

**Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky**

**Technické vylepšenia a rozšírenie
funkcionality systému pre
neurorehabilitáciu vo virtuálnej realite**

Diplomová práca

**Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky**

**Technické vylepšenia a rozšírenie
funkcionality systému pre
neurorehabilitáciu vo virtuálnej realite**

Diplomová práca

Študijný program: Informatika
Študijný odbor: Informatika
Školiace pracovisko: Katedra počítačov a informatiky (KPI)
Školiteľ: doc. Ing. Štefan Korečko, PhD.

Košice 2026

Bc. Andrei Dzemidovich

Abstrakt v SJ

Táto diplomová práca sa zaoberá rozšírením a stabilizáciou existujúceho systému L-NeRVEn pre kolaboratívnu neurorehabilitáciu vo virtuálnej realite. Systém umožňuje terapeutovi a pacientovi spolupracovať v zdieľanom virtuálnom prostredí, kde pacient vykonáva motor imagery cvičenia podporené EEG snímačmi. Na základe analýzy súčasného stavu systému a súvisiacich prác boli identifikované konkrétne problémy a oblasti vyžadujúce zlepšenie. Implementované riešenia zahŕňajú opravu hlasovej komunikácie s izoláciou kanálov pomocou unikátnych identifikátorov, centralizovanú správu serverov s načítavaním z konfiguračného súboru, perzistenciu nastavení pacienta pomocou SQLite databázy a aktualizáciu kľúčových technologických komponentov. Výsledky boli overené testovaním na zariadeniach Oculus Quest 2.

Kľúčové slová v SJ

C#, Mirror Networking, neurorehabilitácia, Unity, virtuálna realita, SQLite

Abstrakt v AJ

This master's thesis deals with the extension and stabilization of the existing L-NeRVEn system for collaborative neurorehabilitation in virtual reality. The system allows a therapist and a patient to collaborate in a shared virtual environment, where the patient performs motor imagery exercises supported by EEG sensors. Based on an analysis of the current system state and related work, specific problems and areas requiring improvement were identified. The implemented solutions include a fix for voice communication with channel isolation using unique identifiers, centralized server management with loading from a configuration file, patient settings persistence using an SQLite database, and updates of key technological components. The results were verified by testing on Oculus Quest 2 devices.

Kľúčové slová v AJ

C#, Mirror Networking, neurorehabilitation, Unity, virtual reality, SQLite

Bibliografická citácia

DZEMIDOVICH, Andrei. *Technické vylepšenia a rozšírenie funkcionality systému pre neurorehabilitáciu vo virtuálnej realite*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2026. 69s. Vedúci práce: doc. Ing. Štefan Korečko, PhD.

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

**ZADANIE
DIPLOMOVEJ PRÁCE**

Študijný odbor: **Informatika**

Študijný program: **Informatika**

Názov práce:

**Technické vylepšenia a rozšírenie funkcionality systému pre
neurorehabilitáciu vo virtuálnej realite**

**Technical Improvements and Functionality Extensions of Virtual Reality
Neurorehabilitation System**

Študent: **Bc. Andrei Dzemidovich**

Školiteľ: **doc. Ing. Štefan Korečko, PhD.**

Školiace pracovisko: **Katedra počítačov a informatiky**

Konzultant práce:

Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

1. Oboznámiť sa so súčasným stavom systému L-NeRVEn pre kolaboratívnu neurorehabilitáciu vo virtuálnej realite (VR) a podobnými riešeniami.
2. Identifikovať problémy a potrebné rozšírenia systému, najmä v oblasti hlasovej komunikácie, správy používateľov a aktuálnosti použitých komponentov.
3. Na základe analýzy existujúcich riešení navrhnúť a implementovať riešenia vybraných problémov v systéme L-NeRVEn.
4. Návrh a implementáciu koordinovať s ďalšími riešiteľmi prác na systéme L-NeRVEn.
5. Implementované riešenia overiť testovaním na relevantnom VR hardvéri a vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: **slovenský**

Termín pre odovzdanie práce: **24.04.2026**

Dátum zadania diplomovej práce: **31.10.2025**



n.z. Liberios Vokorokos
.....
prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry, ako aj s využitím nástrojov generatívnej umelej inteligencie v súlade s Metodickým pokynom o záverečných a kvalifikačných prácach a Etickým kódexom študenta Technickej univerzity v Košiciach. Umelá inteligencia bola použitá ako podporný nástroj bez porušenia zásad akademickej etiky a autorskej integrity.

Podakovanie

Chcem sa poďakovať svojmu vedúcemu práce doc. Ing. Štefanovi Korečkovi, PhD. za odborné vedenie a konzultácie počas vypracovania diplomovej práce. Rovnako ďakujem Damiánovi Chmurovi za spoluprácu na projekte a cenné rady počas vývoja. Osobitne ďakujem svojim rodičom a priateľom za podporu a pomoc počas celého štúdia.

Obsah

1	Úvod	1
2	Analýza súvisiacich prác	4
3	Analýza súčasného stavu systému	7
3.1	Prehľad technologickej architektúry	7
3.2	Analýza kľúčových komponentov	8
3.2.1	Mirror Networking	8
3.2.2	Fish-Networking (FishNet)	11
3.2.3	Unity Engine	14
3.2.4	XR Interaction Toolkit	16
3.2.5	OpenXR Plugin	17
3.2.6	Final IK	19
3.2.7	Výber databázového systému	20
3.3	Záver analýzy a odporúčania pre aktualizáciu	20
3.3.1	Prioritné aktualizácie	21
3.3.2	Odporúčané aktualizácie	21
3.3.3	Voliteľné aktualizácie	21
3.3.4	Implementačná stratégia	21
4	Návrh a implementácia vylepšení systému	23
4.1	Riešenie problémov s hlasovou komunikáciou	24
4.1.1	Identifikácia problému s izolovanými komunikačnými ka- nálmi	25
4.1.2	Implementácia unikátnej identifikácie serverov	25
4.1.3	RefaktORIZÁCIA správy pripojenia k hlasovej komunikácii	26
4.2	Vylepšenia správy serverov a miestností	27
4.2.1	Centralizovaná správa zoznamu serverov	28
4.2.2	Ochrana kritických serverov	29
4.2.3	Dynamická aktualizácia zoznamu miestností	29

4.2.4	Validácia a prevencia duplicity serverov	31
4.2.5	Optimalizácia používateľského rozhrania pre správu serverov	32
4.3	Implementácia profilov pacientov	34
4.3.1	Integrácia SQLite do Unity projektu	34
4.3.2	Zjednodušený model profilu pacienta pre miestnosť	37
4.3.3	Ukladanie nastavení pacienta	40
4.3.4	Automatické načítanie nastavení	43
4.3.5	Testovanie a oprava chýb vo VR prostredí	48
4.3.6	Zhrnutie implementovaných vylepšení	54
4.4	Aktualizácia technologických komponentov na aktuálne verzie	55
4.4.1	Mirror Networking	55
4.4.2	Unity Engine	58
4.4.3	XR Interaction Toolkit	59
4.4.4	OpenXR Plugin	60
4.4.5	FinalIK	60
4.4.6	Testovanie po aktualizáciách	60
4.4.7	Záver aktualizácií	60
5	Vyhodnotenie	62
5.1	Možnosti ďalšieho rozvoja	63
6	Záver	64
	Zoznam použitej literatúry	66
	Zoznam skratiek	70
	Slovník	72
	Zoznam príloh	73
A	Používateľská príručka	74
A.1	Inštalácia softvéru	74
A.1.1	Osobný počítač	74
A.1.2	Oculus Quest 2	74
A.2	Použitie a ukážka aplikácie	76
A.2.1	Kroky pre správne využitie teleportačného systému	77
A.2.2	Správa serverov	77
A.2.3	Hlasová komunikácia	78

A.3	Rola pacienta	79
A.4	Rola terapeuta	79
A.4.1	Nastavenie MI threshold hodnoty	80
A.4.2	Ukazovateľ únavy pacienta	82
A.4.3	Ukladanie nastavení pacienta	82
A.5	Ukážka systému	84
A.5.1	Vzhľad prostredia	84
A.5.2	Ukážka animácií	85
B	Systémová príručka	88
B.1	Importovanie projektu	88
B.2	Preklad aplikácie	89
B.2.1	Preklad pre server	89
B.2.2	Preklad pre stolový počítač	90
B.2.3	Preklad pre Oculus Quest 2	91
B.3	Nasadenie servera	91
B.3.1	Konfigurácia zoznamu serverov	91
B.4	Databáza	92
B.5	Nasadenie release buildov cez Meta Quest Developer Hub	93
B.5.1	Predpoklady	93
B.5.2	Identifikátory kanála	93
B.5.3	Nastavenia buildu v Unity	94
B.5.4	Pre-upload checklist	94
B.5.5	Upload cez Meta Quest Developer Hub	95
B.5.6	Verifikácia na zariadení	95
B.5.7	Build promotion (RC → STABLE)	95
B.5.8	Riešenie problémov	96
B.5.9	Správa keystore	96
B.6	REST API dokumentácia	96
B.7	Referenčná dokumentácia WebSocket protokolu	98
B.7.1	Pripojenie a životný cyklus	99
B.7.2	Formát správ	100
B.7.3	Relácie a autorizačné pravidlá	100
B.7.4	Katalóg udalostí	101
B.7.5	Spracovanie chýb	103
B.7.6	Limity a výkonnosť	103
B.7.7	Príklady správ	104

Zoznam obrázkov

4.1	Diagram tried s vyznačením nových a modifikovaných	
	tried	24
4.2	Popup panel v zatvorenom stave	33
4.3	Popup panel v otvorenom stave	33
4.4	Finálna databázová schéma systému	39
4.5	Tab Adjustments s tlačidlom pre uloženie nastavení	
	pacienta	40
4.6	Tab Training s tlačidlom pre uloženie tréningových	
	nastavení	41
4.7	Tlačidlo pre reset mierky avatara v rozhraní	
	terapeuta	52
4.8	Úspešné načítanie pozícií objektov z databázy pri	
	pripojení pacienta	53
4.9	Nastavenia komponentu NetworkTransformUnreliable	56
4.10	Nové nastavenia komponentu CustomNetworkManager	58
4.11	Nastavenia OpenXR pluginu pre Android platformu	60
A.1		75
A.2		77
A.3		78
A.4		78
A.5		80
A.6		80
A.7		81
A.8		83
A.9		83
A.10		84
A.11		84
A.12		85

A.13		85
A.14		86
A.15		86
A.16		87
B.1		90
B.2		92
B.3		97
B.4		97
B.5		98
B.6		98

Zoznam tabuliek

B.1	Konfigurácia podľa release kanála	93
B.2	Prevádzkové limity WebSocket servera	103

Zoznam výpisov

3.1	Kľúčové informácie z crash dump	10
4.1	Tvorba unikátneho názvu kanála	25
4.2	Pripojenie k hlasovému kanálu po úspešnom pripojení k serveru	26
4.3	Odpojenie od hlasového kanála pri odpojení od servera	27
4.4	Príklad konfiguračného súboru servers.txt	28
4.5	Aktuálny formát konfiguračného súboru servers.json	28
4.6	Načítavanie serverov z JSON súboru	28
4.7	Automatická aktualizácia pri zmene servera	30
4.8	Timeout a error handling pre room list	30
4.9	Kontrola duplicity pri pridávaní servera	31
4.10	Zjednodušený formát servers.json	33
4.11	Serverová inicializácia DatabaseManager	36
4.12	Načítanie miestností z databázy pri štarte servera	39
4.13	Ukladanie nastavení pacienta na serveri	42
4.14	Serializácia Vector3 do JSON s InvariantCulture	43
4.15	Načítanie a aplikácia nastavení pri pripojení pacienta	43
4.16	Čakanie na inicializáciu pred aplikáciou pozícií	44
4.17	Oddelenie Server a Command metód	46
4.18	Automatická aktualizácia UI cez SyncVar hooks	46
4.19	Príklad JSON štruktúry ObjectPositions	47
4.20	Metódy pre rotáciu pacienta	49
4.21	Vykonanie rotácie pacienta	49
4.22	Aplikácia rotácie z databázy	50
4.23	Čakanie na inicializáciu XR trackingu pred aplikáciou pozície	51
4.24	ClientRpc metóda pre synchronizáciu objektov	52
4.25	Aktualizácia referencií na NetworkTransformUnreliable	56
4.26	Nastavenie syncDirection pre objekty scény	56
4.27	Aktualizovaná registrácia handlera správ	57
4.28	Odosielanie správ cez spoľahlivý kanál	57

4.29	Registrácia nových SyncList udalostí	57
4.30	Callback pre počiatočnú synchronizáciu	58
4.31	Pôvodné dočasné riešenie pádu servera	59
B.1	Formát konfiguračného súboru servers.json	91
B.2	Štruktúra obálky WebSocket správy	100
B.3	Check-in — požiadavka	104
B.4	Check-in — potvrdenie	104
B.5	EEG/MI stream	104
B.6	Spustenie kroku cvičenia (prázdny payload)	104
B.7	Serverová notifikácia o začiatku odpočinku	104
B.8	Chybová odpoveď	104

1 Úvod

Táto práca bola riešená v rámci projektov APVV-21-0105 (Dôveryhodná interakcia človek–robot a terapeut–pacient vo virtuálnej realite) a CHIST-ERA ReHaB (Towards an ecologically valid symbiosis of BCI and head-mounted VR displays: focus on collaborative post-stroke neurorehabilitation).

Cievna mozgová príhoda je častou príčinou dlhodobej invalidity, pričom významná časť pacientov trpí pretrvávajúcimi motorickými deficitmi aj po ukončení akútnej fázy liečby. Tradičné rehabilitačné metódy vyžadujú pravidelné osobné stretnutia s fyzioterapeutom, čo je časovo náročné pre pacientov aj zdravotnícke zariadenia. Moderné technológie virtuálnej reality (VR) v kombinácii s Brain-Computer Interface (BCI) systémami založenými na motor imagery ponúkajú inovatívny prístup k neurorehabilitácii, ktorý umožňuje tréning motorických funkcií prostredníctvom mentálnej predstavy pohybov bez potreby ich skutočného vykonania.

Motor imagery je proces, pri ktorom pacient mentálne simuluje pohyb končatiny, čo aktivuje rovnaké oblasti motorickej kôry ako pri reálnom pohybe. Táto aktivácia môže byť detekovaná pomocou elektroencefalografie (EEG) a následne preložená do vizuálneho feedbacku vo virtuálnom prostredí. Imerzívne VR prostredie pritom zvyšuje účasť pacienta a poskytuje realistický kontext pre rehabilitačné cvičenia. Kolaboratívny prístup, kde terapeut a pacient zdieľajú virtuálne prostredie, umožňuje priame monitorovanie pokroku a dynamické prispôbovanie terapie potrebám pacienta.

Systém L-NeRVEn [1, 2] predstavuje platformu na VR založenú neurorehabilitáciu využívajúcu motor imagery, kde pacient v immerzívnom VR prostredí vykonáva cvičenia manipulácie s objektmi (lopta, kocka, kľúč) a dosahové pohyby pod dohľadom terapeuta. Systém je postavený na Unity game engine s podporou kolaboratívnych sedení. Napriek pokročilej funkcionalite základného systému, praktické nasadenie odhalilo viaceré obmedzenia. Primárnym problémom je absencia perzistentného ukladania patientskych údajov. Všetky nastavenia (typ avatara, pozície objektov, rotácia pacienta, parametre cvičení) musia byť manu-

álne konfigurované pri každom sedení, čo spomaľuje prácu terapeuta a znižuje efektivitu terapie. Systém pozostáva zo starších verzií Unity engine a používaných knižníc. Absencia aktualizácie komponentov na najnovšie stabilné verzie ohrozovala dlhodobú udržiateľnosť projektu a kompatibilitu s modernými VR zariadeniami.

Motiváciou pre túto prácu je práve odstránenie týchto prekážok, ktoré bránia širšiemu klinickému nasadeniu systému. Počas vývoja a pravidelných online stretnutí s riešiteľmi projektu boli identifikované a riešené viaceré technické problémy, ktoré ovplyvňovali stabilitu a použiteľnosť systému.

Pôvodná implementácia Vivox systému pre hlasovú komunikáciu medzi terapeutom a pacientom trpela problémami s izolovanými komunikačnými kanálmi. Okrem toho existoval problém, keď aplikácia odosiela požiadavky na nesprávny server bez ohľadu na výber používateľa, čo viedlo k zobrazeniu neaktuálneho alebo nesprávneho zoznamu dostupných miestností. Po aktualizácii Unity engine na verziu 2021.3.35f1 a integrácii Vivox SDK¹ sa objavil kritický problém s nekorrektným ukončovaním serverovej aplikácie. Server počas behu fungoval stabilne, avšak pri pokuse o štandardné zatvorenie programu (CTRL+C, zatvorenie okna) dochádzalo k chybe v natívnom kóde Unity.

Diplomová práca sa zameriava na implementáciu riešení identifikovaných problémov a na rozšírenie systému o novú funkcionality. Systém L-NeRVEn je výsledkom dlhodobého vývoja prebiehajúceho v rámci viacerých nadväzujúcich záverečných prác [3, 4, 5]. Práca sa sústreďuje najmä na rozšírenie systému o perzistenciu patientskych údajov a jeho aktualizáciu na najnovšie verzie knižníc. Výsledkom je stabilnejší a použiteľnejší systém, ktorý odstraňuje kľúčové prekážky v pracovnom toku terapeuta.

Formulácia úlohy

Cieľom tejto diplomovej práce je rozšíriť existujúci kolaboratívny VR systém pre neurorehabilitáciu o perzistentné ukladanie patientskych údajov, aktualizovať technologické komponenty na najnovšie stabilné verzie a vyriešiť identifikované technické problémy, ktoré bránia širšiemu klinickému nasadeniu systému.

Konkrétne úlohy práce zahŕňajú:

1. **Analýza existujúceho systému:** Štúdium architektúry L-NeRVEn platformy, identifikácia technických problémov a limitácií pôvodnej implementácie, a definícia požiadaviek pre vylepšenia systému.

¹<https://unity.com/products/vivox>

2. **Aktualizácia technologických komponentov:** Analýza kompatibility a aktualizácia Unity engine, Mirror Networking a ostatných závislostí na najnovšie stabilné verzie. Testovanie migrácie, riešenie breaking changes a overenie funkcionality po aktualizácii.
3. **Riešenie problémov s hlasovou komunikáciou:** Implementácia unikátnej identifikácie serverov pre zabezpečenie izolovaných komunikačných kanálov medzi sedeniami. RefaktORIZÁCIA správy životného cyklu Vivox hlasového chatu a synchronizácia jeho stavu s pripojením k hernému serveru.
4. **Vylepšenie správy serverov a miestností:** Implementácia automatického načítavania zoznamu serverov z konfiguračného súboru, dynamická aktualizácia zoznamu miestností pri zmene servera, ochrana kritických serverov pred odstránením.
5. **Diagnostika serverovej nestability:** Analýza crash dump súborov a systémových logov pre identifikáciu príčiny nekorektného ukončovania serverovej aplikácie. Testovanie rôznych konfigurácií a dokumentácia nájdených zistení.
6. **Návrh databázovej schémy:** Výber vhodného embedded databázového riešenia, vytvorenie dátového modelu pre ukladanie patientskych profilov, nastavení sedení a tréningových parametrov. Implementácia room-based architektúry, kde každá miestnosť má asociovaný jeden patientsky profil.

Výsledkom práce je upravený a rozšírený systém s perzistentným ukladáním dát, vyriešenými problémami s hlasovou komunikáciou a správou serverov, ktorý poskytuje lepšiu používateľskú skúsenosť pre terapeutov a pacientov.

2 Analýza súvisiacich prác

V tejto kapitole analyzujeme existujúce riešenia v oblasti virtuálnej reality pre neurorehabilitáciu. Cieľom je identifikovať kľúčové prístupy, technológie a obmedzenia súčasných systémov, ktoré slúžia ako kontext pre návrh a implementáciu nášho riešenia. Publikácie boli vyhľadané v databáze IEEE Xplore pomocou dopytov kombinujúcich kľúčové slová *BCI*, *VR*, *neurorehabilitation* a *EEG*, s filtrom na roky 2016–2026. Doplnujúce vyhľadávanie bolo vykonané v Google Scholar s dopytom *VR neurorehabilitation EEG*.

Analyzované riešenia sú porovnávané s naším systémom, ktorý využíva imerzívne VR prostredie s EEG snímačmi pre motor imagery tréning v kolaboratívnych sedeniach terapeuta a pacienta, kde si pacienti predstavujú pohyby končatín bez ich skutočného vykonania. Aplikácia zobrazuje virtuálne rehabilitačné prostredie s avatarmi pacienta a terapeuta, kde pacient vykonáva motor imagery úlohy ako manipulácia s objektmi (lopta, kocka, kľúč) a dosahové pohyby. Terapeut konfiguruje cvičenia, pozície objektov a parametre tréningu.

Guerrero-Mendez a kol. [6] predstavili protokol založený na komplexných motor imagery úlohách pre rehabilitáciu horných končatín u pacientov po mŕtvici. Ich prístup využíva Action Observation, teda pozorovanie a súčasné predstavovanie pohybu, v VR prostredí z pohľadu prvej osoby, kde pacienti sledujú a súčasne si predstavujú manipuláciu s pohárikom v štyroch polohách (vpravo, vľavo, hore, dole). VR aplikácia zobrazuje videozáznamy pohybov ruky na obrazovke počítača, pacient pritom synchronizuje motorické obrazy s pozorovaným pohybom.

Systém implementuje 16-kanálový EEG záznam s Riemannovou geometriou pre klasifikáciu. Ich systém používa Action Observation na PC obrazovke s OpenBCI EEG zariadením, zatiaľ čo náš systém využíva plne imerzívne 3D VR prostredie s headsetom.

Leng a kol. [7] implementovali systém integrujúci VR technológiu s dekódovaním EEG signálov hlbokým učením pre pacientov s poranením miechy. Ich prístup zahŕňa vytvorenie imerzívnych VR scén pomocou Unity platformy a he-

adsetu. VR aplikácia simuluje scenár zbierania ovocia, kde pacienti pomocou motor imagery zbierajú virtuálne ovocie v 3D prostredí z perspektívy prvej osoby.

Technické riešenie používa 64-kanálový EEG systém s architektúrou hlbokého učenia kombinujúcou **CNN** a Transformer modely. Ich systém sa zameriava na EEG klasifikáciu mimo reálneho času s modelmi hlbokého učenia, zatiaľ čo náš systém kladie dôraz na kolaboratívne v reálnom čase sedenia.

McMahon a Schukat [8] vyvinuli nízkonákladovú platformu BCI-VR s otvoreným zdrojovým kódom využívajúcu OpenBCI Cyton board a **OSVR** headset. Systém využíva **ERD/ERS** senzomotorické rytmy z motorickej kôry pre manipuláciu 3D objektov vo VR prostredí. VR aplikácia prezentuje úlohu rotácie keltského symbolu, kde používateľ ovláda rotáciu objektov pomocou motor imagery ľavej/pravej ruky, pričom objekty tvoria keltský symbol pri správnom uhle pohľadu.

Ich prístup implementuje **CSP** priestorovú filtráciu a **LDA** klasifikáciu v OpenViBE, s prepojením cez VRPN protokol do Unreal Engine. Ich systém využíva modulárnu architektúru s oddelenými komponentmi: OpenViBE pre EEG spracovanie, **VRPN** protokol pre komunikáciu, a Unreal Engine pre VR rendering. Na rozdiel od toho náš systém integruje všetky funkcie v Unity s frameworkom Mirror Networking, čo zjednodušuje nasadenie a údržbu.

Ren a kol. [9] navrhli vylepšený **MI**-BCI systém kombinujúci Functional Electrical Stimulation (FES) s VR pre rehabilitáciu dolných končatín. Systém používa MotionStim8 pre FES stimuláciu a Unity3D scenár kopania lopty. VR aplikácia zobrazuje perspektívu z pohľadu prvej osoby, kde virtuálny avatar kopne loptu na základe motor imagery pacienta, čím poskytuje vizuálnu a kinestetickú spätnú väzbu. Ich prístup demonštruje prínos multimodálnej spätnej väzby, zatiaľ čo náš systém zatiaľ využíva len vizuálnu spätnú väzbu.

Parivash a kol. [10] predstavili rozšírenú BCI architektúru s filtračnou jednotkou na elimináciu misklasifikácií vo VR zabudovaných neurorehabilitačných systémoch. Systém detekuje tri neuro-stimulácie (pravá päšť, ľavá päšť, pokoj) pre VR avatara. VR aplikácia zobrazuje imerzívneho avatara, ktorý otvára a zatvára ľavú alebo pravú päšť podľa dekódovaných EEG signálov, pričom filtračná jednotka transformuje nesprávne klasifikácie do pokojového režimu pre zabezpečenie pozitívnej spätnej väzby. Ich prístup s klasifikáciou mimo reálneho času a následným spracovaním kontrastuje s naším systémom pracujúcim v reálnom čase, kde misklasifikácie môžu negatívne ovplyvniť pacientovu skúsenosť priamo počas sedenia.

V porovnaní s uvedenými systémami sa náš systém vyznačuje niekoľkými odlišujúcimi vlastnosťami. Na rozdiel od väčšiny prezentovaných systémov, ktoré využívajú klasifikáciu mimo reálneho času [10, 7] alebo jednostranné scenáre jedného používateľa [8, 9], náš prístup kladie dôraz na kolaboratívnu interakciu v reálnom čase. Terapeut má priamu kontrolu nad parametrami cvičení a môže dynamicky upravovať úroveň zložitosti a poskytovať okamžitú spätnú väzbu pacientovi počas sedenia.

3 Analýza súčasného stavu systému

Kolaboratívny systém pre neurorehabilitáciu vo virtuálnej realite predstavuje komplexné technologické riešenie, ktoré integruje moderné herné technológie s lekáorskými a terapeutickými postupmi. Efektívnosť takéhoto systému je priamo závislá od stability a kompatibility jeho technologických komponentov, keďže aj drobná nekompatibilita môže destabilizovať celý systém a narušiť priebeh terapeutických sedení. Pred implementáciou zmien bolo preto potrebné preveriť, v akom stave sa jednotlivé komponenty nachádzajú. Pre každý komponent bola zdokumentovaná aktuálna verzia, identifikované konkrétne problémy pozorované počas prevádzky a zhodnotené dostupné možnosti aktualizácie. Hodnotenie sa zameriava predovšetkým na funkčnosť, stabilitu a vzájomnú kompatibilitu komponentov.

3.1 Prehľad technologickej architektúry

Kolaboratívny systém L-NeRVEn [1, 2] pre neurorehabilitáciu vo virtuálnej realite je implementovaný ako multiplatformová aplikácia postavená na hernom engine Unity. Systém umožňuje simultánne pripojenie viacerých používateľov v rôznych rolách, terapeuta a pacienta, do zdieľaného virtuálneho prostredia určeného pre rehabilitačné cvičenia.

Architektúra systému je založená na klient-server modeli, pričom jeden z klientov funguje ako server a ostatní sa pripájajú ako klienti. Táto architektúra zabezpečuje centralizované riadenie terapeutických sedení a synchronizáciu stavu všetkých pripojených účastníkov. Server zodpovedá za validáciu a distribúciu všetkých akcií v rámci virtuálneho prostredia, čím sa zabezpečuje konzistentnosť stavu aplikácie pre všetkých používateľov. K serveru sa možno pripojiť zadáním ľubovoľného servera a jeho výberom v úvodnej scéne.

Sieťová komunikácia medzi klientmi a serverom je realizovaná prostredníctvom Mirror Networking knižnice, ktorá poskytuje vysokoúrovňové API pre multiplayer funkcionality. Táto knižnica zabezpečuje automatickú synchronizáciu

objektov, pozícií avatárov a interakcií medzi všetkými pripojenými klientmi.

Používateľské rozhranie systému je navrhnuté s ohľadom na rôzne typy vstupných zariadení. Terapeut môže používať klasické desktopové ovládanie pomocou myši a klávesnice, alebo VR ovládače v prípade použitia VR headsetu. Pacient typicky používa VR headset s ovládačmi, ktoré umožňujú prirodzenú interakciu s virtuálnymi objektmi. Systém automaticky detekuje typ používaného zariadenia a prispôsobí používateľské rozhranie príslušným spôsobom.

Rehabilitačné cvičenia sú implementované ako interaktívne scenáre, ktoré vyžadujú od pacienta predstavu špecifických pohybových úloh bez ich skutočného vykonania. Tieto úlohy sú zamerané na obnovu motorických funkcií horných končatín. Systém využíva pokročilé algoritmy inverznej kinematiky na generovanie prirodzených pohybových animácií, ktoré slúžia ako vizuálna pomôcka pre pacienta.

Sledovanie pokroku pacienta je zabezpečené prostredníctvom integrácie s externými EEG snímačmi a komponentom P-EEG, ktorý spracúva EEG signál a rozpoznáva úspešné vykonanie motor imagery úlohy. Výsledky sú prenášané do systému prostredníctvom WebSocket rozhrania a umožňujú terapeutovi sledovať efektívnosť terapie.

Terapeut môže komunikovať s pacientom v jednej miestnosti alebo prostredníctvom hlasového chatu vytvoreného pomocou Vivox.

3.2 Analýza kľúčových komponentov

Na základe analýzy súčasného stavu implementácie boli identifikované kľúčové technologické komponenty, ktoré majú zásadný vplyv na funkcionality a stabilitu celého systému. Každý z týchto komponentov bude v nasledujúcich podkapitolách podrobne analyzovaný z hľadiska jeho súčasnej verzie, funkcionality, identifikovaných problémov a možností vylepšenia. Osobitná pozornosť bola venovaná kompatibilite medzi jednotlivými komponentmi, pretože nekompatibilita môže viesť k nestabilite systému alebo úplnej nefunkcionalite.

3.2.1 Mirror Networking

Mirror Networking predstavuje kľúčový komponent zodpovedný za sieťovú komunikáciu v rámci kolaboratívneho systému. Táto knižnica zabezpečuje všetky aspekty multiplayer funkcionality, vrátane synchronizácie objektov, komunikácie medzi klientmi a serverom, ako aj správu pripojení a odpojení používateľov.

V súčasnej implementácii je používaná verzia 72.0.0, ktorá poskytuje stabilnú základňu pre sieťovú komunikáciu. Mirror Networking oficiálne podporuje Unity verzie 2020 a 2021 LTS, čo zabezpečuje kompatibilitu s aktuálne používanou verziou Unity 2021.3.35f1 LTS [11].

Funkcionalita a využitie v systéme

Mirror Networking v systéme zabezpečuje nasledujúce kľúčové funkcionality:

- **Synchronizácia avatarov** - pozície a rotácie všetkých pripojených používateľov sa automaticky synchronizujú medzi všetkými klientmi
- **Synchronizácia objektov** - interaktívne objekty vo virtuálnom prostredí sú synchronizované, čo umožňuje kolaboratívnu manipuláciu
- **RPC komunikácia** - umožňuje posielanie špecifických príkazov medzi serverom a klientmi pre riadenie terapeutických cvičení
- **Správa pripojení** - automatické spracovanie pripojenia a odpojenia klientov s príslušným čistením zdrojov

Identifikované problémy

Analýza odhalila kritický problém s nekorektným ukončovaním serverovej aplikácie. Problém sa prejavoval ako neočakávané ukončenie servera s chybovou hláškou pri pokuse o štandardné zatvorenie programu, napríklad použitím klávesovej skratky CTRL+C alebo zatvorením konzolového okna. Server počas behu fungoval stabilne, avšak proces ukončovania konzistentne vyvolával chybu.

Kompiláciou a testovaním buildov z rôznych commitov bolo zistené, že chyba sa objavila po committe obsahujúcom aktualizáciu Unity na verziu 2021.3.35f1, aktualizáciu Mirror Networking na verziu 72.0.0 a integráciu Vivox SDK. Keďže hlasová komunikácia vyžaduje správne uvoľnenie systémových zdrojov, primárne podozrenie padlo na implementáciu Vivox SDK.

S cieľom vylúčiť Vivox ako príčinu boli najprv implementované rôzne prístupy k správne ukončovaniu Vivox služieb. Bolo testované korektné odhlásenie pri ukončovaní aplikácie prostredníctvom metódy `OnDestroy()`, ako aj alternatívne kombinácie poradia ukončovania služieb prostredníctvom `OnApplicationQuit()`. Napriek týmto úpravám problém pretrval. Následne boli odstránené všetky Vivox knižnice a súvisiace skripty z build procesu servera. Ani po ich odstránení však problém nezmizel, čo Vivox ako primárny zdroj chyby vylúčilo.

Po vylúčení Vivox ako príčiny problému bola vykonaná detailná analýza systémových logov. Unity generuje logy v štandardnom adresári pre používateľské dáta aplikácie:

```
%USERPROFILE%/AppData/LocalLow/<CompanyName>/<ProductName>/Logs/Latest/
```

Pri kritických chybách sa vytvárajú detailné crash dump súbory v dočasnom adresári:

```
%USERPROFILE%/AppData/Local/Temp/<CompanyName>/<ProductName>/Crashes/
```

Súbory obsahujú detailné informácie o stave aplikácie v momente zlyhania. Analýza crash dump súborov odhalila nasledujúci typ chyby:

Výpis 3.1: Kľúčové informácie z crash dump

```
ExceptionCode: c0000005 (Access violation)
ExceptionAddress: UnityPlayer+0x1b4e4f4
Failure.Bucket: NULL_CLASS_PTR_READ_INVALID_POINTER_READ
```

Typ chyby c0000005 (Access violation) indikuje pokus o prístup k neplatnej pamäťovej adrese. Konkrétne ide o dereferenčnú chybu nulového ukazovateľa. Kritickým zistením bolo, že chyba vzniká vnútri modulu `UnityPlayer.dll` v hlavnom cykle Unity engine (`UnityMain`), pričom stack trace neobsahoval žiadne priame referencie na používateľský kód, Mirror Networking ani Vivox. Problém je teda lokalizovaný vo vnútornom kóde Unity engine verzie 2021.3.35f1. Problém sa prejavuje výlučne pri ukončovaní servera a neovplyvňuje stabilitu servera počas behu, preto neprekáža terapeutickým cvičeniam. Ako dočasné riešenie môže byť použité programové prinútené ukončenie procesu, ktoré obchádza štandardnú procedúru ukončovania v prostredí Unity.

Možnosti vylepšenia

Novšie verzie Mirror Networking obsahujú vylepšenia v správe sieťových zdrojov a životného cyklu pripojení, ktoré môžu prispieť k stabilite pri ukončovaní servera a priniesť ďalšie vylepšenia:

Verzia 73.0.0 obsahuje opravu “Fixed: Reset of statics in NetworkClient and NetworkServer”, ktorá môže vyriešiť problémy s nekorektným ukončovaním servera [12].

Verzia 81.4.0 prináša dôležité opravy súvisiace s čistením zdrojov:

- “fix: NetworkClient unbatcher is now reset in host mode connect” - zlepšenie čistenia zdrojov
- “fix: KcpServer.Stop now clears connections so they aren’t carried over” - korektné čistenie pripojení při zastavení

Verzia 89.0.0 obsahuje významné vylepšenia:

- “NetworkServer and NetworkClient respect exceptionDisconnect” - zlepšené spracovanie výnimiek pri odpojení
- “NetworkServer.Destroy refactored” - spoľahlivejšie ničenie objektov

Verzia 89.80 prináša kľúčovú opravu: “Batches are now properly returned to NetworkWriterPool before destroying the connection”, ktorá priamo rieši problémy s čistením zdrojov pri ukončovaní pripojení.

Verzia 93.0.0 obsahuje funkcionálne vylepšenia relevantné pre VR aplikácie:

- Pridaný ServerTeleport v NetworkTransform - umožňuje okamžité premiestnenie objektov na serveri, čo je často potrebné vo VR prostredí
- Prepracovaná synchronizácia NetworkTransform - zlepšenie plynulosti pohybu objektov v sieti

Najnovšia dostupná verzia je 96.5.0, ktorá obsahuje všetky uvedené opravy a vylepšenia. Aktualizácia na túto verziu by mala vyriešiť identifikované problémy s ukončovaním servera a zároveň priniesť výkonnostné a funkcionálne vylepšenia pre celý systém.

Záver analýzy Mirror Networking

Problémy s nekorektným ukončovaním servera, ktoré sa vyskytujú v súčasnej verzii, sú v novších verziách systematicky riešené.

Najnovšia verzia 96.5.0 obsahuje všetky kľúčové opravy súvisiace s čistením sieťových zdrojov, zlepšeným spracovaním výnimiek a optimalizovanou synchronizáciou objektov. Tieto vylepšenia sú obzvlášť dôležité pre VR aplikácie, kde je potrebná vysoká responzivnosť a stabilita sieťovej komunikácie.

Aktualizácia na verziu 96.5.0 prinesie nielen riešenie identifikovaných problémov, ale aj nové funkcionality ako ServerTeleport a vylepšenú synchronizáciu NetworkTransform, ktoré môžu byť využité pre ďalšie rozšírenie možností kolaboratívnej neurorehabilitácie [12].

3.2.2 Fish-Networking (FishNet)

Fish-Networking (FishNet) predstavuje alternatívne open-source riešenie pre sieťovú komunikáciu v Unity. FishNet je rýchlejší a ponúka viac funkcií ako akékoľvek iné bezplatné sieťové riešenie [13].

Počas vývoja kolaboratívneho systému pre neurorehabilitáciu bolo identifikovaných niekoľko technických obmedzení súčasného Mirror Networking riešenia. Primárne sa jednalo o pomerne pomalý cyklus aktualizácií Mirror Networking, ako aj problémy s kompatibilitou niektorých potrebných riešení. Napríklad, mnoho populárnych riešení na pridanie hlasového chatu okrem Vivox malo problémy s integráciou s Mirror Networking. Tieto technické limitácie viedli k analýze FishNet ako potenciálnej alternatívy, ktorá ponúka modernú architektúru a širšiu podporu pre tretie strany riešenia [14].

Základné charakteristiky

FishNet oficiálne podporuje Unity verzie 2021.3 a novšie, čo zabezpečuje plnú kompatibilitu s aktuálne používanou verziou Unity 2021.3.35f1 v našom projekte. Táto knižnica ponúka širokú škálu funkcionalít, ktoré pokrývajú požiadavky moderných multiplayer aplikácií.

Kľúčové vlastnosti FishNet zahŕňajú:

- **No-Break Promise** - záväzok, že aktualizácie medzi hlavnými verziami nebudú vyžadovať zmeny v užívateľskom kóde, čo zabezpečuje dlhodobú stabilitu projektu
- **Long-Term Support** - jediné riešenie s bezplatnou LTS podporou, ktorá umožňuje prepínanie medzi stable a development verziami funkcionalít
- **Server Authoritative Design** - architektúra navrhnutá primárne pre dedikované servery s podporou peer-to-peer topológie
- **Tick-Based System** - presný časovací systém pre client-side prediction a plynulú synchronizáciu objektov
- **Zero Hotpath Allocations** - na rozdiel od Mirror a Netcode, FishNet neprodukuje garbage allocations počas behu, čo zabezpečuje lepší výkon

Pokročilé funkcionality

FishNet poskytuje množstvo pokročilých funkcionalít, ktoré presahujú základnú funkcionalitu Mirror Networking:

SyncType Flexibility - rozšírené možnosti konfigurácie synchronizovaných premenných vrátane možnosti posilať reliable/unreliable údaje, nastaviť rôzne intervaly pre každú SyncType, a kontrolovať kto dáta prijíma.

Addressables Integration - na rozdiel od iných riešení má FishNet vstavaný support pre Unity Addressables, čo uľahčuje prácu s dynamicky načítavanými zdrojmi a scénami.

Network Balancing - podpora pre neobmedzený počet NetworkManagers, čo umožňuje klientom pripojiť sa súčasne k viacerým serverom. Každý NetworkManager môže spracovávať odlišné úlohy, svetové prostredia alebo herné módy.

Custom SyncTypes - framework umožňuje vytvárať vlastné synchronizačné typy s prispôbenou logikou, čo poskytuje väčšiu flexibilitu pri návrhu sieťovej architektúry.

Rigidbody Controllers a Physics - vstavaná podpora pre real-time fyzikálne interakcie s rigidbody kontrolérmi, čo je dôležité pre VR aplikácie s fyzikálnymi manipuláciami [13].

Nevýhody a riziká migrácie

Napriek množstvu výhod, ktoré FishNet ponúka, migrácia z Mirror Networking na FishNet predstavuje významné technické a časové výzvy.

Primárnym problémom je potreba komplexného refaktorovania sieťového kódu. Mirror a FishNet používajú odlišné API a architektonické vzory, čo znamená, že všetky sieťové komponenty systému (CustomNetworkManager, NetworkCharacterManager, TherapistMenuManager a ďalšie) by museli byť prepísané od základov.

Druhým kritickým faktorom je riziko zavedenia nových chýb do v súčasnosti stabilného riešenia. Systém neurorehabilitácie je už funkčný a otestovaný s Mirror Networking. Migrácia na novú sieťovú knižnicu prináša nebezpečenstvo destabilizácie systému a vznik nových, ťažko identifikovateľných problémov, ktoré by mohli ovplyvniť terapeutické sedenia s pacientmi.

Záver analýzy FishNet

Na základe vykonanej analýzy FishNet ako alternatívneho sieťového riešenia možno konštatovať, že napriek množstvu pokročilých funkcionalít a výhod, ktoré FishNet ponúka, nie je v súčasnej fáze projektu vhodné pristúpiť k migrácii z Mirror Networking na FishNet.

FishNet by sa mal považovať za vhodnú alternatívu v nasledujúcich prípadoch:

- Pri vývoji nového projektu od začiatku, kde by sa dalo využiť modernú architektúru a lepšiu podporu pre tretie strany riešenia ako Vivox

- Pri potrebe nových funkcionalít ako Network Balancing pre multi-server architektúru
- Ak by sa v budúcnosti objavili neprekonateľné limitácie Mirror Networking

Využitie FishNet je plánované v rámci novej architektúry systému v nadväzujúcich prácach. Jeho moderná architektúra, rozsiahla podpora sieťových transportných protokolov a aktívny vývoj ho robia atraktívnou voľbou pre nové VR a multiplayer aplikácie, obzvlášť v prípadoch kde je potrebná široká podpora pre tretie strany služby. Pre súčasný systém však predstavuje zbytočné riziko a čas, ktorý môže byť efektívnejšie využitý na implementáciu nových rehabilitačných funkcionalít a vylepšení používateľskej skúsenosti.

3.2.3 Unity Engine

Unity Engine predstavuje základnú vývojovú platformu celého systému kolaboratívnej neurorehabilitácie. V súčasnosti je používaná verzia 2021.3.35f1 LTS, ktorá poskytuje stabilnú základňu pre multiplatformový vývoj VR aplikácií [15].

Súčasná verzia a jej vlastnosti

Unity 2021.3 LTS bola vydaná ako Long Term Support verzia s podporou do roku 2024 [15]. Táto verzia obsahuje základnú podporu pre XR technológie a poskytuje stabilné API pre vývoj VR aplikácií. LTS verzie sú navrhnuté pre projekty vyžadujúce dlhodobú podporu a minimálne riziko kompatibilných problémov pri menších aktualizáciách.

Identifikované limitácie

Analýza súčasnej verzie Unity odhalila niekoľko oblastí, kde novšie verzie ponúkajú významné vylepšenia [15]:

- **Zastaraná podpora XR technológií** - novšie verzie obsahujú vylepšenú integráciu s OpenXR
- **Suboptimálny výkon vo VR** - chýbajú optimalizácie pre stereoskopické renderovanie
- **Limitovaná podpora Android XR** - absentuje podpora pre Google Android XR platformu

Možnosti aktualizácie

Unity 2022.3 LTS predstavuje najnovšiu stabilnú LTS verziu s významným vylepšeniami pre VR vývoj. Microsoft oficiálne odporúča použitie najnovšej dostupnej verzie z 2022.3 LTS línie spolu s najnovším OpenXR Mixed Reality pluginom pre optimálnu stabilitu a výkon [16].

Kľúčové výhody Unity 2022.3 LTS línie [15]:

- **Aktualizovaný Universal Render Pipeline (URP)** - lepšia podpora VR grafiky a výkonu
- **Vylepšené renderovanie** - optimalizované osvetlenie a grafické efekty pre VR prostredie
- **Vylepšená kompatibilita** - lepšia integrácia s modernými VR zariadeniami

Unity 6 (2023.x) prináša revolučné zmeny pre multiplatformový a VR vývoj, vrátane vylepšených multiplayer nástrojov, zjednodušeného build procesu a podpory pre Google Android XR [17]:

- **Vylepšené multiplayer nástroje** - zjednodušenie vývoja sieťových aplikácií
- **Zjednodušený build proces** - rýchlejšie a spoľahlivejšie buildovanie aplikácií
- **Podpora Google Android XR** - natívna podpora pre najnovšie Android XR zariadenia

Odporúčania pre migráciu

Aktualizácia na najnovšiu verziu z Unity 2022.3 LTS línie predstavuje kompromis medzi stabilitou a modernými funkciami. Táto verzia je oficiálne podporovaná pre VR vývoj a obsahuje všetky potrebné optimalizácie pre neurorehabilitačné aplikácie [16].

Migrácia na Unity 6 by priniesla najväčšie výhody z hľadiska výkonu a funkcionalite, avšak môže vyžadovať významné prepracovanie existujúceho kódu. Rozhodnutie o migrácii by malo zohľadniť čas potrebný na adaptáciu existujúceho systému verzus prínosy z nových funkcionalít.

Na základe analýzy sa odporúča postupná migrácia na Unity 2022.3 LTS ako prvý krok. Migrácia na Unity 6 bude riešená v rámci novej architektúry systému v nadväzujúcich prácach.

3.2.4 XR Interaction Toolkit

XR Interaction Toolkit predstavuje kľúčový komponent zodpovedný za všetky VR interakcie v rámci systému. Tento komponent zabezpečuje vzájomné pôsobenie medzi používateľmi a virtuálnymi objektmi, vrátane manipulácie objektov, teleportácie a používateľského rozhrania v 3D priestore.

Súčasná implementácia

V súčasnosti je používaná verzia 2.3.2, ktorá je rovnako ako všetky novšie verzie kompatibilná s Unity 2021.3 a novšími verziami [18]. Táto verzia poskytuje základnú funkcionálnu pre VR interakcie.

Súčasná implementácia využíva nasledujúce kľúčové komponenty:

- **XR Direct Interactor** - pre priamu manipuláciu objektov v dosahu ruky
- **XR Ray Interactor** - pre interakciu s objektmi na diaľku pomocou lúča
- **XR Socket Interactor** - pre umiestnenie objektov do preddefinovaných pozícií
- **Locomotion System** - pre pohyb používateľa vo virtuálnom priestore

Možnosti aktualizácie

Konzervatívna aktualizácia na verziu 2.4.3 predstavuje bezpečnú voľbu s minimálnymi zmenami v existujúcom kóde. Táto verzia prináša vylepšenia stability a výkonu bez potreby rozsiahleho prepracovania [19].

XR Interaction Toolkit 3.0 predstavuje stabilnú verziu s revolučnými zmenami v architektúre. Kľúčové vylepšenia zahŕňajú [20]:

- **Near-Far Interactor** - kombinuje funkcionálnu Direct a Ray Interactor do jedného komponenta, umožňujúc plynulé prechody medzi interakciami na blízko a na diaľku
- **Nová Input Reader architektúra** - zjednodušuje spracovanie vstupov a podporuje širokú škálu vstupných zariadení
- **XR Body Transformers a Locomotion Mediator** - zjednodušujú kód a poskytujú väčšiu flexibilitu pri rozširovaní lokomočného systému
- **Vylepšené climb a teleportácia** - podporujú komplexnejšie pohyby, osobitne vertikálnu navigáciu

XR Interaction Toolkit 3.1.2 predstavuje najnovšiu stabilnú verziu, ktorá obsahuje ďalšie vylepšenia a opravy [21]:

- **UI Toolkit podpora** - vylepšená kompatibilita s Unity UI Toolkit
- **Spatial Keyboard sample** - nový vzor pre 3D klávesnice vo VR prostredí

Výkonnostné vylepšenia verzie 3.x oproti aktuálne používanej verzii 2.3.2 zahŕňajú:

- **Ray Endpoint Stabilization** - minimalizuje chvenie lúčov pre plynulejšie používateľské skúsenosti [22]
- **Burst Compiler optimalizácie** - zlepšenie výkonu výpočtov pre VR interakcie
- **Cone Casting** - presnejšia detekcia kolízií pre ray interactor komponenty

Výzvy migrácie

Prechod na **XRI 3.0** vyžaduje významné zmeny v existujúcom kóde [23]:

- **Deprecated komponenty** - mnohé triedy boli nahradené novými implementáciami
- **Namespace zmeny** - reorganizácia namespace štruktúry vyžaduje aktualizáciu scriptov
- **Input System zmeny** - nová input architektúra vyžaduje prepracovanie input handling

Odporúčania

Pre súčasný systém neurorehabilitácie sa odporúča postupná migrácia. Prvým krokom by mala byť aktualizácia na verziu 2.4.3, ktorá prinesie stabilnostné vylepšenia bez potreby rozsiahlych zmien v kóde.

Migrácia na XRI 3.1.2 a využitie jej revolučných funkcionalít bude riešená v rámci novej architektúry systému v nadväzujúcich prácach.

3.2.5 OpenXR Plugin

OpenXR Plugin zabezpečuje univerzálnu podporu pre široké spektrum VR a **AR** zariadení prostredníctvom štandardizovaného OpenXR API. Táto technológia umožňuje systému fungovať na rôznych VR headsetoch bez potreby špecifických **SDK** pre každé zariadenie.

Súčasná implementácia

V súčasnej implementácii je používaná verzia 1.7, ktorá je kompatibilná s Unity 2020 LTS a novšími verziami [24]. OpenXR Plugin zabezpečuje základnú komunikáciu medzi Unity a VR runtime prostredím, vrátane sledovania pozície headsetu a ovládačov, renderovania stereoskopického obrazu a spracovania vstupov.

Kľúčové funkcionality:

- **Device tracking** - sledovanie pozície a orientácie VR zariadení
- **Stereo rendering** - generovanie stereoskopického obrazu pre VR displeje
- **Input handling** - spracovanie vstupov z VR ovládačov
- **Runtime abstraction** - jednotné API pre rôzne VR platformy

Možnosti aktualizácie

Verzia 1.10.0 obsahuje významné vylepšenia špecificky ako [25]:

- **Optimalizácia pre Quest 2, 3 a Quest Pro** - špecializované nastavenia pre tieto zariadenia
- **Vylepšené sledovanie rúk** - Hand Interaction Profile s podporou pokročilých gestov
- **Optimalizácia renderovania** - zlepšená efektivita vykresľovania pre mobilné VR headsety

Najnovšia verzia 1.14.3 prináša najmodernejšie funkcionality a opravy [25]:

- **Performance optimalizácie** - zlepšená správa rendering performance stavov cez XrPerformanceSettings
- **Meta Quest 3 podpora** - natívna podpora pre najnovšie Meta Quest zariadenia
- **Stabilita a opravy** - množstvo opráv súvisiacich s Angular velocity tracking a runtime stability

Odporúčania

Pre neurorehabilitačný systém sa odporúča aktualizácia na verziu 1.14.3, ktorá prinesie značné výkonnostné vylepšenia a rozšírenú podporu pre moderné VR zariadenia. Táto verzia obsahuje všetky optimalizácie potrebné pre stabilný beh na Meta Quest zariadeniach a zároveň zabezpečuje kompatibilitu s budúcimi VR platformami.

Aktualizácia na najnovšiu verziu je obzvlášť dôležitá pre dlhodobú udržateľnosť projektu, keďže obsahuje podporu pre najnovšie OpenXR štandardy a zabezpečuje kompatibilitu s najnovšími VR runtime prostrediami [24].

3.2.6 Final IK

Final IK predstavuje riešenie pre inverse kinematics výpočty v Unity, ktoré zabezpečuje realistické animácie virtuálnych rúk v systéme neurorehabilitácie. Tento komponent je zodpovedný za generovanie prirodzených pohybových animácií na základe cieľových pozícií, čo je kľúčové pre vizuálnu spätnú väzbu pri rehabilitačných cvičeniach.

Súčasná implementácia

V súčasnosti je používaná verzia 2.3, ktorá poskytuje stabilnú podporu pre **VR IK** systém [26]. Táto verzia je kompatibilná so všetkými podporovanými verziami Unity vrátane 2021.3 LTS a novších verzií.

Dostupné vylepšenia

Verzia 2.4 obsahuje vylepšenia pre Interaction System, ako podpory pre plynulé prerušenia interakcií. Táto funkcionálna umožňuje začatie novej interakčnej animácie ešte pred dokončením predchádzajúcej.

Odporúčania

Pre súčasný systém neurorehabilitácie môže byť zvažovaná aktualizácia na verziu 2.4. Plynulé prerušenia interakcií môžu zlepšiť používateľskú skúsenosť pacientov pri vykonávaní rehabilitačných cvičení. Aktualizácia Final IK však nie je kritická pre fungovanie systému, keďže súčasná verzia 2.3 poskytuje všetky potrebné funkcionality pre neurorehabilitačné účely.

3.2.7 Výber databázového systému

Pre implementáciu perzistentného ukladania nastavení pacientov bol potrebný vhodný databázový systém. Kľúčovými požiadavkami boli jednoduchosť nasadenia bez potreby konfigurácie databázového servera, kompatibilita s Unity engine a minimálne nároky na systémové zdroje. Boli porovnané tri databázové systémy:

MySQL [27] je open source relačný databázový systém používaný vo veľkých aplikáciách. MySQL je druhý najpopulárnejší databázový systém a ponúka plnú podporu **ACID** transakcií, vynikajúcu škálovateľnosť a schopnosť spracovať vysoký objem súčasných pripojení. Nevýhodou MySQL je však potreba samostatného serverového procesu a konfigurácie databázového servera, čo by komplikovalo nasadenie systému v rehabilitačných zariadeniach.

LiteDB [28] je .NET NoSQL databáza s jedným dátovým súborom, napísaná v C# kóde s podporou ACID transakcií a šifrovania dát. Táto databáza by bola vhodná pre Unity projekty vďaka svojej .NET kompatibilite, avšak jej NoSQL charakter s dokumentovým modelom bol nad rámec požiadaviek projektu.

SQLite [29] je zabudovaný SQL databázový engine, ktorý nevyžaduje samostatný serverový proces. Databáza ukladá všetky dáta do jedného súboru a ponúka nasadenie bez nutnosti konfigurácie. SQLite je kompaktná knižnica a podporuje plnú SQL syntax s ACID transakciami aj pri výpadku napájania.

Rozhodujúcim faktorom pre výber SQLite bola absencia potreby konfigurácie databázového servera. Zatiaľ čo MySQL vyžaduje inštaláciu a správu samostatného procesu a LiteDB ponúkal NoSQL funkcionality nad rámec požiadaviek, SQLite poskytoval presne to, čo projekt vyžadoval, jednoduchú zabudovanú relačnú databázu s nulovými konfiguračnými nárokmi.

3.3 Záver analýzy a odporúčania pre aktualizáciu

Na základe vykonanej analýzy všetkých kľúčových komponentov systému boli identifikované konkrétne kroky pre vylepšenie technologickej základne kolaboratívneho neurorehabilitačného systému. Okrem technologických komponentov bol analyzovaný aj výber databázového systému pre implementáciu perzistentného ukladania nastavení pacientov, pričom bol zvolený SQLite. Analýza odhalila, že väčšina súčasne používaných technológií je stabilná a funkčná, avšak existujú možnosti zlepšenia výkonu a funkcionality prostredníctvom aktualizácií.

3.3.1 Prioritné aktualizácie

Mirror Networking vyžaduje najvyššiu prioritu aktualizácie z verzie 72.0.0 na verziu 96.5.0. Táto aktualizácia prináša vylepšenia v správe životného cyklu pripojení a čistení zdrojov, ktoré priamo súvisia s identifikovaným problémom pádu servera. V rámci analýzy boli zvážené aj alternatívne sieťové riešenie FishNet. Napriek pokročilým funkciám a výhodám FishNet, analýza preukázala, že migrácia na toto riešenie nie je v súčasnej fáze projektu opodstatnená z dôvodu časovej náročnosti a funkčnej dostatočnosti Mirror Networking po aktualizácii na verziu 96.5.0.

Unity Engine - odporúča sa migrácia na najnovšiu verziu z Unity 2022.3 LTS línie, ktorá prináša vylepšenú stabilitu XR knižníc, optimalizovaný Universal Render Pipeline a opravy v správe systémových zdrojov pri ukončovaní aplikácie. Migrácia na Unity 6 bude riešená v rámci novej architektúry systému v nadväzujúcich prácach.

3.3.2 Odporúčané aktualizácie

XR Interaction Toolkit - aktualizácia na verziu 2.4.3 pre stabilnostné vylepšenia. Migrácia na XRI 3.x bude súčasťou novej architektúry systému v nadväzujúcich prácach.

OpenXR Plugin - aktualizácia na verziu 1.14.3 prinesie vylepšenú podporu pre Meta Quest zariadenia, optimalizácie výkonu a rozšírenú funkcionálnosť pre moderné VR platformy.

3.3.3 Voliteľné aktualizácie

Final IK - aktualizácia na verziu 2.4 nie je kritická, ale môže priniesť mierne vylepšenia v plynulosti interakcií prostredníctvom podpory pre smooth interruptions v Interaction System.

3.3.4 Implementačná stratégia

Odporúča sa postupný prístup k aktualizáciám s dôrazom na testovanie kompatibility po každom kroku. Aktualizácie by mali prebiehať v nasledujúcom poradí pre minimalizáciu rizík a maximalizáciu stability systému.

Ako prvá by mala byť implementovaná aktualizácia Mirror Networking na verziu 96.5.0 kvôli problémom so serverom. Po úspešnej stabilizácii Mirror Networking možno pokračovať aktualizáciou XR Interaction Toolkit na verziu 2.4.3,

ktorá prinesie stabilnostné vylepšenia bez potreby rozsiahlych zmien v kóde. Následne by mala byť aktualizovaná OpenXR Plugin na verziu 1.14.3 pre vylepšenú podporu moderných VR zariadení.

Migrácia na Unity 2022.3 LTS predstavuje najkomplexnejšiu úlohu vyžadujúcu významné zmeny v projekte. Migrácia na Unity 6 a XR Interaction Toolkit 3.x bude riešená v rámci novej architektúry systému v nadväzujúcich prácach.

Táto stratégia zabezpečí minimalizáciu rizík a umožní dôkladné testovanie každej aktualizácie pred pokračovaním na ďalšiu fázu. Výsledkom bude modernizovaný, stabilný a výkonný systém pripravený na ďalší vývoj a rozšírenie funkcionalít.

4 Návrh a implementácia vylepšení systému

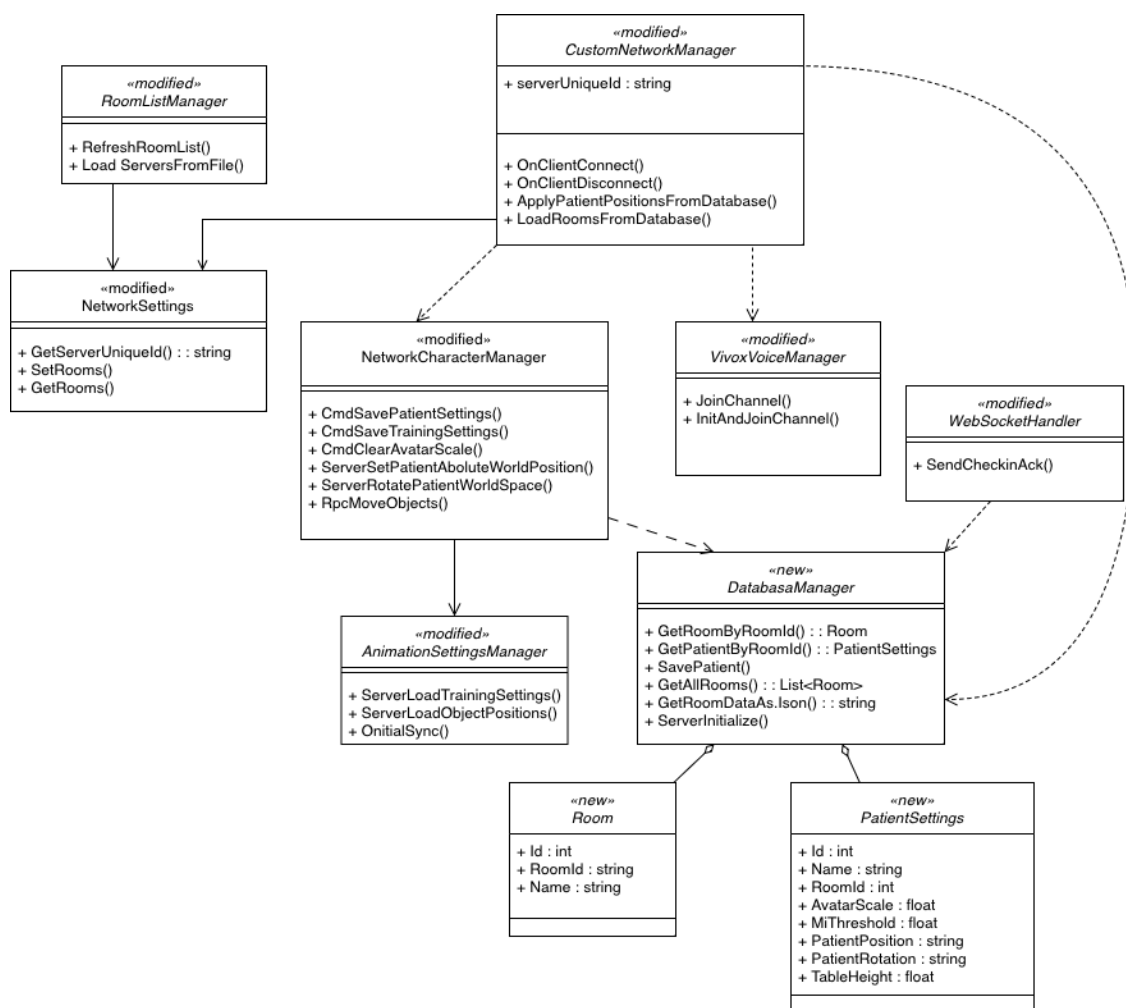
Kolaboratívny systém pre neurorehabilitáciu vo virtuálnej realite je dlhodobý projekt, ktorý si vyžaduje pokračovanie vývoja a opravu chýb. Ako každý softvérový produkt v vývoji, aj tento systém prechádza zmenami a vylepšeniami na základe spätnej väzby od používateľov, požiadaviek vedúceho práce a technických obmedzení identifikovaných počas testovania a nasadenia v reálnom prostredí.

Táto kapitola popisuje implementáciu kľúčových vylepšení, ktoré boli realizované v rámci aktuálnej fázy vývoja projektu. Jednotlivé sekcie dokumentujú proces identifikácie problémov, návrh riešení a konkrétnu implementáciu týchto riešení. Okrem opravy existujúcich a zistených problémov obsahuje kapitola aj aktualizácie používaných technologických komponentov na ich najnovšie stabilné verzie v súlade s vykonanou analýzou, čo zabezpečuje dlhodobú stabilitu projektu a odstránenie známych chýb v používaných knižniciach. Zavedené zmeny tvoria základ pre ďalšie rozširovanie systému a otvárajú možnosti pre budúce funkčné vylepšenia v oblasti neurorehabilitačných cvičení a používateľského zážitku.

Prehľad všetkých tried dotknutých v tejto kapitole zobrazuje diagram tried (obr. 4.1). Triedy označené stereotypom <<new>> boli vytvorené v rámci tejto práce, triedy označené <<modified>> boli rozšírené alebo upravené oproti pôvodnej implementácii.

V rámci tejto kapitoly boli realizované nasledujúce oblasti vylepšení:

- oprava hlasovej komunikácie a izolácia kanálov pomocou **GUID** identifikátorov
- centralizovaná správa serverov a miestností s načítavaním z konfiguračného súboru
- riešenie pádu servera pri ukončení



Obrázok 4.1: Diagram tried s vyznačením nových a modifikovaných tried

- implementácia systému perzistentných profilov pacientov s SQLite databázou
- aktualizácia technologických komponentov na aktuálne stabilné verzie

4.1 Riešenie problémov s hlasovou komunikáciou

Hlasová komunikácia medzi terapeutom a pacientom predstavuje kľúčový aspekt kolaboratívneho neurorehabilitačného systému. Počas testovania existujúcej implementácie využívajúcej Vivox boli identifikované viaceré technické problémy súvisiacich s izolovanými komunikačnými kanálmi a správou životného cyklu pripojenia, ktoré negatívne ovplyvňovali funkcionality a používateľskú skúsenosť. Táto sekcia opisuje identifikované problémy a implementované riešenia, ktoré zabezpečujú spoľahlivú a izolovanú hlasovú komunikáciu v rámci jednotlivých terapeutických sedení.

4.1.1 Identifikácia problému s izolovanými komunikačnými kanálmi

Pôvodná implementácia systému hlasovej komunikácie prostredníctvom Vivox umožňovala náhodne pripojiť niekoľko klientov k jednému serveru hlasového chatu. K tomu dochádzalo napríklad vtedy, keď sa používatelia pripájali k lokálnemu serveru a kanál hlasovej komunikácie sa predtým vytváral podľa názvu miestnosti a IP adresy. Servery pre hlasovú komunikáciu poskytuje Vivox, súčasť Unity, takže sa k nim je možné pripojiť aj lokálne pomocou určitého ID, ak je k dispozícii prístup k internetu. Lokálne to mohlo viesť k rovnakým identifikátorom u viacerých používateľov, ktorí sa ani nenachádzali v rovnakom priestore. Taktiež, ak zadáte inú IP adresu, môžete sa pripojiť k hlasovému chatu, aj keď sa kvôli nastaveniam servera nedostanete do terapeutického priestoru.

4.1.2 Implementácia unikátnej identifikácie serverov

Riešenie tohto problému spočívalo v zavedení unikátnej identifikácie pre každú serverovú inštanciu. Pri inicializácii servera sa generuje globálne unikátny identifikátor (GUID), ktorý je následne použitý ako súčasť názvu hlasového kanála.

Generovanie unikátneho identifikátora pri štarte servera je implementované v triede `CustomNetworkManager` volaním `Guid.NewGuid().ToString()`, ktorý je následne uložený do `NetworkSettings`.

Tento unikátny identifikátor sa kombinuje s identifikátorom miestnosti (room ID) pri vytváraní názvu hlasového kanála v triede `VivoxVoiceManager`:

Výpis 4.1: Tvorba unikátneho názvu kanála

```
private async Task JoinChannel()
{
    if (string.IsNullOrEmpty(networkSettings.GetServerUniqueId()))
    {
        return;
    }
    string serverSpecificRoomId = $"{networkSettings.GetServerUniqueId()}_
{networkSettings.GetRoomId()}";
    await VivoxService.Instance.JoinGroupChannelAsync(
        serverSpecificRoomId, ChatCapability.AudioOnly);
}
```

Výsledný názov kanála má formát `<serverGUID>_<roomID>`, čo zabezpečuje, že každá serverová inštancia vytvorí pre každú miestnosť vlastný izolovaný hlasový kanál. Tento prístup umožňuje prevádzku viacerých nezávislých terapeutických sedení súčasne bez akéhokoľvek prekrývania hlasovej komunikácie.

4.1.3 Refaktorizácia správy pripojenia k hlasovej komunikácii

Okrem problému s izolovanými kanálmi bola identifikovaná potreba prepracovania spôsobu, akým sa systém pripája k hlasovej komunikácii. Pôvodná implementácia obsahovala niekoľko problematických aspektov, ktoré mohli viesť k nekonzistentnému správaniu systému.

Problémy pôvodnej implementácie

Pôvodná implementácia sa pokúšala pripojiť klienta k hlasovému kanálu okamžite pri načítaní scény, bez ohľadu na to, či sa klient úspešne pripojil k serveru. Toto správanie spôsobovalo niekoľko problémov:

Problémom bolo pripojenie k hlasovému kanálu aj v prípade, keď pripojenie k serveru zlyhalo. Používateľ, ktorý sa neúspešne pokúsil pripojiť k serveru (napríklad kvôli nesprávnej IP adrese alebo nedostupnosti servera), sa napriek tomu pripojil do hlasového kanála. To viedlo k situácii, kde používateľ počul iných účastníkov miestnosti, ale sám nebol súčasťou herného sedenia, čo spôsobovalo zmätok a nekonzistentnú používateľskú skúsenosť.

Ďalším aspektom, ktorý vyžadoval úpravu, bolo predvolené nastavenie hlasového chatu. V pôvodnej implementácii bol hlasový chat aktívny ihneď po pripojení, čo nebolo vždy žiaduce.

Tiež bolo potrebné vykonať malý redizajn celej triedy `VivoxVoiceManager`, aby bolo možné správne riadiť inicializáciu a pripojenie k hlasovému chatu.

Implementované riešenie

Riešenie spočívalo v prepojení životného cyklu hlasovej komunikácie so životným cyklom pripojenia k hernému serveru. Pripojenie k hlasovému kanálu sa presunulo do metódy `OnClientConnect()`, ktorá je volaná iba v prípade úspešného pripojenia klienta k serveru:

Výpis 4.2: Pripojenie k hlasovému kanálu po úspešnom pripojení k serveru

```
public override void OnClientConnect()
{
    CharacterMessage characterMessage = new()
    {
        role = roleSettings.CharacterRole,
        hmdType = xrSettings.HmdType,
        // ... ostatné parametre
        roomId = networkSettings.GetRoomId()
    };

    NetworkClient.Send(characterMessage);
    _ = VivoxVoiceManager.Instance.InitAndJoinChannel();
}
```

```
}
```

Táto zmena zabezpečuje, že používateľ sa pripojí k hlasovému kanálu iba vtedy, keď je jeho pripojenie k hernému serveru úspešné. V prípade neúspešného pripojenia k serveru metóda `OnClientConnect()` nie je volaná, a tým pádom nedôjde ani k pripojeniu k hlasovému kanálu.

Analogicky bolo implementované odpojenie od hlasového kanála v metóde `OnClientDisconnect()`:

Výpis 4.3: Odpojenie od hlasového kanála pri odpojení od servera

```
public override void OnClientDisconnect()
{
    base.OnClientDisconnect();

    if (SceneManager.GetActiveScene().buildIndex ==
        SceneUtility.GetBuildIndexByScenePath(menuScene))
    {
        return;
    }

    SceneManager.LoadScene(menuScene);
    _ = VivoxVoiceManager.Instance.LeaveGroupChannel();
}
```

Táto implementácia zabezpečuje automatické odpojenie od hlasového kanála pri ukončení pripojenia k serveru, čím sa predchádza situácii, kde by používateľ ostal pripojený k hlasovému kanálu aj po opustení herného sedenia.

Predvolené nastavenie hlasovej komunikácie bolo zmenené tak, že mikrofón a reproduktory sú pri pripojení automaticky stlmené. Používateľ má možnosť hlasovú komunikáciu manuálne aktivovať prostredníctvom používateľského rozhrania až keď je pripravený komunikovať s ostatnými účastníkmi sedenia. Boli tiež pridané viaceré dodatočné kontroly a spracovanie chýb do metód pripájania pre bezpečnejšie a správnejšie pripájanie

Výsledkom týchto zmien je konzistentné správanie systému, kde stav hlasovej komunikácie je vždy synchronizovaný so stavom pripojenia k hernému serveru. Používatelia sa pripájajú k hlasovému kanálu iba v prípade úspešného pripojenia k serveru a automaticky sa odpájajú pri ukončení sedenia.

4.2 Vylepšenia správy serverov a miestností

Efektívna správa pripojení k serverom a zoznamu dostupných terapeutických miestností je kľúčová pre používateľskú skúsenosť systému. Pôvodná implementácia vyžadovala manuálne zadávanie serverových adries pri novom použití aplikácie, čo bolo nepohodlné pre koncových používateľov. Táto sekcia popisuje

implementované vylepšenia, ktoré zjednodušujú správu serverov a zlepšujú spoľahlivosť získavania informácií o dostupných miestnostiach.

4.2.1 Centralizovaná správa zoznamu serverov

Pri prvom spustení aplikácie museli používatelia ručne zadávať IP adresu a port servera, ku ktorému sa chceli pripojiť. Toto riešenie bolo nepohodlné, najmä vzhľadom na to, že pacienti nie sú vždy schopní to urobiť.

Načítavanie serverov z konfiguračného súboru

Riešenie spočívalo v implementácii automatického načítavania zoznamu serverov z externého konfiguračného súboru. V prvej iterácii bol použitý textový formát (TXT), kde každý riadok obsahoval IP adresu a port servera oddelené dvojbodkou:

Výpis 4.4: Príklad konfiguračného súboru servers.txt

```
147.232.205.23:7778
127.0.0.1:7778
```

V druhej iterácii bol formát súboru zmenený z textového na JSON, ktorý poskytuje lepšiu štruktúrovanosť a rozšíriteľnosť dát a je štandardom. JSON formát umožňuje v budúcnosti jednoduchšie pridať ďalšie atribúty k serverom (napríklad názov, popis, región) bez potreby zmeny parsovacieho kódu:

Výpis 4.5: Aktuálny formát konfiguračného súboru servers.json

```
[
  {
    "serverIp": "147.232.205.23",
    "serverPort": 7778
  },
  {
    "serverIp": "127.0.0.1",
    "serverPort": 7778
  }
]
```

Načítavanie z JSON formátu využíva knižnicu `Newtonsoft.Json`²:

Výpis 4.6: Načítavanie serverov z JSON súboru

```
private void LoadServersFromFile()
{
    if (serversJsonFile != null)
    {
        List<ServersData> loadedServers =
            JsonConvert.DeserializeObject<List<ServersData>>(
```

²<https://www.newtonsoft.com/json>

```

        serversJsonFile.text);

    if (loadedServers != null)
    {
        foreach (var server in loadedServers)
        {
            if (IPAddress.TryParse(server.serverIp, out _))
            {
                // Validácia a pridanie servera
                bool serverExists = false;
                foreach (var existingServer in serversList)
                {
                    if (existingServer.serverIp == server.serverIp
                        && existingServer.serverPort == server.↵
                            serverPort)
                    {
                        serverExists = true;
                        break;
                    }
                }

                if (!serverExists)
                {
                    serversList.Add(server);
                }
            }
        }
    }
}

```

4.2.2 Ochrana kritických serverov

Servery je možné odstrániť aj v prípade, že boli pridané z json. V našej aplikácii je však dôležité zabezpečiť, aby používatelia nemohli odstraňovať konfiguračné servery, ktoré sú definované ako hlavné. Na tento účel bol implementovaný mechanizmus chránených serverov (protected servers).

Servery načítané z konfiguračného súboru sú automaticky označené príznakom `isProtected` a pridané do zoznamu `protectedServers`. V používateľskom rozhraní je tlačidlo pre odstránenie servera deaktivované, ak je vybraný chránený server. Servery pridané používateľom sú uložené v `ScriptableObject` a pretrvávajú medzi spusteniami aplikácie, pričom konfiguračný súbor slúži výhradne ako zdroj chránených serverov.

4.2.3 Dynamická aktualizácia zoznamu miestností

Kritickým problémom bolo, že aplikácia vždy odosielať žiadosť o zoznam miestností na IP adresu, ktorú si používateľ vybral naposledy pred vypnutím, bez ohľa-

du na to, ktorý server si používateľ vybral v užívateľskom rozhraní. To znamenalo, že aj keď používateľ vybral server local-host (127.0.0.1), aplikácia naďalej komunikovala s cloudovým serverom **TUKE**, čo viedlo k zobrazeniu nesprávneho zoznamu miestností. Takisto by mal byť uprednostnený cloudový server TUKE ako hlavný.

Riešenie týchto problémov spočívalo v implementácii dvoch kľúčových zmien. Po prvé, zoznam serverov načítavaný z konfiguračného súboru je usporiadaný tak, že TUKE cloud server je na prvej pozícii, čím sa automaticky stáva predvoľným serverom pri prvom spustení aplikácie. Po druhé, bola implementovaná automatická aktualizácia zoznamu miestností pri zmene vybraného servera:

Výpis 4.7: Automatická aktualizácia pri zmene servera

```
private void ServerDropdownHandler(int index)
{
    // ... nastavenie IP a portu

    if (roomListManager != null)
    {
        roomListManager.RefreshRoomList();
    }
}
```

Pre zlepšenie používateľskej skúsenosti bol pridaný timeout pre **HTTP** requesty na získanie zoznamu miestností. Ak server neodpovie do 20 sekúnd, zobrazí sa chybová hláška a použije sa fallback na lokálne uložené miestnosti:

Výpis 4.8: Timeout a error handling pre room list

```
using (UnityWebRequest webRequest = UnityWebRequest.Get(uri))
{
    webRequest.timeout = 20;

    yield return webRequest.SendWebRequest();

    // ... spracovanie odpovede
}

if (!successfullRequest)
{
    customDebug.LogError("Server not responding - cannot get room list ↩️  
from " + uri);

    List<RoomData> rooms = networkSettings.GetRooms();
    int roomCount = rooms.Count;
    roomsSerialized = new RoomDataSerialized[roomCount];
    for (int i = 0; i < roomCount; i++)
    {
        roomsSerialized[i] = new RoomDataSerialized(rooms[i]);
    }
}
```

Implementácia timeout mechanizmu a fallback riešenia bola obzvlášť dôležitá pre diagnostiku problémov s TUKE cloud serverom. Počas testovania bolo zistené, že requesty na TUKE cloud server boli odosielané okamžite, avšak odpoveď zo servera trvala neprimerane dlho, často prekračujúc 20 sekundový timeout. Tento problém bol identifikovaný ako issue na strane TUKE cloud infraštruktúry a nie na strane aplikácie, keďže localhost server odpovedal v rámci niekoľkých sekúnd. Implementovaný timeout mechanizmus tak nielen zlepšil používateľskú skúsenosť, ale aj umožnil jasne identifikovať zdroj problému s pomalým načítaním zoznamu miestností.

4.2.4 Validácia a prevencia duplicity serverov

Pre zabezpečenie konzistentnosti dát bol implementovaný mechanizmus, ktorý kontroluje duplicitu serverov pri ich pridávaní do zoznamu. Systém kontroluje, či server s rovnakou IP adresou už existuje v zozname:

Výpis 4.9: Kontrola duplicity pri pridávaní servera

```
public void AddServerButtonClick()
{
    string inputText = inputServerField.text.Trim();
    string[] parts = inputText.Split(':');

    if (parts.Length == 2 && IPAddress.TryParse(parts[0], out _)
        && int.TryParse(parts[1], out int port))
    {
        string serverIp = parts[0];

        bool serverExists = false;
        foreach (var server in networkSettings.ServersList)
        {
            if (server.serverIp == serverIp)
            {
                serverExists = true;
                break;
            }
        }

        if (!serverExists)
        {
            networkSettings.ServersList.Add(new ServersData
            {
                serverIp = serverIp,
                serverPort = port
            });
            customDebug.Log($"Server {serverIp}:{port} has been ←
                successfully added");
        }
        else
        {

```

```

        customDebug.LogWarning($"Server {serverIp}:{port} already ↵
            exists");
    }
}
}

```

Ďalej bol opravený problém s výberom servera v dropdown liste. Pôvodná implementácia porovnávala iba IP adresy, čo spôsobovalo nesprávny výber v prípade, že dva servery mali rovnakú IP adresu ale odlišný port. Opravená implementácia porovnáva IP adresu aj port súčasne.

Výsledkom týchto implementácií je robustný systém správy serverov, ktorý poskytuje používateľom jednoduchý spôsob prepínania medzi servermi, zároveň chráni kritické konfiguračné servery pred odstránením a zabezpečuje konzistenciu dát v celej aplikácii.

4.2.5 Optimalizácia používateľského rozhrania pre správu serverov

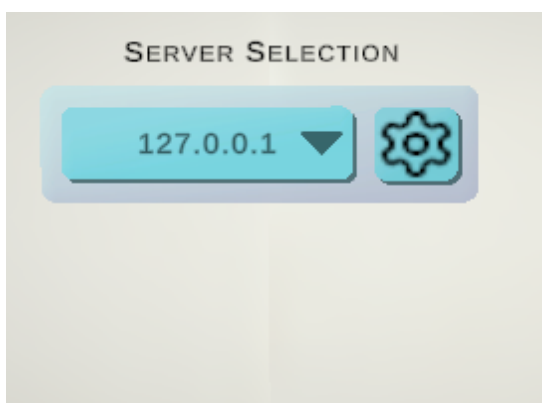
Pôvodné používateľské rozhranie pre správu serverov zobrazovalo všetky ovládacie prvky (dropdown zoznam serverov, vstupné pole pre novú IP adresu, tlačidlá pre pridanie a odstránenie) permanentne na hlavnej obrazovke aplikácie. Tieto prvky zaberali významné miesto v používateľskom rozhraní. Správa serverov nie je operáciou, ktorú používateľ vykonáva často, zvyčajne len pri počiatkoch nastavení aplikácie.

Bolo rozhodnuté, že konfiguračné prvky pre správu serverov budú skryté v popup paneli, ktorý sa zobrazí iba na požiadanie používateľa. Toto riešenie výrazne zlepšuje prehľadnosť hlavnej obrazovky a sústreďuje pozornosť používateľa na primárne funkcie aplikácie: výber miestnosti a pripojenie k sedeniu.

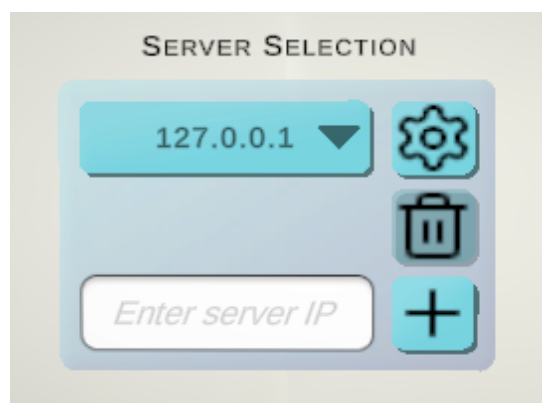
Implementácia popup panelu

Popup panel bol implementovaný ako samostatný GameObject, ktorý je predvolene skrytý a aktivuje sa iba po stlačení tlačidla používateľom. Pri inicializácii aplikácie je popup panel automaticky skrytý.

Používateľské rozhranie v predvolenom stave zobrazuje iba tlačidlo pre otvorenie popup panelu a dropdown zoznam pre výber servera. Po kliknutí na tlačidlo sa popup panel rozbalí a zobrazí všetky konfiguračné možnosti pre správu serverov, vrátane vstupného poľa na pridanie novej IP adresy a tlačidiel na pridanie a odstránenie serverov (obr. 4.2 a 4.3).



Obrázok 4.2: Popup panel v zatvorenom stave



Obrázok 4.3: Popup panel v otvorenom stave

Zjednodušenie konfigurácie serverov

V rámci optimalizácie používateľského rozhrania bola odstránená potreba zadávať port servera pri pridávaní nového servera. Analýza existujúceho systému ukázala, že používaný port bol pevne zapísaný v kóde a server sa vždy spúšťal s rovnakým portom, takže pripojenie na iný port nebolo potrebné. Táto zmena výrazne zjednodušila proces pridávania serverov lebo používateľ teraz zadáva iba IP adresu.

Táto zmena sa prejavila aj v konfiguračnom súbore, ktorý teraz obsahuje iba IP adresy serverov:

Výpis 4.10: Zjednodušený formát servers.json

```
[
  {
    "serverIp": "147.232.205.23"
  },
  {
    "serverIp": "127.0.0.1"
  }
]
```

Bola tiež pridaná podpora lokalizácie všetkých textových prvkov používateľského rozhrania. To zaručuje plynulé fungovanie celej aplikácie pri prepínaní jazyka.

Tieto zmeny prinášajú prehľadnejšie a používateľsky prívetivejšie rozhranie, ktoré neunavuje používateľa zbytočnými konfiguračnými možnosťami a sústreďuje sa na jednoduchosť použitia. Správa serverov je teraz intuitívnejšia a vyžaduje minimálny počet krokov na pridanie nového servera do zoznamu.

4.3 Implementácia profilov pacientov

Z hľadiska praktickej implementácie systému je potrebné zabezpečiť, aby konfigurácia nastavení pacienta nebola vyžadovaná pri každom opakovanom pripojení k rehabilitačnému sedeniu. Terapeut musel opakovane konfigurovať výšku stola, pozíciu pacienta, nastavenia avatara a všetky tréningové parametre, čo bolo časovo náročné a frustrujúce. Bolo rozhodnuté, že systém potrebuje možnosť perzistentného uloženia týchto nastavení.

Počiatočná koncepcia riešenia predpokladala, že používatelia sa budú prihlasovať k serveru pod vlastným používateľským menom, terapeut následne aplikuje na pripojeného pacienta jeden z predtým vytvorených profilov uložených v databáze. Táto myšlienka však bola neskôr prehodnotená a zjednodušená, ako bude popísané ďalej v tejto sekcii.

4.3.1 Integrácia SQLite do Unity projektu

Na základe porovnania databázových systémov vykonaného v analytickej časti práce (v podkapitole 3.2.7) bol pre implementáciu zvolený SQLite.

Problémy s existujúcimi riešeniami

Po výbere SQLite ako databázového systému bolo potrebné nájsť vhodný **plugin**, ktorý by umožnil integráciu SQLite do Unity projektu. Unity samo o sebe neobsahuje natívnu podporu pre SQLite, preto je nutné použiť externú knižnicu.

Prvý pokus o integráciu zahŕňal použitie NuGet balíčka `sqlite-net-pcl` vo verzii 1.9, ktorý je populárnou .NET knižnicou pre prácu so SQLite. Balíček bol stiahnutý a umiestnený do priečinka `Assets/NuGet/` v Unity projekte. Avšak pri pokuse o použitie knižnice sa objavila chyba “Using directive is unnecessary” pri direktíve `using SQLite;`, čo indikovalo, že kompilátor knižnicu nevidí.

Ďalšie pokusy zahŕňali rôzne verzie `sqlite-net-pcl` (1.8.116, 1.9.0) a rôzne spôsoby inštalácie cez Unity Package Manager, avšak žiadne z týchto riešení nefungovalo správne. Problém spočíval v tom, že NuGet balíčky často obsahujú závislosti špecifické pre .NET Framework alebo .NET Core, ktoré nie sú plne kompatibilné s Unity, ktoré používa modifikovanú verziu Mono runtime. Po hľadaní na internete bolo nájdené špecializované riešenie pre Unity – `SQLite4Unity3d` [30].

SQLite4Unity3d

SQLite4Unity3d je plugin vytvorený špeciálne pre Unity projekty, ktorý poskytuje jednoduchý spôsob integrácie SQLite do Unity aplikácií. Plugin využíva knižnicu `sqlite-net` ako základ, čím poskytuje okrem klasických SQL dotazov aj podporu LINQ (Language Integrated Query) pre prácu s dátami a podporuje platformy Android a Windows. Plugin používa iba synchronnú časť `sqlite-net`, čo znamená, že všetky operácie s databázou sú synchronné.

Instalácia SQLite4Unity3d je výrazne jednoduchšia než iné riešenia. Po inštalácii je možné okamžite začať používať SQLite v projekte bez akýchkoľvek dodatočných konfigurácií.

Kritickým krokom pre správnu funkčnosť na rôznych platformách bolo nastavenie Platform Settings pre každú natívnu knižnicu v Unity Inspectore. Pre Windows knižnice (`sqlite3.dll`) boli vybrané platformy Editor a Standalone s architektúrou `x86_64` a operačným systémom Windows. Pre Android knižnice (`libsqlite3.so`) platforma Android s podporou rôznych CPU architektúr (ARM64, ARMv7). Tieto nastavenia zabezpečujú, že Unity použije správnu natívnu knižnicu pre každú cieľovú platformu a predchádzajú chybám typu “`DllNotFoundException`” pri spustení aplikácie.

Návrh databázovej schémy a implementácia

Prvým krokom implementácie bolo vytvorenie databázovej schémy, ktorá by dokázala uchovať všetky potrebné informácie o pacientoch a ich nastaveniach. Databázová schéma bola navrhnutá pomocou online nástroja Miro³, ktorý umožňuje kolaboratívne vytváranie diagramov. Na základe tejto schémy boli následne vytvorené modely dát v C# a implementovaný `DatabaseManager`, ktorý zabezpečuje všetky operácie s databázou.

Inicializácia databázy na serveri

Databázový manager bude inicializovaný výlučne na serverovej strane aplikácie. Táto architektúra zabezpečuje, že databáza slúži ako centrálné úložisko pre všetkých klientov pripojených k danému serveru, čím sa predchádza problémom so synchronizáciou dát.

Inicializácia `DatabaseManager` sa vykonáva pri štarte servera prostredníctvom statickej metódy označenej atribútom `[Server]`, ktorý zabezpečuje, že daný kód sa vykoná iba na serverovej strane:

³<https://miro.com>

Výpis 4.11: Serverová inicializácia DatabaseManager

```
[Server]
public static void ServerInitialize(NetworkSettings networkSettings)
{
    if (_instance != null)
    {
        return;
    }

    GameObject go = new GameObject("DatabaseManager");
    _instance = go.AddComponent<DatabaseManager>();
    DontDestroyOnLoad(go);
    _instance._networkSettings = networkSettings;
    _instance.InitializeServerDatabase();
}

private void InitializeServerDatabase()
{
    _databasePath = Path.Combine(
        Application.persistentDataPath, "neurorehab.db");

    _database = new SQLiteConnection(_databasePath, true);

    CreateTables();
}
```

Metóda `CreateTables()` vytvára databázové tabuľky na základe C# tried označených SQLite atribútmi. Databázový súbor `neurorehab.db` je vytvorený v štandardnom adresári pre perzistentné dáta aplikácie:

```
%USERPROFILE%/AppData/LocalLow/<CompanyName>/<ProductName>/neurorehab.db
```

`DatabaseManager` poskytuje metódy pre všetky CRUD (Create, Read, Update, Delete) operácie s databázovými tabuľkami. Správa databázy zahŕňa vytváranie a aktualizáciu záznamov, vyhľadávanie podľa rôznych kritérií a mazanie neplatných záznamov.

Testovanie funkcionality databázy

Po úspešnej integrácii pluginu bolo vykonané testovanie funkcionality databázy s reálnymi dátami z aplikácie. Na používateľskom rozhraní bola pridaná tlačidlá pre ukladanie nastavení, ktoré aktivovali zápis niektorých aktuálnych hodnôt z tabov do databázy.

Testovanie prebiehalo v Unity Editore a potvrdilo, že databáza funguje správne. Nastavenia boli úspešne ukladané po stlačení tlačidla “Save” a následne mohli byť načítané späť do aplikácie. Toto overenie bolo kritické pre ďalší vývoj, keďže potvrdilo funkčnosť celého databázového riešenia.

Prvá implementácia používateľského rozhrania

Pre počiatočné testovanie bolo vytvorené jednoduché používateľské rozhranie s minimálnou funkcionalitou. Rozhranie obsahovalo vstupné pole, kde terapeut mohol zadať meno profilu pacienta, tlačidlo “Uložiť” pre vytvorenie nového profilu s aktuálnymi nastaveniami pod zadaným menom, a tlačidlo “Načítať” pre obnovenie uloženého profilu podľa mena.

Táto implementácia slúžila primárne ako proof of concept pre overenie, že databázové operácie fungujú správne v kontexte Unity aplikácie. Avšak v súlade s pôvodnou koncepciou malo finálne riešenie fungovať inak, terapeut mal mať možnosť vytvoriť viacero profilov pacientov vopred, a pri pripojení pacienta k sedeniu vybrať príslušný profil zo zoznamu a aplikovať ho na pripojeného používateľa.

Počas analýzy tejto koncepcie sa však ukázalo, že pracovný tok je príliš komplikovaný a nepraktický. Terapeut by musel prejsť viacerými zbytočnými krokmi. Toto pridávalo zbytočnú komplexitu do systému, ktorý mal byť jednoduchý a intuitívny pre terapeutov.

4.3.2 Zjednodušený model profilu pacienta pre miestnosť

Problémy pôvodného návrhu

Počas analýzy pôvodnej koncepcie sa ukázalo, že navrhnuté riešenie je príliš komplikované pre praktické použitie a pre implementáciu. Pôvodný návrh vyžadoval od terapeuta vykonanie viacerých krokov. Bolo potrebné nájsť riešenie, ktoré by eliminovalo tieto zbytočné kroky a zjednodušilo použitie systému pre obe strany.

Nová koncepcia systému

Na základe týchto zistení bola koncepcia radikálne zjednodušená. Nový model vychádza z princípu: jedna terapeutická miestnosť = jeden profil pacienta. Tento prístup reflektuje reálny pracovný tok v rehabilitačnom zariadení, kde je konkrétna miestnosť priradená konkrétnemu pacientovi na určité obdobie.

Nová koncepcia:

1. Terapeut a pacient sa pripoja k vybranej miestnosti
2. Terapeut nakonfiguruje všetky potrebné nastavenia
3. Terapeut stlačí tlačidlo “Uložiť”
4. Pri ďalšom pripojení pacienta sa nastavenia automaticky načítajú

Nové riešenie eliminuje:

- Správu viacerých profilov
- Vyhľadávanie v zoznamoch profilov
- Manuálnu aplikáciu profilov
- Potrebu pamätať si mená profilov

Systém sa stáva intuitívnym. Terapeut jednoducho nastaví potrebné parametre a uloží ich. Pri ďalšom pripojení pacienta k tej istej miestnosti sa všetko automaticky obnoví presne tak, ako bolo naposledy uložené.

Prepracovaná databázová schéma

Na základe novej koncepcie bola databázová schéma upravená. Finálna schéma obsahuje dve hlavné tabuľky s jednoznačným vzťahom medzi nimi (obr. 4.4):

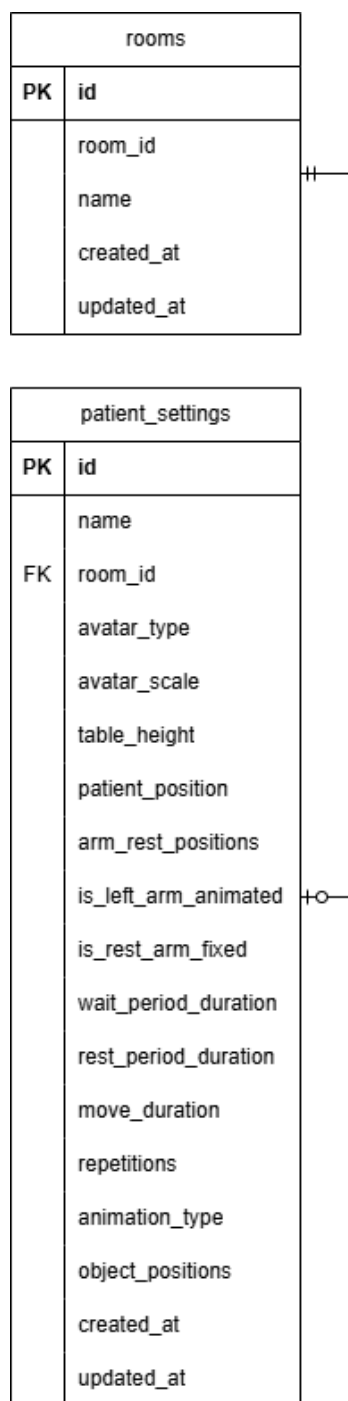
Tabuľka `rooms` obsahuje zoznam terapeutických miestností v systéme. Tabuľka `patient_settings` obsahuje všetky nastavenia pacienta spojené s konkrétnou miestnosťou. `SQLite4Unity3d` automaticky mapuje vlastnosti tried na stĺpce tabuliek na základe atribútov:

Kľúč `RoomId` slúži na vytvorenie vzťahu medzi tabuľkami a zabezpečuje, že každý záznam v tabuľke `patient_settings` je viazaný na presne jednu miestnosť v tabuľke `rooms`. Vzťah medzi tabuľkami je typu jeden k nula alebo jednej, keďže miestnosť môže existovať bez profilu pacienta, ktorý sa vytvorí až pri prvom uložení nastavení terapeutom.

Integrácia správy miestností s databázou

Dôležitou zmenou bolo prepojenie systému správy miestností s databázou. V pôvodnej implementácii boli miestnosti definované staticky v kóde a nebolo možné ich jednoducho spravovať. Po integrácii databázy sa miestnosti stali dynamickými. Pri prvom spustení servera na zariadení sa databáza inicializuje a naplní zoznamom miestností z objektu `NetworkSettings`, ktorý je nakonfigurovaný v build. Ak tento zoznam nie je k dispozícii, použije sa záložný pevne zadaný zoznam predvolených miestností.

Trieda `NetworkSettings` po inicializácii databázy pri každom štarte servera načíta miestnosti z databázy a poskytuje ich klientom cez REST API endpoint. Táto zmena prináša výhodu, že správa miestností je teraz centralizovaná v databáze a je možné do nej vložiť vlastné údaje.



Obrázok 4.4: Finálna databázová schéma systému

Výpis 4.12: Načítanie miestností z databázy pri štarte servera

```

[Server]
private void LoadRoomsFromDatabase()
{
    List<Room> roomsFromDb = DatabaseManager.Instance.GetAllRooms();

    if (roomsFromDb == null || roomsFromDb.Count == 0)
    {
        return;
    }
}
    
```

```

List<RoomDataSerialized> roomsSerialized = new List<
    RoomDataSerialized>();
foreach (var room in roomsFromDb)
{
    roomsSerialized.Add(new RoomDataSerialized
    {
        roomID = room.RoomId,
        roomName = room.Name,
        userCount = 0
    });
}

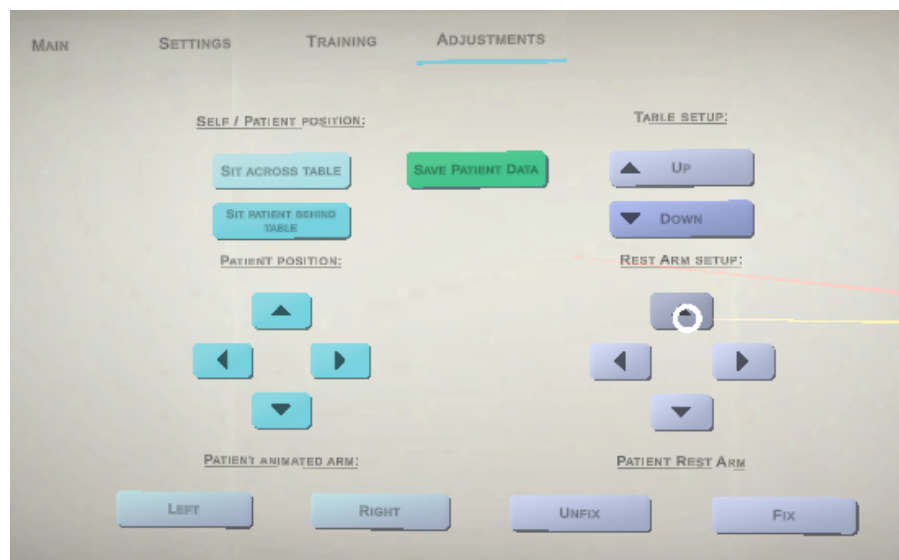
networkSettings.SetRooms(roomsSerialized);
}

```

4.3.3 Ukladanie nastavení pacienta

Používateľské rozhranie pre ukladanie

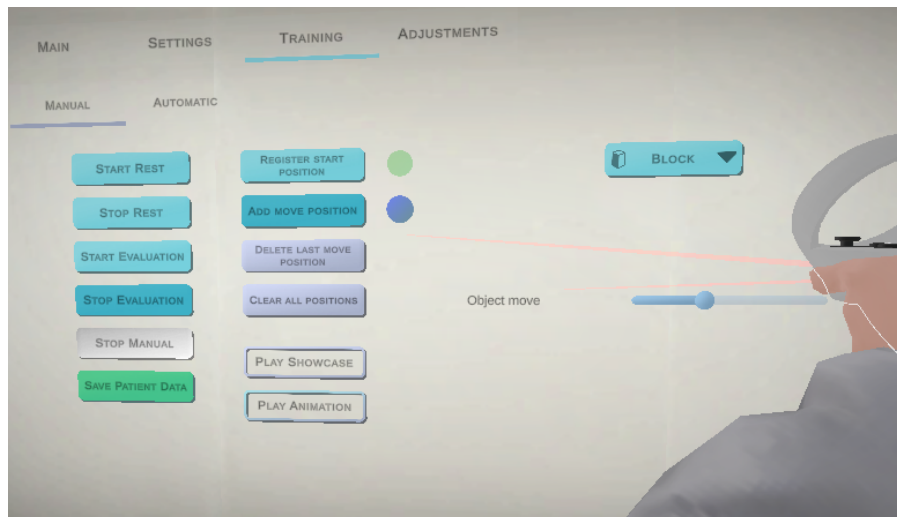
V aplikácii existujú dva hlavné taby s nastaveniami, ktoré je potrebné uložiť: tab “Adjustments” obsahujúci nastavenia pacienta a pozícií, a tab “Training” obsahujúci tréningové parametre. Do oboch tabov boli pridané tlačidlá “Save Patient Data”, ktoré aktivujú uloženie aktuálnych hodnôt do databázy (obr. 4.5 a 4.6).



Obrázok 4.5: Tab Adjustments s tlačidlom pre uloženie nastavení pacienta

Tab “Adjustments” umožňuje uložiť nasledujúce nastavenia:

- Výška stola (TableHeight)
- Pozícia pacienta (PatientPosition)
- Poloha druhej ruky v pokoji (ArmRestPosition)



Obrázok 4.6: Tab Training s tlačidlom pre uloženie tréningových nastavení

- Typ a mierka avatara (AvatarType, AvatarScale)
- Nastavenia animovanej ruky (IsLeftArmAnimated, IsRestArmFixed)

Tab „Training“ obsahuje dva podtaby „Manual“ a „Automatic“, ktoré poskytujú rôzne prístupy k nastaveniu tréningových cvičení. Nastavenia z oboch podtabov sa ukladajú spoločne do jedného profilu. Pri stlačení tlačidla “Save Patient Data” sa uložia nasledujúce tréningové parametre:

- Trvanie periódy čakania (WaitPeriodDuration)
- Trvanie periódy odpočinku (RestPeriodDuration)
- Trvanie pohybu (MoveDuration)
- Počet opakovaní (Repetitions)
- Typ animácie/cvičenia (AnimationType)
- Pozície objektov pre jednotlivé typy cvičení (ObjectPositions)

Všetky pridané textové prvky používateľského rozhrania, ako sú nadpisy a popisy tlačidiel, boli lokalizované rovnako ako ostatné.

Architektúra ukladania cez Mirror Networking

Pre komunikáciu medzi klientom, kde terapeut stláča tlačidlo, a serverom, kde sa nachádza databáza, sa používa Mirror Networking framework, ktorý poskytuje atribúty [Command] a [Server] pre zabezpečenie správnej distribúcie kódu [31].

Atribút [Command] označuje metódu, ktorá je volaná na klientovi, ale vykonávaná na serveri. Metódy s týmto atribútom môžu byť predvolene volané iba z objektu lokálneho hráča.

Atribút [Server] označuje metódu alebo časť kódu, ktorá sa vykoná výlučne na serverovej strane. Kód označený týmto atribútom sa nevykoná na klientských inštanciách aplikácie, čo zabezpečuje, že citlivé operácie vykonávané iba na serveri.

Príkaz `CmdSavePatientSettings` je volaný na klientovi a vykonaný na serveri. Ak pre danú miestnosť ešte neexistuje profil pacienta, vytvorí sa nový záznam:

Výpis 4.13: Ukladanie nastavení pacienta na serveri

```
[Command]
public void CmdSavePatientSettings(string roomId, PatientSettingsData data)
{
    Room room = DatabaseManager.Instance.GetRoomByRoomId(roomId);
    if (room == null) return;

    PatientSettings patient =
        DatabaseManager.Instance.GetPatientByRoomId(room.Id);

    if (patient == null)
    {
        patient = new PatientSettings
        {
            Name = "Patient",
            RoomId = room.Id
        };
    }

    patient.TableHeight = data.TableHeight;
    // ... ostatné polia
    DatabaseManager.Instance.SavePatient(patient);
}
```

Podobný princíp je použitý aj pre ukladanie tréningových nastavení, kde metóda `CmdSaveTrainingSettings` zbiera parametre z `AnimationSettings` a pozície objektov zo všetkých tréningových setupov.

Serializácia pozičných dát

Významným technickým problémom bola serializácia 3D pozícií objektov pre uloženie do databázy vo formáte JSON. Unity používa `Vector3` štruktúru pre reprezentáciu pozícií, ktorú je potrebné previesť do textového formátu.

Kritickou záležitosťou pri serializácii číselných hodnôt je použitie správnej kultúry pre formátovanie. Rôzne krajiny používajú rôzne formáty čísel, napríklad

klad v slovenčine sa desatinná čiarka píše ako “0,5”, zatiaľ čo v anglickom formáte je to “0.5”. `CultureInfo.InvariantCulture` je potrebné použiť pri ukladaní údajov do databázy, aby sa zabezpečil konzistentný formát nezávislý od systémového nastavenia.

Výpis 4.14: Serializácia `Vector3` do JSON s `InvariantCulture`

```
private string FormatVector3ToJson(Vector3 position)
{
    return string.Format(CultureInfo.InvariantCulture,
        "{{\"x\":{0},\"y\":{1},\"z\":{2}}}",
        position.x, position.y, position.z);
}
```

Bez použitia `InvariantCulture` by mohlo dôjsť k situácii, kde pozícia uložená na systéme s jednou lokalizáciou (napríklad “0,5”) by nemohla byť správne načítaná na systéme s inou lokalizáciou, čo by viedlo k chybám pri parsovaní JSON dát.

4.3.4 Automatické načítanie nastavení

Načítanie pri pripojení pacienta k miestnosti

Kľúčovou funkcionalitou systému je automatické obnovenie uložených nastavení pri pripojení pacienta k miestnosti. Tento proces sa vykonáva na serverovej strane v rámci metódy spracovania pripojenia nového používateľa v triede `CustomNetworkManager`.

Keď sa pacient pripojí k miestnosti, server najprv overí, či pre danú miestnosť existujú uložené nastavenia v databáze. Ak áno, systém postupne aplikuje tieto nastavenia na pripojeného pacienta:

Výpis 4.15: Načítanie a aplikácia nastavení pri pripojení pacienta

```
[Server]
private void ApplyPatientPositionsFromDatabase(
    CharacterManager patient, string roomId, Scene roomScene)
{
    Room room = DatabaseManager.Instance.GetRoomByRoomId(roomId);
    if (room == null)
    {
        return;
    }

    PatientSettings settings =
        DatabaseManager.Instance.GetPatientByRoomId(room.Id);

    if (settings == null)
    {
        return;
    }
}
```

```

INetworkCharacter netChar = patient.GetComponent<INetworkCharacter>
    >();
NetworkCharacterManager networkChar = netChar as
    NetworkCharacterManager;

if (settings.AvatarScale > 0)
{
    StartCoroutine(WaitInitAndApplyPositions(
        networkChar, patient, settings, table));
}

if (settings.Repetitions > 0)
{
    AnimationSettingsManager animSettings =
        sceneObjectManager.AnimSettings as AnimationSettingsManager
        ;

    if (animSettings != null)
    {
        AnimationType animType = ParseAnimationType(
            settings.AnimationType);

        StartCoroutine(WaitAndApplyTrainingSettings(
            networkChar, animSettings, settings, animType));
    }
}
}

```

Dôležitým aspektom je validácia načítaných dát. Ak niektoré hodnoty chýbajú alebo sú neplatné (napríklad `AvatarScale` je 0 alebo záporná hodnota), systém ich ignoruje a použije predvolené hodnoty. Toto zabezpečuje, že aplikácia funguje správne aj v prípade čiastočne poškodených alebo neúplných dát v databáze.

Riešenie timing problémov pri načítavaní

Významným technickým problémom bolo časovanie aplikácie nastavení. Nastavenia nie je možné aplikovať okamžite po pripojení pacienta, pretože niektoré komponenty Unity ešte nemusia byť plne inicializované. Napríklad pozíciu pacienta nie je možné nastaviť, kým nie je inicializovaný `XROrigIn` komponent a kamera pacienta.

Pre riešenie tohto problému boli použité Unity coroutines, ktoré umožňujú čakať na splnenie určitých podmienok pred vykonaním operácie:

Výpis 4.16: Čakanie na inicializáciu pred aplikáciou pozícií

```

private IEnumerator WaitInitAndApplyPositions(
    NetworkCharacterManager networkChar,
    CharacterManager patient,
    PatientSettings settings,
    TableHelper table)

```

```

{
    // Čakanie na inicializáciu NetworkCharacterManager
    yield return new WaitUntil(() => networkChar.IsInitialized);

    // Aplikácia vysky stola
    float currentTableHeight = table.transform.position.y;
    float tableHeightOffset = settings.TableHeight - currentTableHeight↵
    ;

    if (Mathf.Abs(tableHeightOffset) > 0.001f)
    {
        networkChar.ServerMoveTable(new Vector3(0, tableHeightOffset, ↵
        0));
    }

    // Čakanie na pripojenie klienta
    yield return new WaitUntil(() => patient.connectionToClient != null↵
    );

    yield return new WaitUntil(() => patient.LocalCamera != null);

    // Aplikácia pozície pacienta
    if (!string.IsNullOrEmpty(settings.PatientPosition))
    {
        Vector3 patientCameraPos = ParseVector3FromJson(
            settings.PatientPosition);

        if (patientCameraPos != Vector3.zero)
        {
            Vector3 currentCameraPos =
                patient.LocalCamera.transform.position;
            Vector3 patientOffset = patientCameraPos - currentCameraPos↵
            ;

            if (patientOffset.magnitude > 0.001f)
            {
                networkChar.ServerMovePatient(patientOffset, patient.Id↵
                );
            }
        }
    }
}

```

Tento prístup zabezpečuje, že každá operácia je vykonaná až v momente, keď sú všetky potrebné komponenty pripravené. Použitie `WaitUntil` s lambda výrazom umožňuje flexibilné definovanie podmienok čakania.

Rovnaký prístup bol použitý aj pre aplikáciu tréningových nastavení, kde sa čaká na plnú inicializáciu `AnimationSettingsManager` a pripravenosť klienta na prijímanie RPC volaní.

Správne použitie Server metód

Pre aplikáciu pozičných zmien bolo potrebné vytvoriť serverové verzie existujúcich metód. Pôvodné implementácie obsahovali iba `[Command]` metódy, ktoré

vyžadovali volanie z klienta. Pre automatické načítanie z databázy však bolo potrebné volať tieto metódy priamo na serveri.

Riešením bolo vytvorenie samostatných [Server] metód, ktoré obsahujú samotnú logiku, a ponechanie [Command] metód ako wrapperov:

Výpis 4.17: Oddelenie Server a Command metód

```
[Server]
public void ServerMoveTable(Vector3 offset, NetworkConnectionToClient
    sender = null)
{
    // logika pohybu stola a synchronizacie pozicii objektov
}

[Command]
public void CmdMoveTable(Vector3 offset, NetworkConnectionToClient
    sender = null)
{
    ServerMoveTable(offset, sender);
}
```

Tento pattern umožňuje použiť rovnakú logiku pri manuálnom nastavení terapeuta aj pri automatickom načítaní z databázy.

Aktualizácia používateľského rozhrania

Dôležitým aspektom bolo zabezpečenie, že používateľské rozhranie terapeuta zobrazuje správne hodnoty okamžite po načítaní nastavení z databázy. V pôvodnej implementácii sa UI prvky (slidery, textové polia) aktualizovali iba pri aktualizácii tabu.

Riešenie zahŕňalo prepojenie načítania hodnôt s Mirror Networking **SyncVar** hooks. Keď server nastaví novú hodnotu do SyncVar, automaticky sa zavolá príslušný hook na všetkých klientoch, ktorý aktualizuje UI:

Výpis 4.18: Automatická aktualizácia UI cez SyncVar hooks

```
[SyncVar(hook = nameof(ChangeRepetitionsHook))]
private int net_repetitions;

private void ChangeRepetitionsHook(int _old, int _new)
{
    animSettings.Repetitions = _new;
    // UI slider sa automaticky aktualizuje cez data binding
}

[Server]
public void ServerLoadTrainingSettings(
    float evalDuration, float restDuration,
    float moveDuration, int repetitions,
    AnimationType animType)
{
    net_evalDuration = evalDuration;
```

```

net_restDuration = restDuration;
net_moveDuration = moveDuration;
net_repetitions = repetitions;
AnimationType oldAnimType = net_animType;
if (SetAnimType(oldAnimType, animType))
{
    SpawnTarget(oldAnimType, animType);
}
}

```

Tento mechanizmus zabezpečuje, že terapeut vidí aktuálne hodnoty načítané z databázy okamžite po pripojení pacienta. Ak sa typ cvičenia v scéne líši od typu uloženého v databáze, je potrebné tiež vyvolať `SpawnTarget`, ktorý aktualizuje objekty v scéne podľa nového typu.

Štruktúra JSON dát pre pozície objektov

Pozície objektov pre rôzne typy cvičení sú uložené v databáze ako JSON string v stĺpci `ObjectPositions`. Tento string obsahuje pozície pre všetky typy cvičení (`Block`, `Cube`, `Cup`, `Key`), pričom každý typ má pole `Vector3` pozícií:

Výpis 4.19: Príklad JSON štruktúry `ObjectPositions`

```

{
  "Block": [
    {"x":0.5,"y":0.1,"z":0.3},
    {"x":0.7,"y":0.1,"z":0.3}
  ],
  "Cube": [
    {"x":0.2,"y":0.15,"z":0.4}
  ],
  "Cup": [
    {"x":0.4,"y":0.1,"z":0.1}
  ],
  "Key": [
    {"x":0.0,"y":0.1,"z":0.2}
  ]
}

```

Deserializácia týchto dát predstavovala technickú výzvu, keďže štandardné Unity JSON nástroje (`JsonUtility`) nepodporujú dynamické štruktúry s rôznymi typmi objektov. Riešenie zahŕňalo manuálny parsing JSON stringu.

Validácia a fallback mechanizmy

Systém obsahuje viacero úrovní validácie a fallback mechanizmov, ktoré zabezpečujú stabilitu aj pri neúplných alebo poškodených dátach. V prvom rade sa overuje existencia miestnosti v databáze, pričom pri jej absencii sa automaticky použijú predvolené nastavenia. Následne systém kontroluje, či existuje profil pacienta pre danú miestnosť, a v prípade jeho neprítomnosti pokračuje s predvolenými

hodnotami. Dodatočne prebieha validácia jednotlivých parametrov, čím sa zabezpečuje konzistentnosť a správnosť všetkých hodnôt, napríklad pomocou podmienok `AvatarScale > 0` a `!string.IsNullOrEmpty(PatientPosition)`. Tento mnohostupňový validačný prístup zabezpečuje, že aplikácia nikdy nezahavaruje kvôli chybným dátam v databáze, ale vždy sa pokúsi použiť čo najviac platných informácií a doplniť chýbajúce hodnoty predvolenými nastaveniami.

4.3.5 Testovanie a oprava chýb vo VR prostredí

Testovanie aplikácie pomocou helmy virtuálnej reality je kľúčovým krokom na zabezpečenie spoľahlivosti a správneho fungovania systému. Simulácia VR na počítači neumožňuje úplne reprodukovat podmienky skutočného používania, preto je na získanie presných a spoľahlivých výsledkov potrebné otestovať aplikáciu na skutočnom VR zariadení.

Testovanie systému prebiehalo spoločne s vedúcim diplomovej práce v VR prostredí pomocou Oculus Quest 2 zariadení. Počas testovania bolo identifikovaných niekoľko problémov, ktoré vyžadovali okamžité riešenie.

Odstránenie načítavania typu avatara z databázy

Pôvodná implementácia ukladala typ avatara do databázy vo forme celého čísla, kde ženské avatary mali offset 50 (napríklad hodnota 51 znamenala ženský avatar typu 1). Problém spočíval v tom, že typ avatara sa určuje v ponuke ešte pred pripojením klienta k serveru, zatiaľ čo načítanie údajov z databázy prebieha priamo v momente pripojenia. V dôsledku toho sa voľba používateľa nezohľadňuje, pretože pri pripojení sa vždy použije avatar uložený v databáze. Po prvom uložení už nie je možné ho zmeniť.

Riešenie spočívalo v odstránení načítavania typu avatara z databázy a použití výlučne hodnôt z `CharacterMessage`, ktoré klient posiela pri pripojení.

Doplnenie parametra `MiThreshold` a serializácie dát miestnosti

Na základe požiadavky Ústavu merania Slovenskej akadémie vied bol do modelu `PatientSettings` doplnený parameter `MiThreshold`. Parameter je ukladaný a načítavaný rovnakým spôsobom ako ostatné tréningové nastavenia a po načítaní sa aplikuje na príslušný slider v rozhraní terapeuta.

Zároveň bola do triedy `DatabaseManager` pridaná metóda `GetRoomDataAsJson`, ktorá serializuje kompletné dáta miestnosti do JSON. Táto metóda je

využívaná pri **WebSocket** komunikácii, konkrétne pri pripojení EEG klienta, keď je výsledný JSON odoslaný v potvrdení pripojenia (`client.checkin.ack`).

Pridanie podpory pre rotáciu pacienta

Na zabezpečenie kompletnej perzistencie pozície pacienta bolo potrebné okrem pozície ukladať aj rotáciu. Do triedy `PatientSettings` bol pridaný nový stĺpec `PatientRotation` pre uloženie Y-osovej rotácie pacienta. Pri ukladaní nastavení sa teraz zaznamenáva aktuálna Y-osová rotácia XR Origin objektu.

Trieda `NetworkCharacterManager` obsahuje novú sadu metód určených na aplikáciu rotácie:

Výpis 4.20: Metódy pre rotáciu pacienta

```
[Server]
public void ServerRotatePatientWorldSpace(float yRotation, uint
patientId)
{
    if (objectProvider.TryGetObject(patientId, out GameObject
patientObject))
    {
        if (patientObject.TryGetComponent(out NetworkIdentity
patientIdentity))
        {
            TargetRotatePatientWorldSpace(
                patientIdentity.connectionToClient, yRotation);
        }
    }
}

[TargetRpc]
protected void TargetRotatePatientWorldSpace(
    NetworkConnection connection, float targetYRotation)
{
    if (objectProvider == null || sceneObjectManager?.LocalChar == null)
    {
        StartCoroutine(WaitForLocalCharAndRotateWorldSpace(
            targetYRotation));
        return;
    }

    PerformPatientRotateWorldSpace(targetYRotation);
}
```

`TargetRpc` metóda zabezpečuje správne časovanie aplikácie rotácie s čakáním na inicializáciu lokálneho charakteru. Samotné vykonanie rotácie využíva metódu `RotateAroundCameraUsingOriginUp` pre XR Origin s použitím `Mathf.DeltaAngle` pre správny výpočet rotačného rozdielu:

Výpis 4.21: Vykonanie rotácie pacienta

```
private void PerformPatientRotateWorldSpace(float targetYRotation)
```

```

{
    GameObject localCharObj = objectProvider.GetObject(
        sceneObjectManager.LocalChar.Id);

    if (localCharObj.TryGetComponent(out XROrigin xrOrigin))
    {
        float currentYRotation = xrOrigin.transform.eulerAngles.y;
        float rotationDiff = Mathf.DeltaAngle(currentYRotation, ↩
            targetYRotation);

        xrOrigin.RotateAroundCameraUsingOriginUp(rotationDiff);
    }
    else
    {
        Vector3 currentRotation = localCharObj.transform.eulerAngles;
        localCharObj.transform.eulerAngles = new Vector3(
            currentRotation.x, targetYRotation, currentRotation.z);
    }
}

```

Aplikácia rotácie v CustomNetworkManager sa vykonáva po nastavení pozície s validáciou rozsahu:

Výpis 4.22: Aplikácia rotácie z databázy

```

if (!string.IsNullOrEmpty(settings.PatientRotation))
{
    float targetYRotation = float.Parse(settings.PatientRotation,
        System.Globalization.CultureInfo.InvariantCulture);

    if (targetYRotation >= 0f && targetYRotation < 360f)
    {
        float currentYRotation = patient.transform.eulerAngles.y;

        if (Mathf.Abs(Mathf.DeltaAngle(currentYRotation, ↩
            targetYRotation)) > 0.1f)
        {
            networkChar.ServerRotatePatientWorldSpace(targetYRotation, ↩
                patient.Id);
        }
    }
}

```

Oprava nesprávneho načítania pozície pacienta

Kritický problém bol identifikovaný pri načítavaní pozície pacienta z databázy na Oculus zariadeniach. Pacient sa vždy objavil o jeden krok dopredu od želanej pozície.

Analýza problému odhalila, že vo VR prostredí nie je garantované, že pacient vždy začína na rovnakej pozícii. Sledovací systém VR môže inicializovať kameru na rôznych pozíciách v závislosti od kalibrácie headsetu a nastavení ochrannej zóny. Preto bolo potrebné implementovať riešenie, ktoré používa absolútne svetové súradnice namiesto relatívnych offsetov.

V triede `NetworkCharacterManager` boli zavedené dve sady metód, z ktorých jedna spracováva absolútne pozície a druhá relatívne offsety.

Kľúčová zmena je v samotnej aplikácii pozície, kde sa namiesto výpočtu offsetu priamo volá `MoveCameraToWorldLocation` na `XROrigin`, čo zaručuje správne umiestnenie bez ohľadu na štartovaciu pozíciu pacienta. Na tento účel bola vytvorená metóda `PerformPatientSetAbsoluteWorldPosition`, ktorú server zavolá na strane konkrétneho klienta cez `TargetRpc`.

Tento prístup zabezpečuje konzistentné správanie naprieč rôznymi VR zariadeniami a konfiguráciami, pretože sa nezakladá na predpokladoch o štartovej pozícii pacienta. Metóda pre relatívne posuny zostáva zachovaná pre potreby terapeutov pri manuálnom nastavovaní polohy pacienta počas sedenia.

Oprava nesprávnej počiatočnej pozície pacienta pri pripojení

Pri testovaní na VR zariadení bol zaznamenaný prípad, keď sa pacient po načítaní pozície z databázy objavil pod podlahou scény. Príčina spočívala v tom, že pri ukladaní sa zaznamenávala aj Y-ová súradnica kamery, ktorá v sebe zahŕňa fyzické posunutie hlavy používateľa od podlahy, ako napríklad výška používateľa alebo kalibrácia sledovacieho systému. Toto posunutie sa medzi sedeniami mení, preto uložená Y-ová hodnota pri obnovení spôsobovala nesprávne umiestnenie `XROrigin` pod úroveň podlahy.

Riešením je pri načítaní ignorovať uloženú Y-ovú súradnicu a nahradiť ju aktuálnou hodnotou z živého XR trackingu. Keďže tracking sa inicializuje s oneskorením, bolo pridané čakanie na stabilizáciu polohy kamery pred aplikáciou pozície:

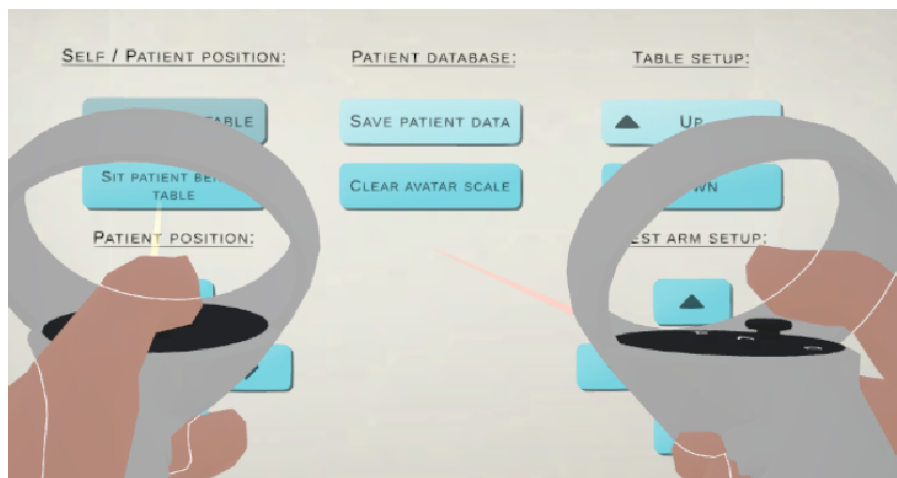
Výpis 4.23: Čakanie na inicializáciu XR trackingu pred aplikáciou pozície

```
yield return new WaitUntil(() =>
    patient.LocalCamera.transform.position.y > 0.1f);
patientCameraPos.y = patient.LocalCamera.transform.position.y;
networkChar.ServerSetPatientAbsoluteWorldPosition(
    patientCameraPos, patient.Id);
```

Pridanie tlačidla pre reset mierky avatara

Počas testovania bolo zistené, že mierku avatara (`AvatarScale`) je možné nastaviť iba v offline scéne, avšak pri pripojení k serveru sa vždy načíta hodnota uložená v databáze. Po prvom uložení teda nebolo možné hodnotu zmeniť inak ako úpravou databázy mimo aplikácie. Pre riešenie tohto problému bolo do rozhrania terapeuta pridané tlačidlo pre reset mierky avatara. Po jeho stlačení sa vykoná príkaz `CmdClearAvatarScale`, ktorý vymaže uloženú hodnotu

mierky avatara z databázy, čím sa pri ďalšom pripojení použije aktuálna hodnota nastavená v offline scéne (obr. 4.7).



Obrázok 4.7: Tlačidlo pre reset mierky avatara v rozhraní terapeuta

Oprava synchronizácie objektov pri zmene výšky stola

Pri testovaní bolo zistené, že keď terapeut zmenil výšku stola, tréningové objekty sa posunuli iba na strane pacienta, nie u terapeuta. To spôsobovalo situáciu, že pri načítaní nastavení z databázy mohol byť objekt uprostred stola z pohľadu terapeuta, zatiaľ čo pre pacienta bol na správnej výške.

Príčina problému spočívala v tom, že sa používala `TargetRpc` metóda, ktorá posielala zmeny iba jednému klientovi (pacientovi). Riešenie zahŕňalo zmenu na `ClientRpc`, ktorá synchronizuje zmeny naprieč všetkými pripojenými klientmi.

Implementácia novej `RpcMoveObjects` metódy:

Výpis 4.24: `ClientRpc` metóda pre synchronizáciu objektov

```
[ClientRpc]
public void RpcMoveObjects(Vector3 offset)
{
    if (objectProvider == null || animSettingsManager == null)
    {
        StartCoroutine(WaitAndMoveObjects(offset));
        return;
    }

    PerformMoveObjects(offset);
}
```

Táto zmena zabezpečuje, že všetci klienti, terapeut aj pacient,vidia objekty na rovnakých pozíciách po zmene výšky stola alebo pri načítaní nastavení z databázy.

Oprava výnimiek pri pridávaní pozícií objektom

Počas testovania sa zistilo, že pri pridávaní položiek k objektom cvičení dochádzalo k výnimkám `NullPointerException` a pacient sa odpojil od servera.

Analýza problému odhalila, že v triede `AnimationSettingsManager` sa pristupovalo k objektom bez dostatočných validácií. Predpokladalo sa, že komponenty `sender.identity` a `NetworkCharacterManager` sú vždy k dispozícii pri validácii pozícií a zasielaní varovaní klientom, čo nebolo vždy pravdou.

Riešenie spočívalo v pridaní explicitných null kontrol pred pristupovaním k týmto objektom v metóde `IsTargetInBounds`.

Následné testovanie potvrdilo, že pozície sa úspešne pridávajú, ukladajú do databázy a načítavajú pri ďalšom pripojení pacienta. Obrázok 4.8 zobrazuje úspešné načítanie všetkých pozícií.



Obrázok 4.8: Úspešné načítanie pozícií objektov z databázy pri pripojení pacienta

Poznámka o probléme s výškou pacienta

Počas prvotného testovania bol zaznamenaný problém, kde sa pacient načítal mierne nižšie, než bolo očakávané, a videl seba s pokrčenými kolenami. Tento problém však nebol reprodukovateľný pri ďalších testovaniach. Je možné, že problém súvisel s dočasnou desynchronizáciou VR sledovacieho systému alebo s jedinečnou konfiguráciou počas konkrétneho testovacieho sedenia. Keďže sa problém neopakoval, nebolo potrebné implementovať špecifické riešenie.

Oprava inicializácie databázy na čistých inštaláciách

Pri testovaní na nových zariadeniach bez predchádzajúcej inštalácie vývojových nástrojov bolo zistené, že databáza sa na serveri neinicializuje. Príčinou bola za-

staraná verzia natívnej knižnice `sqlite3.dll`, ktorá vyžaduje prítomnosť balíka Visual C++ Redistributable. Na čistých inštaláciách systému Windows tento balík nie je štandardne prítomný, čo spôsobovalo zlyhanie pri načítaní knižnice.

Riešením bola náhrada knižnice za aktuálnu predkompilovanú verziu stiahnutú priamo zo stránky SQLite. Nová verzia knižnice nevyžaduje dodatočné systémové závislosti. Po výmene knižnice bola databáza úspešne inicializovaná na všetkých testovaných zariadeniach.

Riešenie problému s neuloženými nastaveniami na VR zariadeniach

Ďalší problém bol identifikovaný pri testovaní na VR zariadeniach. Ak používateľ pridal nový server do zoznamu serverov v menu výberu servera a následne sa pripojil k tomuto serveru, ale potom aplikáciu ukončil priamym zatvorením (bez použitia tlačidla “Disconnect” a následného návratu do hlavného menu), nastavenia neboli uložené. Pri ďalšom spustení aplikácie pridaný server chýbal v zozname.

Problém spočíval v tom, že perzistencia nastavení do súboru sa vykonávala výlučne v metóde `OnApplicationQuit()`. Na VR zariadeniach, ak aplikácia crashne alebo je násilne ukončená napríklad cez systémové menu VR, metóda `OnApplicationQuit()` nemusí byť zavolaná. To znamená, že všetky zmeny vykonané počas aktuálnej relácie vrátane pridania nového servera sa stratia.

Riešenie spočívalo v zavedení proaktívneho ukladania nastavení pri každom pripojení k serveru, namiesto ich ukladania iba pri ukončení aplikácie. V metóde `JoinClient()` triedy `CustomNetworkManager` bolo pridané volanie `SettingManager.Instance.SaveSettings()` bezprostredne po spustení klienta.

Toto riešenie zabezpečuje, že aj v prípade neočakávaného ukončenia aplikácie sú nastavenia uložené najneskôr v momente pripojenia k serveru. Pridaný server je teda perzistentný aj vtedy, keď sa aplikácia na VR zariadení ukončí predčasne.

4.3.6 Zhrnutie implementovaných vylepšení

Cieľom tejto časti vývoja bolo implementovať systém profilov pacientov, ktorý by umožnil terapeutom ukladať a automaticky načítavať nastavenia pre jednotlivých pacientov. Pred implementáciou musel terapeut pri každom novom sedení manuálne nastavovať výšku stola, pozíciu pacienta, pozície objektov a všetky tréningové parametre.

Implementovali sme databázový systém s SQLite, ktorý pre každého pacienta v danej miestnosti ukladá jeho kompletné nastavenia: výšku stola, pozíciu a rotá-

ciu pacienta, pozície objektov pre cvičenia a nastavenia animácií. Keď sa pacient pripojí, systém automaticky načíta jeho profil a aplikuje všetky uložené hodnoty.

Výsledkom je funkčný systém profilov, ktorý výrazne zjednodušuje prácu terapeutov a zabezpečuje konzistentné prostredie pre pacientov naprieč sedeniami.

4.4 Aktualizácia technologických komponentov na aktuálne verzie

Práce na aktualizácii používaných technologických komponentov sa začali súbežne s vývojom a boli dokončené po úspešnej implementácii a otestovaní systému profilov pacientov. Komponenty boli aktualizované na ich najnovšie stabilné verzie, v súlade s analýzou vykonanou v kapitole 3.

Cieľom aktualizácie bolo dosiahnuť stabilnú verziu aplikácie, ktorá bude fungovať dlhodobo. Aktualizácie prebiehali iteratívne. Po každej aktualizácii bola aplikácia otestovaná na Oculus Quest 2 zariadení, aby sa prípadné problémy odhalili čo najskôr. Dôležitým princípom bolo zachovanie existujúceho správania aplikácie: všade, kde to bolo možné, boli pôvodné nastavenia komponentov prenesené do nových verzií bez zmeny, aby aplikácia fungovala rovnako ako pred aktualizáciou.

4.4.1 Mirror Networking

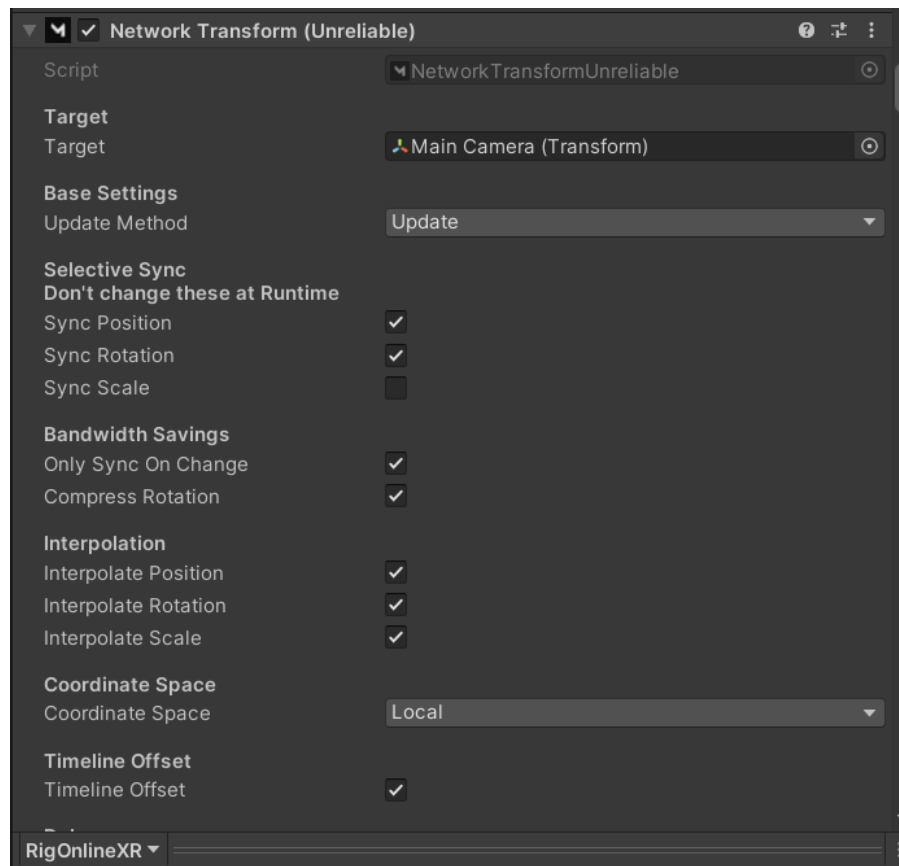
Mirror Networking bol aktualizovaný z verzie 72.0.0 na verziu 96.5.0. Postup aktualizácie spočíval v odstránení pôvodného pluginu a inštalácii novej verzie prostredníctvom `.uni` typ package súboru. Po inštalácii sa objavili chyby kompilácie súvisiace so zmenami v API, ktoré bolo potrebné opraviť.

Prechod na NetworkTransformUnreliable

Jednou z kľúčových zmien v novej verzii Mirror Networking je zavedenie komponentu `NetworkTransformUnreliable`, ktorý nahrádza pôvodný `NetworkTransform`. Tento komponent bol nasadený na XR a desktop rigy, rehabilitačné objekty (Block, Cube, Cup, Key), stôl a objekty interiéru.

`NetworkTransformUnreliable` je pre VR aplikácie vhodnejší ako `NetworkTransformReliable`. Využíva nespoľahlivý (UDP) prenos, ktorý znižuje latenciu pohybu. Pre VR aplikácie je všeobecne uprednostňovaná plynulosť pred garantovaným doručením každého paketu [11].

Pri konfigurácii komponentu boli prenesené pôvodné parametre z `NetworkTransform` a nastavené nové polia. Pre ďalšie vyhladzovanie pohybu bol nastavený parameter `Timeline Offset`. Parameter `Coordinate Space` bol nastavený podľa charakteru každého objektu: pre objekty pohybujúce sa nezávisle v scéne `World`, pre objekty relatívne k rodičovskému objektu `Local` (obr. 4.9).



Obrázok 4.9: Nastavenia komponentu `NetworkTransformUnreliable`

Referencie na `NetworkTransform` v triede `NetworkCharacterManager` boli nahradené komponentom `NetworkTransformUnreliable` pri priradovaní authority nad objektmi a pri ich resetovaní:

Výpis 4.25: Aktualizácia referencií na `NetworkTransformUnreliable`

```
item.gameObject.GetComponent<NetworkTransformUnreliable>().↩
    syncDirection = SyncDirection.ClientToServer;
// ...
item.gameObject.GetComponent<NetworkTransformUnreliable>().Reset();
```

Rovnaká zmena bola vykonaná v triede `CustomNetworkManager` pri nastavovaní `syncDirection` pre objekty scény:

Výpis 4.26: Nastavenie `syncDirection` pre objekty scény

```
if (objectIdentity.TryGetComponent(
    out NetworkTransformUnreliable netTransform))
```

```
{
    netTransform.syncDirection = SyncDirection.ServerToClient;
}
```

Zmeny v CustomNetworkManager

V novej verzii Mirror Networking bolo upravené správanie pri registrácii handlerov správ a odosielaní správ. Registrácia handlera pre CharacterMessage bola aktualizovaná s parametrom `requireAuthentication: false`, čo zodpovedá novému API:

Výpis 4.27: Aktualizovaná registrácia handlera správ

```
NetworkServer.RegisterHandler<CharacterMessage>(OnCreateCharacter, ↩
    false);
```

Pri odosielaní správ bol explicitne špecifikovaný kanál `Channels.Reliable` pre správy, kde je garantované doručenie nevyhnutné, napríklad pri zaslaní unikátneho identifikátora servera klientovi a pri zaslaní správy o vytvorení postavy:

Výpis 4.28: Odosielanie správ cez spoľahlivý kanál

```
conn.Send(new ServerUniqueIdMessage { serverUniqueId = serverUniqueId ↩
    }, Channels.Reliable);
// ...
NetworkClient.Send(characterMessage, Channels.Reliable);
```

RefaktORIZÁCIA SyncList callbackov v AnimationSettingsManager

Nová verzia Mirror Networking priniesla zmeny v spôsobe spracovania udalostí SyncList. Pôvodná implementácia využívala jediný callback `Callback` s parametrom `Operation`, ktorý rozlišoval typ zmeny pomocou `switch` príkazu. Nové API zavádza samostatné udalosti pre každý typ operácie: `OnInsert`, `OnRemove`, `OnSet` a `OnClear`.

Pôvodný kód bol refaktORIZOVANÝ tak, že každá operácia má vlastnú obslužnú metódu:

Výpis 4.29: Registrácia nových SyncList udalostí

```
blockSetup.OnInsert += OnAnimationSetupInserted;
blockSetup.OnRemove += OnAnimationSetupRemoved;
blockSetup.OnSet += OnAnimationSetupSet;
blockSetup.OnClear += OnAnimationSetupCleared;
// rovnako pre cubeSetup, cupSetup, keySetup
```

Dôležitým aspektom bolo zachovanie správania pri počiatočnej synchronizácii. Nové udalosti `OnInsert`, `OnSet` a ostatné sa pri prvotnej synchronizácii stavu zo servera na klienta nevolajú. Z tohto dôvodu bol ponechaný pôvod-

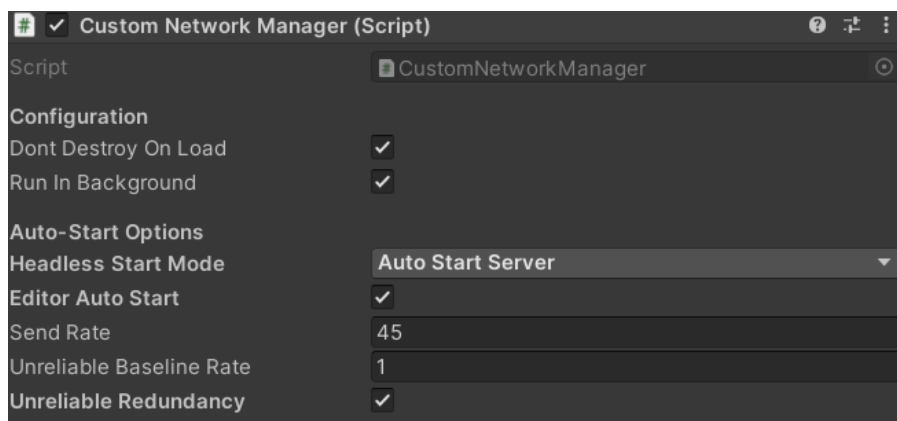
ný Callback delegát, premenovaný na `OnInitialSync`, ktorý pri počiatočnej synchronizácii zavolá `SetupMarkers()` a správne inicializuje markery v scéne:

Výpis 4.30: Callback pre počiatočnú synchronizáciu

```
blockSetup.Callback += OnInitialSync;
// ...
private void OnInitialSync(SyncList<PosRotMapping>.Operation op,
    int index, PosRotMapping oldItem, PosRotMapping newItem)
{
    SetupMarkers();
}
```

Nové nastavenia komponentu

Po aktualizácii sa v komponente `NetworkManager` objavili nové konfiguračné polia. Bolo nastavené `Headless Start Mode: Auto Start Server`, čo zabezpečuje automatické spustenie servera pri štarte v **headless** režime. Zároveň bola aktivovaná možnosť `Editor Auto Start`, ktorá automaticky spúšťa server pri štarte aplikácie v Unity Editore, čo zjednodušuje vývoj a testovanie (obr. 4.10).



Obrázok 4.10: Nové nastavenia komponentu `CustomNetworkManager`

4.4.2 Unity Engine

Unity Engine bol aktualizovaný z verzie 2021.3.35f1 na verziu 2022.3.62f3, čo je najnovšia dostupná verzia z LTS línie 2022.3 dostupná v rámci nášho predplatného Unity. Migrácia na Unity 6 bude riešená v rámci novej architektúry systému v nadväzujúcich prácach, keďže si vyžaduje rozsiahle prepracovanie existujúceho kódu.

Po nainštalovaní novej verzie Unity a reimporte projektu sa objavili problémy. Prvým bol nesprávne nastavený minimálny Android API level. Bolo potrebné ho zvýšiť z úrovne 23 na úroveň 24, čo zodpovedá požiadavkám nových Unity

XR knižníc. Druhým bolo nesprávne generovanie osvetlenia v oboch scénach, čo je štandardný jav po zmene verzie Unity a riešilo sa regeneráciou lightmap pre MenuScene aj MainScene.

Po odstránení týchto problémov bola aplikácia otestovaná na PC aj na Oculus Quest 2 zariadení a fungovala správne.

Pri aktualizácii Unity sa automaticky aktualizoval aj Vivox z verzie 16.0.1 na verziu 16.6.0, keďže je súčasťou Unity Gaming Services.

Vyriešenie pádu servera

Ako bolo popísané v analytickej časti práce (v podkapitole 3.2.1), problém s nekorrektným ukončovaním servera bol identifikovaný ako chyba vo vnútornom kóde Unity engine verzie 2021. Ako dočasné riešenie bol pridaný kód, ktorý vynúti ukončovať proces servera:

Výpis 4.31: Pôvodné dočasné riešenie pádu servera

```
if (!Application.isEditor)
{
    System.Diagnostics.Process.GetCurrentProcess().Kill();
}
```

Po aktualizácii Mirror Networking na verziu 96.5.0 a Unity na verziu 2022.3 sa server ukončuje korektne aj pri stlačení Ctrl+C, bez pádu a bez zanechania crash dump súborov. Toto potvrdilo predpoklad z analýzy, že problém bol v kombinácii starej verzie Unity engine a Mirror Networking. Dočasné riešenie bolo odstránené.

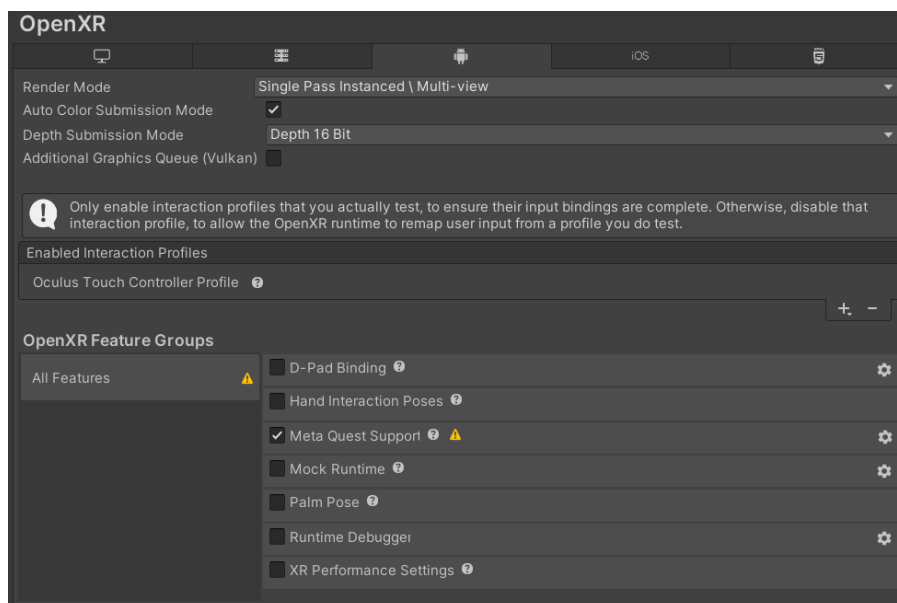
4.4.3 XR Interaction Toolkit

XR Interaction Toolkit bol aktualizovaný z verzie 2.3.2 na verziu 2.6.5. Aktualizácia na verziu 3.0 a novšiu nebola realizovaná, pretože prináša zásadné architektonické zmeny v spôsobe, akým sú konfigurované XR rigy. Ich adaptácia by si vyžadovala rozsiahle úpravy existujúcej konfigurácie a predstavovala by neúmerne veľké riziko pre stabilitu projektu. Verzia 2.6.5 je najnovšia verzia pred týmto veľkým skokom a prináša stabilizačné opravy bez nutnosti rozsiahlych zmien.

Po aktualizácii sa vyskytol problém s orientáciou kurzora Ray Interactoru. Retikel bol zobrazovaný kolmo voči povrchu namiesto správnej orientácie. Problém bol vyriešený úpravou **prefabov** ReticleEmpty a ReticleFilled.

4.4.4 OpenXR Plugin

OpenXR Plugin bol aktualizovaný z verzie 1.7 na verziu 1.14.3 v súlade s odporúčaním analýzy. Po aktualizácii boli nastavenia OpenXR resetované. Bolo potrebné všetko znovu nakonfigurovať pre všetky cieľové platformy (obr. 4.11).



Obrázok 4.11: Nastavenia OpenXR pluginu pre Android platformu

4.4.5 FinalIK

FinalIK bol aktualizovaný z verzie 2.3 na verziu 2.4. Keďže ide o platený asset, aktualizáciu mohol vykonať len jeho vlastník. Po aktualizácii neboli zaznamenané žiadne problémy ani potrebné úpravy kódu.

4.4.6 Testovanie po aktualizáciách

Aktualizácie prebiehali iteratívne, po každej aktualizácii bola aplikácia otestovaná na zariadení Oculus Quest 2, aby sa prípadné problémy odhalili čo najskôr. Po dokončení všetkých aktualizácií bola vykonaná záverečná verifikácia na VR zariadení, ktorá potvrdila, že všetky funkcie systému pracujú správne a správanie aplikácie zostalo konzistentné s verziou pred aktualizáciami.

4.4.7 Záver aktualizácií

Všetky technologické komponenty boli aktualizované na verzie identifikované v analýze. Mirror Networking 96.5.0 priniesol riešenie dlhodobého problému

s pádom servera a opravy súvisiace s čistením sieťových zdrojov pri ukončovaní pripojení, čo zlepšilo stabilitu servera. Unity 2022.3 LTS prinieslo vylepšenú podporu VR grafiky cez aktualizovaný Universal Render Pipeline a lepšiu kompatibilitu s modernými VR zariadeniami. Existujúca funkcionálnosť systému zostala zachovaná bez nutnosti rozsiahlych architektonických zmien.

5 Vyhodnotenie

V rámci tejto práce boli identifikované a vyriešené konkrétne problémy systému L-NeRVEn, ktoré komplikovali jeho reálne nasadenie a každodenné používanie v terapeutickej praxi.

Hlasová komunikácia pred opravou umožňovala pripojenie k hlasovému kanálu aj pri neúspešnom pokuse o pripojenie k serveru. Po zavedení GUID identifikátora pre každú serverovú inštanciu sú hlasové kanály plne izolované a previazané so sieťovým pripojením.

Správa serverov vyžadovala pri každom novom použití aplikácie manuálne zadávanie IP adries, čo bolo problematické najmä pre pacientov. Po implementácii načítavania zo súboru `servers.json` a popup panelu pre správu serverov odpadá táto povinnosť pri bežnom používaní.

Najvýraznejším prínosom z pohľadu každodennej prevádzky je perzistencia nastavení pacienta. Pred implementáciou musel terapeut pri každom sedení manuálne nastaviť výšku stola, pozíciu pacienta, parametre avatara a všetky tréningové hodnoty. Tieto nastavenia sa teraz automaticky obnovia pri pripojení pacienta k miestnosti, čo znižuje réžiu na začiatku každého sedenia.

Nekorektné ukončovanie servera pri pokuse o ukončenie pomocou Ctrl+C bolo dlhodobým problémom systému. Problém bol diagnostikovaný analýzou crash dump súborov, ktorá ukázala že chyba sa nachádza v natívnom kóde Unity a nesúvisí s kódom aplikácie. Ako dočasné riešenie bolo pridané vynútené ukončenie procesu. Po aktualizácii komponentov sa server ukončuje korektne a dočasné riešenie bolo odstránené.

Všetky kľúčové technologické komponenty systému boli aktualizované na ich najnovšie stabilné verzie v súlade s odporúčaniami analytickej časti. Unity bol aktualizovaný z verzie 2021.3 na 2022.3 LTS, Mirror Networking z verzie 72.0.0 na 96.5.0, XR Interaction Toolkit na 2.6.5, OpenXR Plugin na 1.14.3 a FinalIK na verziu 2.4. Po každej aktualizácii bola vykonaná verifikácia funkčnosti na zariadení Oculus Quest 2.

Vývoj prebiehal v spoločnom repozitári s ďalším riešiteľom pracujúcim na sys-

téme L-NeRVEn [32]. Každý riešiteľ pracoval vo vlastnej vývojovej vetve, zmeny sa následne zlúčili do spoločnej vetvy. Z tejto spoločnej vetvy bola vytvorená samostatná vetva obsahujúca aktualizácie komponentov, čím zostala spoločná vetva bez aktualizácií k dispozícii ako stabilná pracovná verzia pre prípad potreby.

Testovanie prebiehalo vo viacerých fázach. Počas vývoja boli zmeny priebežne overované v Unity Editore a pomocou nástroja ParrelSync⁴ pre simultánne spustenie viacerých klientov na jednom počítači. Funkčnosť na reálnom hardvéri bola overovaná osobne na zariadení Oculus Quest 2. Záverečné testovanie prebehlo spolu s ďalším riešiteľom Damiánom Chmurom [32] na fyzických zariadeniach Oculus Quest 2, kde boli overené kľúčové scenáre: pripojenie terapeuta a pacienta do spoločnej miestnosti, izolácia hlasových kanálov pri viacerých súbežných serveroch, automatické načítanie nastavení pacienta a synchronizácia tréningových parametrov medzi klientmi.

Systém je nasadzovaný v prostredí spolupracujúceho pracoviska Ústavu merania Slovenskej akadémie vied, kde prebieha jeho overovanie.

5.1 Možnosti ďalšieho rozvoja

Napriek tomu, že všetky plánované úlohy boli splnené, systém má priestor na ďalší rozvoj. Aktuálna sada rehabilitačných cvičení pokrýva základné motorické úlohy, no systém by bolo možné rozšíriť o ďalšie typy cvičení zamerané napríklad na jemnú motoriku alebo koordináciu. Rozšírenie sady cvičení by si však vyžadovalo spoluprácu s odborníkmi na neurorehabilitáciu, aby nové cvičenia zodpovedali klinickým požiadavkám.

V rámci aktualizácií komponentov boli niektoré prechody na novšie verzie vedome odložené. Migrácia na Unity 6 a XR Interaction Toolkit verzie 3.x si vyžaduje rozsiahle architektonické zmeny v konfigurácii XR rigov a ďalších systémov, ktoré by predstavovali neúmerné riziko pre stabilitu projektu v súčasnej fáze. Tieto migrácie zostávajú prirodzeným ďalším krokom pre nadväzujúce práce na systéme.

Testovanie prebiehalo výhradne na zariadeniach Oculus Quest 2. Keďže systém využíva OpenXR a nie je viazaný na konkrétnu platformu, malo by byť fungovanie na iných headsetoch podporujúcich OpenXR štandard bezproblémové, avšak toto nebolo overené. Otestovanie na iných zariadeniach, napríklad Meta Quest 3 alebo zariadeniach iných výrobcov, zostáva otvorenou úlohou pre ďalší vývoj.

⁴<https://github.com/VeriorPies/ParrelSync>

6 Záver

Cieľom tejto práce bolo rozšíriť a stabilizovať systém L-NeRVEn pre kolaboratívnu neurorehabilitáciu vo virtuálnej realite. Keďže ide o pokračovanie predchádzajúcich prác na tomto systéme, bolo najskôr potrebné analyzovať jeho súčasný stav, identifikovať konkrétne problémy a zároveň preskúmať podobné riešenia v oblasti VR neurorehabilitácie.

Analýza odhalila viaceré problémy: neizolované kanály hlasovej komunikácie, manuálnu správu serverov, chýbajúcu perzistenciu nastavení pacienta, nekorrektné ukončovanie servera a zastaranosť technologických komponentov. Všetky tieto problémy boli v rámci práce vyriešené.

Hlasová komunikácia bola opravená zavedením GUID identifikátora pre každú serverovú inštanciu, čím sa predišlo situácii, kde sa používateľ pripájal k hlasovému kanálu aj bez úspešného pripojenia k serveru. Správa serverov bola zjednodušená načítavaním zoznamu serverov z konfiguračného súboru, čím odpadla potreba manuálneho zadávania adries. Najpraktickejšou zmenou pre každodennú prevádzku je perzistencia nastavení pacienta pomocou SQLite databázy, keďže terapeut už nemusí pri každom sedení nanovo nastavovať pozíciu pacienta, výšku stola a tréningové parametre. Pád servera pri ukončení bol vyriešený aktualizáciou Unity na verziu 2022.3 LTS v kombinácii s Mirror Networking 96.5.0. Spolu s tým boli aktualizované aj XR Interaction Toolkit, OpenXR Plugin a FinalIK na ich najnovšie stabilné verzie.

Vývoj prebiehal v spolupráci s ďalším riešiteľom pracujúcim na tom istom systéme. Implementované zmeny boli priebežne integrované do spoločnej vývojovej vetvy. Funkčnosť všetkých zmien bola overená testovaním na zariadeniach Oculus Quest 2.

Otvorenými otázkami zostávajú migrácia na Unity 6 a XR Interaction Toolkit 3.x, ktoré si vyžadujú rozsiahlejšie architektonické zmeny a boli vedome odložené na nadväzujúce práce. Systém má taktiež priestor na rozšírenie o ďalšie typy rehabilitačných cvičení a overenie funkčnosti na iných VR zariadeniach než Oculus Quest 2.

K aplikácii boli vytvorené systémová a používateľská príručka. Obe príručky nadväzujú na príručky v práci [5], boli aktualizované a rozšírené o časti, ktoré dokumentujú novú funkcionality. Tento postup bol zvolený preto, lebo väčšia časť obsahu pôvodnej práce je stále aktuálna. Obe príručky sú vytvorené rovnakým spôsobom aj v práci spoluriešiteľa Damiána Chmuru [32].

Zoznam použitej literatúry

1. KOREČKO, Štefan; SOBOTA, Branislav; NEHILA, Peter; CHMURA, Damián. Architecture, Enhancements and Perspective of the L-NeRVEn Virtual Environment for Neurorehabilitation. *IEEE 17th International Scientific Conference on Informatics*. 2024. Dostupné z doi: [10.1109/Informatics62280.2024.10900751](https://doi.org/10.1109/Informatics62280.2024.10900751).
2. KOREČKO, Štefan; MATHES, Jaroslav; SOBOTA, Branislav; CHMURA, Damián; ROSIPAL, Roman; VANKO, Martin. L-NeRVEn virtual environment for neurorehabilitation: architecture, functionality, and evaluation. *Open Computer Science* [online]. 2025, roč. 16, č. 1, s. 20250049 [cit. 2026-02-09]. Dostupné z DOI: [10.1515/comp-2025-0049](https://doi.org/10.1515/comp-2025-0049).
3. JAVORKOVÁ, Sára. *Terapia pomocou technológií virtuálnej reality*. 2021. Dostupné tiež z: <https://opac.crzp.sk/?fn=detailBiblioForm&sid=14E8631BC0ED910BF6F4A633AA8B>.
4. GVOŠČOVÁ, Jana. *Terapeutický systém na báze technológií virtuálnej reality*. 2022. Dostupné tiež z: <https://opac.crzp.sk/?fn=detailBiblioForm&sid=83298E4B591633891884D9B92829>.
5. NEHILA, Peter. *Neurorehabilitácia v zdieľanej virtuálnej realite*. 2023. Dostupné tiež z: <https://opac.crzp.sk/?fn=detailBiblioFormChildCRIDA&sid=880D977B60C8641DD3EE229C4FC3>.
6. GUERRERO-MENDEZ, Cristian D.; RIVERA-FLOR, H.; VILLA-PARRA, Ana C.; BASTOS-FILHO, Teodiano F. Exploring Novel Practical Approach to Post-Stroke Upper-Limb Neurorehabilitation Based on Complex Motor Imagery Tasks. 2024. Dostupné z doi: [10.1109/EMBC53108.2024.10782286](https://doi.org/10.1109/EMBC53108.2024.10782286).
7. LENG, Jiancai; LV, Chengyan; LI, Jun; ZHAO, Heng; ZHANG, Bin; ZHANG, Aoning. Integrating Virtual Reality and Multifeature Fusion Deep Learning to Improve EEG-Based Motion Imagery Decoding. *IEEE Transactions on Instrumentation and Measurement*. 2025, roč. 74. Dostupné z doi: [10.1109/TIM.2025.3628441](https://doi.org/10.1109/TIM.2025.3628441).

8. MCMAHON, Michael; SCHUKAT, Michael. A Low-Cost, Open-Source, BCI-VR Game Control Development Environment Prototype for Game Based Neurorehabilitation. 2018. Dostupné z DOI: [10.1109/GEM.2018.8516468](https://doi.org/10.1109/GEM.2018.8516468).
9. REN, Shixin; WANG, Weiqun; HOU, Zeng-Guang; LIANG, Xu; WANG, Jia-xing; SHI, Weiguo. Enhanced Motor Imagery Based Brain-Computer Interface via FES and VR for Lower Limbs. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*. 2020, roč. 28. Dostupné z DOI: [10.1109/TNSRE.2020.3001990](https://doi.org/10.1109/TNSRE.2020.3001990).
10. PARIVASH, Farhad; AMUZADEH, Leila; FALLAHI, Alireza. Design expanded BCI with improved efficiency for VR-embedded neurorehabilitation systems. 2017. Dostupné z DOI: [10.1109/AISP.2017.8324087](https://doi.org/10.1109/AISP.2017.8324087).
11. *Mirror Networking Documentation* [online]. Mirror Networking [cit. 2025-01-24]. Dostupné z: <https://mirror-networking.gitbook.io/docs/>.
12. *Mirror Networking Changelog* [online]. Mirror Networking [cit. 2025-01-24]. Dostupné z: <https://mirror-networking.gitbook.io/docs/manual/general/changelog>.
13. *Fish-Networking: Networking Evolved - Unity Asset Store* [online]. First Gear Games [cit. 2025-11-22]. Dostupné z: <https://assetstore.unity.com/packages/tools/network/fishnet-networking-evolved-207815>.
14. *Fish-Networking Documentation* [online]. First Gear Games [cit. 2025-11-22]. Dostupné z: <https://fish-networking.gitbook.io/docs/>.
15. *Unity Editor Archive* [online]. Unity Technologies [cit. 2025-01-24]. Dostupné z: <https://unity.com/releases/editor/archive>.
16. *Choosing a Unity version and XR plugin* [online]. Microsoft [cit. 2025-01-24]. Dostupné z: <https://learn.microsoft.com/en-us/windows/mixed-reality/develop/unity/choosing-unity-version>.
17. TECHNOLOGIES, Unity. *Unity 6 New Features Overview* [online]. 2024. [cit. 2025-01-24]. Dostupné z: <https://www.youtube.com/watch?v=dsySV1DxkHc>.
18. *XR Interaction Toolkit Documentation* [online]. Unity Technologies [cit. 2025-01-24]. Dostupné z: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1/manual/index.html>.
19. *XR Interaction Toolkit Changelog* [online]. Unity Technologies [cit. 2025-01-24]. Dostupné z: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.4/changelog/CHANGELOG.html>.

20. *What's new in XR Interaction Toolkit 3.0* [online]. Unity Technologies [cit. 2025-01-24]. Dostupné z: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/whats-new-3.0.html>.
21. *XR Interaction Toolkit 3.1.2 Changelog* [online]. Unity Technologies [cit. 2025-01-24]. Dostupné z: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1/changelog/CHANGELOG.html>.
22. XR, Learn. *Unity Interaction Toolkit 2.4.0 & 2.5.0 Versions Are Here - New Big Improvements* [online]. 2023. [cit. 2025-01-24]. Dostupné z: <https://blog.learnxr.io/xr-development/unity-interaction-toolkit-2-5-0-new-big-improvements>.
23. *XRI 3.0 upgrade guide* [online]. Unity Technologies [cit. 2025-01-24]. Dostupné z: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.1/manual/upgrade-guide-3.0.html>.
24. *OpenXR Plugin Documentation* [online]. Unity Technologies [cit. 2025-01-24]. Dostupné z: <https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.14/manual/index.html>.
25. *OpenXR Plugin Changelog* [online]. Unity Technologies [cit. 2025-01-24]. Dostupné z: <https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.14/changelog/CHANGELOG.html>.
26. *Final IK - Unity Asset Store* [online]. RootMotion [cit. 2025-01-24]. Dostupné z: <https://assetstore.unity.com/packages/tools/animation/final-ik-14290>.
27. *What Is MySQL?* [online]. Oracle Corporation [cit. 2025-12-01]. Dostupné z: <https://www.oracle.com/mysql/what-is-mysql/>.
28. DAVID, Maurício. *LiteDB - A .NET NoSQL Document Store* [online]. [cit. 2025-12-01]. Dostupné z: <https://github.com/litedb-org/LiteDB>.
29. *About SQLite* [online]. SQLite [cit. 2025-12-01]. Dostupné z: <https://sqlite.org/about.html>.
30. HUERTAS, Roberto. *SQLite4Unity3d* [online]. [cit. 2025-12-01]. Dostupné z: <https://github.com/robertohuertasm/SQLite4Unity3d>.
31. *Mirror Networking - Remote Actions* [online]. Mirror Networking [cit. 2025-12-01]. Dostupné z: <https://mirror-networking.gitbook.io/docs/manual/guides/communications/remote-actions>.

32. CHMURA, Damián. *Komunikácia, architektúra a ďalšie aspekty systému pre neurorehabilitáciu vo virtuálnej realite*. 2026. Diplomová práca. Technická univerzita v Košiciach. Unpublished.

Zoznam skratiek

ACID Atomicity, Consistency, Isolation, Durability.

API Application Programming Interface.

AR Obohatená realita.

BCI Brain-Computer Interface.

CNN Convolutional Neural Network.

CSP Common Spatial Pattern.

EEG Electroencephalography.

ERD Event-Related Desynchronization.

ERS Event-Related Synchronization.

GUID Globally Unique Identifier.

HTTP Hypertext Transfer Protocol.

LDA Linear Discriminant Analysis.

LTS Long-Term Support.

MI Motor Imagery.

OSVR Open Source Virtual Reality.

RPC Remote Procedure Call.

SDK Software Development Kit.

TUKE Technická univerzita v Košiciach.

UDP User Datagram Protocol.

VR Virtuálna realita.

VRIK Virtual Reality Inverse Kinematics.

VRPN Virtual Reality Peripheral Network.

XR Rozšírená realita.

XRI XR Interaction Toolkit.

Slovník

Headless je režim behu serverovej aplikácie bez grafického rozhrania.

Plugin je softvérový komponent, ktorý pridáva konkrétnu funkciu do existujúceho systému.

Prefab je opakovane použiteľný herný objekt v Unity s preddefinovanými komponentmi a nastaveniami.

SyncVar je atribút Mirror Networking označujúci premennú automaticky synchronizovanú zo servera na všetkých pripojených klientov.

WebSocket je komunikačný protokol umožňujúci obojsmernú komunikáciu medzi klientom a serverom cez jedno trvalé spojenie.

Zoznam príloh

Príloha A Používateľská príručka

Príloha B Systémová príručka

Príloha C CD médium – záverečná práca v elektronickej podobe,

Príloha D Repozitár výsledného projektu je dostupný na adrese

<https://git.kpi.fei.tuke.sk/skor-lirkis-projects/vrarm-unity>

A Používateľská príručka

Táto používateľská príručka nadväzuje na pôvodnú projektovú príručku. Bola aktualizovaná a rozšírená novými časťami, ktorých obsahom je dokumentácia novej funkcionality. Dôvodom pre tento krok je, že sa veľká časť funkcionality nezmenila a obsahovo je stále relevantná.

A.1 Inštalácia softvéru

Aplikácie aktuálne nie je na žiadnej platforme, bežne používanej na šírenie softvéru. Je preto potrebné zabezpečiť prístup k potrebným súborom cez priame spojenie. Kontaktné adresy: stefan.korecko@tuke.sk, damian.chmura@student.tuke.sk, andrei.dzemidovich@student.tuke.sk. Pre fungovanie je potrebné mať zabezpečený funkčný server a správne nastavené klientske inštancie (IP adresa servera). Aktuálne server nie je verejne dostupný.

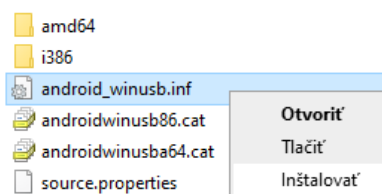
A.1.1 Osobný počítač

Aplikáciu nie je potrebné inštalovať. Spúšťa sa otvorením `NeuroRehabVR.exe`, ktorý je súčasťou priečinka so všetkými súborami.

A.1.2 Oculus Quest 2

Aktuálne nie je možné spúšťať a inštalovať aplikáciu cez platformu na šírenie softvéru (ako napríklad Oculus Store). Je potrebné ju ručne nainštalovať. Postup inštalácie obsahuje niekoľko krokov, na inštaláciu softvéru je potrebné použitie stolového počítača. Postup je nasledovný:

1. **USB debugging** - na inštaláciu aplikácií je potrebné zapnúť developer mode. Je možné ho zapnúť dvoma spôsobmi: použitím aplikácie v smartfóne alebo zmenou nastavení priamo v zariadení Oculus. Pri použití aplikácie Meta Quest na smartfóne je potrebné spárovať zariadenia, následne vybrať zariadenie a prejsť do nastavení zariadenia, kde je možné povoliť developer



Obrázok A.1: Inštalácia ovládača Oculus Go

mode. V prípade nastavenia na headsete priamo, je postup nasledovný: je potrebné prejsť do nastavení “Settings”, následne otvoriť kartu “System”. Ako posledný krok je potrebné prejsť do karty “Developer” na ľavej strane. V tejto karte je potrebné povoliť možnosť “USB Connection Dialog”. Následne sa po pripojení zariadenia do počítača na headsete objaví dialóg “Allow USB debugging?”. Je potrebné povoliť túto možnosť. Ak plánujete používať počítač v spojení s helmou častejšie, odporúčame povoliť možnosť “Always allow from this computer”,

2. **inštalácia ovládačov** - na stolovom počítači je potrebné nainštalovať Oculus Go ovládač⁵. Po stiahnutí adresára, je potrebné ho rozbaľiť. Po rozbalení je možné nainštalovať ovládač. Pravým tlačidlom myši na `android_winusb.inf` sa zobrazí možnosť “Inštalovať” (obr. A.1),
3. **Android Debug Bridge** - skrátene ADB, slúži na komunikácia s android zariadeniami, pripojenými k počítaču. Keďže Oculus Quest využíva android ako operačný systém, je možné tento nástroj použiť na inštaláciu softvéru. ADB je možné stiahnuť z oficiálnej stránky android⁶. Po stiahnutí a rozbalení priečinka je potrebné prejsť do cieľového priečinka, kde sú potrebné súbory rozbalené (odporúčame použiť iný priečinok než ten z predošlého kroku),
4. **inštalácia aplikácie** - inštaláciu je potrebné uskutočniť cez príkazový riadok. Prejdite do priečinka, kde ste rozbalili ADB nástroje v predošlom kroku. Podržaním tlačidla Shift a stlačením pravého tlačidla myši otvoríte dialógové menu. Zvoľte možnosť “Otvoriť tu okno prostredia PowerShell”, prípadne “Otvoriť tu príkazový riadok”. Môže sa to líšiť v závislosti od zariadenia. Následne je potrebné zadať príkaz:

```
.\adb.exe install -r .\Build.apk
```

“.\Build.apk” je potrebné nahradiť celou cestou k súboru aplikácie. Príkaz

⁵<https://developer.oculus.com/downloads/package/oculus-go-adb-drivers/>

⁶<https://developer.android.com/studio/releases/platform-tools>

s celou cestou by mohol vyzeráť nasledovne:

```
.\adb.exe install -r D:\Unity\NeuroRehabVR\Oculus\Build.apk
```

Pre jednoduchosť je preto možné uložiť inštalačný súbor “Build.apk” do rovnakého priečinka, v ktorom sú uložené ADB nástroje. Následne je možné použiť relatívnu cestu, ako je ukázané v prvom príklade príkazu,

5. **spustenie aplikácie** - po inštalácii sa aplikácia objaví medzi nainštalovaným softvérom. Je možné ju nájsť v priečinku “Unknown sources”.

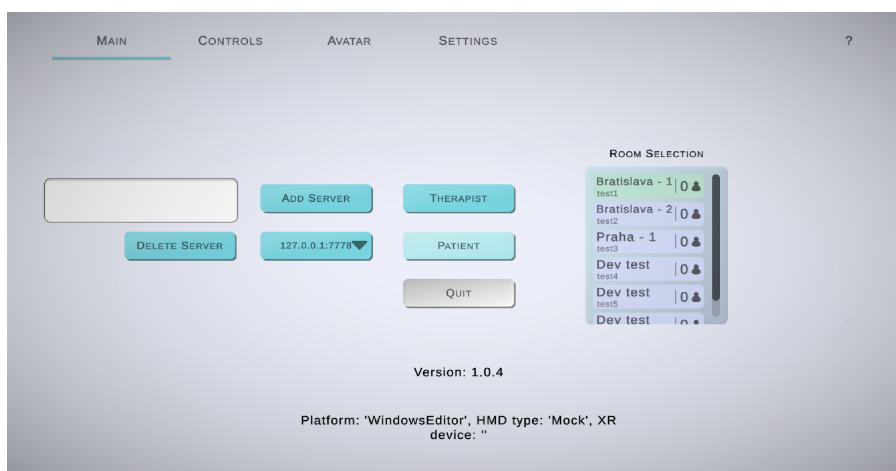
A.2 Použitie a ukážka aplikácie

Po spustení aplikácie sa používateľ nachádza v offline lobby scéne. Používateľ môže pridať novú IP adresu servera na pripojenie, meniť nastavenia, prispôbiť vzhľad avatara, ako aj resetovať výšku avatara. Pohyb v prostredí závisí od platformy, na ktorej sa nachádzame.

V prípade **osobného počítača** je možné ovládať pohyb pomocou klávesov “WASD” alebo kurzorových šípok, otáčanie funguje pomocou myši, interakcia s objektmi (UI aj 3D objekty) pomocou ľavého tlačidla myši. Menu je možné ovládať priamo stlačením tlačidiel v 3D priestore, alebo je možné zobrazíť 2D menu na vrchu obrazu. Po stlačení skratky na zobrazenie menu (klávesa “M”) sa odomkne myš a ovládanie menu je možné štandardným spôsobom. Presúvanie objektu je implementované jeho presúvaním myšou a zároveň držaním ľavého tlačidla myši. Presúvať objekt je možné v bežnom móde pohybu, stlačením klávesy “G” je možné uzamknúť rotáciu kamery. Zároveň sa odomkne kurzor myši. Pred pripojením má používateľ možnosť voľby roly, v ktorej bude vystupovať počas rehabilitácie. Vzhľad úvodného menu je na obr. A.2.

Počas používania **VR helmy** je možný pohyb pomocou chôdze, teleportácie alebo použitím páčky ľavého ovládača. Otáčanie je možné pomocou páčky praveho ovládača, ale vzhľadom na pocit nevoľnosti, ktorý spôsobuje odporúčame otáčať sa len celým telom. Interakcia s menu elementami je možná stlačením hlavného tlačidla ovládača, alebo použitím “Trigger” tlačidla. Interakcia funguje obooma ovládačmi. Presun objektov je možný zamierením ovládača na objekt, alebo dotknutia sa objektu a držaním tlačidla “Grip” ovládača (V prípade Oculus Quest ovládačov to sú tlačidlá na stranách). Zobrazenie menu nad ovládačom je možné pomocou tlačidla menu na ľavom ovládači. Podržaním tohto tlačidla sa objaví menu pred používateľom, ktoré zostane prilepené na danej pozícii.

Systém teleportácie umožňuje rýchly a jednoduchý pohyb v rámci virtuálneho prostredia. Tento spôsob pohybu je určený hlavne pre používateľov, ktorým



Obrázok A.2: Úvodné menu

klasický pohyb pomocou joysticka spôsobuje nevoľnosti.

A.2.1 Kroky pre správne využitie teleportačného systému

1. Aktivácia Teleportácie:

- Uchopte ľavý ovládač VR systému.

2. Výber Teleportačného Miesta:

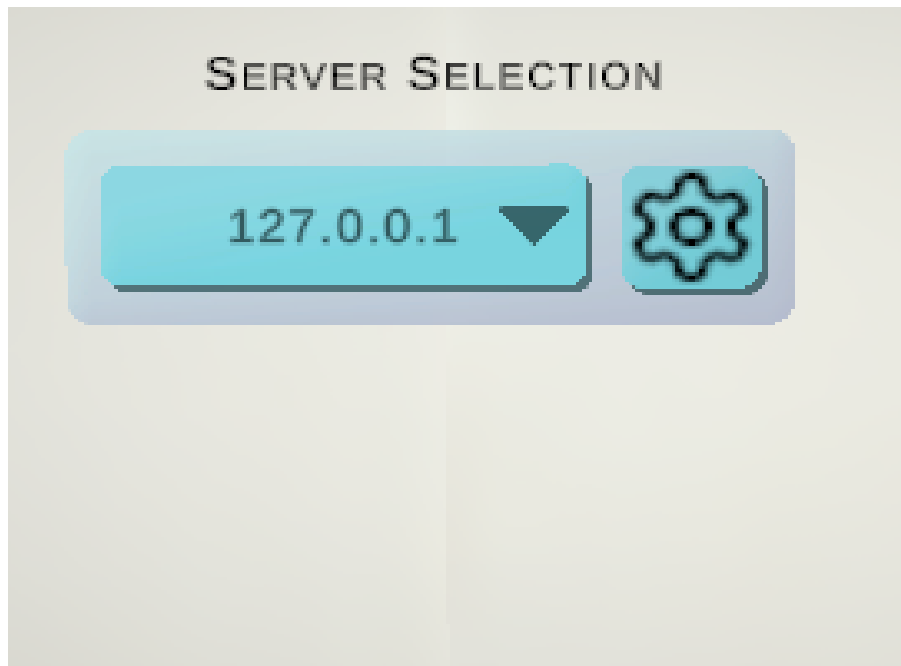
- Nasmerujte ľavý ovládač na miesto, kam sa chcete teleportovať. Farba ukazovateľa sa zmení na žltú, ak sa je na dané miesto možné presunúť.

3. Teleportácia:

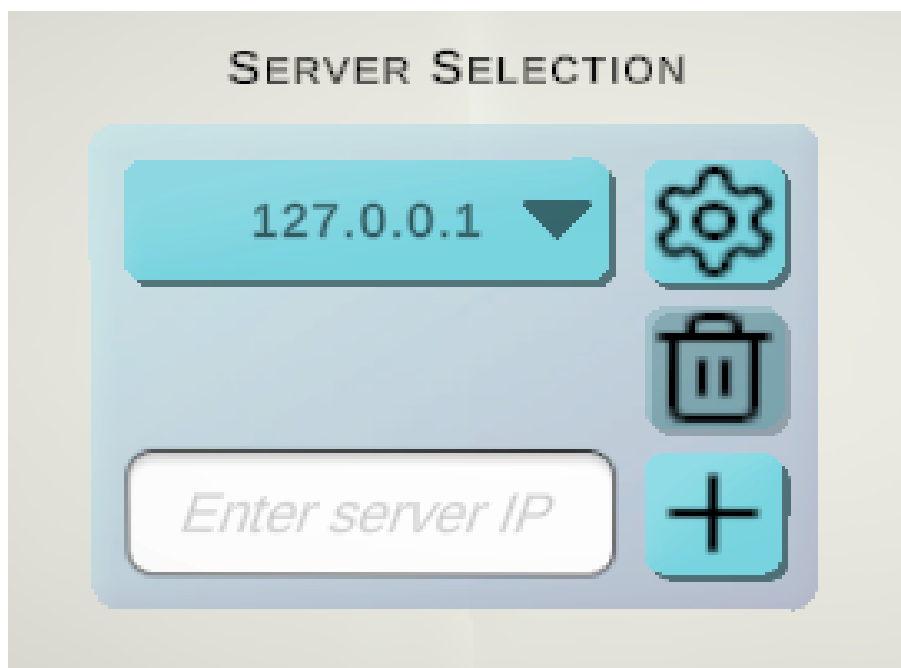
- Stlačte tlačidlo uchopenia na ľavom ovládači, keď je laserový ukazovateľ nasmerovaný na platnú teleportačnú oblasť.
- Po stlačení tlačidla sa okamžite presuniete na zvolené teleportačné miesto.

A.2.2 Správa serverov

Zoznam serverov je predvolene načítaný z konfiguračného súboru. Správa serverov je skrytá v popup paneli, ktorý sa zobrazí po stlačení príslušného tlačidla (obr. A.3 a A.4). V paneli je možné pridať nový server zadáním jeho IP adresy. Servery načítané z konfiguračného súboru sú chránené a nie je možné ich odstrániť. Po výbere servera v rozbaľovacom zozname sa aktualizuje zoznam dostupných miestností.



Obrázok A.3: Správa serverov so zatvoreným panelom



Obrázok A.4: Správa serverov s otvoreným panelom

A.2.3 Hlasová komunikácia

Po úspešnom pripojení k serveru je hlasová komunikácia automaticky inicializovaná, avšak mikrofón aj reproduktory sú predvolene stlmené. Hlasovú komunikáciu je možné aktivovať manuálne prostredníctvom prepínača v používateľskom rozhraní.

A.3 Rola pacienta

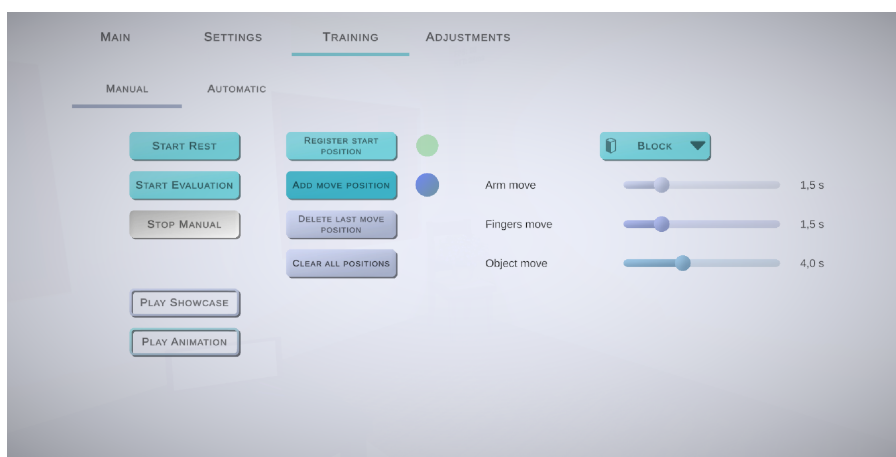
Pacient má počas cvičenia značne obmedzené možnosti. Môže sa hýbať v prostredí, no jeho prácou je len počkať na pokyny terapeuta. Po pripojení je automaticky zvolená ruka, ktorou sa vykonáva cvičenie pravá. Toto nastavenie môže meniť terapeut po pripojení do online prostredia. Terapeut môže pacientovi zafixovať zdravú ruku. Pacient to môže urobiť sám stlačením "N" na klávesnici alebo "Y" a "B" na ovládači.

Ak sú pre danú miestnosť uložené nastavenia pacienta z predchádzajúceho sedenia, server ich automaticky aplikuje po pripojení pacienta bez nutnosti manuálneho zásahu.

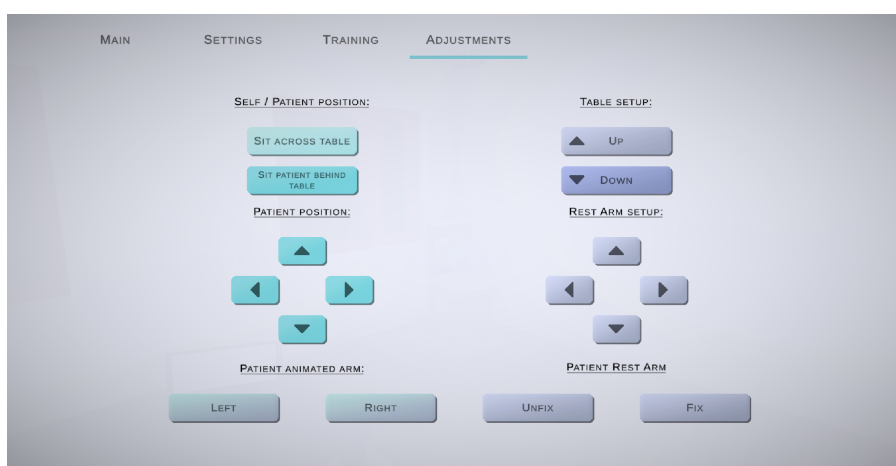
A.4 Rola terapeuta

Terapeut má za úlohu viesť celú rehabilitáciu. V porovnaní s rolou pacienta má tiež po pripojení svoje vlastné menu, kde sa nachádzajú nastavenie týkajúce sa rehabilitácie, ako aj pozícií počas cvičenia. Terapeut môže zmeniť aktívnu ruku pacienta, ktorá je rehabilitovaná alebo zafixovať zdravú. Na obr. A.5 je možné vidieť nastavenia, ktoré sú dostupné počas cvičenia terapeutovi. Môže meniť typ pohybu, ktorý je aktuálne používaný. Na výber je zo štyroch rôznych pohybov. Každý pohyb je špecifický predmetom, ktorý sa pri pohybe používa. Terapeut je taktiež schopný presúvať predmet a pridávať ďalšie pohyby pre rehabilitáciu. Dôležité je aj nastavenie časovania pre animácie, čas čakania alebo aj počet opakovaní. Tieto nastavenia je možné meniť v pravej časti menu. Pred spustením rehabilitácie môže terapeut tiež aj zobraziť ukážkovú animáciu. Po spustení rehabilitácie bude následne server čakať na udalosť z prostredia OpenVibe. Po prijatí správy sa spustí animácia na každom klientovi. Ak sa nepošle animácia v rozmedzí čakania, celé aktuálne cvičenie sa preruší a je potrebné ho znova spustiť.

Podstatnú časť nastavení ešte pred rehabilitáciou tvorí prispôsobenie správnych pozícií. Na obr. A.6 je možné vidieť možnosti, ktoré terapeut má. Môže presunúť svoju postavu bez nutnosti pohybu za stôl. Taktiež dokáže presunúť za stôl aj pacienta. To znamená, že pacient a terapeut sa nemusia fyzicky presúvať. Nie je potrebné sa presúvať ani pomocou ovládačov. Terapeut taktiež dokáže precíznejšie meniť pozíciu pacienta jeho posúvaním vpred, vzad a do strán. Systém taktiež ponúka aj možnosť kalibrácie výšky stola a zmenu oddychovej polohy ruky pacienta.



Obrázok A.5: Menu nastavení rehabilitácie



Obrázok A.6: Menu s nastaveniami pozícií

A.4.1 Nastavenie MI threshold hodnoty

Terapeut môže počas prípravy alebo priebehu cvičenia nastaviť threshold hodnotu pre EEG/MI vyhodnocovanie. Táto hodnota určuje hranicu, voči ktorej sa porovnáva prijímaný EEG signál. Nižšia hodnota znamená, že systém bude prah považovať za dosiahnutý pri slabšej EEG aktivite. Vyššia hodnota vyžaduje výraznejšiu EEG aktivitu.

Postup zmeny threshold hodnoty:

1. Otvorte terapeutické menu s nastaveniami cvičenia.
2. Nájdite ovládací prvok pre threshold hodnotu.
3. Posuňte slider na požadovanú hodnotu.
4. Skontrolujte zobrazenú novú hodnotu.



Obrázok A.7: Nastavenie MI threshold hodnoty v terapeutickom rozhraní

5. Stlačte tlačidlo Send, aby sa hodnota použila v aktuálnej relácii.

Hodnota je zobrazená v normalizovanom rozsahu od 0.00 do 1.00. Napríklad hodnota 0.55 zodpovedá 55 % normalizovanej škály. Rozhranie rozlišuje medzi hodnotou vybranou sliderom a aktuálne aplikovanou hodnotou. Samotné posunutie slidera ešte nemení aktívne nastavenie; hodnota sa aplikuje až po stlačení tlačidla Send.

Ak sa do miestnosti neskôr pripojí ďalší klient alebo sa znovu pripojí EEG klient, systém mu poskytne aktuálne nastavenú threshold hodnotu automaticky.

A.4.2 Ukazovateľ únavy pacienta

Ukazovateľ únavy pacienta zobrazuje priebežne prijímanú EEG/MI hodnotu. Slúži na sledovanie aktuálneho odhadu únavy pacienta počas cvičenia bez toho, aby terapeut musel opustiť terapeutické rozhranie.

Ukazovateľ zobrazuje normalizovanú hodnotu od 0.00 do 1.00:

- 0.00 znamená najnižšiu prijatú úroveň únavy,
- 1.00 znamená najvyššiu prijatú úroveň únavy,
- hodnoty medzi nimi predstavujú pomernú úroveň na rovnakej škále.

Zvislý ukazovateľ sa vyplňa zdola nahor podľa rastúcej hodnoty. Farba hodnoty sa mení od zelenej pri nízkych hodnotách cez stredné odtiene až po červenú pri vysokých hodnotách. Aktuálna hodnota je zároveň zobrazená aj číselne.

Pri prijatí novej hodnoty sa ukazovateľ aktualizuje a predchádzajúca hodnota sa presunie do krátkej histórie. História zobrazuje posledných päť predchádzajúcich hodnôt, pričom najnovšia predchádzajúca hodnota je uvedená navrchu. Terapeut tak môže sledovať, či sa únava v posledných krokoch zvyšuje, znižuje alebo zostáva približne stabilná.

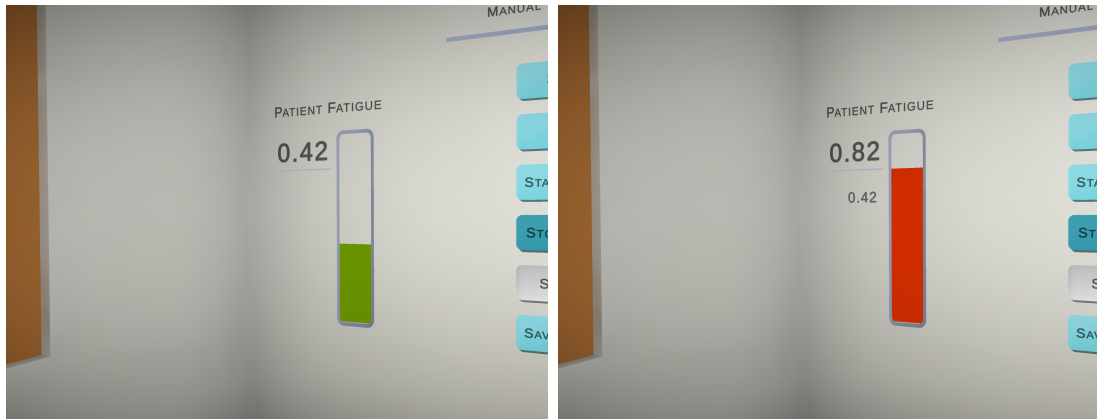
Terapeut môže ukazovateľ únavy použiť najmä na:

- sledovanie nárastu únavy počas cvičenia,
- porovnanie aktuálnej hodnoty s poslednými prijatými hodnotami,
- rozhodnutie, či pacient potrebuje prestávku alebo zníženie intenzity,
- overenie, že systém prijíma EEG údaje počas relácie.

Ak sa ukazovateľ únavy neaktualizuje, je potrebné overiť, či je EEG klient pripojený, či je pripojený do správnej miestnosti a či počas relácie odosiela údaje.

A.4.3 Ukladanie nastavení pacienta

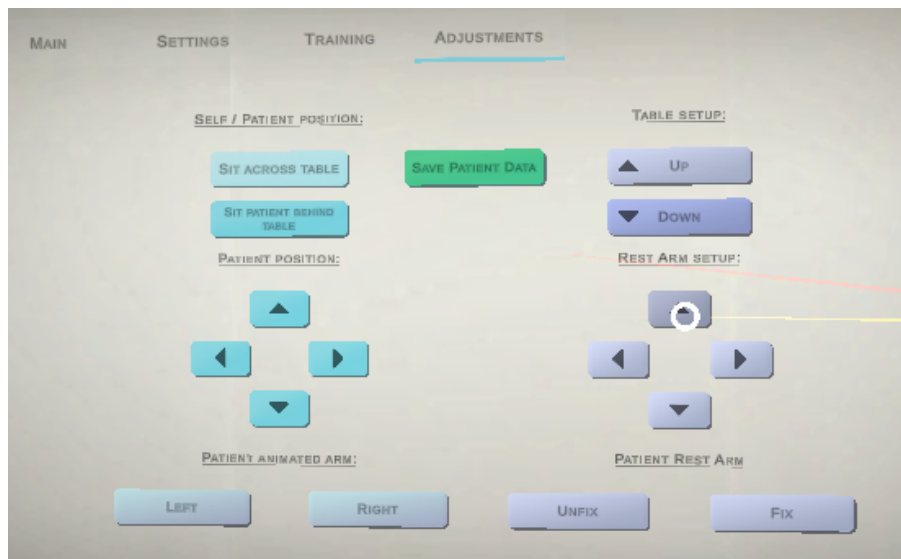
V taboch "Adjustments" a "Training" je k dispozícii tlačidlo "Save Patient Data" (obr. A.9 a A.10). Stlačením tlačidla sa aktuálne nastavenia uložia do databázy a pri ďalšom pripojení pacienta do tej istej miestnosti sa automaticky obnovia. Tab "Adjustments" ukladá výšku stola, pozíciu a rotáciu pacienta, polohu odpočívajúcej ruky a mierku avatara. Tab "Training" ukladá tréningové parametre:



(a) Nízka hodnota únavy

(b) Vysoká hodnota únavy

Obrázok A.8: Ukazovateľ únavy pacienta v terapeutickom rozhraní

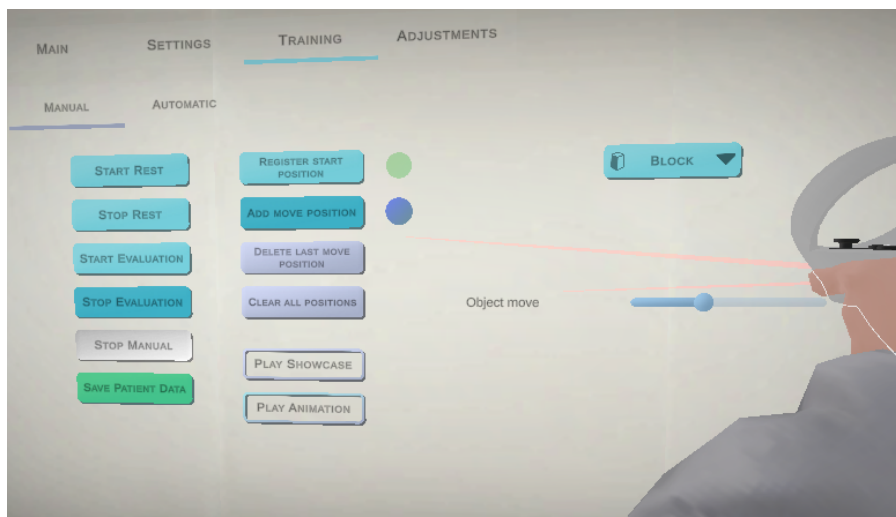


Obrázok A.9: Tab Adjustments s uložením nastavení pacienta

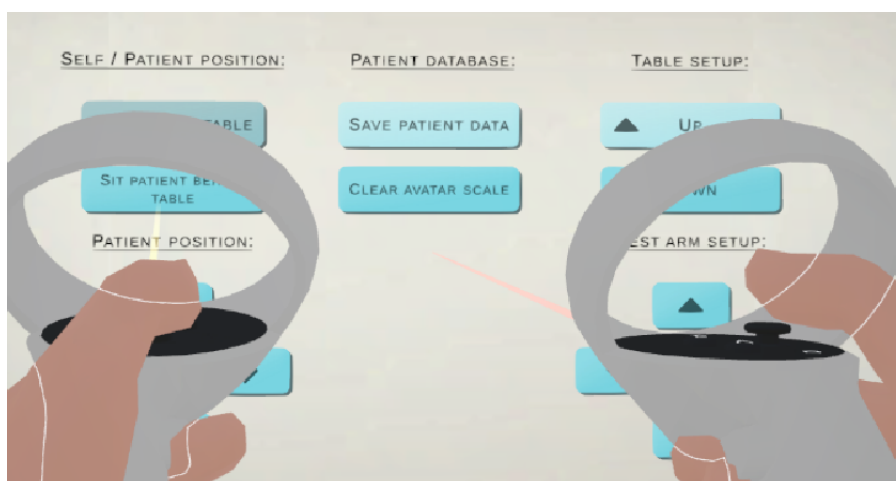
trvanie periódy čakania a odpočinku, trvanie pohybu, počet opakovaní, typ animácie a pozície objektov pre jednotlivé typy cvičení.

Ak je mierka avatara uložená v databáze a je potrebné ju zmeniť, terapeut môže použiť tlačidlo pre reset mierky avatara v tabe “Adjustments” (obr. A.11). Po jeho stlačení sa uložená hodnota vymaže a pri ďalšom pripojení pacienta sa použije hodnota zvolená v offline scéne.

Terapeut je schopný nastaviť viacero pozícií pre pohyb počas animácie. Systém ale má nastavených niekoľko obmedzení. Maximálny počet pozícií pre pohyb je 10, animácia potrebuje mať nastavenú aspoň jednu pozíciu (začiatočnú) aby ju bolo možné spustiť, pozícia sa použije len v prípade, že je v dosahu ruky pacienta. To znamená, ak sa pacient pohne počas rehabilitácie a pozícia už nie je v jeho dosahu, pozícia sa preskočí. Animácia s kľúčom a zámkom má trochu iné nastave-



Obrázok A.10: Tab Training s uložením nastavení pacienta



Obrázok A.11: Tlačidlo pre reset uloženej mierky avatara

nia. Animácia musí mať vždy dve pozície, aby ju bolo možné spustiť začiatočnú pozíciu a cieľovú pozíciu zámku.

A.5 Ukážka systému

V tejto časti ukážeme zopár obrázkov, kde je možné vidieť systém a niektoré vlastnosti, ktorými disponuje.

A.5.1 Vzhľad prostredia

Na obr. A.12 a A.13 je vyobrazené prostredie, v ktorom prebieha rehabilitácia. Na stole je možné vidieť objekt, ktorý sa aktuálne používa počas rehabilitácie. Pri zmene typu animácie sa nový objekt zobrazí na stole.



Obrázok A.12: Prostredie s pohľadom na stôl



Obrázok A.13: Zvyšok miestnosti

A.5.2 Ukážka animácií

Na pod-obrázkoch obr. A.14 je možné vidieť rozdielne pohyby počas vykonávania animácií. Na pod-obrázku (d) sú taktiež vidieť ukazovatele, ktoré vyznačujú pozície pre pohyby. Zelená guľa predstavuje štartovaciu pozíciu, modrá ostatné pozície. Tieto značky vidí len terapeut. Ako bolo spomenuté v predošlej kapitole, pozície animácií sú obmedzené dosahom ruky pacienta. Na obr. A.15 je možné vidieť ukazovateľ dosahu. Tento ukazovateľ je viditeľný len terapeutom počas presúvania objektu. Na obr. A.16 je vidieť animáciu z pohľadu pacienta.



(a) Uchopenie a presun kvádra



(b) Uchopenie kocky



(c) Zdvihnutie pohára



(d) Presunutie kľúča do zámky

Obrázok A.14: Ukážka rôznych typov animácií



Obrázok A.15: Dosah ruky pacienta



Obrázok A.16: Animácia z pohľadu pacienta

B Systémová příručka

Táto systémová príručka je určená pre vývojárov a správcov systému NeuroRehabVR. Nadväzuje na pôvodnú projektovú príručku a dopĺňa ju o postupy pre importovanie projektu, preklad aplikácie, nasadzovanie servera, správu release kanálov a integračné rozhrania.

B.1 Importovanie projektu

Projekt je potrebné možné stiahnuť a importovať do Unity, napríklad použitím aplikácie Unity Hub. Aplikácie aktuálne používa verziu Unity editora 2022.3.62f3⁷, ktorú je možné stiahnuť z oficiálnych stránok Unity. Moduly pre platformy, ktoré je potrebné stiahnuť pri inštalácii sú:

- Android Build Support,
- Windows Build Support,
- Linux Dedicated Server,
- Windows Dedicated Server.

Nie je nutné stiahnuť všetky moduly naraz, keďže je možné ich stiahnuť aj neskôr. Pri prvom otvorení projektu sa automaticky stiahnu všetky potrebné balíčky, ktoré nie sú súčasťou projektu. Po spustení editora je ešte potrebné počkať, kým sa importujú všetky potrebné knižnice a nastaví sa potrebné prostredie. Ďalej je potrebné cez menu “Edit” > “Project Settings” > “Services” vybrať našu organizáciu NeuroRehabVR, aby sa zapol Unity Cloud. Pre spustenie aplikácie a otestovanie plnej funkcionality je potrebné použiť viacero inštancií (aspoň jeden server a jeden klient). Odporúčame použitie rozšírenia ParrelSync do Unity. Pomocou ParrelSyncu je možné vytvoriť si klony a tie používať ako ďalšie inštancie aj na rovnakom počítači. Veľkou výhodou ParrelSyncu je to, že zmeny

⁷<https://unity.com/releases/editor/qa/lts-releases?version=2022.3>

vykonané v hlavnej inštancii sa automaticky aktualizujú aj v klonoch. Pri spúšťaní aplikácie v editore je potrebné dať si pozor na to, akú scénu máme aktuálne otvorenú. Je potrebné spúšťať aplikáciu so scénou "MenuScene" otvorenou. Server spustíte stlačením ServerOnly vľavo hore (v normálnom builde sa server spustí sám, ale v Editore tomu tak nie je). Klientov pripojiť k serveru je možné priamo použitím menu v scéne. Je možné použitie aj HUD vľavo hore, toto menu je ale len zamýšľané na použitie počas vývoja, nie počas prevádzky. Aplikácie funguje len na Windowse alebo Androide (Oculus). OpenXR nepodporuje DX11 na Linuxe z nejakého dôvodu. Pokusy o spustenie na Linuxe pomocou Protonu alebo Wine-u neboli úspešné. Jediný spôsob, ako poskytnúť funkcionality na Linuxe je zmena nastavení (nižšie opísané v sekcii **B.2**).

Ak by bol nejaký problém, napr. nefungujúce pripojenie klientov na server, je potrebné skontrolovať niekoľko parametrov, ktoré môžu pomôcť opraviť tento problém:

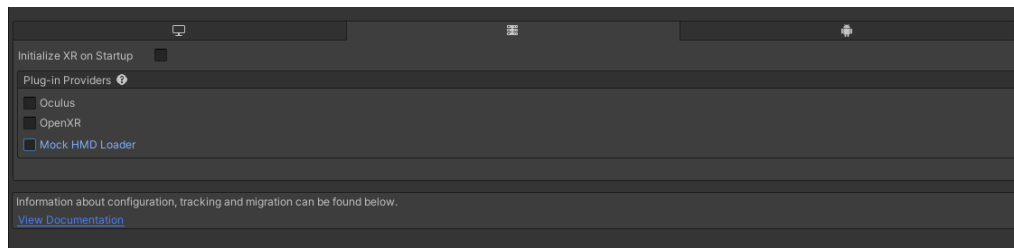
- správne IP adresy (je možné použiť localhost) - server aj klienti musia byť nastavený na rovnakú IP adresu,
- antivírusový softvér - niekedy antivírusový softvér blokuje porty (aktuálne server používa port 7777),

B.2 Preklad aplikácie

Preklad v Unity sa vykonáva cez menu "File" > "Build Settings". Otvorí sa okno, kde je možné meniť nastavenia pre preklad. Je dôležité dať si pozor na to, aby boli zvolené všetky scény, ktoré sú v aplikácii používané. Pri preklade aplikácie je potrebné nastaviť XR Plugin, ktorý poskytuje funkcionality XR. V aplikácii používame dva pluginy: OpenXR a Mock HMD. OpenXR zabezpečuje fungovanie VR headsetov, Mock HMD Loader používame pri simulovaní XR na desktope.

B.2.1 Preklad pre server

Z ľavej časti je potrebné vybrať možnosť Dedicated Server. Následne je potrebné vybrať Target platform. Je možné použiť aj Linux aj Windows, aplikácia je otestovaná na Windowse primárne, Linux je otestovaný čiastočne. Pred tým, než môžeme vygenerovať potrebné súbory je potrebné zmeniť nastavenia projektu. Do nastavení môžeme prejsť dvoma spôsobmi, v okne určenom na build stlačením tlačidla "Player settings", alebo cez menu "Edit" > "Project Settings". Nastavenia kvality a iných špecifikácií už sú vytvorené. Najdôležitejšie nastavenie



Obrázok B.1: Nastavenia XR Plug-in management pre server

je ale v karte “XR Plug-in management”. V otvorenom menu je potrebné vybrať pod-kartu určenú pre dedikované servery (obr. B.1). V prípade prekladu pre servery je potrebné vypnúť loader-y. Ak ostanú loader-y povolené predlžuje to čas buildovania a aj veľkosť vygenerovaných súborov. OpenXR loader taktiež nie je kompatibilný s operačným systémom Linux. Tieto loadre nie sú vôbec potrebné pre fungovanie servera, keďže XR modul nikdy nezapíname. Následne je možné zmeniť platformu a spustiť generovanie buildu.

B.2.2 Preklad pre stolový počítač

V okne “Build Settings” zvolíme možnosť pre “Windows, Mac, Linux”. Target platform zvolíme Windows. Ak sme pred tým menili nastavenia pre dedikovaný server je potrebné prejsť do nastavení projektu, konkrétne “XR Plug-in management” a zapnúť loadre nanovo. Dôvodom je, že pri zmene nastavení “XR Plug-in management” pre server sa zmenia aj pre desktop. Podľa informácií, ktoré sme našli na fóre je to takto “by design”. Možno sa to v budúcnosti zmení. OpenXR plugin sa použije v prípade, že používame VR headset pripojený k stolovému počítaču.

Linux

Aj napriek tomu, že OpenXR nefunguje na Linuxe, je možné preložiť aplikáciu bez tohto pluginu. V nastaveniach pre “XR Plug-in management” je potrebné vypnúť loader OpenXR. Problémom je potreba čiastočne prispôsobiť kód. Bez pluginu OpenXR bude možné aplikáciu používať v desktop móde, prípadne použitie simulovaného VR (keďže táto funkcionality je zabezpečená pluginom Mock HMD). Aplikácie ale nie je otestovaná na Linuxe, takže je potrebné tomu venovať pozornosť v prípade, že chceme ponúkať podporu pre Linux.

B.2.3 Preklad pre Oculus Quest 2

Vytvorenie apk súboru pre Oculus Quest 2 je možné aj bez pripojeného zariadenia. Ak je ale zariadenie aj pripojené je možné hneď nainštalovať aplikáciu priamo na Oculus. Ohľadom loadrov, ktoré sme museli meniť v prípade prekladu na stolový počítač alebo server, to na android nie je potrebné. Nesmie ale byť povolený Mock HMD Loader, keďže jeho spustenie by znefunkčnilo OpenXR plugin. Nastavenia prekladu je možné do určitej miery upravovať. Jedným z nich je Run Device. Ak je Oculus Quest 2 pripojený k počítaču, je možné prepnúť na tento VR headset. Následne Stlačením tlačidla “Build nad Run” sa spustí preklad aplikácie. Nová verzia aplikácie sa zároveň aj nainštaluje na zariadenie. Ďalším nastavením je “IL2CPP Code Generation” . Je možné zvoliť z dvoch možností, možnosť pre rýchlejší beh aplikácie a možnosť pre rýchlejší preklad. Počas vývoja je možné používať druhú možnosť, keďže generuje menšie inštalačné súbory a taktiež aj preklad trvá kratšie. Ak chceme inštalovať priamo na Oculus je potrebné mať povolený developer mode. Rozdiel v trvaní prekladu ale nie je príliš významný. Na obr. B.2 je možné vidieť nastavenia, ktoré sme používali počas vývoja.

B.3 Nasadenie servera

Nasadenie servera je jednoduché, jediné čo je potrebné spraviť je spustiť správny build. Server umožňuje spustenie na konkrétnej IP adrese, takže nie je potrebné vytvárať nový build len kvôli zmene IP adresy (klientske inštancie je potrebné vygenerovať nanovo, aktuálne neponúkajú možnosť konfigurácie). Spustenie servera so špecifickou IP adresou je možné prepínačom `serverip`. Celý príkaz by mohol vyzeráť nasledovne:

```
.\NeuroRehabVR.exe -serverip 192.168.1.210
```

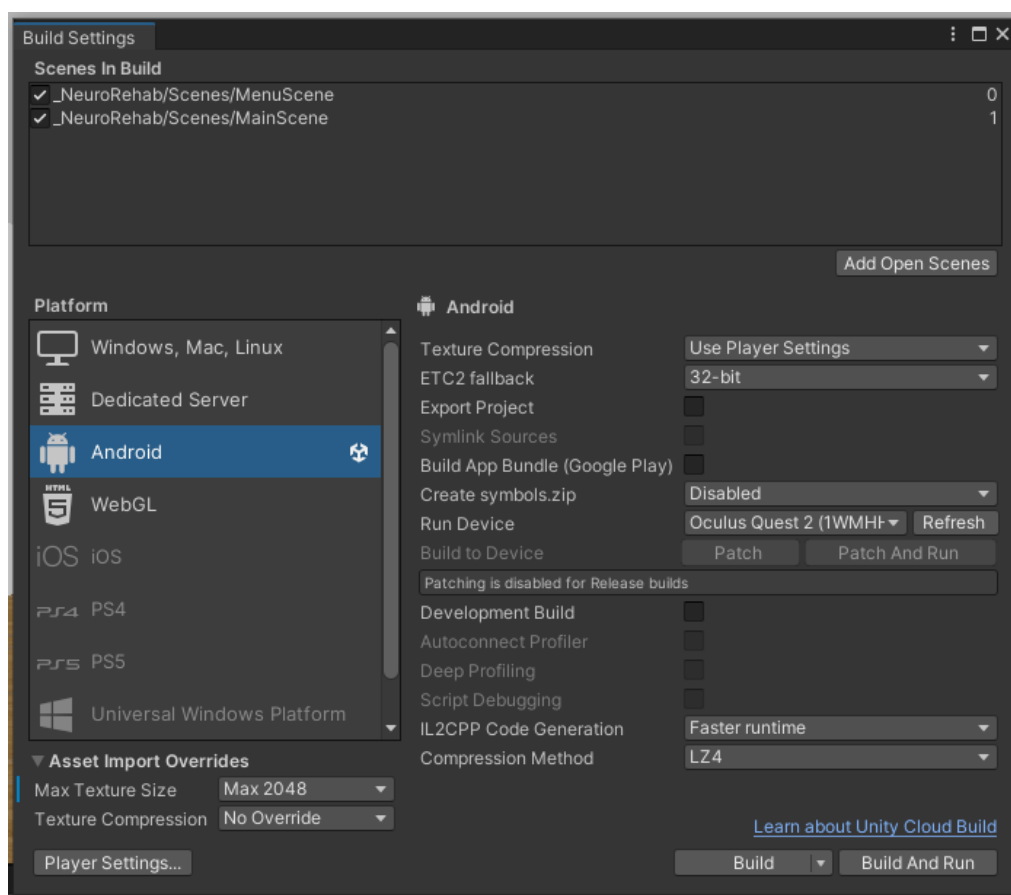
Namiesto IP adresy je možné taktiež použiť kľúčové slovo “localhost”.

B.3.1 Konfigurácia zoznamu serverov

Zoznam predvolených serverov pre klientske inštancie je definovaný v konfiguračnom súbore `servers.json`, ktorý je súčasťou projektu. Súbor má nasledujúci formát:

Výpis B.1: Formát konfiguračného súboru `servers.json`

```
[
  {
```



Obrázok B.2: Nastavenia na preklad pre Android

```

    "serverIp": "147.232.205.23"
  },
  {
    "serverIp": "127.0.0.1"
  }
]

```

Server na prvej pozícii v súbore sa pri prvom spustení aplikácie stane predvoľným. Servery definované v tomto súbore sú označené ako chránené, preto ich používatelia nemôžu odstrániť prostredníctvom používateľského rozhrania.

B.4 Databáza

Systém používa SQLite databázu na serveri pre ukladanie profilov pacientov. Databáza je inicializovaná automaticky pri prvom spustení servera na danom zariadení a naplní sa zoznamom terapeutických miestností z nakonfigurovaného objektu NetworkSettings. Databázový súbor sa nachádza v adresári:

```
%USERPROFILE%\AppData\LocalLow\<CompanyName>\<ProductName>\
```

Obsah databázy je možné prezerať a upravovať pomocou nástrojov pre SQLi-

te, napríklad DB Browser for SQLite. Priama úprava databázy je vhodná v prípade potreby zmeny zoznamu miestností alebo opravy nesprávnych hodnôt. Pred priamou úpravou odporúčame zálohovať súbor databázy.

B.5 Nasadenie release buildov cez Meta Quest Developer Hub

Táto časť opisuje postup nasadzovania nových verzií aplikácie NeuroRehabVR do beta vetvy Meta Store. Postup vychádza z procesov popísaných v kapitole ??.

B.5.1 Predpoklady

Pred zostavením a nahraním buildu je potrebné overiť:

- prístup k Meta developer organizácii (účet s oprávneniami na upload),
- nainštalovaný Meta Quest Developer Hub (MQDH),
- prístup k správnomu Android keystore súboru (vrátane aliasu a hesla) pre cieľový kanál,
- telefón pre SMS verifikáciu pri uploade,
- Unity projekt s korektnou konfiguráciou.

B.5.2 Identifikátory kanála

Podľa cieľového release kanála je potrebné nastaviť správne hodnoty:

Tabuľka B.1: Konfigurácia podľa release kanála

Kanál	Package Name	Product Name	Bundle Code rada
LTS	com.lirkis.neurorehabvr	NeuroRehabVR [LTS]	10000+ (zmrazený)
STABLE	com.lirkis.neurorehabvr.stable	NeuroRehabVR [STABLE]	11000+
RC	com.lirkis.neurorehabvr.rc	NeuroRehabVR [RC]	12000+

B.5.3 Nastavenia buildu v Unity

1. Player Settings → Android:

- *Package Name* — nastaviť podľa tabuľky B.1,
- *Product Name* — nastaviť s príslušnou príponou kanála,
- *Version* — viditeľné číslo verzie (napr. 1.2.5),
- *Bundle Version Code* — inkrementovať v rámci rady kanála, musí byť vyšší ako predchádzajúci upload.

2. Build Settings:

- *Development Build* — **vypnúť**.

3. XR Plugin Management:

- overiť, že OpenXR plugin je aktívny pre platformu Android.

4. Keystore (Publishing Settings):

- vybrať správny .keystore súbor pre daný kanál,
- zadať alias a heslo.

5. Vizuálna identita:

- overiť, že ikona (logo) a banner zodpovedajú príslušnému kanálu.

Build sa zostaví cez *File → Build Settings → Android → Build*. Výstupom je súbor .apk alebo .aab.

B.5.4 Pre-upload checklist

Pred nahraním buildu do MQDH je nutné potvrdiť všetky nasledujúce body:

- ☐ Správne package name podľa cieľového kanála
- ☐ Správna prípona product name ([LTS], [STABLE], [RC])
- ☐ Správne viditeľné číslo verzie
- ☐ Bundle Version Code inkrementovaný oproti poslednému uploadu
- ☐ Build podpísaný správnym keystore pre daný kanál
- ☐ Development mode vypnutý
- ☐ OpenXR plugin aktívny

B.5.5 Upload cez Meta Quest Developer Hub

1. Otvoriť MQDH a prihlásiť sa Meta účtom s prístupom do správnej developer organizácie.
2. Prejsť do sekcie *App Distribution* a zvoliť správnu organizáciu a aplikáciu.
3. Zvoliť cieľový release kanál (beta branch).
4. Kliknúť na *Upload* a zvoliť príslušný súbor .apk alebo .aab.
5. Pred potvrdením overiť všetky body z checklistu (sekcia [B.5.4](#)).
6. Dokončiť SMS verifikáciu (vyžaduje prístup k telefónu).
7. Počkať na spracovanie buildu platformou.
8. Overiť, že build sa objavil v správnom kanáli.

B.5.6 Verifikácia na zariadení

Po úspešnom uploade je potrebné overiť build priamo na Meta Quest zariadení:

1. Nainštalovať alebo aktualizovať aplikáciu na zariadení.
2. Overiť, že názov aplikácie obsahuje správnu príponu kanála.
3. Overiť základnú funkčnosť aplikácie.

B.5.7 Build promotion (RC → STABLE)

Postup presunu buildu z testovacieho kanála do stabilného:

1. Otestovať RC build na cieľových zariadeniach.
2. Ak sa objavia kritické chyby, vytvoriť nový RC build so zvýšeným Bundle Code a testovanie zopakovať.
3. Po úspešnom overení pripraviť rovnaký stav kódu ako STABLE build:
 - zmeniť package name na `com.lirkis.neurorehabvr.stable`,
 - zmeniť product name na `NeuroRehabVR [STABLE]`,
 - inkrementovať Bundle Version Code v rade STABLE (11000+).
4. Nahrať STABLE build do príslušného release kanála podľa postupu v sekcii [B.5.5](#).
5. Doplniť release notes.

B.5.8 Riešenie problémov

Bundle Version Code conflict — MQDH odmietne build so zhodným alebo nižším Bundle Code. Riešenie: zvýšiť kód, znovu zostaviť, upload zopakovať.

24-hodinový lockout účtu — chyba *too many requests*. Riešenie: počkať 24 hodín, neskúšať opakovane nahrávať.

Cooldown po zmazaní aplikácie — po zmazaní app line nie je možné znovu použiť rovnaké package name niekoľko dní. Riešenie: aplikácie nemazať, ponechať ako zmrazený *fallback*.

Strata keystore — bez správneho podpisovacieho kľúča nie je možné aktualizovať daný app line (poučenie z kanála [LTS](#)).

B.5.9 Správa keystore

Keystore súbory, aliasy a heslá nesmú byť uložené v repozitári. Musia byť evidované v bezpečnom úložisku spolu s informáciou, ku ktorému app line patria.

- Každý kanál má vlastný keystore.
- Keystore je zviazaný s package name — strata znamená nemožnosť aktualizovať daný app line.
- Súbor `.gitattributes` chráni deployment identifikátory (package name, product name) pred nechceným prepísaním pri mergovaní vetiev.
- Pri zakladaní nového app line alebo verzie „V2“ musí byť správa keystore zavedená od prvého dňa vývoja.

B.6 REST API dokumentácia

Systém obsahuje prístup pomocou REST API k serveru zvonku. API obsahuje niekoľko referencií. POST Request bude vždy úspešný, ak sa podarí kontaktovať server. Mení sa response body. Hodnota “result” je rovná “true” len v prípade, že sa naozaj podarí vykonať funkciu spustenú POST requestom. Na obr. [B.3](#) je vyobrazená referencia na spustenie animácie počas tréningu. Na obr. [B.4](#) je request pre obyčajný pohyb, tento je možné vykonať pokiaľ nebeží tréning. Request na obr. [B.5](#) zmení aktuálny typ animácie na “Off” a tým skryje objekty určené na animácie. Request `/spawn` slúži na nastavenie typu animácie, typ animácie sa prepne do posledného typu animácie. (obr. [B.6](#)).

POST **/training/move** Progress move during training

If training is running, will progress step and show animation

Responses

Code	Description	Links
200	Successful connection to server, result is true when move is successfully started. Otherwise it's false. Media type application/json Controls Accept header. Example Value Schema <pre>{ "result": true }</pre>	No links
400	Failed to connect	No links

Obrázok B.3: API dokumentácia - /training/move

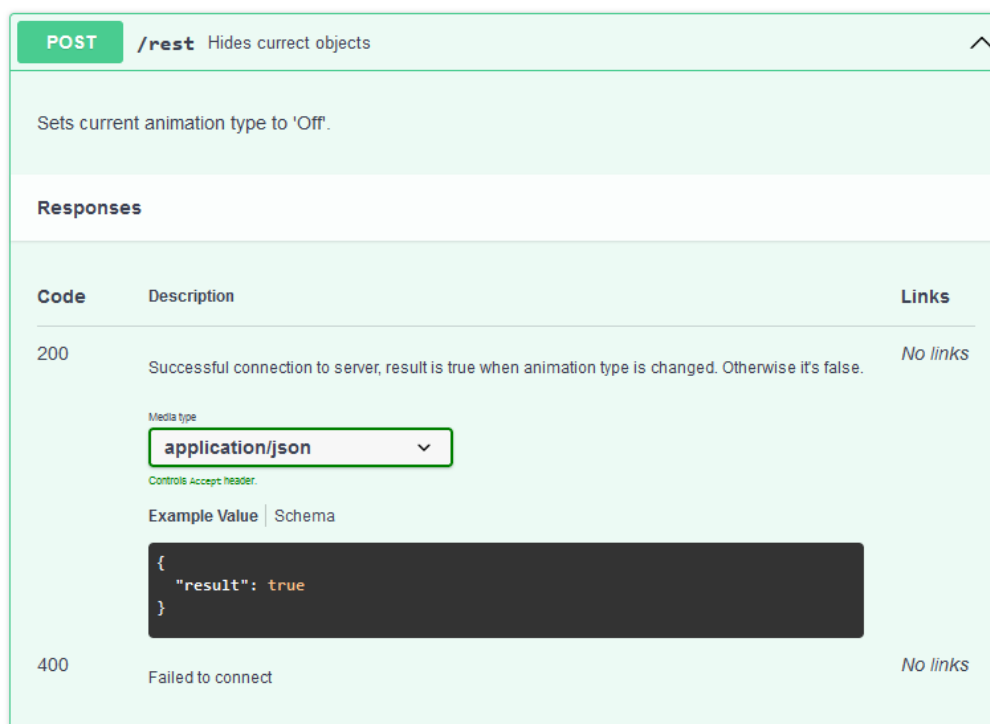
POST **/move** Show animation outside of training sequence

If training is not running, will show animation

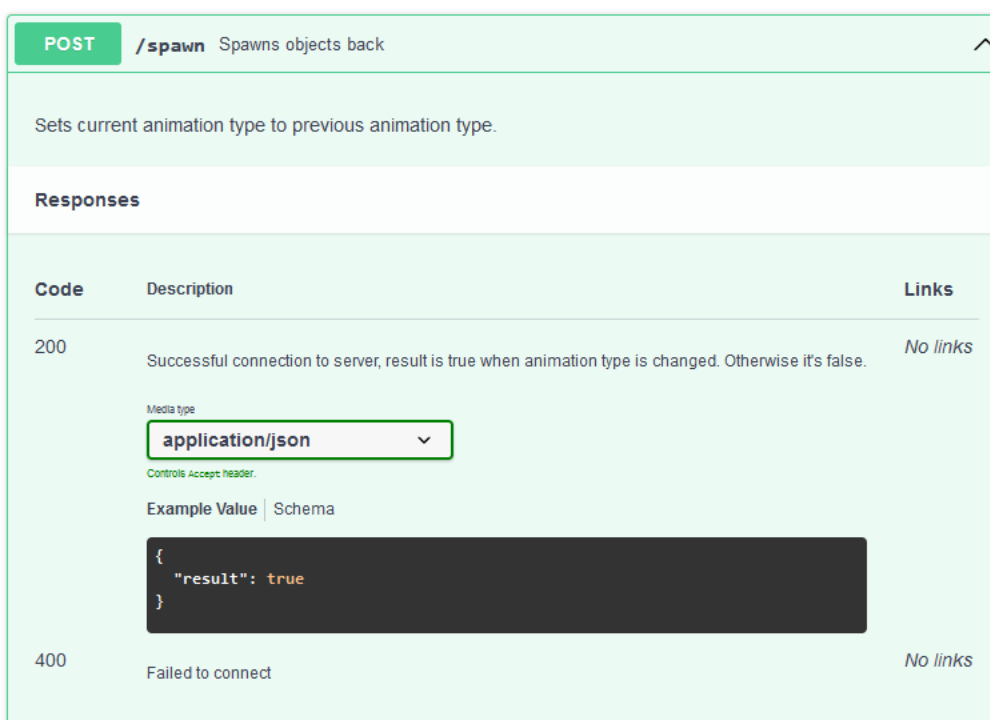
Responses

Code	Description	Links
200	Successful connection to server, result is true when move is successfully started. Otherwise it's false. Media type application/json Controls Accept header. Example Value Schema <pre>{ "result": true }</pre>	No links
400	Failed to connect	No links

Obrázok B.4: API dokumentácia - /move



Obrázok B.5: API dokumentácia - /rest



Obrázok B.6: API dokumentácia - /spawn

B.7 Referenčná dokumentácia WebSocket protokolu

Táto časť obsahuje referenčnú dokumentáciu WebSocket protokolu používaného serverom NeuroRehabVR. Slúži ako integračná špecifikácia pre externých kon-

zumentov, predovšetkým OpenViBE/EEG klienta. Architektúra, návrh a zdôvodnenie rozhodnutí sú popísané v kapitole ??.

B.7.1 Pripojenie a životný cyklus

Adresa servera

WebSocket server počúva na adrese:

`ws://{bindAddress}:{port}`

Hodnoty sú odvodené z konfiguračného objektu `NetworkSettings`:

- **Port** — `WebSocketPort`, predvolená hodnota 7778. Možno prepísať parametrom príkazového riadku `-socketport {port}`.
- **Bind adresa:**
 - prázdna alebo medzera → `0.0.0.0`,
 - `localhost` → `127.0.0.1`,
 - inak sa použije explicitne nakonfigurovaná IP adresa.

Šifrovaný kanál `wss://` nie je v aktuálnej verzii podporovaný (viď sekciu ??).

Životný cyklus servera

1. Server sa spúšťa v `CustomNetworkManager.OnStartServer()` volaním `WebSocketManager.StartServer()`.
2. Pri štarte sa čistia všetky aktívne session objekty.
3. Server sa zastavuje v `CustomNetworkManager.OnStopServer()` alebo pri zničení objektu v `WebSocketManager.OnDestroy()`.

Transportné správanie

Transportná vrstva je implementovaná pomocou knižnice `Fleck` (viď sekciu ??).

- Prichádzajúce správy sú parsované ako JSON do obálky `WebSocketMessage`.
- Správy presahujúce `MaxMessageBytes` sú zahodené bez odpovede.
- Neplatný JSON je zahodený bez odpovede.

B.7.2 Formát správ

Všetky správy v oboch smeroch používajú rovnakú JSON obálku:

Výpis B.2: Štruktúra obálky WebSocket správy

```
{
  "roomId":    "room-123",
  "eventType": "client.checkin",
  "timestamp": 1700000000000,
  "payload": {
    "clientType": "eeg"
  }
}
```

Polia obálky

roomId (*string*, povinné) — identifikátor miestnosti; slúži na smerovanie a autorizáciu.

eventType (*string*, povinné) — typ udalosti; rozlišuje veľké a malé písmená; musí zodpovedať jednej z podporovaných udalostí (sekcia [B.7.4](#)).

timestamp (*number*) — unixové milisekundy. Nastavuje odosielateľ; server generuje vlastnú hodnotu pri odchádzajúcich správach. Server túto hodnotu nevaliduje.

payload (*object* alebo *null*) — telo správy. Povinné pre handlersy vyžadujúce typovaný obsah (viď sekciu [B.7.4](#)). Ak je prítomné, musí byť objekt.

Dvojfázová deserializácia

Transportná vrstva vždy najprv deserializuje obálku. Pole `payload` sa ponechá ako surový JSON objekt a deserializuje sa na konkrétny typ až po výbere handlera podľa `eventType`.

B.7.3 Relácie a autorizačné pravidlá

Požiadavka na check-in

Prvá správa odoslaná klientom **musí** byť `client.checkin`. Ak klient odošle akúkoľvek inú správu pred úspešným check-inom, server odpovie `server.error` a správu zahodí.

Viazanosť na miestnosť

Po úspešnom check-ine sa spojenie naviaže na jednu roomId. Ak klient neskôr odošle správu s odlišnou roomId, server odpovie `server.error` a správu zahodí.

Opätovné pripojenie

Ak sa rovnaké identifikačné spojenie prihlási opätovne, server ho považuje za *re-connect* a vracia stav `status = "reconnected"` v správe `client.checkin.ack`.

B.7.4 Katalóg udalostí

Táto sekcia uvádza kompletný zoznam podporovaných udalostí. Udalosť neuvedená v tomto zozname sa považuje za nepodporovanú a server na ňu odpovie `server.error`.

Klient → Server (prichádzajúce)

client.checkin Registrácia klienta do miestnosti a priradenie typu zariadenia.

Pole payloadu	Popis
<code>clientType (string enum)</code>	"patient" "therapist" "eeg"

Odpoveď: `client.checkin.ack` pri úspechu; `server.error` pri zlyhaní validácie.

Validačné pravidlá:

- `payload` musí byť prítomný,
- `clientType` nesmie byť `Unknown`,
- `roomId` musí existovať na serveri.

mi.stream.fatigue Odoslanie priebežnej EEG/MI hodnoty na server.

Pole payloadu	Popis
<code>value (float)</code>	nameraná EEG/MI hodnota

Odpoveď: žiadna pri úspechu; `server.error` pri neplatnom vstupe alebo chýbajúcej autorizácii.

Validačné pravidlá:

- payload musí byť prítomný,
- roomId musí existovať na serveri.

exercise.action.move Spustenie jedného kroku cvičenia v miestnosti.

Pole payloadu	Popis
<i>prázdny objekt alebo vynechaný</i>	

Odpoveď: žiadna pri úspechu; `server.error` pri neplatnom vstupe alebo chýbajúcej autorizácii.

Validačné pravidlá:

- roomId musí existovať na serveri,
- ak je payload prítomný, musí byť prázdny objekt (`{}`).

Server → Klient (odchádzajúce)

client.checkin.ack Potvrdenie registrácie klienta.

Pole payloadu	Popis
<code>clientType (string enum)</code>	"patient" "therapist" "eeg"
<code>status (string enum)</code>	"connected" "reconnected"

mi.threshold.set Nastavenie aktuálnej MI threshold hodnoty pre miestnosť. Server túto správu odosiela automaticky po úspešnom check-ine EEG klienta, bezprostredne po `client.checkin.ack`.

server.error Štandardizovaná chybová odpoveď.

Pole payloadu	Popis
<code>error (string enum)</code>	"generic"
<code>type (string, voliteľné)</code>	eventType, ktorý chybu spôsobil
<code>description (string)</code>	textový popis chyby

exercise.rest.start / exercise.rest.stop Notifikácia klientov v miestnosti o začiatku, resp. ukončení odpočinkovej fázy. Payload je prázdny objekt.

exercise.eval.start / exercise.eval.stop Notifikácia klientov v miestnosti o začiatku, resp. ukončení fázy EEG vyhodnocovania. Payload je prázdny objekt.

B.7.5 Spracovanie chýb

Server odpovedá správou `server.error` v nasledujúcich prípadoch:

- chýbajúci alebo nepodporovaný `eventType`,
- neplatný alebo chýbajúci `payload` pre typované handlers,
- neplatná alebo neexistujúca `roomId`,
- akákoľvek správa odoslaná pred úspešným check-inom,
- *cross-room* správa (nesúladiť `roomId` so `session`),
- validačné alebo prevádzkové chyby handlera.

Transportné chyby **negenerujú** odpoveď:

- neplatný JSON,
- správa presahujúca `MaxMessageBytes`.

B.7.6 Limity a výkonnosť

Prevádzkové limity sú definované v `NetworkSettings.Adapter`:

Tabuľka B.2: Prevádzkové limity WebSocket servera

Parameter	Hodnota	Popis
<code>MaxMessageBytes</code>	1 MB	maximálna veľkosť jednej správy
<code>MaxQueueActionsPerFrame</code>	500	max. akcií spracovaných za snímok
<code>QueueWarnThreshold</code>	2 000	varovná hranica veľkosti frontu

Prichádzajúce správy sú dispečované v hlavnom vlákne Unity s časovým rozpočtom ~5 ms na snímok. Pri záťaži sa správy odkladajú do nasledujúcich snímkov.

B.7.7 Príklady správ

Výpis B.3: Check-in — požiadavka

```
{
  "roomId": "room-123",
  "eventType": "client.checkin",
  "timestamp": 1700000000000,
  "payload": {
    "clientType": "eeg"
  }
}
```

Výpis B.4: Check-in — potvrdenie

```
{
  "roomId": "room-123",
  "eventType": "client.checkin.ack",
  "timestamp": 1700000000100,
  "payload": {
    "clientType": "eeg",
    "status": "connected"
  }
}
```

Výpis B.5: EEG/MI stream

```
{
  "roomId": "room-123",
  "eventType": "mi.stream.fatigue",
  "timestamp": 1700000000200,
  "payload": {
    "value": 0.42
  }
}
```

Výpis B.6: Spustenie kroku cvičenia (prázdny payload)

```
{
  "roomId": "room-123",
  "eventType": "exercise.action.move",
  "timestamp": 1700000000300,
  "payload": {}
}
```

Výpis B.7: Serverová notifikácia o začiatku odpočinku

```
{
  "roomId": "room-123",
  "eventType": "exercise.rest.start",
  "timestamp": 1700000000400,
  "payload": {}
}
```

Výpis B.8: Chybová odpoveď

```
{
  "roomId": "room-123",
  "eventType": "server.error",
  "timestamp": 17000000000500,
  "payload": {
    "error": "generic",
    "type": "mi.stream.fatigue",
    "description": "Check-in required before sending messages."
  }
}
```