONG!",
	RpcTarget.Single(rpcParams.Receive.SenderClientId, RpcTargetUse.Temp)
	);
}

[Rpc(SendTo.SpecifiedInParams)]
void PongRpc(int pingCount, string message, RpcParams rpcParams)
{
	// Něco vykonej
}
```
[59]

2.1.3 NetworkVariables

Na vyšší úrovni je NetworkVariable způsob synchronizace vlastnosti (proměnné) mezi serverem a klienty bez nutnosti používat vlastní zprávy nebo RPC. [59]

Jelikož je NetworkVariable obal (kontejner) uložené hodnoty typu T, je nutné použít vlastnost NetworkVariable.Value k přístupu ke skutečné synchronizované hodnotě. Hodnota NetworkVariable.Value je synchronizována: [59]

- S nově připojenými klienty (tzv. „Late Joining Clients“)
  - Když je přidružený NetworkObject s NetworkBehaviour s vlastnostmi NetworkVariable vytvořen, je aktuální stav (hodnota) jakéhokoliv NetworkVariable automaticky synchronizován na straně klienta.
- S připojenými klienty
  - Když se hodnota NetworkVariable změní, budou o této změně informováni všichni připojení klienti, kteří se před změnou hodnoty přihlásili k události NetworkVariable.OnValueChanged.

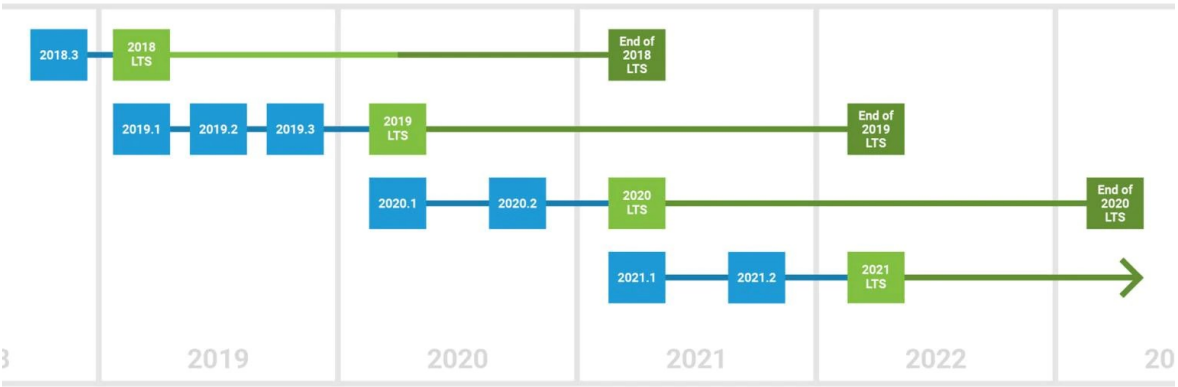
2.1.3.1 Podporované typy hodnot

NetworkVariable poskytuje podporu pro následující typy: [59]

- Jednoduché typy hodnot v jazyce C#
  - bool, byte, sbyte, char, decimal, double, float, int, uint, long, ulong, short, and ushort
- Typy hodnot věstavěné v Unity
  - Vector2, Vector3, Vector2Int, Vector3Int, Vector4, Quaternion, Color, Color32, Ray, Ray2D
- Typy hodnot enum
- Libovolný typ, který implementuje rozhraní INetworkSerializable.
- Libovolný strukturovaný typ, který implementuje rozhraní INetworkSerializeBy-Memcpy.
- „Fixed string“ typy v Unity
  - FixedString32Bytes, FixedString64Bytes, FixedString128Bytes, Fixed-String512Bytes, FixedString4096Bytes

2.2 Mirror Networking

Mirror je vysokoúrovňová síťová knihovna, vydaná v roce 2018. Byla vytvořena s cílem opravit opomíjený UNET (oficiální síťová implementace od Unity do verze 2019.3) a poskytnout platformu pro další vývojáře, kteří by mohli rozšířit jeho funkcionality, předkládat vylepšení a spolupracovat. Byla vystavěna na nízko úrovňové knihovně Telepathy a je koncipována tak, aby vývojářům poskytovala jednoduchý a efektivní způsob implementace síťových funkcí ve svých hrách. [40][49][62]



Obrázek 14. Podporované verze Unity pro knihovnu Mirror [62]

Mirror využívá síťovou strukturu Command-RPC, kde každé spojení vyžaduje autoritu nad objektem pro jeho úpravu v síti. „Je vynikající volbou pro malá studia i velké AAA studia, která chtějí vytvořit kvalitní síťové řešení na úrovni MMO.“ [40][49]

Architektura Command-RPC znamená, podobně jako u již probírané knihovny NGO, že klienti a servery komunikují prostřednictvím příkazů (Client-to-Server) a vzdálených volání procedur (Server-to-Client). Příkazy umožňují klientům provádět vzdálenou metodu na serveru vůči objektům, nad kterými mají autoritu. Jedná se o způsob, jak klient požádá server o provedení určitého příkazu. Naopak RPC umožňují serveru provádět metody na klientovi vzdáleně. Tato vzájemná komunikace mezi serverem a klientem tvoří základ pro síťové dorozumění, což umožňuje posílat širokou škálu podporovaných datových typů a odkazů po síti. [49]

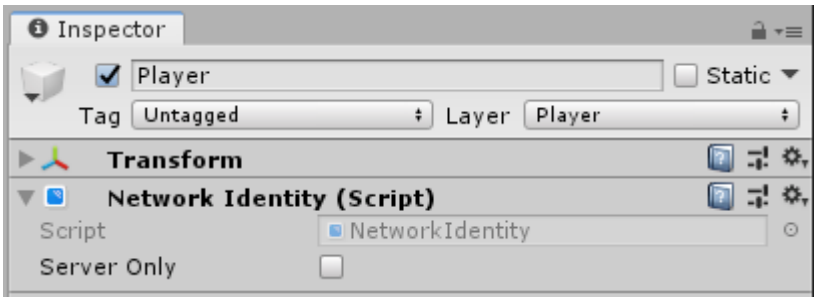
Skvělou vlastností Mirroru je také to, že klient i server jsou součástí jediného projektu, což přispívá ke zvýšení produktivity během vývoje. [49][62]

2.2.1 Komponenty

2.2.1.1 NetworkIdentity

Komponenta Network Identity je jádrem síťového vysokoúrovňového API pro Unity. Kontroluje jedinečnou identitu herního objektu v síti a využívá tuto identitu k tomu, aby systém věděl o existenci daného herního objektu. [62]

Disponuje i možností vytvořit GameObject, ke kterému je přidána, pouze na serveru:



Obrázek 15. Komponenta NetworkIdentity v Unity editoru [62]

To se může hodit v případech kdy si server potřebuje držet objekty, které řídí herní pravidla, skórování nebo události ve světě, které nemusí být vizualizovány pro klienty, ale potřebují být na serveru pro výpočty a aktualizace.

Popř. komplexní umělá inteligence – složité chování umělé inteligence, které vyžaduje přístup ke kompletnímu stavu hry nebo provádí výpočty, které by byly neefektivní na jednotlivých klientech, můžou se tedy provádět a udržovat pouze na straně serveru. Klienti pak obdrží aktualizace polohy a akcí NPC ze serveru.

Je důležité poznamenat, že Mirror nepodporuje síťové identity na vnořených GameObjectech. Podřazené GameObjecty mohou přistupovat k síťové identitě rodiče pomocí vestavěného skriptovacího funkce GetComponentInParent v Unity. [62]

V síťovém prostředí Mirroru musí server vytvářet síťové GameObjecty s NetworkIdentity pomocí metody NetworkServer.Spawn. Tímto způsobem jsou tyto objekty automaticky vytvořeny na klientech připojených k serveru a je jim přiřazen identifikátor netId. [62]

Pro správné fungování sítě je nutné umístit komponentu Network Identity na všechny Prefaby, které jsou vytvářeny za běhu. [62]

2.2.1.2 NetworkManager

Network Manager je komponenta určená pro správu důležitých síťových aspektů multiplayerové hry vyvíjené pomocí knihovny Mirror. Seskupuje množství užitečných funkcí do jednoho místa a zjednodušuje vytváření, provoz a ladění multiplayerových her na maximum možností. [62]

Mezi funkce Network Manageru patří: [62]

- Správa herního stavu
- Správa spawnování
- Správa scén
- Poskytování ladících informací
- Možnost úprav

V každé scéně může být vždy pouze jeden aktivní Network Manager, jelikož funguje jako singleton. [62]

NetworkManager také umožňuje jednoduchou záměnu transportního protokolu. Oddělení transportního protokolu do vlastní komponenty umožňuje vývojářům her vybrat takový, který nejlépe vyhovuje potřebám jejich hry. Pro změnu protokolu je zapotřebí výměna komponenty na objektu NetworkManager a následné přiřazení do pole Transport. [62]

Hra vyvíjená v knihovně Mirror může běžet ve třech režimech – jako klient, jako dedikovaný server nebo jako hostitel, který je současně klientem a serverem. [49][62]

Pro spouštění těchto režimů slouží podobně jako u knihovny NGO tyto tři metody: [62]

- StartServer()
- StartHost()
- StartClient()

V režimu klienta se hra pokouší připojit k síťové adrese, která je specifikována. Lze také použít plně kvalifikované doménové jméno (*.com). V režimu serveru nebo hostitele hra naslouchá příchozím spojením na localhost, což zahrnuje lokální síťovou IP adresu serverového stroje. [62]

NetworkManager také slouží k řízenému vytváření NetworkObjectu z Prefabu, pro tento účel obsahuje slot pro hráčský Prefab. Pokud je nastaven hráčský Prefab, je automaticky vytvořen GameObject z tohoto Prefabu pro každého připojeného uživatele ve hře. To platí pro lokálního hráče na hostovaném serveru a vzdálené hráče. Před přiřazením této položky je nutné k hráčskému Prefabu připojit komponentu NetworkIdentity. [62]

Po zastavení hry jsou všechny hráčské objekty zničeny. Pokud je spuštěna další kopie hry a připojí se klient, NetworkManager vytváří další hráčský GameObject. Po zastavení tohoto klienta se zničí i jeho GameObject. [62]

Kromě hráčského Prefabu je nutno také registrovat další Prefaby, které se budou dynamicky vytvářet během hraní. Ty jsou definovány v poli Registered Spawnable Prefabs v rámci komponenty NetworkManager. Předlohy lze také registrovat v kódu pomocí metody NetworkClient.RegisterPrefab. [62]

Jelikož většina her obsahuje více než jednu scénu je NetworkManager navržen tak, aby automaticky řídil stav scény a přechody mezi scénami v síťovém prostředí. [62]

V komponentě NetworkManageru jsou dva sloty pro scény: [62]

- Offline scéna
- Online scéna

Když je server nebo host spuštěn, je načtena Online scéna. Tato scéna se pak stává aktuální síťovou scénou. Jakýkoli klient, který se připojí k tomuto serveru, je instruován, aby také načetl tuto scénu. [62]

Když je připojení přerušeno, buďto zastavením serveru nebo odpojením klienta, je načtena Offline scéna. To například umožňuje, aby se hra automaticky vrátila do hlavního menu nebo začáteční scény po odpojení z multiplayerové hry. Také je možné změnit scény během aktivní hry voláním ServerChangeScene. To způsobí, že všichni aktuálně připojení klienti také změní scénu. [62]

2.2.1.3 NetworkTransform

Komponenta NetworkTransform umožňuje replikovat transformaci připojeného herního objektu pro další připojené uživatele. Tato komponenta vyžaduje připojení komponenty NetworkIdentity k danému hernímu objektu (řešeno automaticky, pokud není nalezena). [49][62]

V současnosti Mirror poskytuje dvě varianty síťové transformace: [62]

- Spolehlivá (Reliable)
  - nízké využití šířky pásma, stejná latence jako Rpcs/Cmds atd.
- Nespolehlivá (Unreliable)
  - vysoké využití šířky pásma, extrémně nízká latence

Zde dokumentace Mirror Networking doporučuje používat NetworkTransformReliable, pokud není zapotřebí opravdu velmi nízké latence. [62]

Ve výchozím nastavení je síťová transformace server-autoritativní, pokud není provedena změna v poli synchronizačního směru na „Client To Server“. Autorita klienta se poté vztahuje jak na GameObjecty, tak na ne-hráčské objekty, které byly explicitně přiřazeny klientovi. S tímto nastavením jsou změny pozice odesílány z klienta na server. [49][62]

2.2.2 Atributy

Síťové atributy jsou přidávány k členským funkcím skriptů, které dědí od třídy NetworkBehaviour, aby bylo možné určit, zda mají být spuštěny na klientovi nebo serveru.

- [Server] – Metodu může volat pouze server (vyvolává varování, když je volána na klientovi).
- [Client] – Metodu může volat pouze klient (vyvolává varování, když je volána na serveru).
- [Command] – Tato funkce je volána na klientovi, a spouští se na serveru. Zavolat tuto funkci ze serveru není možné.

- [ClientRpc] – Server používá vzdálené volání procedur (RPC) k provedení této funkce na klientech.
- [TargetRpc] – Tento atribut umožňuje spouštění funkcí na klientech ze serveru. Na rozdíl od atributu [ClientRpc] jsou tyto funkce volány na jednotlivém cílovém klientovi, nikoli na všech připojených klientech.
- [SyncVar] – Používá se k synchronizaci proměnné ze serveru na všechny klienty. Tyto hodnoty by neměly být nulové. Lze použít typy:
    - int, long, float, string, Vector3 atd. (jednoduché typy)
    - Síťovou identitu
    - GameObject

2.2.3 Časová synchronizace

Pro mnoho algoritmů je potřeba, aby byl čas synchronizován mezi klientem a serverem. Knihovna Mirror tento proces provádí automaticky. [62]

Pro získání aktuálního času se použijte tento kód: [62]

```
double now = NetworkTime.time;
```

Tento kód vrátí stejnou hodnotu jak na klientovi, tak na serveru. Tato proměnná začíná na hodnotě 0 při spuštění serveru. Čas je typu double a je nutné podotknout, že by nikdy neměl být přetypován na float. Přetypování na float znamená, že hodiny po určité době ztrácejí přesnost: [62]

- po 1 dni je přesnost v rozmezí 8 ms
- po 10 dnech je přesnost v rozmezí 62 ms
- po 30 dnech je přesnost v rozmezí 250 ms
- po 60 dnech je přesnost v rozmezí 500 ms

Mirror má také možnost vypočítat RTT, jak jej vidí aplikace: [62]

```
double rtt = NetworkTime.rtt;
```

2.2.4 Správa zájmů (Interest Management)

Při vytváření multiplayerových her je prvním zjevným přístupem jednoduše vysílat stav světa každému hráči. Což je základní chování knihovny Mirror, pokud nejsou použity žádné komponenty pro správu zájmu. [62]

Možnost mít stovky hráčů připojených k herní seanci je díky knihovně Mirror dosažitelná, ale to neznamená, že všichni připojení klienti budou schopni zvládnout takový počet aktivních hráčů najednou. Na zpracování hráčských objektů se podílí mnoho procesů, od fyziky přes vykreslování až po obecnou herní logiku. V mnoha případech je tedy vhodné omezit, kolik hráčů může klient vidět, např. pouze ty kteří jsou v jeho bezprostřední blízkosti. Z jiného úhlu pohledu, může titulu kompetitivnějšího charakteru pomoci, omezit podvodné chování tím, že nebudou posílány údaje o polohách všech hráčů na herní mapě. Pokud je celý stav světa znám v paměti, mohou hackeři tuto informaci např. využít k zobrazení hráčů za zdí. [49][62]

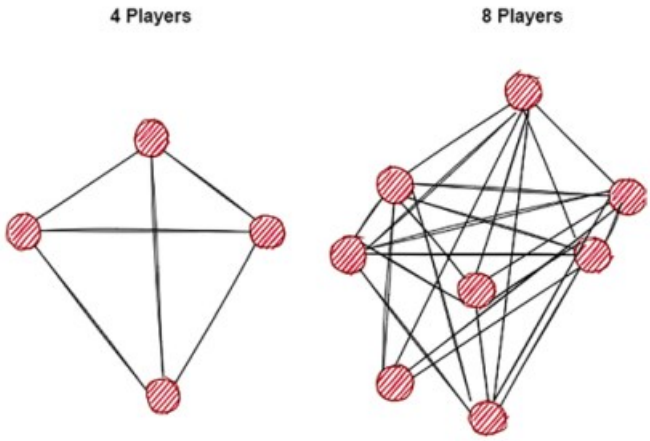
2.2.4.1 Správa zájmů založená na vzdálenosti (Distance Interest Management)

Správa zájmu pomocí vzdálenosti je tradiční a jednoduchý přístup k řešení této problematiky. Pro zprovoznění je přidána komponenta pro správu zájmu podle vzdálenosti ke komponentě NetworkManager. [40][49][62]

Obrázek 16. Komponenta pro Distance Interest Management [49]

- Vis Range (Rozsah viditelnosti)
  - Definuje poloměr kolem hráče, ze kterého přijímá aktualizace světa. [40][49][62]
- Rebuild Interval (Interval obnovování)
  - Je vyjádřen v sekundách a určuje, jak často Mirror znovu vypočítává viditelnost objektů pro klienty. [40][49][62]

Hlavní problém ohledně vzdálenostní správy zájmu je, že se příliš dobře neškáluje. Každý hráč provádí kontrolu vzdálenosti proti každému jinému hráči. [40][49]

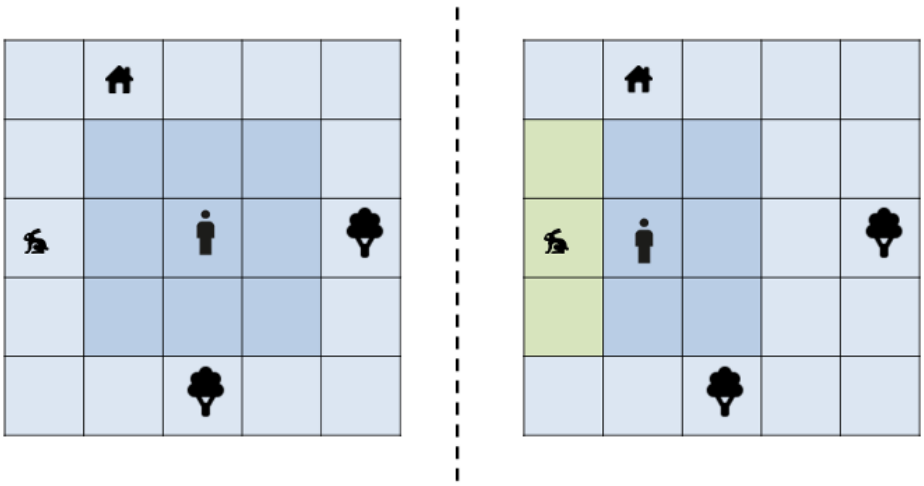


Obrázek 17. Znázornění problému při škálování u vzdálenostní správy zájmu [49]

Jak lze vidět, přidání více hráčů věci komplikuje. Tato špatná škálovatelnost výrazně omezuje výkon. [49]

2.2.4.2 Prostorové hashování (Spatial Hashing)

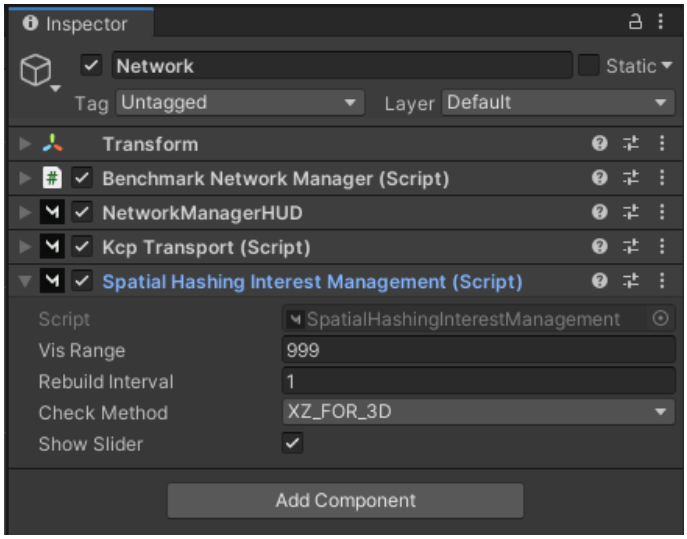
Na místo používání Vector3.Distance k ověření každého vytvořeného objektu proti každému připojení jako u vzdálenostní správy zájmů, je v této správě zájmů svět rozdělen do mřížky. Z této mřížky jsou klientovi sdíleny pouze sousední mřížky, dle tohoto systému se určuje, která připojení jsou v dosahu a mají obdržet aktualizace stavu. [49][62]



Obrázek 18. Grafické znázornění prostorového hashování [63]

Každému mřížkovému bloku je přiřazena hash hodnota. Poté může server porovnat viditelné hashe každého hráče s ostatními síťovými entitami ve scéně a určit, které stavy by měly být synchronizovány. [49]

Implementace prostorového hashování je velice přímá; jde opět o pouhé přiřazení komponenty Spatial Hashing Interest Management k NetworkManageru. [49][62]



Obrázek 19. Komponenta pro Spatial Hashing uvnitř Unity editoru [62]

- Vis Range (Rozsah viditelnosti)
  - Rozsah viditelnosti, ve které by měly probíhat aktualizace stavu.
- Rebuild Interval (Interval obnovování)
  - Interval obnovování v sekundách, který určuje, jak často budou probíhat kontroly změn v pozici mřížky.
- Check Method (Metoda kontroly)
  - Metoda kontroly určuje, zda je mřížka na rovině XZ nebo XY.
- Show Slider (Zobrazit posuvník)
  - Přepne testovací posuvník, kterým je možné upravovat rozsah viditelnosti.

2.2.4.3 Ostatní správy zájmů

- Scénová správa zájmu (Scene Interest Management)
    - Používá se s přídavnými scénami k NetworkObjectům ve vedlejších scénách s izolací fyziky. To znamená, že i když máte na serveru několik instancí stejné podscény, srážky a podobné události mezi objekty se dějí pouze v rámci této podscény, aniž by narušovaly ostatní. [62]
- Týmová správa zájmu (Team Interest Management)
    - Umožňuje, aby byly NetworkObjecty viditelné pouze pro ty, kteří jsou ve stejném týmu. Tento tým se určuje pomocí hodnoty TeamId v komponentě NetworkTeam na hráčském objektu. [62]

2.2.5 SyncVars

SyncVars jsou vlastnosti tříd, které dědí od NetworkBehaviour a jsou synchronizovány ze serveru na klienty. Když je vytvořen GameObject nebo se k probíhající hře připojí nový hráč, jsou jim zaslány nejnovější stavy všech SyncVars na síťových objektech, které jsou jim viditelné. Je důležité zmínit že pouze server může upravit synchronizovanou vlastnost. [49][62]

Stav SyncVars je aplikován na GameObjecty klientů předtím, než je zavolána metoda OnStartClient(), takže stav objektu je vždy aktuální když se tato metoda spouští (nově připojený uživatel má k dispozici aktuální informace). [62]

SyncVars mohou používat libovolný typ podporovaný knihovnou Mirror. Na jednom skriptu NetworkBehaviour můžete mít až 64 SyncVars. [62]

SyncVars také fungují s dědičností tříd:

```
class Pet : NetworkBehaviour
{
    [SyncVar]
    public string name;
}

class Cat : Pet
{
    [SyncVar]
    public Color32 color;
}
```
[62]

2.2.6 SyncVar Hooks

SyncVar Hooks jsou elegantním řešením pro detekci aktualizací proměnných klientů v síti. Umožňují se „zakotvit“ do proměnné, což vyvolá upozornění pokaždé, když se tato proměnná změní a zavolá vývojářem určenou metodu. [49][62]

Pro použití SyncVar Hooks je potřeba použití atributu SyncVar s dodatečným krokem, a to doplnění názvu metody, která se má provést při změně proměnné: [62]

```
public class PlayerController : NetworkBehaviour
{
	[SyncVar(hook = nameof(SetColor))]
	Color playerColor = Color.black;

	// Materiál pro hráče
	Material playerMaterial;

	public override void OnStartServer()
	{
		base.OnStartServer();

		// Změna barvy -> změna SyncVar -> vyvolání metody SetColor
		playerColor = Random.ColorHSV(0f, 1f, 1f, 1f, 0.5f, 1f);
	}

	// Na vstupu jsou dva parametry (musí být stejného typu jako SyncVar)
	// 1. parametr – předešlá hodnota
	// 2. parametr – nová hodnota
	void SetColor(Color oldColor, Color newColor)
	{
		if (playerMaterial == null)
			playerMaterial = GetComponent().material;

		// Nastavení nové barvy do materiálu hráče
		playerMaterial.color = newColor;
	}
}
```

[49][62]

2.3 Photon (Fusion 2)

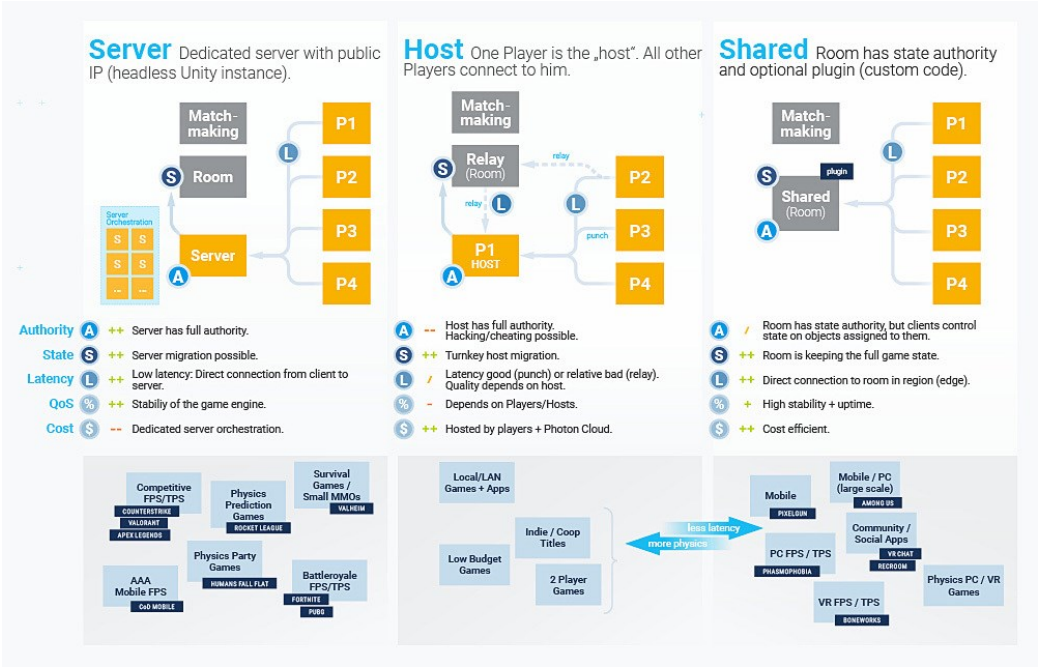
Photon je framework pro vývoj real-time multiplayer her. Skládá se ze serveru a několika klientovských SDK pro různé použití, programovací jazyky a enginy. [64]

Od roku 2023 byla, do té doby hlavní síťová knihovna firmy Photon, PUN 2 přesunuta do „režimu údržby“. Zásadní aktualizace nebo podpora nových verzí Unity po verzích 2022.x nejsou plánovány. Z vyjádření firmy Photon vzešlo, že oprava všech nedostatků, které současná implementace obsahuje by mohla vzniknout nová knihovna. Což vedlo k vytvoření novějšího a vylepšeného Fusion SDK. [64]

Fusion je nová síťová knihovna pro synchronizaci stavu v Unity. „Fusion je navržen s jednoduchostí na paměti, aby se přirozeně integroval do běžného pracovního postupu v Unity, zároveň nabízí pokročilé funkce jako komprese dat, předpovídání na straně klienta a kompenzace zpoždění“. [65]

API Fusionu je navrženo tak, aby bylo podobné běžnému kódu Unity MonoBehaviour. Například RPC a síťový stav jsou definovány atributy na metodách a vlastnostech MonoBehaviours bez nutnosti explicitního kódu serializace, a síťové objekty mohou být definovány jako prefaby pomocí všech nejnovějších funkcí prefabu Unity, jako jsou vnoření a varianty. [65]

2.3.1 Topologické režimy



Obrázek 20. Topologie knihovny Fusion Network [65]

2.3.1.1 Serverový režim

V režimu serveru má server plnou a exkluzivní autoritu nad všemi objekty, bez výjimek. Klienti mohou modifikovat síťové objekty pouze tím, že odesílají své vstupy serveru (a ten na ně reaguje) nebo požadují změnu pomocí RPC. [65]

Serverová aplikace je vytvořena z Unity projektu a spouští se jako kompletní headless verze hry (bez grafického rozhraní). Tato headless verze musí být hostována na serverovém stroji nebo v cloudu. [65]

2.3.1.2 Hostitelský režim

V tomto režimu funguje hostitel jako server i klient. Celkově je tento režim ekvivalentní režimu serveru, ale je mnohem levnější na provoz, protože nevznikají žádné náklady na hostování dedikovaného serveru. To však přichází za cenu důvěryhodnosti autority stavu – jinými slovy, nepoctivý hostitel může podvádět. [65]

Protože je relace vlastněna hostitelem, bude ztracena, pokud se hostitel odpojí. Fusion poskytuje mechanismus migrace hostitele, který umožňuje přenos autority stavu na nového klienta v situacích jako je tato, avšak to není automatické v režimu hostitele (na rozdíl od režimu sdíleného) a vyžaduje speciální zacházení v klientském kódu. [65]

2.3.1.3 Sdílený režim

Zde je autorita nad síťovými objekty distribuována mezi všechny klienty. Konkrétně každý klient má ze začátku autoritu nad objekty, které vytvoří. Klienti však mají i možnost tuto autoritu uvolnit a předat ji jiným klientům. Volitelně může být klientům umožněno, aby si autoritu stavu převzali kdykoliv. [65]

Funkce jako je předpovídání na straně klienta a rollback nejsou v tomto režimu k dispozici. Simulace vždy postupuje vpřed se stejným tikovým rychlostem na všech klientech. [65]

Síťová relace v režimu sdíleného módu je vlastněna Photon cloudem a zůstává aktivní tak dlouho, dokud je k ní připojen alespoň jeden klient. [65]

2.3.2 Komponenty

2.3.2.1 NetworkRunner

NetworkRunner je centrální komponenta Fusion, která představuje jednoho síťového klienta. Všechna komunikace, připojování, vytváření, simulace a replikace stavu jsou zorchestrovány touto komponentou. V jedné instanci Unity lze spustit více instancí NetworkRunner, přičemž každá z nich představuje jednotlivého klienta. [65]

Network Runner GameObject lze vytvořit třemi způsoby: [65]

- Vytvořený z prefabu za běhu
- Načtený jako objekt scény
- Dynamicky vytvořený za běhu přidáním komponenty NetworkRunner k hernímu objektu

Jakmile je instance NetworkRunneru vytvořená může vytvořit/připojit se k místnosti. [65]

Volání metody StartGame() z instance NetworkRunneru vytváří klienta, který se připojí nebo vytvoří místnost podle argumentu StartGameArgs. [65]

```
NetworkRunner.StartGame(new StartGameArgs
{
      // Parametry pro nastavení hry
      SessionName = [string],
      SessionProperties = [Dictionary<string, SessionProperty>],
      CustomLobbyName = [string],
      EnableClientSessionCreation = [bool],
      PlayerCount = [int],
      IsOpen = [bool],
      IsVisible = [bool],
      MatchmakingMode = [MatchmakingMode],
});
```
[65]

V režimu jednoho hráče se žádné připojení k Photon serverům nevytváří a žádná místnost není vytvořena. V režimu více připojených klientů jsou všechny objekty scény a vytvořené objekty umístěny jako potomci dedikovaného GameObjectu a jsou přidány do PhysicsScene/PhysicsScene2D přidružené k tomuto NetworkRunnerovi (klientovi). [65]

NetworkRunner lze použít pouze jednou. Jakmile se tento NetworkRunner odpojí z herní session nebo se mu nepodaří připojit, měl by být zničen, a na jeho místě vytvořena nová instance NetworkRunneru k zahájení nových herních sezení. [65]

Kromě případů dedikovaného serveru nebo sdíleného herního serveru se předpokládá, že všichni vrstevníci představují lidské hráče, kteří poskytují vstupy. Každý NetworkRunner má proto přidruženou strukturu PlayerRef. Místní hodnota PlayerRef je vrácena pomocí Runner.LocalPlayer. V případech, kdy není k dispozici žádný hráč (dedikovaný server nebo sdílený herní server), je hodnota PlayerRef.None. [65]

PlayerRef se používá k označení toho, který klient má vstupní a stavovou autoritu síťových objektů a slouží k zaměřování klientů pro vzdálené volání procedur. [65]

2.3.2.2 NetworkObject

Jedná se o komponentu, která je přidělována GameObjectu. Dohromady poté představují jednu síťovou entitu v místnosti. [65]

Server této komponentě přiřazuje hodnotu NetworkId, což je unikátní celočíselný identifikátor pro tento objekt v místnosti. NetworkId je konzistentní mezi všemi klienty a slouží k odkazování na objekt v síti. [65]

NetworkObjecty mohou výt vytvořeny následujícími způsoby: [65]

- Voláním metody Runner.Spawn()
    - Při volání metody Spawn se vytvoří GameObject s komponentou NetworkObject, a akce Spawn je replikována v síti.
    - INetworkObjectProvider, který byl předán ve volání Runner.StartGame(), obsahuje implementaci toho, jak je nový objekt vytvořen:
        - Vytvoření Prefabu.
        - Klonování existujícího objektu.
        - Vytvoření nového vlastního objektu pomocí kódu nebo vytažení objektu z poolu.
- Načtením scény, která už obsahuje síťové objekty.

Oba způsoby vedou k vytvoření a připojení objektu na server a následnou replikaci jeho stavu všem zainteresovaným klientům. [65]

Kterým hráčům se síťový objekt replikuje, lze korigovat pomocí různých mechanismů správy zájmů, jako je například oblast zájmu (Area Of Interest). Správa zájmů byla již rozebírána u knihovny Mirror, jedná se o mechanismus redukce dat, který vylučuje aktualizace objektů pro určité hráče, což je velmi důležité pro hry s velkým počtem síťových objektů a/nebo vysokým počtem hráčů. [65]

Fusion podporuje i vnořování síťových objektů. Vnoření zde znamená, že jeden NetworkObject je potomkem jiného v hierarchii. Po připojení je každý síťový objekt zpracováván jako samostatná entita, je tedy oddělen od svého rodiče a dětských síťových objektů. [65]

2.3.2.3 NetworkBehaviour

NetworkBehaviour je odvozena od třídy MonoBehaviour a rozšiřuje ji o síťovou funkcionalitu: [65]

- Odkaz na příslušný NetworkRunner pomocí vlastnosti „Runner“.
- Odkaz na příslušný NetworkObject pomocí vlastnosti „Object“.
- Zpracování pro „Networked Properties“. (synchronizované síťové proměnné)
- Zpracování pro vzdálené volání procedur. (RPC)
- Callback virtuálních událostí:
    - `Spawned();`
    - `Despawned(NetworkRunner runner, bool hasState);`
    - `FixedUpdateNetwork();`
    - `Render();`

Na NetworkObject a jeho potomkové transformace lze přidat libovolný počet komponent NetworkBehaviour. [65]

Každá komponenta NetworkBehaviour připojená k síťovému objektu má jedinečný síťový identifikátor. Tento identifikátor může být samotný síťován k odkazování na NetworkBehaviour pomocí NetworkProperty nebo RPC. [65]

2.3.3 Transferování dat

V rámci knihovny Fusion existují tři hlavní způsoby přenosu dat mezi klienty: [65]

1. Networked Properties
2. Remote Procedure Calls
3. Vstup od hráče – Relevantní pouze pro režim server-klient, není použito v režimu sdíleného serveru.

2.3.3.1 Networked Properties

Networked Properties jsou vlastnosti třídy odvozené od NetworkBehaviour, které definují síťový stav přidruženého síťového objektu. [65]

Pro definici je přidán atribut [Networked] k vlastnosti třídy odvozené od NetworkBehaviour. Fusion poté automaticky generuje IL kód, který propojuje getter a setter těchto vlastností s paměťovým bufferem stavu přidruženého síťového objektu. [65]

```
public class PlayerBehaviour : NetworkBehaviour
{
	[Networked]
	public float Health { get; set; }
}
```
[65]

Změny těchto stavů by měly být prováděny v metodě FixedUpdateNetwork() jako součást simulace. Tím se zajistí, že se budou chovat správně při předpovídání klienta a změny budou na tick přesně reprodukovány. [65]

```
public override void FixedUpdateNetwork()
{
	// Regenerace životů v metodě FixedUpdateNetwork() zaručuje,
	// že v jeden daný tick je hodnota přesná pro všechny klienty
	Health += Runner.DeltaTime * HealthRegen;
}
```
[65]

Povolenými typy pro Networked Properties jsou: [65]

- Primitivní: byte, sbyte, short, int, long, ushort, uint, ulong, float, double
- Textové řetězce (musí být stanovena maximální délka pomocí atributu [Capacity])
- Unity struktury: Vector2, Vector3, Vector4, Quaternion Matrix4x4, Vector2Int, Vector3Int, BoundingSphere, …
- Typy Enum
- System.Guid
- Uživatelem definované INetworkStructs
- Fusionem definované INetworkStructs: NetworkString<>, IFixedStorage<>, NetworkBool, Ptr, Angle, …
- Typy Fusionu: NetworkObject, NetworkBehaviour, PlayerRef

Pro vlastnosti označené jako [Networked] je také možné nastavit výchozí hodnoty a pomocí atributu [Capacity] definovat maximální velikosti pro NetworkArray, NetworkDictionary, NetworkLinkedList, NetworkString a textové řetězce. [65]

```
public class Player : NetworkBehaviour
{
	// Maximální kapacita pro řetězec Name je 20 znaků a jeho
	// výchozí hodnota je nastavena jako John
	[Networked, Capacity(20)]
	string Name { get; set; } = "John";

	// Pole AvailableSlots může obsahovat maximálně 8 bool hodnot
	[Networked, Capacity(8)]
	NetworkArray<bool> AvailableSlots { get; set; }
}
```
[65]

2.3.3.2 Remote Procedure Calls

Ideální pro sdílení důležitých jednorázových událostí ve hře. Oproti tomu např. Networked Properties jsou preferované řešení pro sdílení stavu mezi síťovými klienty, které procházejí kontinuálními změnami. Dále jsou zde struktury vstupu, které jsou odesílány jako nejisté zprávy, tzn. že pakety mohou být ztraceny. To je zřídka pozorovatelné u situací, které vyžadují neustálý vstup, jako je například pohyb postavy. Nicméně, v případě, že důležitá událost nebyla vykonána kvůli ztracenému paketu s informací o této události, což může být klíčové pro postup ve hře, to může výrazně negativně ovlivnit zážitek hráče. [65]

Obecně existují tři hlavní typy RPC: instanční RPC, statické RPC a cílené RPC [65]

Podmínky pro deklaraci instančního RPC: [65]

- Metoda, která bude sloužit jako RPC musí být mít návratovou hodnotu void nebo RpcInvokeInfo (když je metoda RPC vyvolána, návratová hodnota RpcInvokeInfo bude obsahovat informace o volání a odeslání RPC operací.)
- Jméno metody musí obsahovat předponu nebo příponu „RPC“ (není brán ohled na velikost písmen)
- Metodě musí být přidělen atribut [Rpc]
- Konfigurace parametrů RpcSources a RpcTargets tak, aby určily, odkud může být RPC voláno a kde bude prováděno.

```
[Rpc(RpcSources.InputAuthority, RpcTargets.StateAuthority)]
public void RPC_Configure(string name, Color color)
{
      playerName = name;
      playerColor = color;
}
```
[65]

RpcSources a RpcTargets jsou filtry. RpcSources definují, které uzly mohou odeslat RPC, zatímco RpcTargets určují, na kterých uzlech je RPC vykonáno. [65]

- All: může být odesláno / vykonáno všemi uzly v relaci (včetně serveru). [65]
- Proxies: může být odesláno / vykonáno uzlem, který nemá ani vstupní autoritu, ani autoritu stavu nad objektem. [65]
- InputAuthority: může být odesláno / vykonáno uzlem s InputAuthority nad objektem. [65]
- StateAuthority: může být odesláno / vykonáno uzlem se StateAuthority nad objektem. [65]

Oproti tomu statické RPC slouží k jinému účelu. Ignorují parametry RpcSources a RpcTargets, neexistuje tedy filtr zdroje a cíle, statické RPC mohou být volány z libovolného klienta a budou odeslány všem klientům. Je třeba si však uvědomit, že je stále možné zaměřit RPC na konkrétního PlayerRef, pro kontrolu, kdo obdrží volání. [65]

```
[Rpc]
public static void RPC_MyStaticRpc(NetworkRunner runner, int a) { }
```
[65]

Pokud má být RPC vykonán výhradně na konkrétním stroji, používají se cílené RPC (Targeted Rpc). Jak instanční RPC, tak i statické RPC mohou být převedeny na cílené RPC přidáním parametru PlayerRef za atributem [RpcTarget]. Typickým použitím může být týmový chat, kde je zpráva určena pouze pro konkrétní hráče ve vlastním týmu. [65]

```
[Rpc(sources: RpcSources.InputAuthority, targets: RpcTargets.All)]
public void Rpc_TargetedInstanceMessage(
[RpcTarget] PlayerRef player,  // Cíl
string message                 // Zpráva
) { };

[Rpc(sources: RpcSources.InputAuthority, targets: RpcTargets.All)]
public static void Rpc_MyTargetedStaticMessage(
NetworkRunner runner,          // Nutný odkaz na NetworkRunner (Static RPC)
[RpcTarget] PlayerRef player,  // Cíl
string message                 // Zpráva
) { };
```
[65]

2.3.3.3 Vstup od hráče

Jedná se o vstupní data získaná každý tick pomocí metody INetworkRunnerCallbacks.OnInput(), která jsou následně replikována na serveru. Tyto vstupy jsou ukládány do bufferu a používají se při každém simulovaném tiku. Vstupy jsou získávány pomocí metody GetInput() uvnitř metody FixedUpdateNetwork(), která vrací vstupy poskytnuté autoritou daného síťového objektu. [65]

2.3.4 Management scén

Knihovna Fusion neobsahuje žádnou implementaci pro manipulaci s Unity scénami, avšak poskytuje rozhraní INetworkSceneManager, kde může vývojář definovat, jak Fusion reaguje na různé události související se scénami, jako jsou např. změny scén. [65]

Implementace INetworkSceneManager musí být přiřazena v metodě NetworkRunner.StartGame() pomocí pole StartGameArgs.SceneManager. Pokud nebyla poskytnuta žádná implementace (null), Fusion vytvoří instanci třídy NetworkSceneManagerDummy a zaznamená chybu, ve které sdělí, že nebude schopen propojit scénové objekty. [65]

S Fusion je dodávána výchozí implementace nazvaná NetworkSceneManagerDefault, která umožňuje: [65]

- Načítání a uvolňování scén.
- Podporu běžného načítání scén a načítání scén pomocí adresování.
- Možnost znovunačtení aktuálně aktivních scén.
- Načítání až 8 scén současně pomocí přídavného načítání scén.

2.4 Fish-Net: Networking Evolved

„Fish-Networking (Fish-Net) je bezplatná a všestranná síťová knihovna v Unity, vybudovaná od základů nezávisle na jiných řešeních, která nabízí více funkcí než jakékoliv jiné bezplatné řešení.“ [66]

Fish-Net je od svého návrhu serverem autoritativní, což umožňuje použití dedikovaných serverů, ale zároveň umožňuje uživatelům jednat jako server a klient pro rychlejší vývoj a testování. Je podporován jakýkoliv typ síťové topologie. Transporty mohou využívat různé technologie pro komunikaci mezi serverem, klientem, a dokonce i třetími stranami. [66]

Vysokoúrovňové API umožňuje rychlý přístup k synchronizaci stavů, logiky, objektů a dalších prvků. Je však i možné využívat nízko úrovňovou funkčnost za pomocí přiložených událostí nebo dědičnosti. [66]

Fish-Networking je navíc jediné řešení (z řešení, které byly popsána v této práci), které nabízí bezplatnou dlouhodobou podporu (LTS). [66][67][68]

2.4.1 Odlišnosti Fish-Net od ostatních síťových knihoven pro Unity

Všechny ostatní knihovny diskutované v této práci jsou založeny na předchozích projektech, z nichž některé jsou již zastaralé. Fish-Networking je originální a od základů vyvinuté řešení, které není zatíženo omezeními předchozích řešení. [66][68]

Jednou z hlavních odlišností a výhodou nad ostatními knihovnami, a i důvod proč byla tato knihovna vybrána pro projekt UTW je management scén v Unity. Zatímco jiná řešení jako Mirror, Fusion a Netcode jsou omezena na základní správu jednotlivých scén, Fish-Networking poskytuje vestavěnou logiku pro kontrolu oddělených klientů v rámci vícero scén. To např. umožňuje vytvářet několik paralelních místností (scén) pro rozdělení hráčů do samostatných herních instancí. [66][68]

Mezi další odlišnosti patří: [66][68]

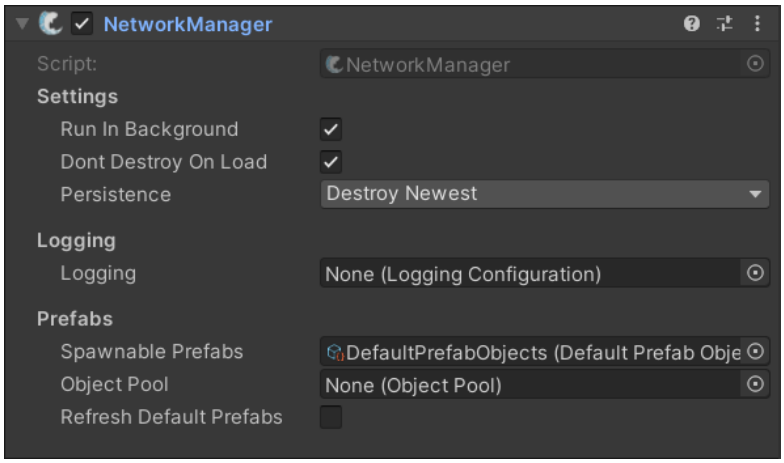
- Síťová úroveň detailu
    - Implementace omezuje frekvenci odesílání dat pro objekty vzdálené od hráčů, aby snížila využití serverového šířky pásma.

- „Lokální“ vzdálené volání procedur
    - Fish-Networking umožňuje vzdálená volání, která se spouštějí nejen na svém zamýšleném cíli, ale také na volající straně. Obvykle by bylo zapotřebí logiku zkopírovat do několika metod. Fish-Networking však umožňuje provést vše v jedné metodě. To výrazně snižuje potenciální chyby v kódu, počet řádků kódu a zlepšuje jeho použitelnost.
- Bufferovaná vzdálená volání
    - Často musí být poslední hodnota vzdáleného volání odeslána nově připojeným klientům. Fish-Networking je jedním z mála řešení, která tuto schopnost nabízejí.
- Zpracování velkých paketů
    - V některých scénářích může být potřeba odeslat velké množství dat buď klientovi, nebo serveru. „Testy ukazují, že Fish-Networking byl schopen účinně odeslat po síti od několika málo megabajtů až po několik stovek megabajtů. Jiná řešení, jako je například Mirror a dokonce i Fusion, vyvolávají chyby při stejné úloze.“
- Síťové komponenty uvnitř potomků
    - Díky této funkcionalitě je možno vytvořit snazší návrh a lépe organizovat kód pomocí komponent jako jsou síťové objekty nebo síťová chování na objektech potomků. Například u Mirroru je vyžadováno, aby všechny síťové komponenty byly umístěny na kořenovém objektu, což by mohlo rychle zkomplikovat přehlednost přojektu.
- Automatické zjišťování Prefabů
    - Jiná síťová řešení vyžadují explicitní specifikaci a organizaci Prefabů síťových objektů. Tato úloha zabírá čas a často vyžaduje přetahování každého prefabu do kolekce nebo hledání prefabů podle řetězce před jejich vytvořením přes síť.
    - U Fish-Networking probíhá automatické zjišťování síťových Prefabů pomocí vlastního frameworku. Změny provedené na síťových objektech jsou automaticky rozpoznány a nakonfigurovány, aniž by uživatel musel podniknout další kroky.

2.4.2 Síťový manažeři

2.4.2.1 NetworkManager

NetworkManager je nezbytnou součástí pro provoz klienta a serveru. Působí jako most mezi hlavními komponentami a konfigurací sítě. I přestože tato komponenta spravuje síť, sama o sobě by neměla být síťovým objektem a nesmí obsahovat komponentu NetworkObject na stejném objektu ani jako rodičovský prvek. [66]



Obrázek 21. Komponenta NetworkManager knihovny Fish-Net

NetworkManager základní nastavení přímo v editoru Unity, jako např.: [66]

- Run In Background – umožňuje aplikaci běžet na pozadí, pokud je nastavena na true. Běh na pozadí je často nezbytný pro klienty a zejména pro servery.
- Don't Destroy On Load – zajistí, že Network Manager přetrvává při změnách scén. Vhodné pro případy, kdy je používán pouze jeden NetworkManager.
- Persistence – určuje, jak se chovat, pokud je současně spuštěno více Network Manažerů.

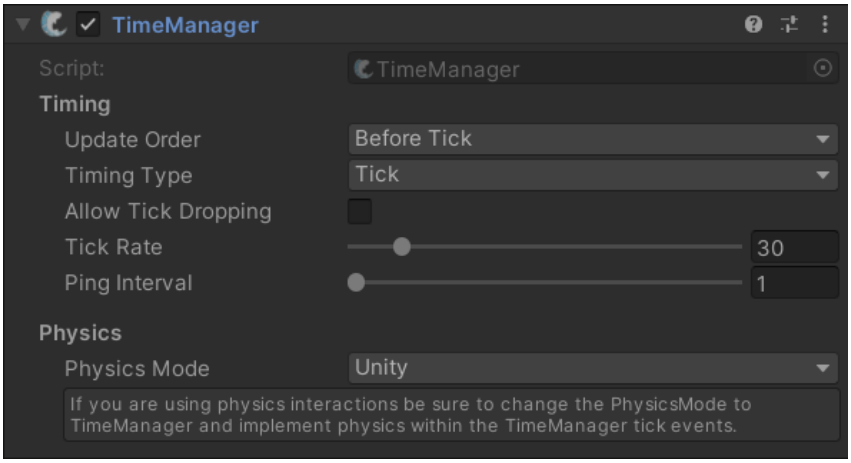
Logování řídí, jak je konfigurován síťový záznam. Určuje, které akce mají být logovány pro build (sestavenou aplikaci), editor (v rámci Unity) a headless (server) režim. Když pole není vyplněno, použijí se výchozí nastavení logování. [66]

Spawnable Prefabs určuje, kterou kolekci Prefabů použít pro síťové objekty. Výchozí hodnota tohoto pole je automaticky nastavena na DefaultPrefabObjects. Je však i možnost vytvořit vlastní třídu PrefabObjects s upravenými pravidly a aplikacemi. Object Pool označuje, který skript pro objektové poolování použít. Když není nastaveno, je automaticky

přidán DefaultObjectPool. Refresh Default Prefabs zajistí že každým spuštěním hry se obnoví kolekce DefaultPrefabCollection. [66]

2.4.2.2 TimeManager

TimeManager zpracovává a poskytuje zpětná volání související s časováním v síti. [66]



Obrázek 22. Komponenta TimeManager knihovny Fish-Net

Update Order řídí, kdy bude TimeManager vyvolávat své verze Unity zpětných volání (callbacks). Volba BeforeTick umožňuje získávat vstupy v metodě OnUpdate než dojde k tiku. [66]

Timing Type řídí, kdy jsou data odesílána a přijímána. Když je nastaveno na Tick, data jsou zpracována pouze ve snímcích, které také tikají. Při výběru možnosti Variable budou data odesílána a přijímána každý snímek, pokud jsou k dispozici. [66]

Allow Tick Dropping umožní klientovi přeskakovat tiky, když se vyskytnou několikrát během jednoho snímku. Toto může zabránit klientovi ve zvětšování počtu simulací na snímek, což by vedlo k větší ztrátě výkonu. [66]

Tick Rate je průměr, kolikrát za sekundu TimeManager vyvolává události tiků, a také jak často mohou být data odesílána nebo přijímána. [66]

Ping Interval určuje, jak často v sekundách uživatel obdrží aktualizační ping. Nastavení této možnosti na vyšší hodnotu s sebou nese šanci, že synchronizace tiků serveru ztratí přesnost. [66]

Physics Mode určuje, jak jsou prováděny fyzikální výpočty. Volba Unity umožní enginu spravovat fyziku. TimeManager simuluje fyziku na tiky. Pro predikci založenou na fyzice musíte použít nastavení TimeManageru. [66]

2.4.2.3 TransportManager

TransportManager řídí komunikaci s transportními prostředky. Stará se také o posílání, přijímaní, nebo i úpravu zpracovávaných paketů. [66]

Disponuje i simulátorem latence, což umožňuje vývojáři simulovat různé problematické scénáře jako: zpoždění (nastavitelné v milisekundách), odeslání paketu mimo pořadí (procentuální šance) nebo ztráta paketu (procentuální šance). [66]

Ve výchozím nastavení používá Fish-Net transport Tugboat, který je součástí instalačního balíčku knihovny. Tento transportní prostředek umožňuje konfiguraci serveru, například specifikaci IPv4/IPv6 adresy, na které bude server naslouchat, určení portu na této adrese a nastavení maximálního počtu připojených klientů. Dále umožňuje nastavit adresu, kterou klient použije k připojení k serveru. [66]

2.4.2.4 ServerManager

ServerManager se zabývá validací klientů a řadou nastavení, která jsou aplikovatelná pouze na server. [66]

Nabízí možnost specifikace autentifikátoru, kterým musí klient úspěšně projít, aby mohl komunikovat se serverem. V případě, že není žádný autentifikátor definován, příchozí připojení nejsou podrobeny žádné formě ověření. [66]

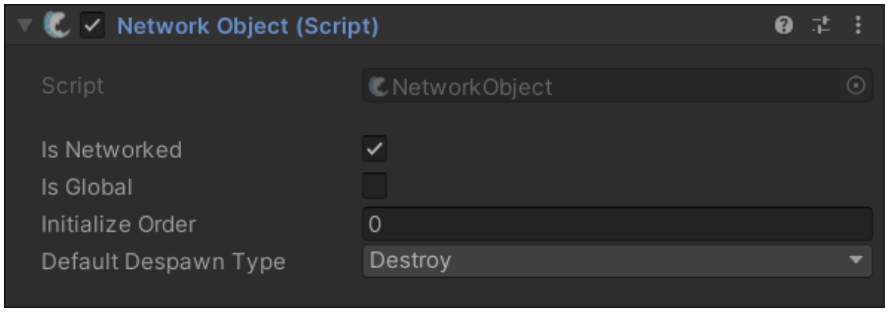
Další volitelné možnosti jsou: [66]

- Remote Client Timeout – určuje, zda by server měl odpojit klienty, kteří se zdají neaktivní.
- SyncType Rate – nastavení frekvence aktualizace pro síťové proměnné (SyncTypes). To lze nastavit pro každý typ zvlášť.
- Share Ids – umožňuje klientům, aby byli informováni o ostatních klientech ve hře a o objektech vlastněných jinými klienty. V opačném případě jsou objekty vlastněné jinými klienty známy pouze tehdy, jsou-li dostupné lokálnímu klientovi, například prostřednictvím systému pozorovatele.

2.4.3 Komponenty

2.4.3.1 NetworkObject

Tato komponenta je automaticky připojena k objektu vždy, když je použit skript, který dědí od třídy NetworkBehaviour. NetworkObject je centrem síťových objektů. [66]



Obrázek 23. Komponenta NetworkObject knihovny Fish-Net

- IsNetworked označuje, zda má být objekt považován za síťový objekt. Pokud je nastavena na false, objekt se inicializuje jako nesíťový a je tím pádem pouze lokální. To se může hodit např. pokud si server potřebuje držet své mezi výpočty, které by klient neměl vidět. Vždy, když je objekt spawnut pomocí metody ServerManager.Spawn, se vlastnost IsNetworked automaticky nastaví na true pro tento vytvořený objekt. [66]
- IsGlobal určuje, zda bude objekt známý všem klientům po celou dobu. Tato hodnota může být změněna za běhu. [66]
- Initialize Order určuje pořadí, v němž budou inicializační zpětná volání objektu provedena jako první, pokud jsou vytvořeny s jinými síťovými objekty ve stejném tiku. Výchozí hodnota je 0, záporné hodnoty zajistí, že se zpětná volání provedou jako první. [66]
- Default Despawn Type je výchozí chování při odstranění objektu. [66]

2.4.3.2 NetworkTransform

Komponenta NetworkTransform v rámci knihovny Fish-Net je zodpovědná za synchronizaci pozice a rotace objektů přes síť ve vaší. Tím umožňuje všem klientům vidět stejný pohyb objektů, i kdyby došlo k určitému zpoždění. [66]

Možnost Synchronize a „Snappování“ umožňuje vybrat, které vlastnosti se mají synchronizovat. Pouze zvolené hodnoty budou odeslány přes síť. Snapping umožňuje aby se transformace stala okamžitě (skok na pozici), namísto toho aby se postupně vyhlazovala.

Tato funkce je např. běžně používána u 2D her (překlopení osy Y pro okamžitou rotaci). [66]

Možnost Smoothing umožňuje nastavení toho, jak se síťová transformace vyhlazuje pro ostatní klienty. Interpolace určuje délku bufferu, který se vytváří při replikaci transformace. Větší hodnoty interpolace snižují šanci sekavého pohybu, pokud dojde k síťovému zpoždění, na úkor toho, že transformace bude dál v minulosti. Extrapolace (placená funkce) určuje, jak dlouho se transformace bude snažit předpovídat pohyb, když se očekávají nová data, ale nedorazí. Použití nízké hodnoty interpolace v kombinaci s extrapolací je skvělý způsob, jak získat plynulý pohyb bez zobrazení síťového zpoždění. [66]

Povolení teleportace odhalí a povolí hodnotu prahu teleportace. Práh teleportace určuje, jak daleko se musí transformace přesunout v jediném aktualizačním kroku, aby došlo k teleportaci místo vyhlazování. [66]

Pokud je volba Client Authoritative nastavena na true, umožňuje vlastníkovi objektu provádět změny na své transformaci lokálně, a tyto změny budou odeslány na server a ostatním klientům. Když je nastaveno na false, server musí provádět změny na transformacích, aby byly odeslány klientům. [66]

2.4.3.3 NetworkObserver

NetworkObserver využívá podmínky k určení, zda se klient kvalifikuje jako pozorovatel objektu. Lze použít libovolný počet podmínek. Komponenta NetworkObserver může být použita k přepsání ObserverManageru nebo k přidání dalších podmínek speciálně k objektu, na který je komponenta NetworkObserver přidána. [66]

Ve výchozím stavu jsou již implementovány různé podmínky, které mohou být použity společně. Taktéž je možné dědit ze třídy ObserverCondition a vytvořit vlastní podmínky. Když jsou všechny podmínky splněny, objekt se stane viditelným pro klienta. [66]

- Podmínka vzdálenosti je pravdivá, pokud jsou klienti v určité vzdálenosti od objektu.
- Podmínka mřížky je podobná podmínce vzdálenosti a je pravdivá, pokud jsou klienti v určité vzdálenosti od objektu. Podmínka mřížky je méně přesná, ale více výkonná. Tato podmínka vyžaduje umístění HashGridu na nebo pod objektem NetworkManageru.
- Podmínka scény je pravdivá, pokud klient sdílí scény s objektem.

- Podmínka zápasu je pravdivá, když hráči nebo objekty sdílejí stejný zápas. Do zápasů lze přidat jak vlastněné, tak nevlastněné objekty. Objekty nebo hráči, kteří nejsou přidáni do zápasů, budou mít svá data synchronizována se všemi, pokud tomu nebude bránit jiná podmínka.
- Podmínka pouze vlastníka je pravdivá, když hráč vlastní objekt. Objekt je tedy viditelným pouze pro vlastníka. Pokud neexistuje vlastník, objekt nebude viditelný pro žádného klienta.
- Podmínka pouze hostitele je pravdivá, když je hráč clientHost. Jakékoliv připojení, které není clientHost, tuto podmínku nesplní.

2.4.4 Vytváření a ničení objektů

Objekty, které mají existovat přes síť, musí mít připojenou komponentu NetworkObject a musí být vytvořeny na serveru. K vytvoření objektu musí být nejprve vytvořena instance, která je předána do metody Spawn. [66]

Vytvoření bez vlastníka se provádí předáním hodnoty null u argumentu vlastníka nebo prostým vynecháním argumentu:

```
GameObject go = Instantiate(prefabName);

InstanceFinder.ServerManager.Spawn(go);
// Nebo
InstanceFinder.ServerManager.Spawn(go, null);
```
[66]

Vytvoření s vlastníkem:

```
GameObject go = Instantiate(prefabName);
InstanceFinder.ServerManager.Spawn(go, ownerConnection);
```
[66]

Přistupovat ke spawn metodě je možné i v libovolném skriptu, který dědí ze třídy NetworkBehaviour, nebo přistupovat přímo k NetworkObjectu:

```
GameObject go = Instantiate(prefabName);

base.Spawn(go, ownerConnection); // NetworkBehaviour.
// Nebo
networkObject.Spawn(go, ownerConnection); // Reference na NetworkObject.
```
[66]

Odstranění objektu lze provést stejnými způsoby jako jeho vytvoření. To znamená pomocí třídy NetworkBehaviour, odkazem na NetworkObject nebo přímo skrze ServerManager.

```
base.Despawn(); // NetworkBehaviour.
networkObject.Despawn(); // Reference na NetworkObject.
InstanceFinder.ServerManager.Despawn(go); // ServerManager.
```
[66]

2.4.5 Atributy

V rámci knihovny Fish-Net existuje řada atributů, které přidávají funkcionalitu a pomáhají ke zrychlení vývojového procesu. [66]

Umístění atributu Client nad metodu zajistí, že metoda nemůže být volána, pokud není lokální klient aktivní:

```
[Client(Logging = LoggingType.Off, RequireOwnership = true)]
private void PlayVFX() { }
```
[66]

Server nepotřebuje přehrávat vizuální efekty – vizuální efekt (např. v podobě particle effect) se přehrají pouze tehdy, pokud je aktivní klient. Pokud by byla tato metoda volána bez aktivního klienta, bude zobrazeno varování v konzoli Unity. Typ logování lze změnit pomocí vlastnosti Logging. Je také možné povolit volání metody pouze vlastníkem objektu nastavením RequireOwnership na true; tato hodnota je výchozím nastavením false. [66]

Atribut Server poskytuje totožné funkce jako varianta Client, avšak pro server:

```
[Server(Logging = LoggingType.Off)]
private void ValidateHit() { }
```
[66]

Atribut NonSerialized je součástí namespace System a může být použit k zabránění serializace pole přes síť:

```
public class PlayerStats
{
    public float Health;
    public float MoveSpeed;
    // ControllerIndex je zapotřebí pouze na lokálním prostředí.
    [System.NonSerialized]
    public int ControllerIndex;
}
```
[66]

2.4.6 Scene Stacking

Scene Stacking je schopnost serveru nebo hostitele načíst současně více instancí scény, obvykle s různými klienty/nebo pozorovateli v každé scéně. [66]

Skvělým příkladem pro scene stacking je provoz několika instancí herních doupat na jediném serveru. Pokud do stejného doupěte vstoupí dva klienti, každý z nich načte svou vlastní kopii daného doupěte a bude mít individuální herní objekty a stav pro jeho scénu. Pro server to však znamená současné načtení dvou instancí téže scény. [66]

```
// Pole připojení, která se připojí k nové scéně.
NetworkConnection[] conns = new NetworkConnection[] {connA, ConnB};

// K vytváření stacked scén je nutné použít její název.
SceneLoadData sld = new SceneLoadData("DungeonScene");

// Samozřejmostí je přepnutí možnosti Options.AllowStacking na true.
sld.Options.AllowStacking = true;

// Je zde i možnost oddělit fyziku pouze pro danou scénu.
sld.Options.LocalPhysics = LocalPhysicsMode.Physics3D;

// Nakonec je zavolána metoda LoadConnectionScene, které jsou
// předány parametry, v podobě klientů, kteří budou připojeni a
// samotné nastavení scény.
base.SceneManager.LoadConnectionScene(conns, sld);
```
[66]

Existuje mnoho způsobů, jak získat referenci na načtené scény, např. pomocí připojených klientů. Každý klient si udržuje seznam scén, ke kterým je připojen:

```
int clientToLookup;
InstanceFinder.ServerManger.Clients[clientToLookup].Scenes;
```
[66]

2.4.7 Vzdálené volání procedur

Knihovna Fish-Net disponuje třemi hlavními typy RPC: ServerRpc, ObserversRpc a TargetRpc. Tyto typy lze i libovolně kombinovat (Multi-Purpose Rpc). Vzhledem k tomu, že vzdálené volání procedur je vázáné na objekt, musí být volány na skriptech, které dědí ze třídy NetworkBehaviour. [66]

Metody RPC ve Fish-Net nemusí mít předpony ani přípony „Rpc“, jak je tomu u ostatních knihoven, ale dle dokumentace je to doporučená praktika (lepší přehlednost kódu). [66]

ServerRpc umožňuje klientovi spouštět logiku na serveru. Klient zavolá metodu ServerRpc a data pro provedení této metody jsou odeslána na server. Pro použití ServerRpc musí být klient aktivní a metoda musí mít atribut ServerRpc.

```
[ServerRpc]
private void RpcSendChat(string msg)
{
	Debug.Log($"Received {msg} on the server.");
}

// RequireOwnership, nastavuje zda je nutné aby RPC posílal vlastník objektu
[ServerRpc(RequireOwnership = false)]
// Volající klient lze identifikovat pomocí NetworkConnection na posledním
// parametru s výchozí hodnotou null. Tato proměnná bude automaticky
// vyplněna při volání.
private void RpcSendChat(string msg, NetworkConnection conn = null)
{
	Debug.Log($"Message: {msg}, from client: {conn.ClientId}.");
}
```
[66]

ObserversRpc umožňuje serveru provádět logiku na klientech. Logika bude spuštěna pouze u klientů, kteří jsou pozorovateli daného objektu. Pozorovatelé jsou určeni pomocí komponenty NetworkObserver. Pro použití ObserversRpc je přidán atribut ObserversRpc k metodě. Pokud je zapotřebí poslat RPC nově připojeným klientům, přidá se k atributu ObserversRpc možnost BufferLast = true. [66]

```
[ObserversRpc(BufferLast = true)]
private void RpcSendNumber(int num)
{
	Debug.Log($"Received number {num} from the server.");
}
```
[66]

Poslední hlavní typ RPC ve Fish-Net je TargetRpc. TargetRpc slouží k provedení logiky na konkrétním klientovi. Je implementována přidáním atributu TargetRpc. Při odesílání TargetRpc musí být prvním parametrem metody vždy NetworkConnection. Tento parametr určuje, na které připojení jsou data odesílána. [66]

```
[TargetRpc]
private void RpcSetAmmo(NetworkConnection conn, int newAmmo)
{
      myAmmo = newAmmo;
}
```
[66]

Je možné, aby jedna metoda byla současně jak TargetRpc, tak ObserversRpc. To může být velmi užitečné v případech kdy je třeba poslat RPC všem pozorovatelům nebo jednotlivci. Příkladem může být odesílání zpráv v chatu: [66]

```
[ObserversRpc][TargetRpc]
private void DisplayMessage(NetworkConnection target, string sender,
string message)
{
      Debug.Log($"{sender}: {message}.");
}

[Server]
private void SendDebugMessage()
{
      // Pošle zprávu pouze vlastníkovi
      DisplayMessage(base.Owner, "Bob", "Hello world");
      // Pošle zprávu všem pozorovatelům
      DisplayMessage(null, "Bob", "Hello world");
}
```
[66]

Fish-net umožňuje i specifikovat maximální potenciální velikost dat v atributu RPC. Tuto funkci podporuje každý typ RPC. [66]

```
[ServerRpc(DataLength = 3500)]
private void ServerSendBytes(byte[] data) { }
```
[66]

2.4.8 SyncTypes

Podobně jako u předešle diskutovaných knihoven jsou SyncTypes dalším typem komunikace. Jedná se o pole, která se automaticky synchronizují klientům přes síť, když je server změní. K dispozici je řada různých typů SyncTypes: SyncVar, SyncDictionary, SyncList, vlastní SyncTypes a další. [66]

Při provedení změn na SyncType jsou odeslány pouze tyto změny. Například, pokud existuje SyncList o 10 hodnotách a je přidána další, bude odeslána pouze právě přidaná položka. [66]

Jako příklad je uvedena SyncVar, použita k synchronizaci jednoho pole. Toto pole může být téměř cokoli: hodnotový typ, struktura nebo třída. [66]

```
public class MyClass
{
    private readonly SyncVar<float> _health = new SyncVar<float>();
}
```
[66]

SyncTypes mohou být také upraveny s dodatečnými možnostmi. Mezi tyto možnosti patří oznámení o změně hodnoty, změna frekvence synchronizace SyncType a další. [66][69]

```
// Interval posílání SyncVar _health je vždy alespoň 1f
private readonly SyncVar<float> _health = new SyncVar<float>(
new SyncTypeSettings(1f));

// V metodě Awake je přidán listener pro _health.OnChange
// To zajistí, že se při změně hodnoty zavolá metoda HealthChanged()
private void Awake()
{
      _health.OnChange += HealthChanged;
}

// Parametry metody zpětného volání musí vždy mít tuto podobu:
// prev – předešlá hodnota
// next – nová hodnota
// asServer – zda se má vykonat na serveru či klientovi
private void HealthChanged(float prev, float next, bool asServer) { }
```
[66]

II. PRAKTICKÁ ČÁST

3 PROJEKT UTW

Tato úvodní kapitola si dává za cíl uvedení čtenáře do projektu UTW. Je zaměřena na popis a důvod vzniku projektu a na objasnění očekávání uživatelů, pro které je tento simulátor určen.

3.1 Historie projektu

Tento projekt byl částečně iniciován určitými hráčskými komunitami, které vyžadovaly více hráčovou herní platformu pro vizualizaci tankových bitev, jež měly vliv na vlastnictví půdy na virtuální mapě.

Tyto komunity jsou současně omezeny na hry jako např. War Thunder, které vyžadují velké časové zapojení od uživatele, aby byl schopen vlastnit potřebné vybavení (tanky), které pak může použít v soukromých bitvách ve své komunitě. Zároveň je zde velice omezená modifikovatelnost samotné techniky či jiných herních elementů, což je důležité pro komunity, které chtějí jednoduše přidávat nebo upravovat používanou techniku.

Značné množství uživatelů také projevilo zájem o implementaci systému vícečlenné posádky v rámci jednoho tanku. Tento požadavek je například realizován ve hře Squad 44 (dříve známé jako Post Scriptum). I když tato hra umožňuje určitou míru modifikace herních mechanik a vozidel, je třeba zdůraznit, že tato činnost není pro nezkušeného uživatele jednoduchá.

3.2 Projektové požadavky na síťovou architekturu

Projekt UTW si klade za cíl vytvořit uživatelsky přístupné herní „sandbox“ prostředí, které umožní pořádání tankových bitev, snadnou modifikaci herních assetů a umožní současný běh více her na jednom serveru.

Uživatel bude schopen se připojit na oficiální server, který je zprovozněn na doméně UTB FAI (nyní pouze přístupný pro univerzitní studenty s přístupem k VPN). Samozřejmostí je i možnost založení vlastního serveru na statické IP adrese, kterou budou následně jiní uživatelé moci zadat v menu a připojit se tak např. na jejich komunitní server.

V rámci serveru bude uživatelům umožněno vytvářet lobby (místnosti) kde se následně mohou shlukovat a vytvářet jednotlivé týmy, komunikovat s ostatními hráči v lobby pomocí in-game chat systému, vybírat si svůj počáteční bod (spawnpoint) ze skupiny těchto bodů, rozřazovat se do jednotlivých pozic tanků, kde každá pozice může hrát jinou roli (řidič,

střelec, velitel). Po úspěšném rozřazení a naplnění herního pole, může majitel lobby spustit hru, což přesune všechny hráče na jejich zvolené pozice a předá jim kontrolu nad ovládacími prvky, které přísluší dané pozici.

Tato koncepce podporuje teamovou spolupráci a komunikaci v rámci hry. Sociální aspekt bude hrát velkou roli, jelikož budou hráči muset spolupracovat pro dosažení vítězství.

3.3 Open-Source

Projekt UTW je koncipován jako open-source herní projekt, což znamená, že jeho zdrojový kód je veřejně přístupný na platformě Github. Tento přístup umožňuje komukoliv, aby se podílel na vývoji, testování nebo modifikaci hry. Open-source charakter projektu přináší řadu výhod, jako je rychlejší identifikace a oprava chyb, inovativní vývoj díky příspěvkům od komunity a vysoká úroveň transparentnosti vývojového procesu. [70]

Tento způsob vývoje také povzbuzuje komunitu kolem projektu UTW, která může přispívat nejen kódem, ale i nápady a zpětnou vazbou, což vede k neustálému zlepšování hry. Dále open-source licence umožňuje uživatelům studovat, jak jsou implementovány různé funkce, což může sloužit jako vzdělávací nástroj pro nové i zkušené programátory.

Projekt UTW navíc posiluje svou komunitu prostřednictvím vlastního Discord serveru, který slouží jako centrální místo pro komunikaci, spolupráci a sdílení mezi vývojáři a hráči. Na tomto serveru mohou členové komunity diskutovat o nejnovějších aktualizacích, sdílet nápady a navrhovat změny, které by mohly hru vylepšit. Discord server také poskytuje prostředí pro technickou podporu, kde si mohou uživatelé pomáhat s řešením problémů, které při hraní nebo vývoji hry mohou nastat. Využívání Discord serveru pro tyto účely zajišťuje efektivní a rychlou komunikaci, a tím i dynamický rozvoj projektu UTW. [71]

4 NÁVRH ŘEŠENÍ ARCHITEKTURY PRO PROJEKT UTW

Tato kapitola je zaměřena na analýzu funkčních požadavků a návrh client-server architektury pro projekt UTW. Hlavním cílem této části je prezentace konceptuálních a technických požadavků na architekturu projektu UTW a navržení vhodného řešení, které by tyto požadavky efektivně a spolehlivě splnilo. Součástí návrhu jsou i aktivitní diagramy, které graficky popisují procesy uvnitř simulátoru.

4.1 Architektura Client-Server

Jak již bylo zmíněno, simulátor UTW využívá autoritativní síťovou architekturu client-server. Tento přístup byl zvolen především z důvodu centralizace. S ohledem na vysokou úroveň modifikovatelnosti UTW je nezbytné, aby centrální entita, v tomto případě server, určila podobu hry na daném shardu. Jinými slovy, uživatel musí mít všechny potřebné assety, které po něm server vyžaduje k tomu, aby se mohl připojit. Tento proces je realizován v rámci navázání spojení mezi serverem a klientem, kdy je provedeno ověření hash hodnot asset databáze, tento proces je podrobněji popsán v kapitole 4.1.3.2.

Mezi další výhody této architektury pro projekt UTW patří:

- Kontrola prostředí – server, má úplnou kontrolu nad herním prostředím. To umožňuje zajistit konzistenci herního světa a řídit chování a interakce mezi hráči.
- Řízení komunikace – server umožňuje regulovat komunikaci. Hráči nemusejí mít všechny informace o dalších připojených uživatelích s nimi na serveru.
- Ochrana proti podvodům – server může provádět kontrolu a validaci akcí hráčů a přijímat opatření proti nekalým praktikám.

4.2 Hlavní menu

Při startu hry bude uživateli představena počáteční scéna „Hlavní menu“. Tato scéna není, jakkoliv síťově propojená a běží plně na lokálním klientovi. Uživatel si zde bude moci vybrat z následujících možností:

- Připojit se na oficiální server
- Připojit se na komunitní server pomocí IP adresy
- Zadat své uživatelské jméno

4.2.1 Připojení na oficiální server

Pokud uživatel zvolí tuto možnost bude automaticky, připojen na oficiální server, kde proběhne ověření. Pokud úspěšně, je uživatel přesunut do shard scény serveru. S touto možností se prozatím váže i nutnost připojení přes univerzitní VPN.

4.2.2 Přímé připojení

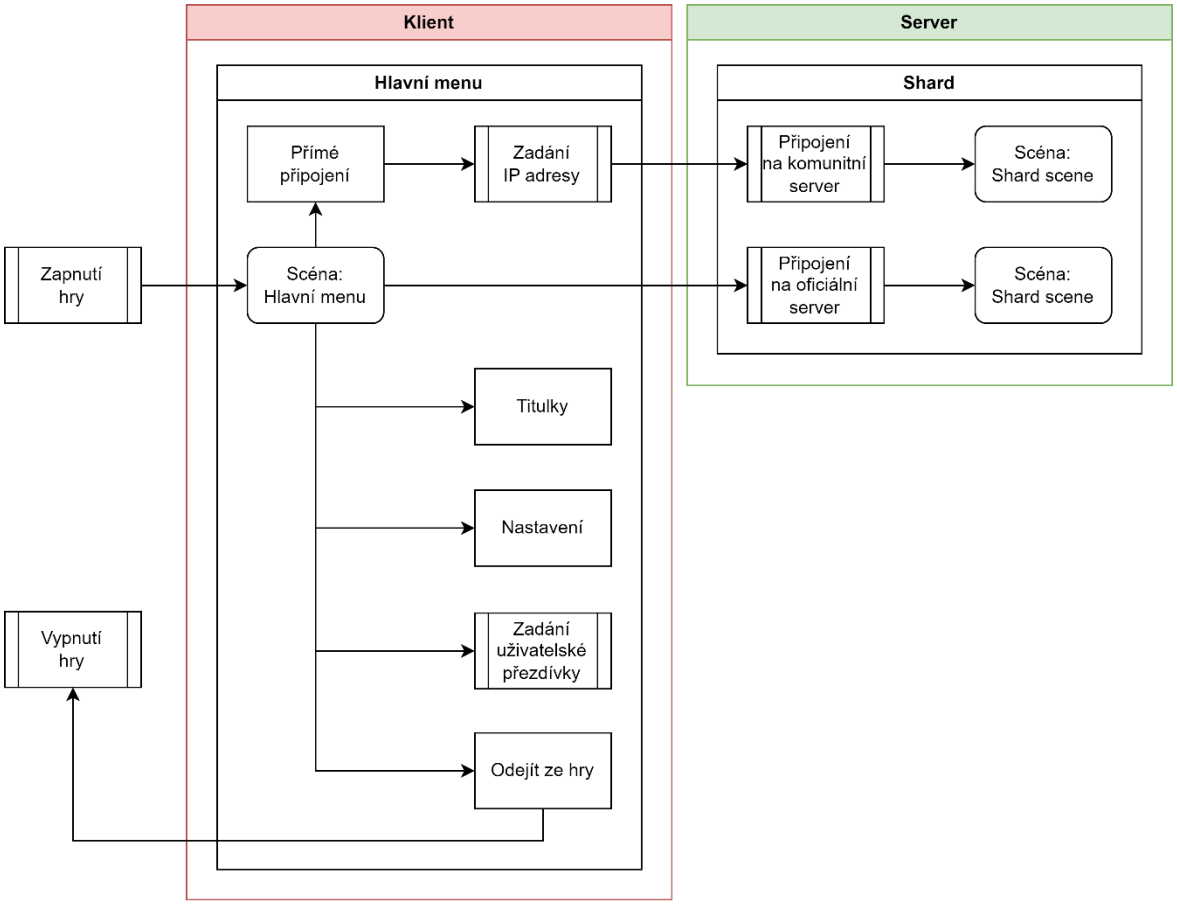
Uživatel bude po vybrání této možnosti vyzván k zadání IP adresy serveru, na který se chce připojit. Po potvrzení se klient pokusí navázat spojení se serverem na požadované adrese, pokud úspěšně, proběhne opět ověření, pokud úspěšně, bude uživatel transportován do shard scény cílového serveru.

4.2.3 Zadání uživatelského jména

Jedná se o prozatímní systém pro personalizaci uživatele. Uživatel si bude moci zvolit svou přezdívku, kterou následně uvidí ostatní uživatelé, připojení na stejný server.

Pomocí této přezdívky jsou také vytvářeny a ukládány serializované soubory, díky kterým bude server schopen identifikovat vracejícího se uživatele a přiřadit mu příslušné hodnoty, jako např. id jeho frakce, poslední zvolený preset tanku, apod.

Tento systém bude v budoucnu nahrazen systémem SteamID, který poskytne lepší způsob identifikace a zajistí legitimitu každého připojení bez nutnosti ukládání hesel či jiných citlivých informací na straně serveru. [72]



Obrázek 24. Diagram aktivit scény hlavního menu

4.3 Připojení na shard

Po výběru buďto oficiálního či komunitního shardu uživatelem následuje pokus o navázání spojení. Tento proces zahrnuje provedení dvou ověření:

- Ověření, zda uživatel se stejným uživatelským jménem je již připojen na shard
- Ověření, zda má připojující se uživatel stejnou databázi assetů jako server

4.3.1 Ověření uživatelského jména

Shard provádí kontrolu seznamu registrovaných uživatelů a v případě nalezení shody mezi uživatelskými jmény jednoho z připojených uživatelů a aktuálně se připojujícím uživatelem vybere zda:

- Odmítne spojení
    - Tento stav nastává, pokud je uživatel se stejným uživatelským jménem aktuálně aktivní na shardu. To znamená, že uživateli, který se právě pokouší připojit, je odmítnuto spojení, jelikož již existuje aktivní spojení s touto přezdívkou.
- Přijme spojení
    - Tato akce probíhá tehdy, když není detekováno žádné aktivní spojení s uživatelským jménem, které se právě pokouší připojit.

V případě, že žádná shoda není nalezena, probíhá registrace nového uživatele, která zahrnuje vytvoření nového serializovaného souboru na serverovém úložišti.

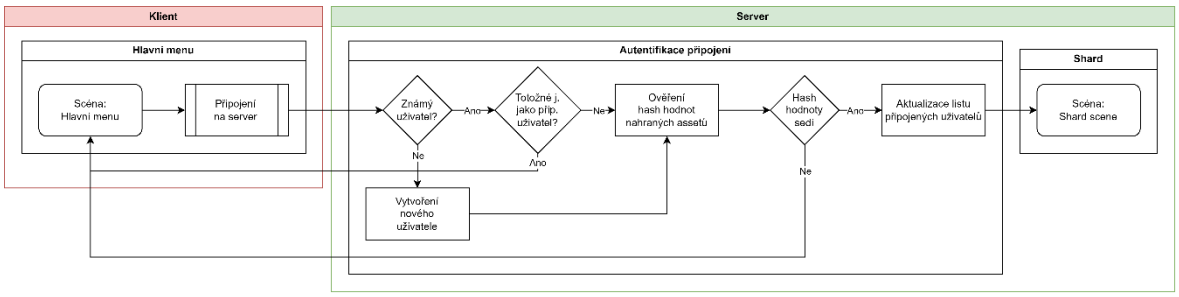
4.3.2 Ověření hash hodnot asset databáze

Při spuštění simulátoru UTW se inicializuje tzv. Asset databáze. Tato databáze je sestavena ze souborů umístěných v adresáři StreamingAssets a obsahuje serializované prefaby nezbytné pro správný chod hry. Při připojení uživatele k shardu je tedy nezbytné ověřit, zda má na svém klientovi všechny potřebné soubory, které shard vyžaduje.

Klient může mít i více těchto souborů, kontroluje se zde pouze shoda mezi soubory, které vlastní server.

4.3.3 Aktualizace listu připojených uživatelů

Po úspěšném dokončení všech ověřovacích procesů je uživatel považován za legitimního účastníka a je zařazen do seznamu připojených uživatelů. Tento seznam se aktualizuje znovu při každém dalším ověřeném připojení nebo v případě, že jeden z připojených uživatelů ukončí spojení. Tím se zajišťuje aktuálnost a přesnost seznamu uživatelů připojených k shardu.



Obrázek 25. Diagram aktivit připojování na shard

4.4 Shard scéna

Po úspěšném navázání spojení se serverem je uživatel přenesen do shard scény, kde jsou mu představeny následující možnosti:

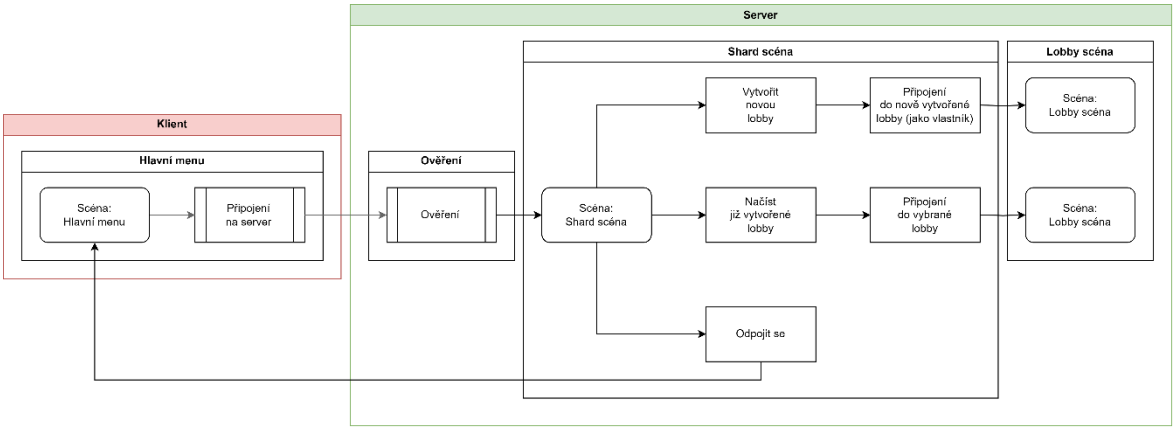
- Založit novou lobby
- Připojit se do již existujících lobby na daném shardu

4.4.1 Založení nové lobby

Pokud uživatel zvolí možnost vytvoření nového lobby, je mu automaticky přidělena role vlastníka lobby. Tato role mu dává exkluzivní práva, což znamená, že pouze tento uživatel má oprávnění k zahájení hry, úpravě jejích parametrů a v případě, že se odpojí, dojde k automatickému ukončení této lobby. Všichni ostatní uživatelé, kteří byli připojeni k této lobby, jsou poté přemístěni zpět na scénu shardu.

4.4.2 Připojení do již existující lobby

Při připojení na shard si klient vyžádá seznam všech aktuálně probíhajících lobby a má možnost vybrat, ke které se připojí. Důležité je poznamenat, že jakmile je hra spuštěna v určité lobby, již se tato lobby nezobrazuje v seznamu dostupných lobby na shardu. To znamená, že uživatel nemá možnost připojit se do již probíhajících her.



Obrázek 26. Diagram aktivit uvnitř shard scény

4.5 Lobby scéna

Lobby scéna slouží pro zorganizování připojených hráčů do místnosti, kde se mohou:

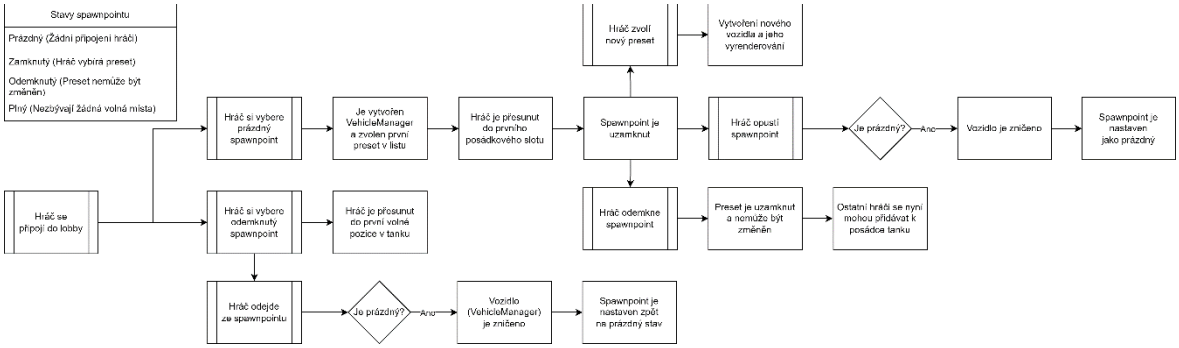
- Rozdělit do posádek a určit jaký hráč bude obsluhovat jakou pozici v rámci fungování bojové techniky (řidič, střelec, …)

- Určit startovní pozice jednotlivých posádek a jejich tanku
- Komunikovat v rámci chatu s ostatními hráči v lobby

Jakmile jsou všichni hráči na svých pozicích a jsou připraveni, může zakladatel lobby spustit hru.

4.5.1 Management spawnpointu a posádky

Řešení chování spawnpointů a plnění posádek je podrobně popsáno v rámci aktivitního diagramu (viz. Obrázek 27).



Obrázek 27. Diagram aktivit managementu spawnpointu a posádky

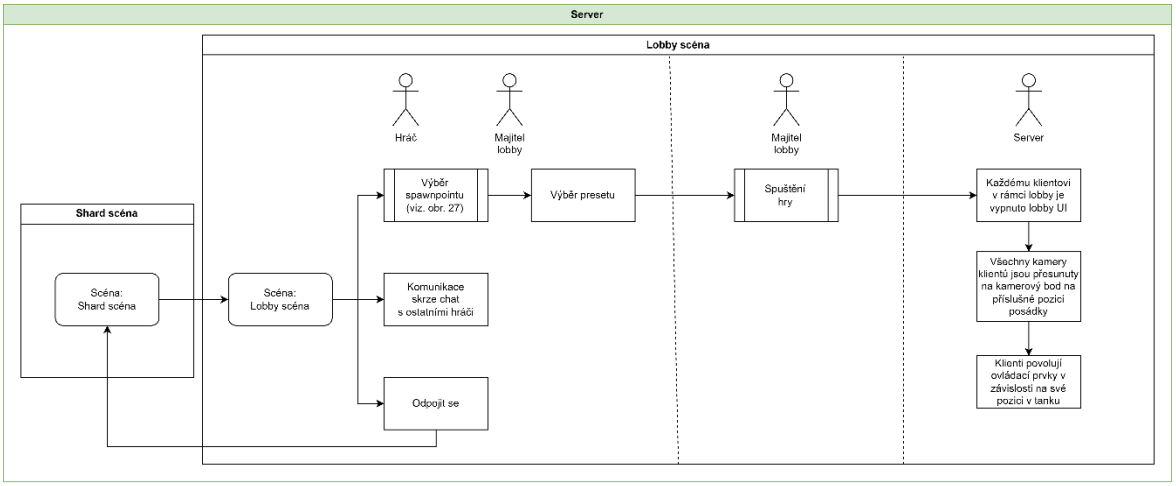
4.5.2 Chat

Pro účely komunikace hráčů v rámci lobby bude implementován chatovací systém. Díky této funkcionalitě uživatelé nemusí využívat externí aplikace pro textovou či hlasovou komunikaci, a mohou se tak domlouvat přímo v rámci simulátoru UTW.

V plánu je také implementace systému hlášek hráčských postav. Hráč bude mít možnost vybrat si z předem připraveného seznamu dialogů, které jeho postava následně přehraje jakožto audio záznam. Tato funkcionalita bude sloužit jako alternativa pro hráče, kteří neradi používají mikrofon, ale přesto chtějí komunikovat se svým týmem nebo posádkou.

4.5.3 Start hry

Po spuštění hry zakladatelem lobby již nedochází k přechodu do žádné další scény, a celý herní proces se odehrává v rámci lobby scény, do které se hráči připojili. Kamery klientů jsou umístěny do jejich tanků na příslušné pozice (pokud si hráči vybrali spawnpoint; jinak jsou vráceni zpět na scénu shardu). Následně hráči získávají kontrolu nad odpovídajícími částmi tanku (například řidič ovládá pohyb korby, střelec ovládá věž atd.) a mohou začít hrát.



Obrázek 28. Diagram aktivit pro Lobby scénu

4.5.4 Ukončení hry

Jediný, kdo má pravomoc ukončit probíhající hru/lobby je její vlastník. Ostatní účastníci nemají oprávnění k ukončení hry, avšak mají možnost kdykoli se z ní odpojit, přičemž není možné se po odpojení do hry vrátit.

4.6 GDPR

V kontextu GDPR, které se zaměřuje na ochranu osobních údajů a práva jednotlivců v rámci EU, je důležité zajistit, že veškeré zpracování osobních údajů je prováděno v souladu s předpisy. V případě projektu UTW, kde dochází k ukládání uživatelských jmen připojených klientů na server ve formě serializovaných souborů, je klíčové vytvořit dokumentaci, která popisuje zpracování tohoto osobní údaje.

Vznikl tedy dokument Minimální záznam o činnosti zpracování OÚ, který v bodech popisuje, mimo jiné, za jakým účelem jsou data sbírána, jak bude s daty nakládáno, kdo je odpovědná osoba a kdo jsou subjekty osobních údajů. Vzor tohoto dokumentu je dostupný na DPO Univerzity Tomáše Bati.

5 IMPLEMENTACE ŘEŠENÍ DLE NÁVRHU

Tato kapitola má za cíl popis implementace projektu UTW v prostředí Unity s využitím knihovny Fish-Net: Networking Evolved. Rovněž zde bude popsán proces připojení a správy serveru na doméně UTB FAI. Cílem této části je poskytnout čtenáři komplexní vhled do implementace projektu včetně praktických aspektů spojených s jeho provozem a správou v prostředí Unity a na serveru domény UTB FAI.

5.1 Třídy správců UTW

Tyto třídy představují základní síťové pilíře projektu UTW. Jsou zodpovědné za správu komunikace a synchronizaci dat mezi různými klienty a serverem. Tímto způsobem umožňují hladký průběh multiplayerové hry, zajišťují přenos informací o stavu hry, včetně pohybu objektů, akcí hráčů a dalších relevantních událostí. Díky nim je možné vytvořit stabilní a spolehlivou síťovou infrastrukturu, která podporuje provoz hry a umožňuje hráčům interagovat v reálném čase.

5.1.1 SceneManager

Třída SceneManager je odpovědná za řízení scén a správu přechodů mezi nimi. Je navržena tak, aby centralizovala veškerou logiku spojenou se scénami, včetně správy lobby, připojení a odpojení hráčů, a přechodu mezi hlavními herními scénami a scénami pro lobby.

Nejdůležitější funkcí třídy je správa scén v lobby, kde hráči interagují před začátkem hry. Třída umožňuje vytváření nových lobby, připojování hráčů k existujícím lobby, inicializaci správců lobby (LobbyManager, ChatManager), a řízení spouštění hry z lobby. Dále poskytuje metody pro odpojení hráčů z lobby a aktualizaci stavu lobby po odpojení.

Kromě správy lobby třída také obsahuje metody pro přesun hráčů mezi scénami, především z lobby do Shard scény. Tyto metody jsou aktivovány v reakci na události připojení a odpojení hráčů. Dále třída poskytuje metody pro získání a zobrazení dat o lobby, jako jsou informace o hráčích a stavu lobby.

5.1.2 LobbyManager

Třída LobbyManager má za úkol řídit herní lobby jak ze strany serveru, tak i ze strany klienta. Je určena pro správu výběru spawnpointů, inicializaci mapy, řízení připojení hráčů a jejich přidělování do posádek.

Na straně serveru LobbyManager vytváří a inicializuje herní mapu, určuje spawnpointy a přiděluje je hráčům. Také obsahuje logiku pro výběr spawnpointu, blokování a odemykání spawnpointů v závislosti na stavu (prázdný, uzamčený, odemčený, plný) a interakci s vozidly hráčů. Klientům potom umožňuje vybírat spawnpointy a odesílat požadavky na přidání nebo odebrání se z týmu.

Na straně klienta LobbyManager zajišťuje vizuální zobrazení dostupných spawnpointů a umožňuje klientům vybírat, kde se chtějí spawnout. Také se stará o nastavování kamery a vizuálních efektů (změna materiálu) při výběru spawnpointu.

5.1.3 GameManager

Třída GameManager má na starosti řízení dat hráčů a frakcí. Umožňuje načítání informací o hráčích a frakcích z externích serializovaných souborů ve formátu JSON, vytváření nových hráčů a aktualizaci jejich údajů. Dále umožňuje získání informací o hráčích a frakcích na základě jejich identifikátorů nebo jména. Tato třída také sleduje stav připojení klientů a aktualizuje příslušná data, například identifikátor klienta, při odpojení.

5.1.4 ChatManager

Třída ChatManager řídí zobrazování a odesílání zpráv v chatovacím rozhraní v rámci lobby. Při startu získává přístup k instanci chatu (NetworkObject) a jménu uživatele (z listu třídy GameManager). Umožňuje uživatelům psát zprávy, které jsou odesílány na server a distribuovány všem klientům. Dále umožňuje uživatelům smazat své vlastní zprávy. Zprávy jsou zobrazovány ve scrollovatelném rozhraní spolu s informacemi o odesílateli a časovém razítku.

5.1.5 VehicleManager

Tato třída je odpovědná za správu vozidel ve hře. Její hlavní funkce zahrnují řízení interakce s posádkou vozidla a manipulaci s jeho částmi. Třída zajišťuje uživatelské rozhraní pro zobrazení informací o posádce vozidla. Dále definuje prefaby, a synchronizuje jejich aktuální stav pomocí proměnných.

Pro řízení posádky vozidla třída implementuje metody pro připojení, odpojení a zjištění stavu členů posádky. Díky použití třídy CrewData a synchronizovaného slovníku _tankCrew uchovává informace o členech posádky, včetně jejich síťových spojení, stavu a přiřazených částech tanku.

Důležitou částí funkcionality je také možnost žádosti o výměnu pozice v tanku mezi členy posádky.

Kromě toho třída obsahuje metody pro spawnování tanku s ohledem na zvolený preset a příslušné části, které jsou následně přiřazeny členům posádky. Pro komunikaci s klienty a zpracování událostí týkajících se změn v posádce vozidla a stavu tanku jsou implementovány různé metody s anotacemi ServerRpc, TargetRpc a ObserversRpc.

5.1.6 Inicializace správcovských tříd

K inicializaci tříd SceneManager a GameManager dochází již spuštění serveru, jelikož jsou persistentní a drží si nutné stavy a hodnoty pro chod serveru.

Ve funkci Start je přidána metoda InitializeAllManagers jako posluchač události OnServerConnectionState z objektu ServerManager instance třídy NetworkManager. Tato událost je spuštěna při změně stavu připojení na serveru.

```
private void Start()
{
	InstanceFinder.NetworkManager.ServerManager.OnServerConnectionState +=
	InitializeAllManagers;
}
```

Funkce InitializeAllManagers je zavolána, když je stav připojení na serveru nastaven na Started. Poté projde seznam prefabů (prefabList) a inicializuje je pomocí funkce Initialize.

```
[SerializeField]
private List<GameObject> prefabList;

private void InitializeAllManagers(ServerConnectionStateArgs args)
{
	if (args.ConnectionState != LocalConnectionState.Started) return;

	Debug.Log("Server started... Initializing all managers");

	foreach (GameObject prefab in prefabList)
	{
		Initialize(prefab);
	}
}
```

Funkce Initialize vytváří instanci objektu ze zadaného prefabu a nastavuje mu jméno odpovídající jménu prefabu. Poté získává komponentu NetworkObject a používá metodu

Spawn ze třídy ServerManager instance NetworkManager, aby zaregistrovala a synchronizovala objekt přes síť. Na konzoli se pak vypisuje úspěšná inicializace objektu.

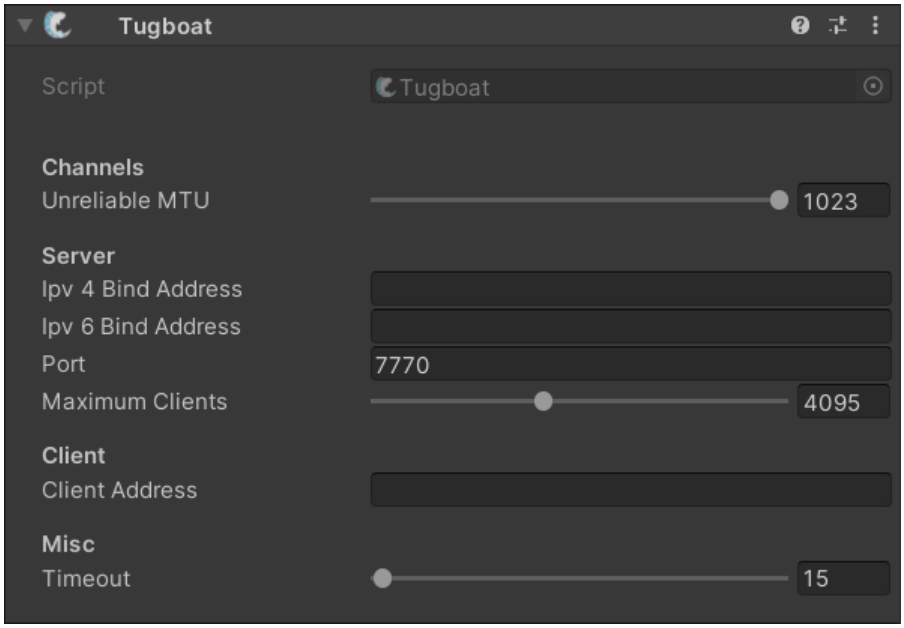
```
public void Initialize(GameObject prefab)
{
	GameObject go = Instantiate(prefab);
	go.name = prefab.name;
	NetworkObject no = go.GetComponent<NetworkObject>();

	InstanceFinder.NetworkManager.ServerManager.Spawn(no);

	Debug.Log($"{prefab.name} successfully initialized.");
}
```

5.2 Připojení uživatele na server

5.2.1 Tugboat

Jak již bylo uvedeno v teoretické části této práce, Tugboat slouží jako výchozí transport v knihovně FishNet a je využíván k navigaci uživatele v rámci UTW.

Nejprve byl Tugboat umístěn do MainMenuScene, jakožto komponenta GameObjectu NetworkManager (společně s dalšími komponentami jako např. NetworkManager, SceneManager, …)



Obrázek 29. Komponenta Tugboat

I když by se daly hodnoty IP adres zadat přímo do komponenty v rámci Unity editoru, je nutné díky různorodé interakci uživatele s prostředím nastavovat hodnoty za běhu.

```
private Tugboat GetTugboat()
{
	if (InstanceFinder.NetworkManager.TryGetComponent(out Tugboat t))
		return t;
	else
		return null;
}
```

Tímto způsobem je dostupný odkaz na komponentu Tugboat a je možné nastavit její hodnoty pomocí příslušných metod. Metoda StartConnection poté zahájí připojovací proces s parametrem bool server = false. Hodnoty pro adresu a port jsou pouze ilustrační.

```
private const string sAddress = "8.8.8.8";
private const ushort sPort = 1111;
public void JoinOfficialShard()
{
	var tugboat = GetTugboat();

	if (tugboat == null)
	{
		Debug.LogError("Couldn't find Tugboat component!");
		return;
	}

	tugboat.SetClientAddress(sAddress);
	tugboat.SetPort(sPort);

	tugboat.StartConnection(false);
}

public void DirectConnect()
{
	var tugboat = GetTugboat();

	if (tugboat == null)
	{
		Debug.LogError("Couldn't find Tugboat component!");
		return;
	}

	tugboat.SetClientAddress("127.0.0.1");

	tugboat.StartConnection(false);
}
```

Z testovacích účelů slouží nyní přímé připojení pouze pro připojení na localhost.

5.2.2 Autentifikace klienta

Na autentifikaci klienta se primárně podílejí dvě třídy: GameManager a Authenticator.

5.2.2.1 GameManager

Instance třídy GameManager je vytvořena při startu hry a obsahuje statickou vlastnost Instance, na kterou se odkazují další třídy:

```
public static GameManager Instance { get; private set; }
```

V metodě Awake je do této vlastnosti pouze jednou uložen odkaz na instanci této třídy, pokud by došlo k inicializaci další třídy GameManager je přeskočen zápis do Instance a nový GameManager je automaticky zničen. Společně se spuštěním inicializačních metod a napojením na obsluhu události při odpojení uživatele.

Instance třídy GameManager udržuje dvě důležité kolekce dat – SyncDictionary objekty pro uživatele (PlayerData) a frakce (Faction). Tyto kolekce jsou navrženy tak, aby umožňovaly sledování a aktualizaci dat prostřednictvím sítě, což je klíčové pro správu připojených uživatelů a frakcí na daném serverovém shardu.

```
[SyncObject]
private readonly SyncDictionary<string, PlayerData> _playersData = new();
[SyncObject]
private readonly SyncDictionary<int, Faction> _factions = new();
```

Při spuštění hry jsou tyto kolekce inicializovány na základě dat načtených ze serializovaných souborů JSON, které jsou uloženy na serverovém úložišti.

5.2.2.2 Authenticator

Třída Authenticator je vložena do NetworkManageru jakožto script, to znamená že při startu hry dojde k její inicializaci a vykoná se kód uvnitř metody InitializeOnce.

```
public override void InitializeOnce(NetworkManager networkManager)
{
	base.InitializeOnce(networkManager);

	// Event handler pro zaznamenání nového připojení
	// a registrace brodcastů na klientovi i serveru
	NetworkManager.ClientManager.OnClientConnectionState
	+= ClientManager_OnClientConnectionState;
```

```
        NetworkManager.ServerManager
        .RegisterBroadcast<UserBroadcast>(OnUserBroadcast, false);

        NetworkManager.ClientManager
        .RegisterBroadcast<ResponseBroadcast>(OnResponseBroadcast);
}
```

Při změně klientského připojení, tedy v okamžiku, kdy se klient pokouší připojit ke vzdálenému serveru, je vyvolána obsluha události ClientManager_OnClientConnection-State. Tato událost následně zajišťuje naplnění struktury uživatelského broadcastu a odesílá ji na server. Tento uživatelský broadcast obsahuje jméno uživatele a seznam hash hodnot ve formátu řetězce, které poté slouží k ověření vůči hash hodnotám na serveru.

```
public struct UserBroadcast : IBroadcast
{
    public string Username;
    public List<string> Hashes;
}

private void ClientManager_OnClientConnectionState(ClientConnectionStateArgs
args)
{
        if (args.ConnectionState != LocalConnectionState.Started)
                return;

        var input = GameObject.Find("UsernameInputText")
        .GetComponent<TMP_Text>().text;

        UserBroadcast ub = new UserBroadcast()
        {
                Username = input,
                Hashes = Database.hashes
        };

        NetworkManager.ClientManager.Broadcast(ub);
}
```

Vzhledem k tomu, že má server zaregistrovanou obsluhu pro příjem broadcastu od uživatele, je nyní schopen reagovat v metodě OnUserBroadcast. Tato metoda je v podsatě celá vyobrazené na grafu z obrázku 25. Jako parametry obdrží NetworkConnection a UserBroadcast.

V prvé řadě zkontroluje, zda uživatel se jménem, které obdrží v broadcastu existuje, pokud ano tak načte jeho data ze SyncDictionary _playerdata, pokud ne, tak jej do tohoto slovníku přidá.

```
// Odkaz na metodu třídy GameManageru
var player = GameManager.Instance.CreateOrSelectPlayer(ub.Username);
```

Následuje kontrola, zda již není uživatel s tímto jménem připojen na server. Metoda provádějící tuto kontrolu prochází synchronizovaným slovníkem a pokud najde shodné jméno, které bylo obdrženo v broadcastu s identifikátorem klienta odlišným od „-2“, vrátí bool = true. Toto označuje, že dané jméno je již připojeno, protože má přidělené své id, a uživateli je zaslán ResponseBroadcast s chybovou zprávou.

```
private bool IsAlreadyConnected(PlayerData player)
{
	if (GameManager.Instance.GetPlayerByName(player.PlayerName)
	.ClientConnectionId == -2)
		return false;

	return true;
}

if (IsAlreadyConnected(player))
{
	Debug.Log($"Player {player.PlayerName} is already connected!");

	SendAuthenticationResponse(conn, false, "Player with this name is
	already connected!");
	OnAuthenticationResult?.Invoke(conn, false);

	return;
}
```

Jakmile klient úspěšně prošel první kontrolou, začíná další kontrola nad druhou položkou v jeho broadcastu. Jedná se o porovnání dvou seznamů hash hodnot asset databáze obou stran. Metoda provádějící tento krok tedy obdrží seznam z uživatelského broadcastu a seznam z lokální instance tohoto seznamu na serveru. Pokud seznam klienta obsahuje všechny hash hodnoty, které má server, návratová hodnota je kladná. Klient může být připojen na server. V opačném případě je stejně jako v předešlé metodě, klientovi zaslán ResponseBroadcast s chybovou zprávou a není mu umožněno se připojit.

```
private bool UserHasRequiredHashes(List<string> hashes)
{
	HashSet<string> userHashes = new HashSet<string>(hashes);
	HashSet<string> serverHashes = new HashSet<string>(Database.hashes);

	return userHashes.IsSupersetOf(serverHashes);
}
```

```
if (!UserHasRequiredHashes(ub.Hashes))
{
	Debug.Log($"Player {player.PlayerName} has different AssetDB!");

	SendAuthenticationResponse(conn, false, "You have a different
	AssetDB!");
	OnAuthenticationResult?.Invoke(conn, false);

	return;
}
```

Pokud jsou všechny kontroly úspěšně dokončeny, je uživatel považován za autentifikovaného. Následně je zaznamenán identifikátor připojení (NetworkConnection.clientId) uživatele u jména které mu nyní přísluší v rámci slovníku _playersData třídy GameManager.

```
player.ClientConnectionId = conn.ClientId;
GameManager.Instance.UpdateDictionary(player.PlayerName);

Debug.Log($"Player {player.PlayerName} authenticated!");

SendAuthenticationResponse(conn, true, "Authentication complete.");
OnAuthenticationResult?.Invoke(conn, true);
```

Metoda UpdateDictionary poté provede aktualizaci pole s klíčem jména daného uživatele. Tento postup je nezbytný pro zajištění toho, že budou změny zapsány i pro ostatní uživatele, a slovník tak bude skutečně synchronizovaný.

```
[Server]
public void UpdateDictionary(string name)
{
	_playersData.Dirty(name);
}
```

5.2.3 Odpojení klienta

Při změně stavu připojení klienta dojde k odchycení této události na instanci GameManageru, který následně zareaguje vyvoláním obsluhy ve formě metody ServerManager_OnRemoteConnectionState, do které jsou dodány parametry – NetworkConnection odpojujícího se klienta a stav jeho připojení, v tomto případě je očekáván stav RemoteConnectionState.Stopped. Pokud se tento stav naplní je klient s daným jménem označen jako odpojený, tudíž je jeho clientId změněno na hodnotu „-2“ a nakonec je provedena aktualizace slovníku _playersData.

```
private void ServerManager_OnRemoteConnectionState(NetworkConnection conn,
RemoteConnectionStateArgs args)
{
	if (args.ConnectionState == RemoteConnectionState.Stopped)
	{
		Debug.Log($"The user with id: {conn.ClientId} has
		disconnected!");

		try
		{
			PlayerData p = GetPlayerByConnection(conn.ClientId);
			p.ClientConnectionId = -2;

			UpdateDictionary(p.PlayerName);
		}
		catch (System.Exception)
		{
			Debug.LogWarning("Couldn't find a matching connection.");
		}
	}
}
```

5.3 Scéna hlavního menu

Tato scéna představuje jednoduché lokální prostředí, které primárně slouží k navigaci uživatele. Klient zde má možnost zadat své uživatelské jméno, které bude následně uloženo na serveru, ke kterému se rozhodne připojit. Toto jméno bude viditelné i pro ostatní klienty, kteří jsou připojeni ke stejnému serveru.

Poté zbývá pouze volba mezi připojením na oficiální shard nebo jiný server. V současné době umožňuje druhá možnost pouze připojení na localhost pro usnadnění testování, avšak v budoucnu bude implementován jednoduchý pop-up s textovým polem, kam uživatel zadá požadovanou IP adresu pro připojení na konkrétní server.

Obrázek 30. Scéna hlavního menu

5.4 Shard scéna

Funkcionalitu této scény na pozadí zpracovává třída SceneManager. O obsluhu uživatelských událostí na front-endu se stará třída ShardController a z části i třída LobbyController.

Do Shard scény je klient přesunut po úspěšné autentifikaci pomocí obsluhy události, přiřazené při inicializaci instance SceneManager.

```
private void MoveClientToShardScene(NetworkConnection conn, bool passed)
{
	if (!passed) return;

	SceneLoadData data = new SceneLoadData(GameSceneUtils.SHARD_SCENE);
	data.ReplaceScenes = ReplaceOption.All;

	InstanceFinder.SceneManager.LoadConnectionScenes(conn, data);
}
```

Na obrázku 31. je zachycena scéna shardu (1). V této fázi již došlo k vytvoření dvou lobby místností na daném shardu. Jedna z těchto místností má připojené dva uživatele, zatímco druhá místnost má pouze jednoho uživatele. Tyto informace jsou klientovi po připojení k danému shardu okamžitě dostupné.

Klient má nyní možnost vybrat si, zda se připojí do již existující lobby, nebo vytvoří vlastní a stane se jejím vlastníkem.

Obrázek 31. Shard scéna (1) a scény lobby (2, 3, 4)

Při načtení shard scény, a tedy i inicializaci ShardControlleru dojde ke spuštění korutiny, která následně volá metodu RefreshLobby. Tato metoda, jak již název naznačuje, obnovuje seznam vytvořených lobby danému klientovi.

```
private void Start()
{
	ChangeStatusText("Refreshing...");
	StartCoroutine(LateStart());
}

public IEnumerator LateStart()
{
	yield return new WaitForSeconds(1);
	RefreshLobby();
}

public void RefreshLobby()
{
	if (InstanceFinder.IsServer) return;

	ChangeStatusText("Refreshing...");
	ClearButtons();
	UTW.SceneManager.Instance.GetLobbyData(InstanceFinder.ClientManager
	.Connection);
}
```

V metodě RefreshLobby je volána metoda instance SceneManageru GetLobbyData s parametrem klientského připojení, aby server věděl, který uživatel žádá o obnovení seznamu. Tato metoda je volána jako ServerRpc.

```
private Dictionary<int, SceneData> lobbyData;

[ServerRpc(RequireOwnership = false)]
public void GetLobbyData(NetworkConnection conn)
{
	GetLobbyDataResponse(conn, lobbyData);
}
```

Server poté obratem posílá data (pokud existují) o vytvořených lobby zpět na připojení, které o tato data žádalo, prostřednictvím TargetRpc. Server identifikuje ShardController příslušného klienta a vzdáleně spouští metodu CreateLobbyButtons s předanými informacemi o lobby místnostech.

```
[TargetRpc]
public void GetLobbyDataResponse(NetworkConnection conn, Dictionary<int,
SceneData> lobbyData)
{
	ShardController sc = FindObjectOfType<ShardController>();

	if (sc != null)
	{
		sc.CreateLobbyButtons(lobbyData);
	}
}
```

5.4.1 Vytvoření nové lobby scény

Vytváření nové lobby místnosti je iniciováno klientem při zvolení této možnosti v shard menu. Jakmile se uživatel rozhodne pro vytvoření nové lobby je zavolána metoda CreateLobby na instanci SceneManageru CreateLobby s parametrem klientského připojení.

```
public void CreateLobby()
{
	UTW.SceneManager.Instance.CreateLobby(InstanceFinder.ClientManager
	.Connection);
}
```

Tato metoda na serveru provede načtení scény pomocí metody LoadScene, jíž jsou předány parametry – připojení, scéna, která má být načtena, a nakonec bool hodnota allowStacking, která určuje, zda se uživatel připojuje k nové scéně nebo již existující.

```
[ServerRpc(RequireOwnership = false)]
public void CreateLobby(NetworkConnection conn)
{
	Debug.Log($"Creating new lobby by client ID: {conn.ClientId}...");
	LoadScene(conn, new SceneLookupData(GameSceneUtils.LOBBY_SCENE),
	true);
}

private void LoadScene(NetworkConnection conn, SceneLookupData lookupData,
bool allowStacking)
{
	SceneLoadData sceneLoadData = new SceneLoadData(lookupData);
	sceneLoadData.Options.AllowStacking = allowStacking;
	sceneLoadData.ReplaceScenes = ReplaceOption.OnlineOnly;
	sceneLoadData.PreferredActiveScene = lookupData;
	InstanceFinder.SceneManager.LoadConnectionScenes(conn, sceneLoad-
	Data);
}
```

5.4.1.1 Načtení lobby scény

Po načtení scény dojde i k inicializaci LobbyControlleru, který je v lobby scéně umístěn jako GameObject. V Awake metodě této třídy je přidána obsluha události dokončení načtení scény, stejně jako přidělení odkazu na statickou instanci SceneManageru a NetworkConnetion klienta.

```
private void Awake()
{
	if (InstanceFinder.IsServer) return;

	InstanceFinder.SceneManager.OnLoadEnd += SceneLoadEnd;

	sceneManager = UTW.SceneManager.Instance;
	conn = InstanceFinder.ClientManager.Connection;

	[…]
}
```

Jakmile je načítání scény u klienta dokončeno je zavolána obsluha této události ve formě metody SceneLoadEnd. Tato metoda vzdáleně provede inicializaci potřebných komponent na serveru (LobbyManager, Chat) metodami InitializeLobbyManagers a InitializeChatManager. Dále přiřadí klienta do dané scény metodou Connected.

```
private void SceneLoadEnd(SceneLoadEndEventArgs args)
{
	sceneManager.InitializeLobbyManagers(lobbyManagerPrefab, chatPrefab,
	conn);
	sceneManager.InitializeChatManager(chatManagerPrefab, conn);
	sceneManager.Connected(conn);
}
```

5.4.1.2 Metoda InitializeLobbyManagers

Jak bylo již zmíněno, metoda InitializeLobbyManagers má na starosti vytvoření instance LobbyManager a Chat. Nejprve zjišťuje, ve které scéně má tyto komponenty vytvořit. Pokud daná scéna neexistuje (není zaznamenána ve slovníku lobbyData), pokračuje běh metody dál. V případě, že scéna již existuje, je běh metody přerušen, aby nedošlo k duplikaci těchto GameObjectů. Tento případ nastává, když se do existující lobby připojuje nový klient.

Následně dochází k vytvoření záznamu pro danou scénu ve slovníku lobbyData. Tento slovník obsahuje objekty typu SceneData, které uchovávají informace o jednotlivých lobby scénách, jako jsou identifikátor, název lobby, vlastník lobby, seznam připojených klientů atd.

Poté se provádí samotné vytvoření instance LobbyManageru a Chatu pomocí metody Spawn třídy ServerManager. Parametry této metody jsou: go (prefab daného GameObjectu), conn (připojení, které bude vlastníkem vytvořené instance) a scene (scéna, ve které bude objekt vytvořen).

```
[ServerRpc(RequireOwnership = false)]
public void InitializeLobbyManagers(GameObject lobbyManagerPrefab,
GameObject chatPrefab, NetworkConnection conn)
{
	Scene scene = GetSceneForClient(conn, GameSceneUtils.LOBBY_SCENE);

	if (lobbyData.ContainsKey(scene.handle)) return;

	CreateNewLobbyData(conn, scene);

	GameObject go = Instantiate(lobbyManagerPrefab);
	go.name = lobbyManagerPrefab.name;
	InstanceFinder.ServerManager.Spawn(go, conn, scene);
	Debug.Log($"{go.name} successfully initialized.");

	go = Instantiate(chatPrefab);
	go.name = chatPrefab.name;
	InstanceFinder.ServerManager.Spawn(go, conn, scene);
```

```
        Debug.Log($"{go.name} successfully initialized.");

        ActivateStartButtonForLobbyOwner(conn);
}
```

5.4.1.3 Metoda InitializeChatManager

Tato metoda provádí v podstatě stejnou činnost jako InitializeLobbyManagers, s tím rozdílem, že vytváří ChatManager, který se ve scéně může vyskytovat mnohokrát (podobně jako například VehicleManager). Každý klient připojený do lobby má svou instanci třídy ChatManager, která slouží pro řízení odesílání a mazání zpráv v rámci chat systému.

Každá nově vytvořená instance je zařazena do druhého slovníku třídy SceneManager – chatManagers. Na základě tohoto slovníku se poté vyhledává GameObject pro ChatManager, který je nutné odstranit, například v případě odpojení jeho majitele.

```
[ServerRpc(RequireOwnership = false)]
public void InitializeChatManager(GameObject chatManagerPrefab,
NetworkConnection conn)
{
        Scene scene = GetSceneForClient(conn, GameSceneUtils.LOBBY_SCENE);
        GameObject go = Instantiate(chatManagerPrefab);
        go.name = chatManagerPrefab.name;

        InstanceFinder.ServerManager.Spawn(go, conn, scene);
        chatManagers[conn] = go;

        Debug.Log($"{go.name} successfully initialized.");
}
```

5.4.1.4 Metoda Connected

Metoda Connected je volána jako třetí a poslední z těchto hlavních inicializačních metod. Slouží jako začátek procesu zapsání klienta do seznamu připojených uživatelů dané scény.

```
[ServerRpc(RequireOwnership = false)]
public void Connected(NetworkConnection conn)
{
        AddClientData(conn);
        OnClientJoinLobby?.Invoke(conn);
}
```

V následujících metodách dochází k vyhledání scény, ve které se dané připojení (klient) nachází, dále k jeho zařazení do listu připojení scény a inkrementaci počtu připojení scény.

```
private void AddClientData(NetworkConnection conn)
{
	var data = GetData(conn);
	if (data == null) return;
	data.playerCount++;
	data.clients.Add(conn);
}

private SceneData GetData(NetworkConnection conn)
{
	Scene scene = GetSceneForClient(conn, GameSceneUtils.LOBBY_SCENE);

	foreach (var dataPair in lobbyData)
	{
		if (dataPair.Key == scene.handle)
		{
			return dataPair.Value;
		}
	}

	return null;
}

public Scene GetSceneForClient(NetworkConnection conn, string sceneName)
{
	return conn.Scenes.First(x => x.name.Equals(sceneName));
}
```

Odstranění klienta z lobby se nese ve velice podobném duchu, pouze se zrcadlenými operacemi v metodě AddClientData -> RemoveClientData.

5.4.2 Připojení se do existující lobby

Každé lobby tlačítko v rámci scény shardu (viz Obrázek 31, 1. panel) obsahuje identifikátor příslušné lobby scény. Tento identifikátor je při vytváření tlačítka přidělen jako parametr pro metodu ConnectToLobby, která má být zavolána při kliknutí.

```
button.onClick.AddListener(() => ConnectToLobby(sceneData.handle));
```

Skrze lokální metodu ConnectToLobby je následně volána vzdálená metoda ConnectToLobby pomocí ServerRpc. Tato metoda se pokouší pomocí parametru handle spojit s

existující lobby místností. Pokud spojení proběhne úspěšně (handle existuje v rámci slovníku lobbyData), klientovi je načtena scéna pomocí metody LoadScene, tentokrát však s parametrem allowStacking nastaveným na false. Poté je proces připojování identický s procesem při vytváření lobby.

Pokud by příslušná místnost nebyla nalezena, klientovi je odeslána zpráva o neúspěšném pokusu. K takové situaci může dojít například v případě, že má uživatel načtený zastaralý seznam a pokusí se připojit na lobby, které již jeho vlastník stihl zrušit.

```
[ServerRpc(RequireOwnership = false)]
public void ConnectToLobby(NetworkConnection conn, int handle)
{
	Debug.Log($"Connecting client ID: {conn.ClientId} to lobby...");

	if (lobbyData.TryGetValue(handle, out SceneData sceneData))
	{
		LoadScene(conn, new SceneLookupData(handle), false);
	}
	else
	{
		LogResponse(conn, "Cannot connect to lobby.");
	}
}
```

5.4.3 Odstranění lobby

K odstranění/skrytí lobby dojde ve dvou případech:

- Odpojí se majitel lobby
  - V tomto případě je lobby zničena.
- Majitel lobby spustí hru
  - V tomto případě je lobby skryta a novým klientům není dovoleno se připojit.
  - Tento případ bude popsán v kapitole 5.5.

Během procesu odpojování klienta je na úrovni třídy SceneManager zavolána metoda Disconnect, která přijímá jako jediný parametr NetworkConnection uživatele, který se právě odpojuje. Nejprve je využita již probíraná metoda GetData, která díky NetworkConnection zjistí, v jaké scéně se klient nachází.

Následně dochází ke kontrole, zda připojení, které vyvolalo tuto metodu, je vlastníkem lobby:

- Pokud je připojení vlastníkem lobby, všichni klienti v lobby jsou přesunuti zpět do scény Shard. K tomuto účelu je upravena metoda LoadScene, která namísto jednoho připojení, pro které má načíst novou scénu, přijímá pole všech připojení v dané lobby.
- Pokud tomu tak není, je shard scéna načtena pouze připojení, které vstoupilo jako parametr do metody Disconnect a dojde k jeho odpojení od lobby.

```
[ServerRpc(RequireOwnership = false)]
public void Disconnect(NetworkConnection conn)
{
	var data = GetData(conn);
	if (RemoveLobbyData(conn, data))
	{
		Debug.Log($"Disconnecting all clients from lobby...");
		LoadScene(data.clients.ToArray(), new SceneLookupData(
		GameSceneUtils.SHARD_SCENE), false);
		return;
	}

	Debug.Log($"Updating data for cliend ID: {conn.ClientId}");
	RemoveClientData(conn);

	Debug.Log($"Despawning ChatManager for cliend ID: {conn.ClientId}");
	DespawnChatManager(conn);

	Debug.Log($"Disconnecting client ID: {conn.ClientId} from lobby...");
	LoadScene(conn, new SceneLookupData(GameSceneUtils.SHARD_SCENE),
	false);

	OnClientDisconnectLobby?.Invoke(conn);
}
```

5.4.4 Odpojení od Shardu

K odpojení od shardu dojde v případě, kdy uživatel vybere příslušné tlačítko pro odpojení v rámci scény shardu.

Nejprve je zavoláno ServerRpc na metodu DisconnectFromShard třídy SceneManager, která jednoduše provede odpojení klienta, který poslal žádost. Poté je uživatel považován za nepřipojeného a je použita třída SceneManager, konkrétně metoda LoadScene, která změní lokální scénu zpět na hlavní menu.

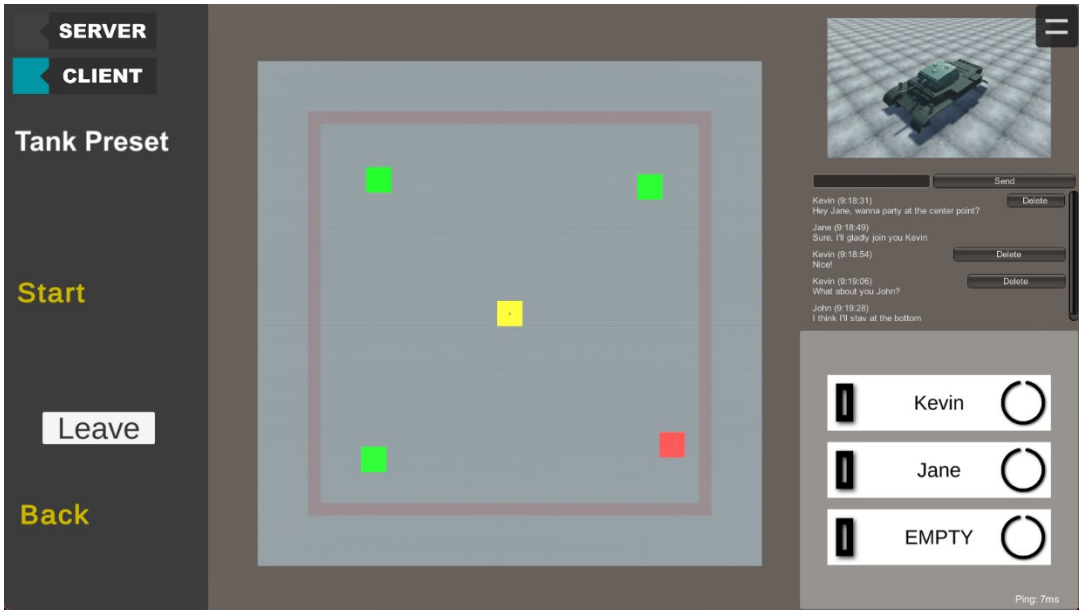
```
// Klient
private void Disconnect()
{
      UTW.SceneManager.Instance.DisconnectFromShard(InstanceFinder
      .ClientManager.Connection);

      SceneManager.LoadScene(GameSceneUtils.MAIN_MENU_SCENE);
}

// SceneManager (server)
[ServerRpc(RequireOwnership = false)]
public void DisconnectFromShard(NetworkConnection conn)
{
      conn.Disconnect(true);
}
```

5.5 Lobby scéna

Na funkci této scény se podílí hned několik tříd:

- Za zajištění back-end funkcionality jsou odpovědné třídy LobbyManager, VehicleManager a SceneManager, spolu s dalšími.
- O front-end se převážně stará třída LobbyController, která zpracovává UI prvky. Důležitou roli také hraje třída MapController, která se zabývá všemi událostmi souvisejícími s mapou a výběrem spawnpointů.

Po připojení do lobby je načtena prozatímní testovací mapa, kde si může uživatel vybrat z pěti spawnpointů.



Obrázek 32. Lobby scéna

5.5.1 Inicializace mapy a spawnpointů

K inicializaci mapy dochází v metodě InitializeMap, která je volána na serveru ihned po načtení scény Lobby.

```
private void Start()
{
	if (InstanceFinder.IsServer)
	{
		InitializeMap(defaultMap);
	}
[…]
```

V této metodě dochází k vytvoření mapy, která je definovaná jako vstupní parametr ve formě GameObjectu. Jedná se o prefab dané mapy, a je dosazen do serializovaného pole v rámci Unity editoru.

Prefab mapy obsahuje další prefaby reprezentující jednotlivé spawnpointy, o které se následně stará následující metoda InitializeSpawnpoints.

```
private void InitializeMap(GameObject mapToSpawn)
{
	GameObject map = Instantiate(mapToSpawn);
	activeMap = map;

	Spawn(map, null, gameObject.scene);

	InitializeSpawnpoints();
}
```

Metoda InitializeSpawnpoints nejprve hledá všechny prefaby s tagem odpovídajícím spawnpointu (GameTagsUtils.MAP_SPAWNPOINT). Zde však může potencionálně dojít k nalezení spawnpointů v jiných scénách. Tomu zabrání filtr .Where, který porovnává scénové handles a zaručí že jsou vybrány pouze ty spawnpointy, které se nacházejí ve stejné scéně jako GameObject LobbyManager.

Po následném seřazení pole bodů, je pro každý spawnpoint vytvořen nový klíč (jméno spawnpointu) a také GameObject, který je vyjmut z aktuální pozice v poli. Tato data jsou poté přidána synchronizovaného slovníku _spawnpoints, který udržuje data o spawnpointech v dané lobby a synchronizuje je s klienty.

```
[SyncObject]
private readonly SyncDictionary<string, MapSpawnpointData> _spawnpoints =
new();

private void InitializeSpawnpoints()
{
	GameObject[] spawnpoints = GameObject
	.FindGameObjectsWithTag(GameTagsUtils.MAP_SPAWNPOINT)
	.Where(x => x.scene.handle == gameObject.scene.handle).ToArray();

	Array.Sort(spawnpoints.Where(x => x.scene.handle == gameOb-
	ject.scene.handle).ToArray(), (a, b) => a.name.CompareTo(b.name));

	for (int i = 0; i < spawnpoints.Length; i++)
	{
		string newKey = spawnpoints[i].name;
		GameObject spawnpoint = spawnpoints[i];
		_spawnpoints.Add(newKey, new MapSpawnpointData(spawnpoint, spa-
		wnpoint.transform));
	}
}
```

5.5.2 Management spawnpoint bodů a posádky

Pro zajištění kontroly nad stavem spawnpointů a posádkami na těchto bodech jsou založeny synchronizované slovníky ve třídách LobbyManager (viz. Kapitola 5.5.1.) a VehicleManager:

```
[SyncObject, HideInInspector]
public readonly SyncDictionary<int, CrewData> _tankCrew =
new SyncDictionary<int, CrewData>();
```

Každý spawnpoint má svou vlastní instanci třídy VehicleManager (pokud je inicializována, připojením klienta) v rámci objektu MapSpawnpointData a tedy i nepřímý přístup k datům o posádce na daném bodě.

5.5.2.1 Výběr spawnpointu

Při výběru spawnpointu klientem je pomocí ServerRpc volána metoda SelectSpawnpoint s parametry připojení klienta, newKey = klíč klientem vybraného spawnpointu a activeKey = klíč spawnpointu, na kterém se klient právě nachází.

V této metodě se následně kontroluje, v jakém stavu se nachází nově zvolený bod a podle něj se následně vykonávají požadované akce.

```
[ServerRpc(RequireOwnership = false)]
private void SelectSpawnpoint(NetworkConnection conn, string newKey, string
activeKey)
{
	switch (_spawnpoints[newKey].spawnpointState)
	{
		case SpawnpointState.EMPTY:
		{
			ChangeSpawnpoint(conn, newKey, activeKey);
		}
		break;
		case SpawnpointState.LOCKED:
		{
			LogResponse(conn, "Cannot join... CREW is not ready.");
		}
		break;
		[…]
	}
}
```

Pokud je bod prázdný (EMPTY) je spuštěna metoda ChangeSpawnpoint, která zjišťuje, zda klient nemá záznam v parametru activeKey, což by znamenalo že je již přihlášen na jiném spawnpointu a funkce se tedy neprovede. Pokud tomu tak není, je uživatel přihlášen k tomuto bodu a je pro něj a jeho nový spawnpoint inicializována instance VehicleManageru.

```
private void ChangeSpawnpoint(NetworkConnection conn, string newKey, string
activeKey)
{
	if (activeKey is not null)
		return;

	if (activeKey is null)
	{
		InitializeVehicleManager(conn, newKey);
		return;
	}
}
```

Pokud je spawnpoint zamknutý (LOCKED), není možné se k tomuto bodu připojit, neboť vlastník bodu vybírá preset. Není tedy známo, kolik hráčů bude bod schopen obsáhnout, jinými slovy, kolik bude mít tank pozic. Uživateli je tedy poslán důvod pro odmítnutí.

Pokud je spawnpoint odemčený (UNLOCKED), je volána metoda JoinSpawnpoint, která v prvé řadě kontroluje, zda se uživatel již nenachází na jiném spawnpointu a v jiné posádce. Pokud ano, metoda se nevykoná.

V opačném případě, je přiřazen k jeho nově zvolenému bodu a taktéž do prvního volného místa posádky, která je k tomuto bodu přiřazena, pomocí metody JoinCrew. Pokud metoda JoinCrew vrátí kladnou hodnotu bool, je spuštěna metoda SetSpawnpointFull, jelikož je posádka plná a nemělo by tedy být dále možné se k bodu přiřadit.

V dalších krocích jsou klientovi od serveru odeslány dvě TargetRpc metody. První s názvem SetSpawnpointCamera zobrazí náhledovou kameru nad bodem klienta, aby si mohl prohlédnout jeho tank a počáteční pozici. Druhá metoda SetKey provede přepsání lokálního záznamu klienta – activeKey na klíč jeho nového spawnpointu.

```
private void JoinSpawnpoint(NetworkConnection conn, string newKey, string
activeKey)
{
	if (activeKey is not null && _spawnpoints[activeKey]
	.vehicleManager.IsInCrew(conn))
	{
		LogResponse(conn, "Cannot join... Already in CREW.");
		return;
	}

	if (_spawnpoints[newKey].vehicleManager.JoinCrew(conn))
		SetSpawnpointFull(newKey);

	SetSpawnpointCamera(conn, newKey);
	Debug.Log($"Client ID: {conn.ClientId} joined CREW at position
	{_spawnpoints[newKey].transform.position}.");
	SetKey(conn, "CREW joined.", newKey);
}
```

Čtvrtý a poslední stav byl již lehce zmíněn a jedná se o takový stav kdy je spawnpoint, respektive posádka na něm, plná (FULL). V takovémto případě je uživateli stejně jako ve stavu LOCKED poslána zpráva o odmítnutí a uveden důvod.

Všechny tyto stavy jsou signalizovány změnou materiálu (barvy) daného bodu, aby měl uživatel přehled, kam se může připojit a kam ne. Operaci změny materiálu vykonává metoda SetSpawnpointMaterial, která je volána pokaždé když dojde ke změně v synchronizovaném slovníku _spawnpoints.

5.5.2.2 Opuštění spawnpointu

Pokud se například uživatel rozhodne, že nechce hrát s posádkou do které se přidal nebo chce začít na jiném bodě musí nejprve odejít ze současného spawnpointu.

Při kliknutí na tlačítko „Leave“ je volána lokální metoda LeaveSpawnpoint, která skryje UI prvky, které by měly být dostupné pouze tehdy, když je uživatel na spawnpointu. Poté vyhledává ve scéně objekt typu LobbyManager a prostřednictvím řetězce metod spouští kód na serveru. Do serverové metody jsou předány dva parametry – NetworkConnection a activeKey (identifikátor spawnpointu).

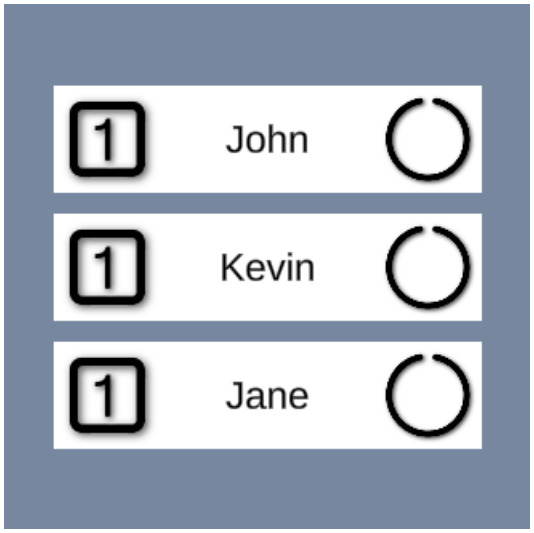
Metoda kontroluje, zda je klient poslední člen posádky a pokud ano, odstraní vytvořenou instanci VehicleManageru a nastaví spawnpoint do stavu EMPTY metodou Unlock. V opačném případě nastaví stav na UNLOCKED.

V poslední řadě skryje náhledovou kameru nad spawnpointem a nastaví klientův activeKey na null.

```
private void LeaveSpawnpoint(NetworkConnection conn, string activeKey)
{
	if (activeKey is null) return;
	MapSpawnpointData data = _spawnpoints[activeKey];
	if (data.vehicleManager.LeaveCrew(conn))
	{
		data.vehicleManager.Despawn();
		Unlock(conn, activeKey);
	}
	else
	{
		data.spawnpointState = SpawnpointState.UNLOCKED;
		_spawnpoints.Dirty(activeKey);
	}

	SetSpawnpointCamera(conn, activeKey, false);
	Debug.Log($"Client ID: {conn.ClientId} left CREW at position
	{data.transform.position}.");
	SetKey(conn, $"CREW left", null);
}
```

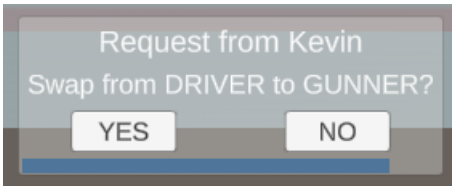
5.5.2.3 Změna pozic v tanku

V rámci posádky může uživatel změnit svoji pozici tím, že vybere buď prázdnou pozici nebo pozici, ke které je již přihlášen jiný uživatel.



Obrázek 33. Panel posádky

Jako příklad lze uvést situaci kdy chce klient Kevin (2. pozice) obsadit pozici řidiče (1. pozice), tuto pozici však vlastní John. Když Kevin klikne na tlačítko na první pozici, Johnovi se zobrazí výzva od Kevina o změně pozic. John se nyní může rozhodnout, zda vyhoví žádosti či nikoli.



Obrázek 34. Žádost o výměnu pozic (Pohled klienta „John“)

Štítky (viz. Obrázek 33.) jsou vytvářeny dynamicky pomocí slovníku _tankCrew. Každému je přiděleno tlačítko s obsluhou události v podobě metody SwapRequest. Parametry této metody jsou LocalConnection, jelikož žadatel bude vždy lokální klient a klíč dané pozice v tanku.

```
go.GetComponentInChildren<Button>().onClick.AddListener(() =>
SwapRequest(LocalConnection, key));
```

Metoda začíná získáním dat o cílové pozici pomocí klíče key. Dále získává klíč aktuální pozice žadatele.

Dále probíhá kontrola validnosti požadavku.

- Pokud žadatel již požaduje výměnu pozice jinde, vyvolá se odpověď s logem a metoda končí.
- Pokud je požadující připojení již na pozici, kterou chce získat, loguje se odpovídající zpráva a metoda končí.
- Pokud je cílová pozice prázdná, dojde k provedení výměny pomocí metody Swap a metoda končí.
- Pokud jiný klient již požaduje cílovou pozici, loguje se zpráva o tomto stavu a metoda končí.

Pokud žádné z předchozích podmínek neplatí, metoda pokračuje nastavením flagů pro výměnu na obou pozicích pomocí metody SetSwapping(key, oldKey, true).

Následně je uživateli na požadované pozici zobrazen UI prvek, kde může žádost přijmout nebo zamítnout skrze TargetRpc metodu SwapRequestPopup (viz. Obrázek 34.).

```
[ServerRpc(RequireOwnership = false)]
public void SwapRequest(NetworkConnection requestConn, int key)
{
	CrewData data = _tankCrew[key];
	int oldKey = GetKey(requestConn);
	if (_tankCrew[oldKey].swapRequest)
	{
		LogResponse(requestConn, "Cannot swap while requesting another
		position.");
		return;
	}
	if (data.conn == requestConn)
	{
		LogResponse(requestConn, "Already in this position.");
		return;
	}
	if (data.empty)
	{
		Swap(requestConn, key);
		return;
	}
	if (data.swapRequest)
	{
		LogResponse(requestConn, "Another client is requesting this
		position.");
		return;
	}

	SetSwapping(key, oldKey, true);
	SwapRequestPopup(data.conn, requestConn, key, oldKey);
	Debug.Log($"Client ID: {requestConn.ClientId} requesting position
	{key} in tank.");
}
```

Odpověď od majitele požadovaného místa dorazí ve formě ServerRpc metody SwapRequestResponse.

Pokud je parametr swap nastaven na true, znamená to, že klient souhlasí s výměnou. V takovém případě metoda volá funkci Swap, která provádí samotnou výměnu pozic. Po úspěšné výměně metoda končí (return).

Pokud je swap nastaven na false, znamená to, že klient odmítá provést výměnu. Metoda poté zavolá SetSwapping, s parametrem isSwaping = false, což nastaví dočasné flagy, které určují, zda probíhá výměna zpět na původní hodnoty (false).

```
[ServerRpc(RequireOwnership = false)]
public void SwapRequestResponse(NetworkConnection requestConn, int key, int
oldKey, bool swap)
{
	if (swap)
	{
		Swap(requestConn, key);
		return;
	}
	SetSwapping(key, oldKey, false);
	LogResponse(requestConn, "Client declined swap");
}
```

Metoda Swap započne načtením všech potřebných informací ve formě objektů CrewData z pozic vstupních klíčů ve slovníku _tankCrew. Následně proběhne prohození pozic mezi klienty a předání vlastnictví NetworkObjectu tankPart.

```
private void Swap(NetworkConnection requestConn, int key)
{
	int oldKey = GetKey(requestConn);
	if (oldKey == -1) return;
	CrewData oldData = _tankCrew[oldKey];
	CrewData requestedData = _tankCrew[key];

	var conn = requestedData.conn;
	oldData.conn = conn;
	oldData.empty = conn is null;
	oldData.tankPart.GiveOwnership(conn);

	requestedData.conn = requestConn;
	requestedData.empty = false;
	requestedData.tankPart.GiveOwnership(requestConn);

	_tankCrew.Dirty(key);
	_tankCrew.Dirty(oldKey);

	SetSwapping(key, oldKey, false);
}
```

Tato funkce zůstává aktivní i po spuštění hry. To znamená, že tank teoreticky může ovládat pouze jeden člověk. Nicméně myšlenka spočívá v tom, že hráč, který takto ovládá svůj tank, bude mnohem méně efektivní než hráči v tanku, kde na každé pozici sedí individuální obsluha.

5.5.3 Inicializace VehicleManager

Nejprve metoda načte data o spawnpointu podle klíče newKey ze slovníku _spawnpoints. Poté je na pozici tohoto spawnpointu vytvořen nový objekt VehicleManager z prefabu. Objekt je následně zaregistrován v síťovém prostředí a odpovídající scéně pomocí volání metody Spawn.

Následně je na nově vytvořené instanci VehicleManageru volaná metoda SetCrewData, která mimo jiné vytvoří samotný objektu tanku z výchozího presetu. Systém presetů je v této fázi vývoje pouze testovací a bude se v budoucnu měnit.

```
private void InitializeVehicleManager(NetworkConnection conn, string newKey)
{
	MapSpawnpointData data = _spawnpoints[newKey];

	GameObject go = Instantiate(vehicleManagerPrefab, data.transform
	.position, Quaternion.identity);
	Spawn(go, null, gameObject.scene);
	Debug.Log($"{go.name} successfully initialized.");

	VehicleManager vehicleManager = go.GetComponent<VehicleManager>();
	vehicleManager.SetCrewData(Preset.CreateDefaultPreset(), conn,
	data.transform.position);
	vehicleManager.JoinCrew(conn);

	SetSpawnpointCamera(conn, newKey);

	data.vehicleManager = vehicleManager;
	data.spawnpointState = SpawnpointState.LOCKED;
	data.locked = true;
	_spawnpoints.Dirty(newKey);

	Debug.Log($"Client ID: {conn.ClientId} created CREW at position
	{data.transform.position}.");
	SetKey(conn, "CREW created.", newKey);
}
```

5.5.4 Chat

O funkcionalitu této mechaniky se stará třída Chat. Tato třída je vytvořena jako GameObject při načtení lobby scény (pomocí prefabu) společně s objektem LobbyManager. Pro ukládání zpráv slouží synchronizovaný list _messages, do kterého server na vyžádání klienta přidává zprávy. Uživateli je také umožněno svou zprávu smazat.

```
[SyncObject]
private readonly SyncList<ChatMessage> _messages = new();
```

Klienti si následně mohou z tohoto listu číst pomocí IReadOnlyList. Účelem této vlastnosti je poskytnout přístup k seznamu zpráv reprezentovaných typem ChatMessage. Tímto způsobem může vnější kód prohlížet zprávy, aniž by mohl seznam zpráv, jakkoliv modifikovat. Tuto schopnost by měl mít pouze server.

```
public IReadOnlyList<ChatMessage> Messages
{
	get => _messages;
}
```

Každému klientovi je při připojení do lobby scény také vytvořen jeho vlastní GameObject ChatManager, který zodpovídá za renderování samotného chatu a slouží pro volání ServerRpc metod třídy Chat.

```
[ServerRpc(RequireOwnership = false)]
public void ServerPostMessage(NetworkConnection conn, DateTime timestamp,
string username, string content)
{
	PostMessage(conn, timestamp, username, content);
}

[Server]
public void PostMessage(NetworkConnection conn, DateTime timestamp, string
sender, string content)
{
	_messages.Add(new ChatMessage(conn, timestamp, sender, content));

	Debug.Log($"{sender} sent a message: {content} ({timestamp:T})");
}
```

5.5.5 Spuštění hry

Majitel lobby spustí hru pomocí tlačítka, které je viditelné pouze pro něj. Jakmile tento proces začne je na serveru, v rámci třídy SceneManager, spuštěna metoda StartGame, která zjistí, v jaké scéně se připojení nachází.

Poté proběhne zničení GameObjectu ChatManageru pro všechny klienty v lobby, neboť tato funkce není v rámci hry povolena. Následně je spuštěna metoda StartGame v GameObjectu LobbyManageru, který je spojen s touto scénou. Metoda je volána s atributem ObserversRpc, tak aby byla provedena na všech klientech v lobby.

Nakonec je tato lobby označena jako probíhající (LobbyState.ONGOING), což znamená, že už není umožněno se dalším klientům k této lobby připojit.

```
[ServerRpc(RequireOwnership = false)]
public void StartGame(NetworkConnection conn)
{
	SceneData data = GetData(conn);
	DespawnChatManager(data.clients);
	data.lobbyManager.StartGame();
	HideLobbyData(data);
}

[ObserversRpc]
public void StartGame()
{
	_spawnpoints[activeSpawnpointKey].vehicleManager.StartGame();
}
```

Třetí a poslední metoda v tomto řetězci metod StartGame je lokalizována ve třídě VehicleManager. Tato metoda a menší navazující metody jsou zodpovědné za skrytí uživatelského rozhraní a aktivaci herních ovladačů, kamery a audio posluchače pro lokálního klienta.

```
public void StartGame()
{
	FindObjectOfType<LobbyController>().menuPanel.SetActive(false);
	ActivateController(GetActualTankPosition().Value, true);
}
```

6 TESTOVÁNÍ IMPLEMENTACE

6.1 Vzdálený server

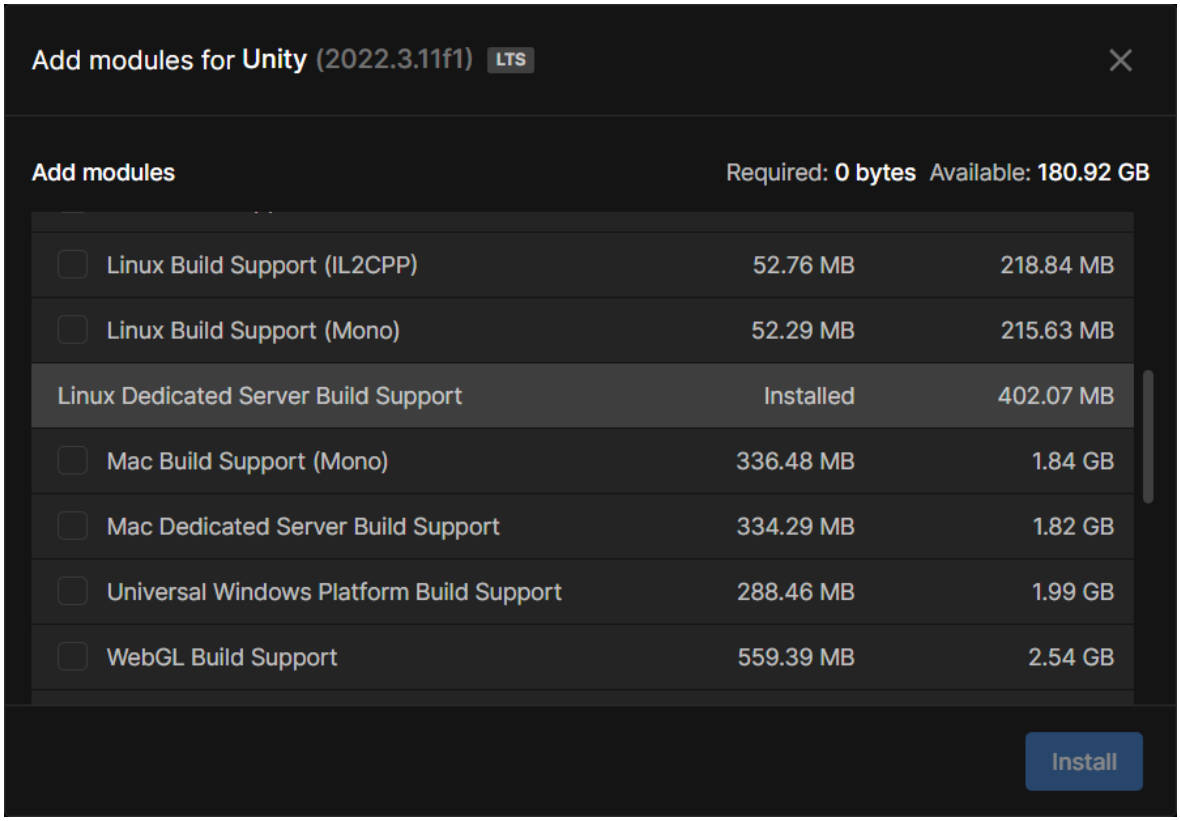
Pro účely otestování vzdáleného připojení byl na Fakultě Aplikované Informatiky ve Zlíně zřízen soukromý server, ke kterému mají povolený vstup pouze studenti s přístupem k univerzitní VPN.

6.1.1 Přihlašování

K připojení k serveru je využíván nástroj Putty. Jedná se o populární klient pro vzdálený přístup k serverům pomocí SSH. Pro autentizaci je využívána metoda založená na asynchronním šifrování, kdy soukromý klíč zůstává uložen na klientském počítači a veřejný klíč je nainstalován na server. Při pokusu o připojení, server požádá klienta, aby prokázal vlastnictví soukromého klíče, pokud jej poskytne je klient připuštěn na server.

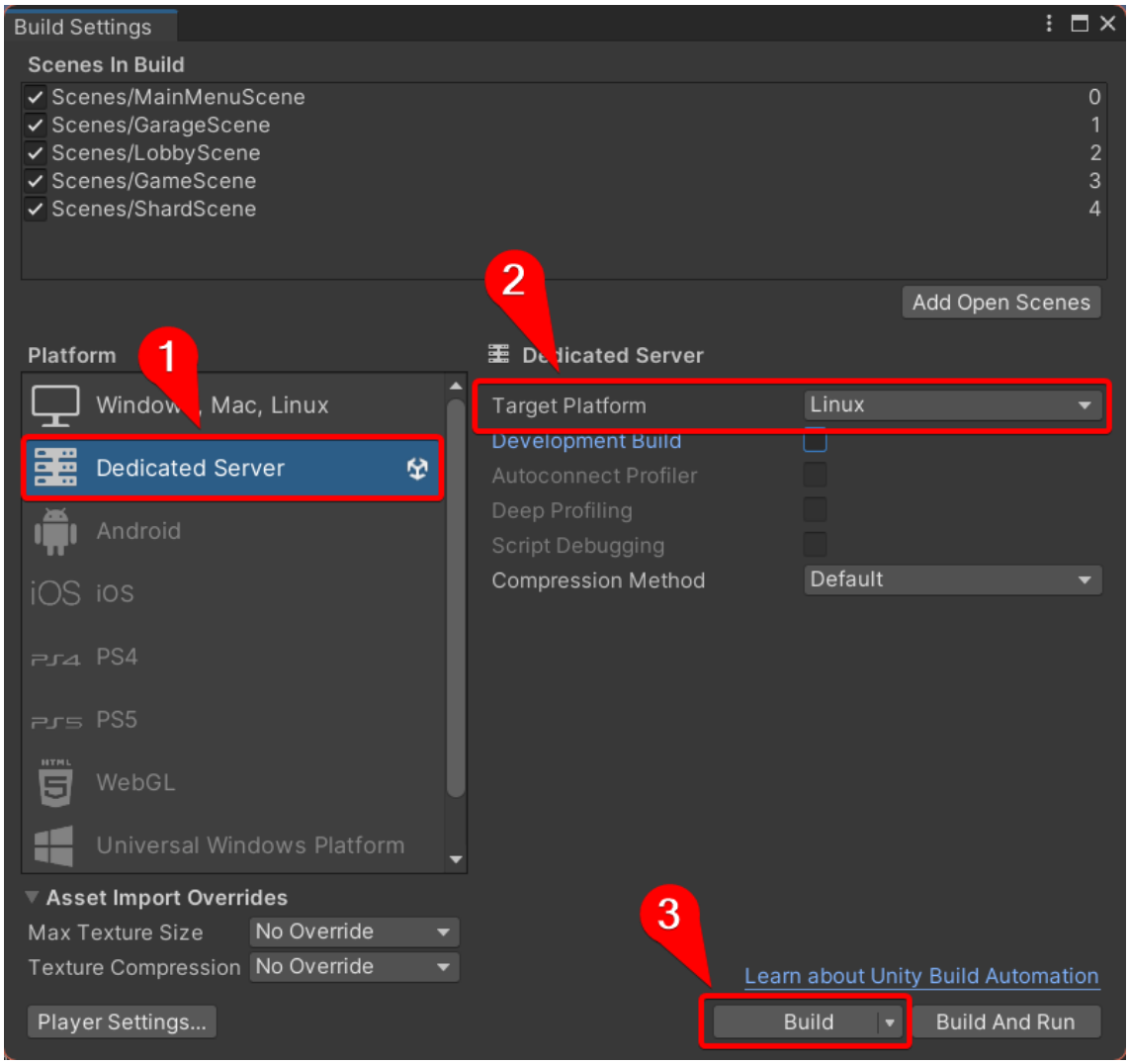
6.1.2 Build a nahrávání nové verze

Pro sestavení serverové aplikace pro operační systém Linux je nezbytné do aktuální instalace přidat modul „Linux Dedicated Server Build Support“.



Obrázek 35. Modul Linux Dedicated Server Build Support

K samotnému sestavení serverového buildu dochází přímo v editoru Unity, kde je nutné pouze změnit hodnotu pole pro určení operačního systému a sestavit aplikaci.



Obrázek 36. Postup při vytváření serverové aplikace pro OS Linux

Pro nahrání aplikace na server byl použit software WinSCP, který poskytuje prostředí pro vzdálenou správu souborových systémů Windows a Linux.

6.1.3 Spouštění serveru a service

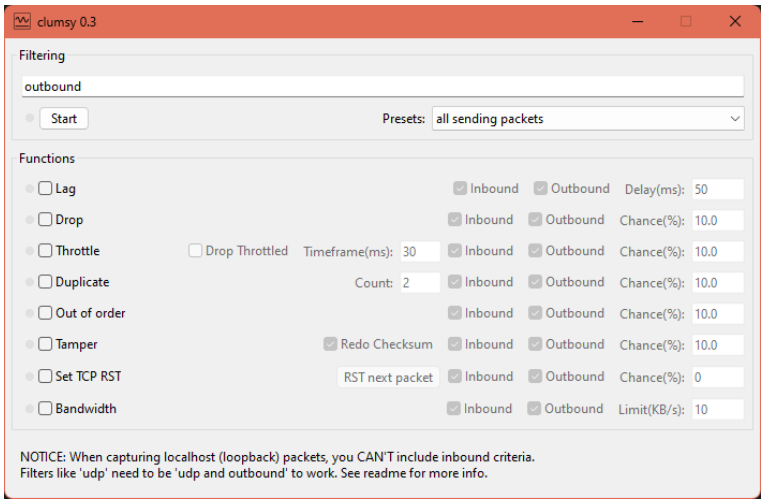
Pro potřeby projektu UTW byl vytvořen bash skript, jehož úkolem je aktivovat serverovou aplikaci v novém tmux sezení. Tmux udržuje aplikace běžící v pozadí i když je terminál odpojen, a je tedy v tomto případě velice užitečný.

Skript je nastaven tak, aby se automaticky spustil při startu serveru, respektive je zaregistrována service, která jej spouští. Dojde-li tedy například k výpadku, je serverová aplikace po restartu serveru znovu spuštěna bez nutnosti zásahu administrátora.

6.2 Simulace síťových problémů na vzdáleném serveru

Knihovna Fish-Net sice disponuje zabudovaným simulátorem různých síťových problémů, nastavitelných v komponentě TransportManager. Pro simulaci byl však zvolen nástroj Clumsy. Jelikož se jedná o software třetí strany, nabízí více možností a lze díky němu testovat i vzdálený build serveru.

Clumsy umožňuje emulovat různé situace, jako např. zpoždění, ztráty paketů, duplikace paketů, záměna pořadí paketů a mnoho dalších. Tyto možnosti umožní lépe porozumět, jak se hra chová při různých síťových podmínkách.



Obrázek 37. Software clumsy

6.2.1 Lag

Pro lepší přehled o latenci byla do projektu přidána komponenta knihovny Fish-Net „PingDisplay“. Tato komponenta zobrazuje aktuální dobu odezvy (RTT), což umožňuje testovat, zda omezení skutečně funguje.

Po nastavení zpoždění na 100 ms pro příchozí i odchozí pakety byla v aplikaci zaznamenána průměrná hodnota odezvy kolem 230 ms. Tato hodnota, s tolerancí 30 ms navíc, odpovídá nastavenému limitu.

Přestože bylo zaznamenáno větší zpoždění, hra zůstává stále hratelná. To je pravděpodobně důsledek specifického žánru hry. Například ve hře typu FPS by hráč s takovýmto zpožděním čelil nepřesnostem, což by pro něj mohlo být velmi frustrující.

6.2.2 Packet loss

Ztráta paketů v rámci software clumsy, je nastavována v procentuální šanci na „upuštění“ při každém odeslaní paketu nebo obdržení paketu.

Šance byla při testu nastavena na poměrně vysokou hodnotu při 50 % a to pouze pro odchozí pakety. Hra byla při této limitaci viditelně méně responsivní. Např. při vytváření nové lobby bylo zapotřebí čekat mnohonásobně déle, než byly klientovi pakety ověřeny, a mohl být připojen do své nové lobby scény.

V jednom testovacím případě došlo k odpojení klienta, který se pokoušel opakovaně vytvořit lobby místnost a zahlcoval tak server neplatnými požadavky.

6.2.3 Duplicate

Tato funkce zapříčiní odeslání duplicitních paketů. Je zde možnost nastavit kolik duplicitních paketů bude odesláno, zda budou ovlivněny příchozí i odchozí pakety a šance na výskyt.

Pro tento test byly hodnoty nastaveny na:

- Kvantita duplikace: 2
- Ovlivnění příchozích i odchozích paketů
- Šance: 50 %

Během tohoto testu nebyly zaznamenány žádné anomálie. Hra běžela plynule a bez problémů jak v shard scéně, tak v lobby scéně. Během hraní také nebyly pozorovány žádné problémy.

6.2.4 Out of order

Nastavení simulace prohození pořadí odeslaných nebo přijatých paketů je identické jako při zpoždění (kapitola 6.2.1) nebo ztrátě paketů (kapitola 6.2.2).

V tomto testu byly zaznamenány totožné výsledky jako při testu duplikace (kapitola 6.2.3).

6.2.5 Tampering

Funkce tampering (manipulace) se týká úmyslných změn obsahu datových paketů, které se přenášejí v síti. V běžném provozu by se taková aktivita neměla vyskytnout, neboť může představovat bezpečnostní hrozbu. Tato činnost obvykle souvisí s útoky na síť, při

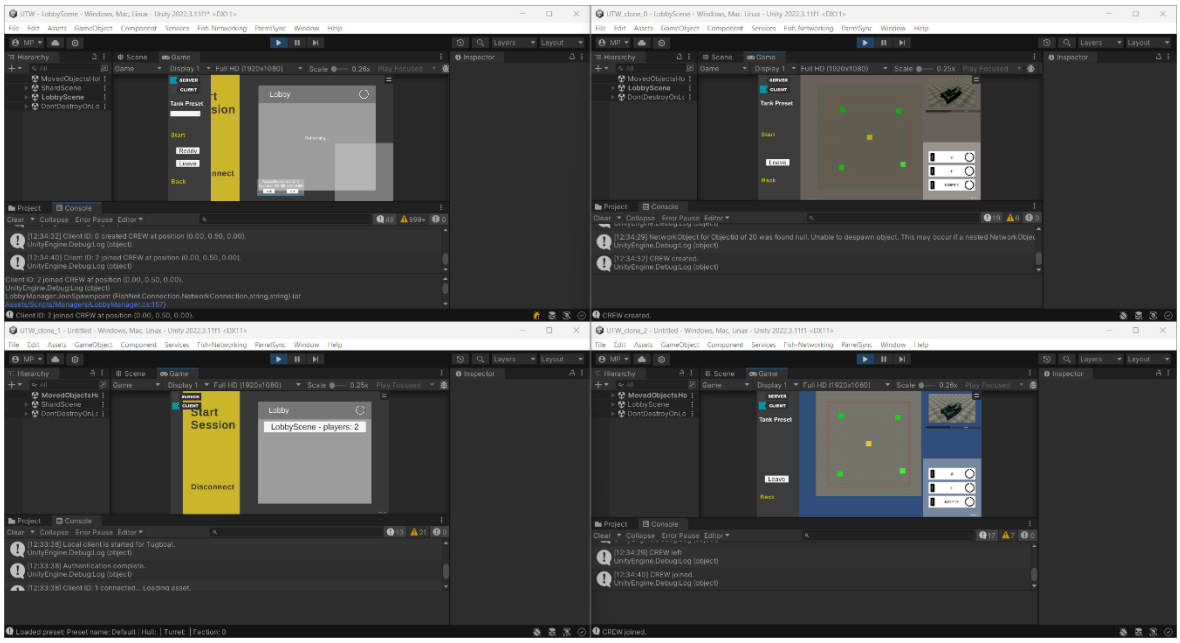
kterých útočník může upravovat, mazat nebo vkládat data do paketů za účelem získání neoprávněného přístupu, narušení komunikace nebo získání citlivých informací.

Během testování této funkce došlo k odhlášení z univerzitní VPN, což způsobilo odpojení od serveru. Z tohoto důvodu není možné tuto funkci na vzdáleném serveru testovat.

6.3 Debugging v rámci vývoje

Pro testování a debuggování většiny funkcionalit při vývoji architektury simulátoru UTW, znatelně pomáhal nástroj ParallelSync.

ParallelSync je nástroj navržený speciálně pro Unity editor, který se zaměřuje na efektivní testování a debugging her pro více hráčů přímo v prostředí vývojového editoru. Jeho hlavní přínos spočívá v možnosti simultánního testování a sledování změn v kódu ve více instancích Unity editoru současně, což značně zjednodušuje a urychluje vývoj her pro více hráčů. Eliminuje například potřebu neustálého sestavování celého projektu při každé změně v kódu, což tradičně zabírá značné množství času.



Obrázek 38. Čtyři instance Unity editoru synchronizované díky ParallelSync

Na obrázku 38 lze vidět čtyři instance Unity editoru. Krom serverové instance (vlevo nahoře), představuje každá individuálního klienta s jeho vlastním výpisem v logu.

ZÁVĚR

V rámci této práce byla provedena rozsáhlá rešerše zaměřená na architektury síťové komunikace ve hrách pro více hráčů. Byly analyzovány současné techniky a postupy při návrhu těchto architektur, včetně komunikačních protokolů, síťových topologií a klíčových výzev, jako jsou latence, predikce, synchronizace a bezpečnostní hrozby.

Dále byl proveden rozbor hlavních síťových knihoven pro herní engine Unity, včetně Netcode for GameObjects (NGO), Mirror Networking, Photon (Fusion 2) a Fish-net: Networking Evolved. Každá z těchto knihoven byla podrobně zkoumána z hlediska poskytovaných funkcionalit, komponent a efektivity v implementaci síťové komunikace.

Po důkladné analýze a porovnání dostupných síťových knihoven pro herní engine Unity byla vybrána ta nejvhodnější pro implementaci do existujícího projektu UTW. Tato volba byla provedena s ohledem na potřeby projektu a požadované funkcionality.

Následně bylo navrženo konkrétní řešení architektury síťové komunikace pro tento projekt. Tento návrh zahrnoval požadavky projektu, aktivitní diagramy, určení topologie a autorit při spouštění interních procesů.

Navržené řešení bylo poté do projektu UTW implementováno. Během implementace byly využity nejnovější techniky a postupy v oblasti síťového programování, z části i díky stáří vybrané knihovny. Důraz byl kladen na efektivitu, spolehlivost a snadnou rozšiřitelnost systému.

Výsledky práce potvrzují, že správná architektura síťové komunikace je klíčovým faktorem pro úspěšné fungování her pro více hráčů. Implementované řešení pro projekt UTW splňuje požadavky na spolehlivou síťovou komunikaci, což vede k lepšímu uživatelskému zážitku a vyšší kvalitě herního prostředí.

SEZNAM POUŽITÉ LITERATURY

[1] Communication Protocol. *Techopedia* [online]. 2023 [cit. 2024-02-16]. Dostupné z: https://www.techopedia.com/definition/25705/communication-protocol

[2] Standardised Communications Protocols. *ARDC* [online]. 2022 [cit. 2024-02-16]. Dostupné z: https://ardc.edu.au/resource/standardised-communications-protocols/

[3] MADHAV, Sanjay a Josh GLAZER. Multiplayer Game Programming: Architecting Networked Games (Game Design). Addison-Wesley Professional, 2015. ISBN 978-0134034300.

[4] Network Communication Protocols. *KIO* [online]. c2024 [cit. 2024-02-16]. Dostupné z: https://www.kio.tech/en-us/blog/data-center/network-communication-protocols

[5] Co je TCP? *SPRÁVA SÍTĚ* [online]. c2022 [cit. 2024-02-16]. Dostupné z: https://www.sprava-site.eu/tcp/

[6] Co je TCP/IP? *SPRÁVA SÍTĚ* [online]. c2022 [cit. 2024-02-16]. Dostupné z: https://www.sprava-site.eu/tcp-ip/

[7] TCP/IP. *Informační systém Masarykovy univerzity* [online]. c1999-2024 [cit. 2024-02-16]. Dostupné z: https://is.muni.cz/do/ics/el/sitmu/law/html/tcpip.html

[8] TCP VS UDP. *ProFiber Networking* [online]. c2007-2024 [cit. 2024-02-16]. Dostupné z: https://www.profiber.eu/cz/aplikace/tcp-vs-udp/

[9] TCP vs UDP: What's the Difference and Which Protocol Is Better? *Avast* [online]. c1988-2024 [cit. 2024-02-20]. Dostupné z: https://www.avast.com/c-tcp-vs-udp-difference#topic-2

[10] Transmission Control Protocol (TCP). *Khan Academy* [online]. c2008-2024 [cit. 2024-02-20]. Dostupné z: https://cs.khanacademy.org/computing/informatika-pocitace-a-internet/x8887af37e7f1189a:internet/x8887af37e7f1189a:tcp-protokol/a/transmission-control-protocol--tcp

[11] Co je UDP? *SPRÁVA SÍTĚ* [online]. c2022 [cit. 2024-02-20]. Dostupné z: https://www.sprava-site.eu/udp/

[12] What is the User Datagram Protocol (UDP/IP)? *CLOUDFLARE* [online]. c2024 [cit. 2024-02-20]. Dostupné z: https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/

[13] Client-Server. *HEAVY.AI* [online]. c2022 [cit. 2024-02-22]. Dostupné z: https://www.heavy.ai/technical-glossary/client-server

[14] Client-server. *TechTarget* [online]. c2000-2024 [cit. 2024-02-22]. Dostupné z: https://www.techtarget.com/searchnetworking/definition/client-server

[15] Guide to Client-server Architecture or Model. *Liquid Web* [online]. 2023 [cit. 2024-02-22]. Dostupné z: https://www.liquidweb.com/blog/client-server-architecture/

[16] Dedicated vs. listen servers — what's the difference in gaming? *Liquid Web* [online]. 2024 [cit. 2024-02-24]. Dostupné z: https://www.liquidweb.com/kb/dedicated-vs-listen-server/

[17] Real-Time Game Server Internals: Basic Theory, Architecture, Optimization, Auto-Scaling. Medium [online]. 2022 [cit. 2024-02-22]. Dostupné z: https://betterprogramming.pub/real-time-game-server-internals-basic-theory-architecture-optimization-auto-scaling-b2070aa803d9

[18] Dedicated Server vs Listen Server. Cedric-Neukirchen.net [online]. c2023 [cit. 2024-02-24]. Dostupné z: https://cedric-neukirchen.net/docs/multiplayer-compendium/dedicated-vs-listen-server/

[19] VOGELTANZ, Tomáš. Síťová komunikace a multiplayer v počítačových hrách [PDF]. *UTB ve Zlíně*, 2023 [cit. 2024-02-24]. Dostupné z: https://moodle.utb.cz/mod/resource/view.php?id=592855

[20] Dedicated Servers In Gears of War 3: Scaling to Millions of Players. *Microsoft Studios* [PDF]. 2012 [cit. 2024-02-24]. Dostupné z: https://www.gdcvault.com/play/1015337/Dedicated-Servers-In-Gears-of

[21] Peer-to-Peer Gaming. *Medium* [online]. 2023 [cit. 2024-02-25]. Dostupné z: https://medium.com/tashi-gg/peer-to-peer-gaming-9991600c6707

[22] What is Peer-to-Peer Gaming, and How Does it Work? *VGKAMI* [online]. 2021 [cit. 2024-02-25]. Dostupné z: https://vgkami.com/what-is-peer-to-peer-gaming-and-how-does-it-work/

[23] What Is Peer-To-Peer? Meaning, Features, Pros, and Cons. *Spiceworks* [online]. 2023 [cit. 2024-02-25]. Dostupné z: https://www.spiceworks.com/tech/networking/articles/what-is-peer-to-peer/

[24] Building a Peer-to-Peer Multiplayer Networked Game. *Envato tuts+* [online]. 2013 [cit. 2024-02-25]. Dostupné z: https://code.tutsplus.com/building-a-peer-to-peer-multiplayer-networked-game--gamedev-10074t

[25] STEINMETZ, Ralf a Klaus WEHRLE. Hybrid Peer-to-Peer Systems. In: Peer-to-Peer Systems and Applications. Springer Science & Business Media, 2005, s. 353-366. ISBN 9783540291923.

[26] DESPARAT, Caroline, Hervé LUGA a Jean-Pierre JESSEL. Hybrid client-server and P2P network for web-based collaborative 3D design [online]. 23rd International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2015), 2015 [cit. 2024-02-26]. Dostupné z: https://hal.science/hal-01303837/document

[27] Why latency and ping in gaming are crucial to player experience. *Servers.com* [online]. 2022 [cit. 2024-02-27]. Dostupné z: https://www.servers.com/news/blog/why-latency-and-ping-in-gaming-are-crucial-to-player-experience

[28] What is Latency? In Gaming, It's the Enemy. *T-Mobile* [online]. c1999-2024 [cit. 2024-02-27]. Dostupné z: https://www.t-mobile.com/home-internet/the-signal/speed/what-is-latency-in-gaming

[29] How to improve latency (ping). *CenturyLink* [online]. c1930-2024 [cit. 2024-02-27]. Dostupné z: https://www.centurylink.com/home/help/internet/how-to-improve-gaming-latency.html

[30] Bandwidth. *Verizon* [online]. 2023 [cit. 2024-02-27]. Dostupné z: https://www.verizon.com/articles/internet-essentials/bandwidth-definition/

[31] Bandwidth (network bandwidth). *TechTarget* [online]. 2021 [cit. 2024-02-27]. Dostupné z: https://www.techtarget.com/searchnetworking/definition/bandwidth

[32] Latency vs Throughput. *Edgeuno* [online]. 2023 [cit. 2024-02-27]. Dostupné z: https://edgeuno.com/latency-vs-throughput/

[33] Latency vs Throughput – Understanding the Difference. *Comparitech* [online]. 2024 [cit. 2024-02-27]. Dostupné z: https://www.comparitech.com/net-admin/latency-vs-throughput/

[34] What is a good ping for online gaming? Is there any way to lower my ping? *Reddit* [online]. 2016 [cit. 2024-02-27]. Dostupné z: https://www.reddit.com/r/pcmasterrace/comments/3zy895/what_is_a_good_ping_for_online_gaming_is_there/

[35] Lag! Top 5 Reasons your Ping is so High. *HP* [online]. 2020 [cit. 2024-02-27]. Dostupné z: https://www.hp.com/us-en/shop/tech-takes/5-reasons-your-ping-is-so-high

[36] What's a good ping speed for gaming and how to test it. *COX* [online]. c1998-2024 [cit. 2024-02-27]. Dostupné z: https://www.cox.com/residential/internet/guides/gaming-performance/ping-testing.html

[37] What is a good ping speed? *Virgin Media* [online]. 2023 [cit. 2024-02-27]. Dostupné z: https://www.virginmedia.com/blog/gaming/what-is-a-good-ping

[38] What Is a Good Internet Speed for Gaming? HighSpeedInternet [online]. 2023 [cit. 2024-02-27]. Dostupné z: https://www.highspeedinternet.com/resources/how-much-speed-do-i-need-for-online-gaming

[39] Unreal Networking Architecture. *UDK* [online]. c2001-2012 [cit. 2024-02-28]. Dostupné z: https://docs.unrealengine.com/udk/Three/NetworkingOverview.html

[40] CHIU, Chihming. Massively Multiplayer Game Programming With Unity 3d and Mirror: The Ultimate Guide to Building and Hosting Your MMOGS. Tellwell Talent, 2021. ISBN 978-0228844105.

[41] Fast-Paced Multiplayer (Part II): Client-Side Prediction and Server Reconciliation. *Gabriel Gambetta* [online]. c2024 [cit. 2024-02-28]. Dostupné z: https://www.gabrielgambetta.com/client-side-prediction-server-reconciliation.html

[42] BERNIER, Yahn W. Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization [PDF]. *Valve*, 2001 [cit. 2024-02-28]. Dostupné z: https://web.cs.wpi.edu/~claypool/courses/4513-B03/papers/games/bernier.pdf

[43] HOW DO MULTIPLAYER GAMES WORK? FROM SIMPLE TO COMPLEX. *N-iX* [online]. 2023 [cit. 2024-02-28]. Dostupné z: https://gamestudio.n-ix.com/how-do-multiplayer-games-work-from-simple-to-complex/

[44] Reconciliation in multiplayer games. *Elympics* [online]. 2023 [cit. 2024-03-01]. Dostupné z: https://docs.elympics.cc/concepts/reconciliation/index.html

[45] Fast-Paced Multiplayer (Part III): Entity Interpolation. *Gabriel Gambetta* [online]. c2024 [cit. 2024-03-01]. Dostupné z: https://www.gabrielgambetta.com/entity-interpolation.html

[46] Fast-Paced Multiplayer (Part IV): Lag Compensation. *Gabriel Gambetta* [online]. c2024 [cit. 2024-03-07]. Dostupné z: https://www.gabrielgambetta.com/lag-compensation.html

[47] 21 Server Security Tips to Secure Your Server. *PhoenixNAP* [online]. 2023 [cit. 2024-03-11]. Dostupné z: https://phoenixnap.com/kb/server-security-tips

[48] How to protect the server structure of your fintech business — seven tips that are unlikely to become obsolete. *Soft-FX* [online]. c2018-2024 [cit. 2024-03-11]. Dostupné z: https://www.soft-fx.com/blog/how-to-protect-the-server/

[49] ENGELBRECHT, Dylan. Building Multiplayer Games in Unity Using Mirror Networking. Johannesburg, South Africa: Apress, 2022. ISBN 978-1-4842-7474-3.

[50] How to prevent DDoS attacks | Methods and tools. CLOUDFLARE [online]. c2024 [cit. 2024-03-11]. Dostupné z: https://www.cloudflare.com/learning/ddos/how-to-prevent-ddos-attacks/

[51] What is Anycast? | How does Anycast work? CLOUDFLARE [online]. c2024 [cit. 2024-03-11]. Dostupné z: https://www.cloudflare.com/learning/cdn/glossary/anycast-network/

[52] Packet Sniffing Explained: Definition, Types, and Protection. Avast [online]. c2024 [cit. 2024-03-11]. Dostupné z: https://www.avast.com/c-packet-sniffing

[53] Hackers, Scammers and Thieves – Understanding Cybersecurity in the Gaming Industry. *UKTN* [online]. 2023 [cit. 2024-03-11]. Dostupné z: https://www.uktech.news/other_news/hackers-scammers-and-thieves-understanding-cybersecurity-in-the-gaming-industry

[54] Server-side vs. Client-side Vulnerabilities. *Medium* [online]. 2023 [cit. 2024-03-12]. Dostupné z: https://radiumhacker.medium.com/server-side-vs-client-side-vulnerabilities-16ff5743b35b

[55] HARDY, Robert Stafford. Cheating in Multiplayer Video Games [PDF]. *Virginia Polytechnic Institute and State University*, 2009 [cit. 2024-03-15]. Dostupné z: https://vtechworks.lib.vt.edu/server/api/core/bitstreams/39ff6dde-bd1b-4aa2-a514-fe0f444c9b47/content

[56] Medal of dishonour: why do so many people cheat in online video games? *The Guardian* [online]. 2021 [cit. 2024-03-15]. Dostupné z: https://www.theguardian.com/games/2021/feb/24/medal-of-dishonour-cheating-video-games

[57] AUTOMATING AN MMORPG. *Chimpeon* [online]. c2024 [cit. 2024-03-15]. Dostupné z: https://chimpeon.com/games/mmorpg-automation

[58] How The Multiplayer Gaming Industry Is Evolving Due to Hacks & Cheats. *Readwrite* [online]. 2023 [cit. 2024-03-18]. Dostupné z: https://readwrite.com/how-the-multiplayer-gaming-industry-is-evolving-due-to-hacks-cheats/

[59] UNITY TECHNOLOGIES. *Unity Multiplayer Networking* [online]. 2023 [cit. 2024-03-19]. Dostupné z: https://docs-multiplayer.unity3d.com/.

[60] How to choose the right netcode for your game. *Unity* [online]. 2020, aktualizováno 2023 [cit. 2024-03-28]. Dostupné z: https://blog.unity.com/games/how-to-choose-the-right-netcode-for-your-game

[61] Beginner Netcode for GameObjects Tutorial | Unity Gaming Services. *Youtube* [online]. 27.12.2022 [cit. 2024-03-28]. Dostupné z: https://youtu.be/dKrdrPjJG04?si=w0TBETkjGKD3pPHz. Kanál uživatele Unity.

[62] MIRRORNETWORKING. *Mirror* [online]. 2023 [cit. 2024-04-02]. Dostupné z: https://mirror-networking.gitbook.io/docs/.

[63] Interest management for multiplayer online games. *DynetisGames* [online]. 2017 [cit. 2024-04-04]. Dostupné z: https://www.dynetisgames.com/2017/04/05/interest-management-mog/

[64] Photon Unity Networking 2. *Photon Unity Networking 2* [online]. [cit. 2024-04-05]. Dostupné z: https://doc-api.photonengine.com/en/pun/current/index.html

[65] PHOTON. *Photon Engine Documentation* [online]. 2023 [cit. 2024-04-05]. Dostupné z: https://doc.photonengine.com/

[66] FIRSTGEARGAMES. *Fish-Net: Networking Evolved* [online]. 2023 [cit. 2024-04-08]. Dostupné z: https://fish-networking.gitbook.io/docs/.

[67] Mirror LTS. *Unity Asset Store* [online]. [cit. 2024-04-08]. Dostupné z: https://assetstore.unity.com/packages/tools/network/mirror-lts-102631

[68] Unity Networking Solutions. *Dokumenty Google* [online]. 2022, aktualizováno 7. 4. 2024 [cit. 2024-04-08]. Dostupné z: https://docs.google.com/spreadsheets/d/1Bj5uLdnxZYlJykBg3Qd9BNO-tvE8sp1ZQ4EgX1sI0RFA/edit#gid=233715429

[69] FishNet: Api Documentation [online]. [cit. 2024-04-08]. Dostupné z: https://firstgeargames.com/FishNet/api/api/

[70] UTW Repository. Online. *Github*. c2024. Dostupné z: https://github.com/Thomas-Bata-University/UTW. [cit. 2024-04-17].

[71] UTW – Official Discord Server. Online. *Discord*. c2024. Dostupné z: https://discord.com/channels/891973183274631198/1120750406910365708. [cit. 2024-04-17].

[72] SteamID, Steam Account Names, Merging Accounts and Deleting Accounts. *Steam Support* [online]. c2024 [cit. 2024-04-17]. Dostupné z: https://help.steampowered.com/en/faqs/view/2816-BE67-5B69-0FEC

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

DDoS	Distributed Denial-of-Service Attack
DPO	Data Protection Officer
FOW	Fog of war
FPS	First-person shooter
IP	Internet Protocol
LTS	Long-Term Support
MITM	Man-in-the-Middle
NGO	Netcode for GameObjects
NPC	Non-Playable Character
OSI	Open System Interconnection
P2P	Peer-to-Peer
PCI DSS	Payment Card Industry Data Security Standard
PvP	Player versus player
RPC	Remote Procedure Call
RTS	Real-time strategy
RTT	Round Trip Time
TCP	Transmission Control Protocol
TMUX	Terminal Multiplexer
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: NÁZEV PŘÍLOHY

Přiložené CD obsahuje:

- Elektronická verze diplomové práce
    - DP_PluskalMichael_2024.pdf
- Sestavené aplikace
    - SERVER_BUILD.zip
    - CLIENT_BUILD.zip
- Relevantní zdrojové kódy pro síťovou část projektu UTW
    - SOURCE_CODE.zip