ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA STROJNÍ

Studijní program: **N0715A270012 – Strojní inženýrství**
Studijní obor: **Průmyslové inženýrství a management**

DIPLOMOVÁ PRÁCE

Virtuální kolaborativní trénink průmyslových procesů

Autor: Bc. Kryštof KORTUS

Vedoucí práce: Doc. Ing. Petr HOŘEJŠÍ, Ph.D.

Konzultant: Ing. Matěj DVOŘÁK

Akademický rok 2023/2024

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta strojní
Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Kryštof KORTUS**
Osobní číslo:	**S22N0068P**
Studijní program:	**N0715A270012 Průmyslové inženýrství a management**
Téma práce:	**Virtuální kolaborativní trénink průmyslových procesů**
Zadávající katedra:	**Katedra průmyslového inženýrství a managementu**

Zásady pro vypracování

1. Úvod
2. Obdobné světové aplikace
3. Shrnutí hlavních možností existujících řešení
4. Analýza vlastního řešení
5. Popis vývoje vlastního řešení
6. Popis funkcionality vlastního řešení
7. Závěr

Rozsah diplomové práce: **50 až 70 stran**
Rozsah grafických prací: **–**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Okita, A., Learning C# Programming with Unity 3D, second edition, Routledge 2019, ISBN-13: 978--1138336810
2. Sung, K., Smith, G., Basic Math for Game Development with Unity 3D: A Beginner's Guide to Mathematical Foundations, Apress 2019, ISBN 978-1484254424
3. Linowes, J., Unity 2020 Virtual Reality Projects: Learn VR development by building immersive applications and games with Unity 2019.4 and later versions, 3rd Edition, Packt Publishing 2020, ISBN 978-1839217333
4. LaValle, S. M., Virtual Reality, Cambridge University Press 2023, dostupné online na http://lavalle.pl/vr/
5. Oficiální Unity3D návody dostupné na https://learn.unity.com/

Vedoucí diplomové práce: **Doc. Ing. Petr Hořejší, Ph.D.**
Katedra průmyslového inženýrství a managementu

Konzultant diplomové práce: **Ing. Matěj Dvořák**
Katedra průmyslového inženýrství a managementu

Datum zadání diplomové práce: **16. října 2023**
Termín odevzdání diplomové práce: **24. května 2024**

L.S.

Doc. Ing. Vladimír Duchek, Ph.D.
děkan

Doc. Ing. Michal Šimon, Ph.D.
vedoucí katedry

Prohlášení o autorství

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě strojní Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

V Plzni dne: ……. …….

podpis autora

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu diplomové práce, panu Doc. Ing. Petrovi Hořejšímu, Ph.D. a svému konzultantovi Ing. Matěji Dvořákovi za jejich ochotu a podporu při vypracování mé diplomové práce. Hlavně bych chtěl poděkovat za jejich konzultace, které byly vždy okamžité a postačují k dokončení mé diplomové práce.

ANOTAČNÍ LIST DIPLOMOVÉ PRÁCE

AUTOR	**Příjmení** Kortus	**Jméno** Kryštof	
STUDIJNÍ PROGRAM	N0715A270012 Průmyslové inženýrství a management		
VEDOUCÍ PRÁCE	**Příjmení (včetně titulů)** Doc. Ing. Hořejší, Ph.D.	**Jméno** Petr	
PRACOVIŠTĚ	ZČU - FST - KPV		
DRUH PRÁCE	**DIPLOMOVÁ**	~~**BAKALÁŘSKÁ**~~	**Nehodící se škrtněte**
NÁZEV PRÁCE	Virtuální kolaborativní trénink průmyslových procesů		

FAKULTA	strojní		**KATEDRA**	KPV		**ROK ODEVZD.**	2024

POČET STRAN (A4 a ekvivalentů A4)

CELKEM	89		**TEXTOVÁ ČÁST**	89		**GRAFICKÁ ČÁST**	0

STRUČNÝ POPIS (MAX 10 ŘÁDEK) **ZAMĚŘENÍ, TÉMA, CÍL POZNATKY A PŘÍNOSY**	Diplomová práce se zaměřuje na vývoj a implementaci aplikace pro virtuální kolaborativní trénink průmyslových procesů, využívající technologii virtuální reality na platformě Android pro zařízení Meta Quest 2 a Meta Quest 3. Cílem je poskytnout efektivní, bezpečný a interaktivní způsob tréninku bez fyzických rizik spojených s reálným průmyslovým prostředím. Práce přináší nové poznatky o integraci VR technologií v průmyslovém vzdělávání a demonstruje, jak může virtuální realita zlepšit přístup k tréninku, snížit náklady a zvýšit bezpečnost pracovníků.
KLÍČOVÁ SLOVA **ZPRAVIDLA JEDNOSLOVNÉ POJMY, KTERÉ VYSTIHUJÍ PODSTATU PRÁCE**	VR, AR, Unity 3D, Photon, RPC

SUMMARY OF DIPLOMA SHEET

AUTHOR	**Surname** Kortus	**Name** Kryštof	
STUDY PROGRAMME	N0715A270017 Design engineering of machines and technical devices		
SUPERVISOR	**Surname (Inclusive of Degrees)** Doc. Ing. Hořejší, Ph.D.	**Name** Petr	
INSTITUTION	ZČU - FST - KPV		
TYPE OF WORK	**DIPLOMA**	~~**BACHELOR**~~	**Delete when not applicable**
TITLE OF THE WORK	Virtual Collaborative Industrial Process Training		

FACULTY	Mechanical Engineering		**DEPARTMENT**	Machine Design		**SUBMITTED IN**	2024

NUMBER OF PAGES (A4 and eq. A4)

TOTALLY	89		**TEXT PART**	89		**GRAPHICAL PART**	0

BRIEF DESCRIPTION **TOPIC, GOAL, RESULTS AND CONTRIBUTIONS**	This thesis focuses on the development and implementation of an application for virtual collaborative training of industrial processes, utilizing virtual reality technology on the Android platform for Meta Quest 2 and Meta Quest 3 devices. The goal is to provide an effective, safe, and interactive method of training without the physical risks associated with real industrial environments. The work brings new insights into the integration of VR technologies in industrial education and demonstrates how virtual reality can improve access to training, reduce costs, and increase worker safety.
KEY WORDS	VR, AR, Unity 3D, Photon, RPC

Obsah

Seznam obrázků

Seznam tabulek

Použité zkratky

CAD (Computer-Aided Design) - počítačem podporované projektování
3D (Three Dimensions) – tři dimenze
VR (Virtual Reality) – virtuální realita
AR (Augmented Reality) – rozšířená realita
XR (Extended Reality) – rozšířená realita
VRTK (Virtual Reality Toolkit) - sada nástrojů a knihoven pro vývoj virtuální reality
VE (Virtual Environment) – virtuální prostředí
CVE (Collaborative Virtual Environment) - kolaborativní virtuální prostředí
HMD (Head-Mounted Display) – displejová náhlavní souprava
LAN (Local Area Network) – lokální počítačová síť
EOS (Epic Online Services) – epic online servis
P2P (peer-to-peer) – klient – klient (počítačová síť)
UE4 – Unreal Engine 4
VC (Virtual Collaboration) – virtuální spolupráce
UX (User Experience) – uživatelská zkušenost
OVR (Oculus Virtual Reality) – Oculus virtuální realita
SDK (Software Development Kit) – softwarový vývojářský balíček
RPC (Remote Procedure Call) – vzdálené volání procedur

Úvod

V posledních letech došlo k rapidnímu vývoji technologií virtuální reality (VR) v mnoha odvětvích, včetně průmyslu.[1, 2] Průmyslové podniky stojí před výzvou adaptace na rychle se měnící tržní podmínky, což zahrnuje neustálé zlepšování dovedností jejich pracovní síly.[3] Tradiční metody průmyslového tréninku jsou často nákladné, časově náročné a mohou nést riziko chyb, které mohou vést k finančním ztrátám nebo bezpečnostním incidentům. V tomto kontextu přináší virtuální realita řešení, které může transformovat způsob, jakým jsou pracovníci školeni a jak spolupracují.[4, 5]

Synchronní a asynchronní spolupráci podporují kolaborativní virtuální prostředí (CAVE), která mohou rovněž zlepšit kvalitu interakcí, výměny znalostí a komunikace mezi různými zúčastněnými stranami a mezioborovými týmy.[6–9] Několik výzkumných studií[10, 11] uvádí, že průmysloví pracovníci mají na technologie virtuální reality příznivý názor, což vedlo k nárůstu jejich motivace k jejich využívání. Podle nedávné studie[6] mohou rozhodování v raných fázích návrhu účinně podpořit imerzní aplikace VR, které rovněž zvyšují zapojení týmu. Podobně jako jiná studie[12] ukázala, jak systém VR usnadňuje ověřování postupů instalace, údržby i komunikaci mezi inženýry a montážními operátory.

Diplomová práce "Virtuální kolaborativní trénink průmyslových procesů" se zaměřuje na vývoj a implementaci aplikace pro VR, která umožní uživatelům trénovat a spolupracovat na průmyslových procesech v bezpečném, kontrolovaném a imerzním prostředí. Cílem je nejen snížení nákladů na trénink a eliminace rizik spojených s fyzickým školením, ale také zvýšení efektivity učení díky využití interaktivních a poutavých VR technologií.[10, 13]

Uveďme si cíle práce:

1. Analyzovat současný stav a potřeby tréninku v průmyslu.
2. Navrhnout uživatelské rozhraní a interakce pro efektivní kolaboraci a učení.
3. Implementovat systém na platformě Android kompatibilní s brýlemi pro virtuální realitu.
4. Otestovat efektivitu aplikace ve spolupráci s průmyslovými partnery.
5. Zhodnotit přínosy aplikace pro průmyslové podniky a uživatele.

Praktická část diplomové práce se opírá o teoretické základy kolaborativní práce a pedagogických principů virtuálního učení, zkoumá technologické možnosti VR a jejich aplikaci v průmyslovém tréninku. V rámci diplomové práce tedy budou analyzována stávající řešení, dále budou identifikovány jejich nedostatky a bude navrženo vlastní softwarové řešení Zkoumání se zaměří na uživatelské rozhraní, interakční design, simulaci realistických průmyslových prostředí a integraci týmové spolupráce do virtuálního prostoru.

1. Analýza virtuální kolaborace

Virtuální spolupráce (VC - virtual collaboration) je forma spolupráce, která využívá virtuální realitu (VR) k propojení lidí na dálku. VC umožňuje lidem spolupracovat na společných projektech, aniž by museli být fyzicky v jednom prostoru.

1.1. Uživatelská zkušenost ve virtuální spolupráci

Uživatelská zkušenost (UX – user experience) je důležitým faktorem při určování úspěchu virtuální spolupráce (VC). UX ve VC se zaměřuje na způsob, jakým uživatelé interagují s VR kolaborativní prostorem. UX by měla být snadno ovladatelná, intuitivní a pohlcující.

Snadná ovladatelnost je klíčová. VC systémy by měly být snadno ovladatelné, aby uživatelé mohli rychle a snadno provádět požadované úkoly. To je důležité zejména pro uživatele, kteří nejsou obeznámeni s VR nebo kteří mají omezené fyzické schopnosti. Intuitivní ovládání je rovněž důležité. Ovládací prvky VR systémů by měly být intuitivní, aby uživatelé mohli rychle pochopit, jak je používat. To je zvláště klíčové pro uživatele, kteří chtějí začít pracovat s VC co nejdříve. Pohlcující prostředí je dalším klíčovým prvkem. VC systémy by měly poskytovat pohlcující prostředí, které uživatelům umožňuje cítit se, jako by byli přítomni v daném prostoru. Tento aspekt je důležitý pro zlepšení spolupráce a komunikace mezi uživateli. Kromě snadné ovladatelnosti, intuitivního ovládání a pohlcujícího prostředí existují další faktory, které mohou ovlivnit UX ve VC. Mezi tyto faktory patří kvalita grafického zobrazení, která může ovlivnit to, jak uživatelé vnímají VR prostor, latence sítě, která může způsobit, že interakce v VR prostoru budou zpomalené nebo „trhané“ a pohodlí VR headsetu, které může ovlivnit to, jak dlouho uživatelé mohou VR používat [14–16]

Závěrem lze říct, že UX je důležitým faktorem, který je třeba zvážit při vývoji VR systémů pro spolupráci. VR systémy by měly být snadno ovladatelné, intuitivní, pohlcující a nabízet vysokou kvalitu grafického zobrazení a zpětnou vazbu v reálném čase.

1.2. Prezence a imerze

VR má potenciál zlepšit spolupráci mezi lidmi, kteří jsou od sebe vzdáleni. VR systémy mohou uživatelům poskytnout pocit přítomnosti ve sdíleném prostoru, což může usnadnit komunikaci a spolupráci.

Prezence a imerze jsou klíčové pojmy. Prezence je pocit, že jste v místě, které není skutečné, zatímco imerze je pocit, že jste pohlceni něčím, co vás obklopuje. Obě tyto koncepce jsou důležité pro VR spolupráci, neboť mohou pomoci uživatelům ke společnému pracovnímu prostoru. [15, 16]

Existuje několik výhod VR spolupráce oproti běžným technikám, jako jsou videokonference nebo telefonní hovory. VR systémy mohou uživatelům poskytnout vyšší pocit přítomnosti a imerze, díky čemuž se cítí jako ve sdíleném prostoru, což usnadňuje komunikaci a spolupráci. Kromě toho mohou VR systémy usnadnit sdílení informací a návodů, což vede k vyšší míře spolupráce. Navíc mohou uživatelům pomoci soustředit se na práci a eliminovat rušivé faktory, což zvyšuje jejich produktivitu. [17]

Nevýhody VR spolupráce zahrnují náročnost na hardware, jelikož VR systémy mohou být nákladné a vyžadují výkonný hardware. Dále je tu nedostatek sdílených nástrojů, protože existuje jejich omezený počet pro VR spolupráci. Také se může vyskytnout závrať, když někteří uživatelé používají VR systémy. [18]

Závěrem lze říct, že kolaborace ve VR má potenciál zlepšit spolupráci mezi lidmi, kteří jsou od sebe vzdáleni, a poskytnout uživatelům pocit přítomnosti a imerze, což může usnadnit komunikaci a spolupráci. Nicméně je důležité si být vědom i nevýhod, jako jsou náročnost na hardware a nedostatek sdílených nástrojů.

1.3. Nástroje virtuální kolaborace

VR nabízí řadu nástrojů, které mohou usnadnit spolupráci mezi lidmi, kteří jsou od sebe vzdáleni. Tyto nástroje mohou být použity v široké škále aplikací, od vzdělávání a podnikání až po zdravotnictví a vývoj produktů.

Tabule a ukazovátka jsou jedním z nejběžnějších nástrojů pro spolupráci. Ve VR mohou být virtuální tabule použity ke sdílení nápadů, kreslení diagramů a vytváření prezentací. Ukazovátka mohou být použita k označení objektů nebo k vedení pozornosti ostatních účastníků. 3D modely mohou být použity k vizualizaci komplexních konceptů nebo k simulaci situací. Ve VR mohou být 3D modely použity ke spolupráci na projektech, jako je navrhování produktů nebo plánování stavebních projektů. Avataři jsou virtuální reprezentace uživatelů. Ve VR mohou být avataři použiti k vyjádření identity a k vytvoření pocitu přítomnosti. Audio a video jsou důležitými nástroji pro komunikaci ve VR. Může být použito k hovoru, k přehrávání zvuků a k vytváření atmosféry. Gesta mohou být použita k ovládání VR prostředí a k interakci s ostatními účastníky. [14–17]

Nástroje kolaborace ve VR mohou být využity v široké škále aplikací. Vzdělávání je jednou z oblastí, kde VR může být použita k poskytování interaktivních učebních zkušeností. Učitelé mohou využívat tabule a ukazovátka k vysvětlování konceptů, 3D modely k vizualizaci složitých témat a avatary k interakci s žáky. Podnikání nachází využití ve VR, kde může být použita k usnadnění spolupráce na projektech, jako je návrh produktů nebo jejich plánování. Účastníci mohou používat tabule a ukazovátka ke sdílení nápadů, 3D modely k simulaci situací a avatary k vyjádření identity a k vytvoření pocitu přítomnosti. Ve zdravotnictví může VR sloužit k poskytování terapie a rehabilitace, kde lékaři využívají tabule a ukazovátka k vysvětlování léčebných postupů, 3D modely k simulaci operací a avatary k interakci s pacienty. Vývoj produktů je další oblastí, kde může být VR využita k testování produktů a získání zpětné vazby od uživatelů. Účastníci zde mohou využívat tabule a ukazovátka ke sdílení nápadů, 3D modely k simulaci používání produktů a avatary k interakci s uživateli. [15, 16]

Závěrem lze říct, že VR nástroje mohou pomoci uživatelům cítit se, jako by byli ve sdíleném prostoru, což může usnadnit komunikaci a spolupráci.

1.4. Účinek kolaborace na týmovou spolupráci

Spolupráce ve VR má potenciál zlepšit týmovou spolupráci v řadě oblastí. VR může pomoci týmovým členům lépe komunikovat tím, že poskytne společný prostor pro komunikaci a spolupráci, což je obzvláště užitečné pro týmy, které jsou geograficky vzdálené. Dále může VR

pomoci týmu vizualizovat a interaktivně pracovat s komplexními daty, což usnadňuje porozumění a spolupráci. Například může být použita k simulaci stavebního projektu nebo k vytvoření modelu produktu. Navíc VR může poskytnout platformu pro tým lidí, aby spolupracovali v reálném čase, což je užitečné pro rychlé řešení problémů nebo přijímání rozhodnutí. [15, 17]

Konkrétní příklady toho, jak může být VR použita ke zlepšení týmové spolupráce, jsou různorodé. Ve vzdělávání může VR sloužit k poskytování interaktivních učebních zkušeností, které napomáhají usnadnit spolupráci mezi studenty a učiteli. V podnikání může VR usnadnit spolupráci na projektech, jako je navrhování produktů nebo plánování stavebních projektů. V oblasti zdravotnictví může VR přispět k poskytování terapie a rehabilitace, což podporuje spolupráci mezi lékaři a pacienty. A vývoj produktů může využít VR k testování produktů a sběru zpětné vazby od uživatelů. [15, 17]

1.5. Zabezpečení a etika

Stejně jako u všech technologií, i u VR je důležité zvážit bezpečnostní a etické aspekty jejího používání.

Bezpečnost využívání VR zahrnuje několik klíčových oblastí. Jednou z nich je ochrana osobních údajů, jelikož VR může shromažďovat citlivé informace o uživatelích, jako je jejich poloha, pohyby a hlas. Je důležité, aby tyto údaje byly chráněny před neoprávněným přístupem nebo zneužitím. Současně je třeba brát v úvahu zdravotní rizika spojená s používáním VR, jako je možná únava, nevolnost a další potenciální zdravotní problémy. Dodržování doporučené maximální doby používání podle manuálu hardwaru je klíčové pro minimalizaci těchto rizik. Dále je důležitá i fyzická bezpečnost, protože VR může způsobit, že uživatelé ztratí kontakt se svým okolím. Je nutné používat VR pouze v bezpečném prostředí a být si vědom možných rizik spojených s ignorováním okolí. [18]

Etická hlediska používání VR jsou rovněž důležitá. Je třeba dbát na soukromí, aby nebylo narušováno sledováním nebo špehováním lidí bez jejich souhlasu. Dále je nutné zabránit diskriminaci, která by mohla být podněcována či umožněna prostřednictvím VR, a uplatňovat spravedlivý a inkluzivní přístup. Kromě toho je třeba mít na paměti možnost sociální izolace, kterou může používání VR přinést, a používat tuto technologii způsobem, který podporuje zdravé sociální vztahy a nedává přednost virtuálnímu světu před reálným.

1.6. Technické problémy

Latence sítě, omezení hardwaru a kompatibilita softwaru jsou důležité faktory, které je třeba zvážit při plánování spolupráce ve VR. Tyto faktory mohou ovlivnit vnímanou realitu a plynulost spolupráce.

Latence sítě může mít významný dopad na vnímanou realitu a plynulost spolupráce ve VR. Vysoká latence může vést ke zpožděním uživatelských akcí a pocitu odpojení od ostatních účastníků. Zejména v reálném čase je nízká latence zásadní pro kvalitní spolupráci. Latence může způsobit zpoždění při příjmu a odesílání dat na komunikaci, stejně jako zpoždění při manipulaci se sdílenými objekty. Hardwarová omezení zařízení VR a AR mohou mít významný dopad na rozsah a kvalitu spolupráce. Omezený výpočetní výkon, zorné pole a přesnost sledování mohou omezit počet účastníků, složitost virtuálních prostředí a přesnost interakcí. Nekompatibilní formáty softwaru, komunikační protokoly a uživatelská rozhraní mohou bránit

uživatelům z různých platforem nebo zařízení v účasti na společných zkušenostech v rozšířené realitě (XR). To může vést k tomu, že uživatelé nebudou moci sdílet data, interagovat s virtuálními objekty a efektivně spolupracovat. [14–17]

Obtíže s otevřením souborů vytvořených jinými uživateli, problémy s komunikací s ostatními účastníky a obtíže při manipulaci s virtuálními objekty jsou konkrétními příklady, jak mohou problémy s kompatibilitou softwaru ovlivnit spolupráci v XR.

1.7. Didaktický pohled

Spolupráce ve virtuální a rozšířené realitě (VR/AR) přináší mnoho pedagogických výhod, které lze využít v různých prostředích, od vzdělávání a školení po vzdálenou práci a kreativní projekty. Učitel ve VR může pracovat s jakýmikoliv virtuálními pomůckami a může navazovat na různé scénáře. Vše lze opakovat. Na druhou stranu, vzdělání realizované skutečnou osobou má větší dopad než vzdělání realizované osobou virtuální.

Zvýšené zapojení je jednou z těchto výhod, neboť VR/AR mohou poskytnout pohlcující a poutavý zážitek, což může vést k většímu zapojení účastníků, zvláště v situacích, kde je spolupráce v reálném světě obtížná či neefektivní. Druhou výhodou je zlepšená komunikace, neboť VR/AR mohou usnadnit komunikaci mezi účastníky na různých místech, což je užitečné zejména pro vzdálenou spolupráci a podporu týmů pracujících na dálku. Posílené prostorové porozumění je další výhodou, protože VR/AR mohou účastníkům pomoci lépe pochopit prostorové vztahy, což je užitečné zejména pro úkoly vyžadující spolupráci na společném projektu nebo úkolu. Další výhodou je snížení sociálních bariér, neboť VR/AR mohou přispět ke snížení sociálních rozdílů mezi účastníky, což je užitečné pro skupiny, které jsou jinak odděleny geograficky či kulturně. Zvýšená kreativita je poslední z výhod, neboť VR/AR mohou poskytnout novou platformu pro kreativní spolupráci, což je využitelné zejména při projektech, které vyžadují spolupráci na nových nápadech nebo konceptech. [14–17]

1.8. Případové studie

Existuje celá řada společností, které již běžně využívají virtuální kolaboraci. Tyto společnosti využívají VR a AR k řešení různých problémů a úkolů, jako je design, údržba, výroba a školení.

Společnost Airbus používá VR pro spolupráci na designu letadel. VR umožňuje inženýrům z různých zemí sdílet prostor a spolupracovat na stejném modelu letadla. To může výrazně zkrátit dobu vývoje a zlepšit komunikaci mezi inženýry. Společnost Siemens používá VR pro spolupráci na údržbě zařízení. VR umožňuje technikům sdílet informace a spolupracovat na úkolech, aniž by museli být fyzicky přítomni na místě. To může zlepšit bezpečnost a efektivitu údržby. Společnost Ford používá AR pro spolupráci na výrobě automobilů. AR umožňuje zaměstnancům zobrazovat si pokyny a informace přímo v reálném světě. To může pomoci zaměstnancům provádět úkoly přesněji a rychleji. [19–21]

Tyto případové studie ukazují, že VR/AR má potenciál být cenným nástrojem pro spolupráci v průmyslu. Může pomoci zlepšit efektivitu, bezpečnost a komunikaci v různých oblastech průmyslu.

2. Obdobné světové aplikace

V předchozí kapitole jsme prozkoumali základní aspekty virtuální kolaborace. Nyní se podíváme na konkrétní aplikace, které v průmyslu využívají tato řešení. Zabývat se budeme především funkcionalitou a možnostmi využití a okrajově budou zmíněny hardwarové, softwarové a serverové řešení.

Spolupráce týmu na jednom místě má klíčový vliv na zlepšení týmové dynamiky a efektivity. V dnešní době podniky i jednotlivci využívají digitální média, jako je internet, k práci, učení, zábavě a objevování nových, personalizovaných zdrojů [22]. V podobném duchu byly promyšleně vytvořeny četné platformy a nástroje pro komunikaci, které umožňují spojení mezi jednotlivci po celém světě v reálném čase. Jedním z těchto nástrojů je vytváření 3D virtuálních světů. Virtuální světy se v průběhu let rozvinuly díky počítačovým simulacím, které vytvářejí neuvěřitelně realistické vizuální efekty, jež působí autenticky. Obsáhlý přehled virtuálních světů je uveden v [23].

Virtuální realita (VR) prokázala schopnost pomáhat v návrhové fázi vývojového cyklu výrobku[6, 24] a zahrnuje následující scénáře: společné přezkoumání návrhu, imerzní testování výrobku, řízení výrobku a hodnocení výrobního procesu.[8] Protože VR nabízí lepší vizualizační schopnosti a přirozený způsob interakce s virtuálními věcmi v měřítku 1:1 podobném realitě, může být schopna zkrátit dobu návrhu a eliminovat jeho chyby. V rané fázi procesu vývoje výrobku zlepšují společná hodnocení návrhu pro udržitelnost, což má pozitivní vliv na optimalizaci návrhu a snižuje celkové náklady. [2]

Obr. 2-1 Ukázka virtuální reality v průmyslu [25]

2.1. COVE-VR platforma

COVE-VR je platforma pro virtuální spolupráci, která umožňuje uživatelům z různých míst spolupracovat na společných projektech ve virtuální realitě. Platforma nabízí řadu funkcí, které usnadňují a zpříjemňují spolupráci.

Sdílení virtuálního prostoru umožňuje uživatelům spolupracovat na stejných objektech a procesech. Synchronizace stavu zajišťuje, že stav virtuálního prostředí je synchronizován mezi všemi uživateli, a tak všichni vidí stejné věci. Hlasový chat, jako klíčová funkce, umožňuje uživatelům efektivně komunikovat a koordinovat činnosti, což zásadně usnadňuje spolupráci. Sdílení obsahu zpřístupňuje uživateli text, obrázky a videa od jiných uživatelů, což usnadňuje komunikaci a spolupráci. Platforma COVE-VR je navržena tak, aby podporovala asynchronní spolupráci např. mezi globálními týmy pracujícími v různých časových pásmech. To umožnilo týmům pracovat samostatně a zanechat své komentáře a poznámky ostatním, aby si je mohli prohlédnout, uložit a později v nich pokračovat. Jsou také schopni vytvářet digitální obsah, jako jsou texty, obrázky a videa, které lze opět použít k předání pokynů nebo k dalšímu dialogu. [26]

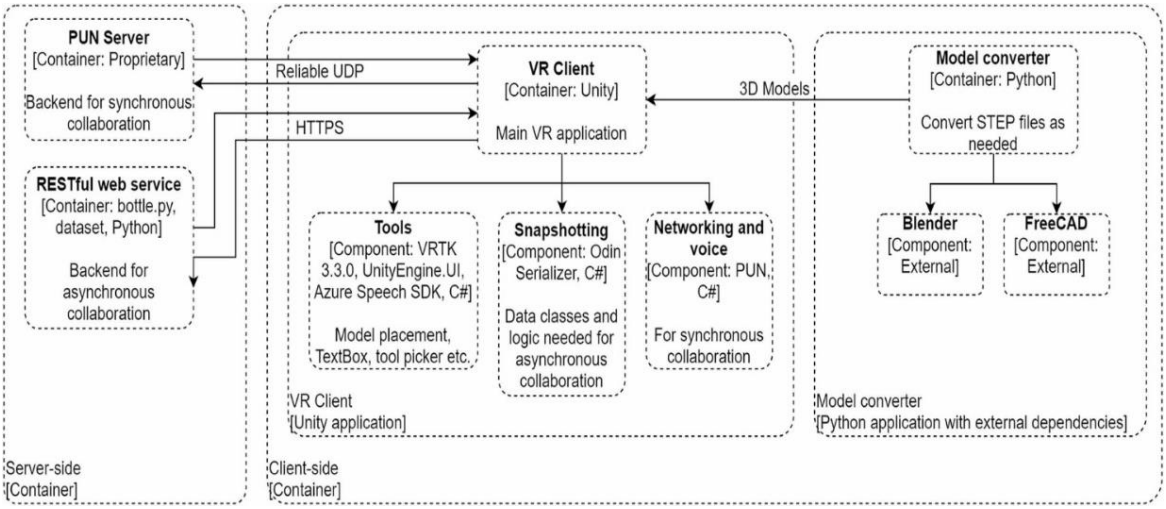
COVE-VR je cenným nástrojem pro týmy, které potřebují spolupracovat na společných projektech, i když se nacházejí v různých časových pásmech.

2.1.1 Hardwarové, softwarové a serverové řešení

Aplikace COVE-VR je založena na robustní softwarové a hardwarové architektuře. Využívá komerční hotové komponenty, které poskytují spolehlivé a výkonné řešení. Článek [26] neuvádí, na jakých zařízeních byla aplikace testována, ani minimální hardwarové požadavky pro její plynulý chod.

K vývoji VR klienta byl použit herní engine Unity 3D a sada nástrojů VRTK 3.3.0, protože obsahují většinu součástí potřebných pro VR aplikaci, jako je vykreslování, fyzika objektů, skriptovací spouštění, vizuální editor, vstupní systém, importéry 3D modelů, multiplatformní sestavovací systém a komponenty uživatelského rozhraní VR. Protože server PUN nepodporuje dlouhodobou perzistenci dat, byla dále využita PUN k synchronizaci činnosti v prostředí mezi více uživateli (v rámci relace) a vytvořila se webová služba RESTful pro ukládání stavu VE mezi relacemi k asynchronní spolupráci. [26]

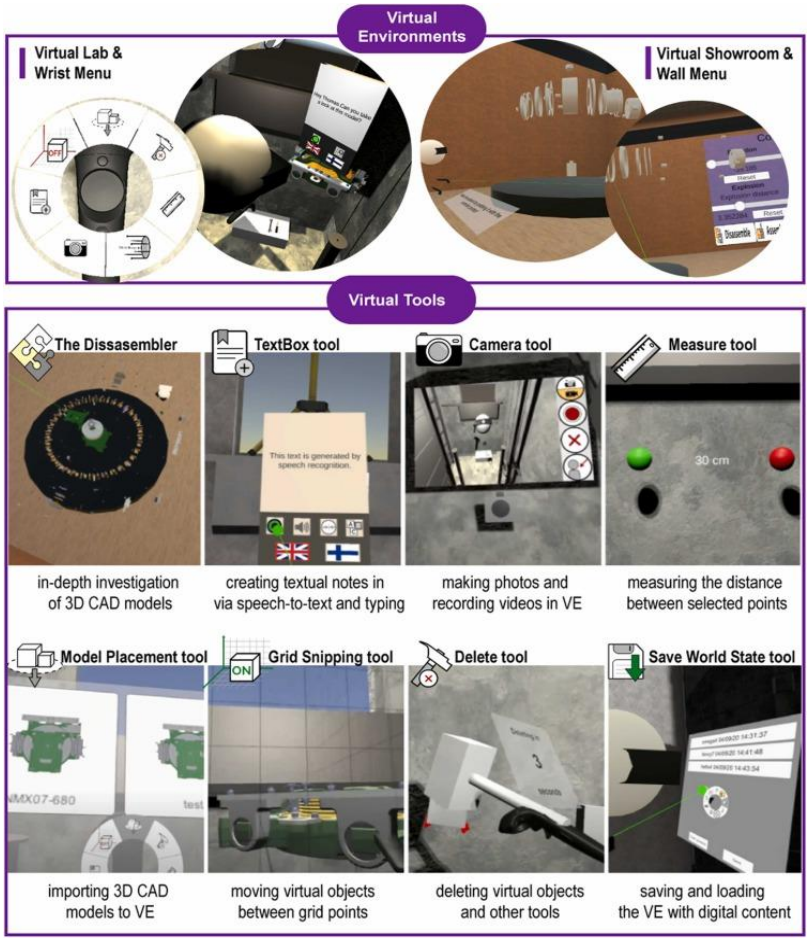
Systém COVE-VR funguje na architektuře klient-server, kde asynchronní a synchronní spolupráci řídí dva různé servery (Obr. 2-2). Vyvinutý převodník samotného modelu, webová služba RESTful, klient VR a server PUN (Photon Unity Networking), což je hotová komerční komponenta. Společně tyto komponenty tvoří systém. [26]



Obr. 2-2 Architektura klient-server [26]

2.1.1 Funkce a možnosti využití

Z hlediska uživatelského ovládání aplikace využívá standardní ovládací prvky VR, jako jsou pohybová gesta, ovládání pomocí ovladačů a hlasové ovládání. Ovládací prvky v aplikaci jsou intuitivní a snadno se používají. Platforma COVE-VR zahrnuje osm nástrojů, které jsou uživatelům k dispozici, a dva předefinované virtuální prostory pro podporu průmyslových situací. Na Obr. 2-3 jsou zobrazeny součásti platformy.



Obr. 2-3 COVE-VR součásti platformy [26]

Virtuální prostor je kompaktní pracovní prostor určený pro samostatnou i společnou práci. Aby byl umožněn bezpečný přístup do virtuálního prostoru - proces, který je v reálném životě nebezpečný a časově náročný - napodobuje skutečné pracovní prostředí, kterým je výtahová šachta založená na aktuálním 3D modelu CAD. Showroom je větší prostor určený k prezentaci klientů a ke spolupráci. Demontážní zařízení, které je součástí vybavení Showroomu, umožňuje hloubkové zkoumání 3D modelů jejich rozložením na jednotlivé součásti a nastavením jejich velikosti, natočení a vertikální polohy (prostřednictvím nástěnného menu). [26]

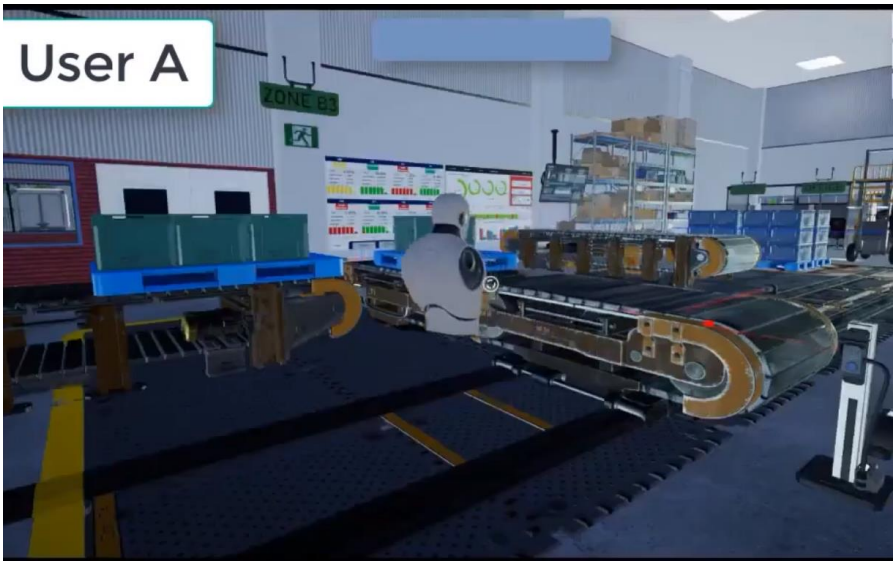
Menu na zápěstí virtuální ruky poskytuje přístup k osmi nástrojům, které usnadňují práci ve virtuálním prostředí. Kdekoli ve virtuálním prostředí lze importovat 3D modely CAD pomocí funkce Umístění modelu. Díky nástroji TextBox lze vytvářet textové poznámky pomocí rozpoznávání hlasu nebo psaním na simulované klávesnici. Pomocí nástroje Fotoaparát lze pořizovat snímky a videa. Otevírá se v režimu selfie a má vestavěný časovač. Nástroj Vyjmutí mřížky dodává přesnost tím, že umožňuje přesouvat objekty přes body mřížky, zatímco nástroj Měření určuje délky mezi dvěma body. Uživatelé mohou odstranit virtuální objekty i ostatní nástroje pomocí nástroje Odstranit a mohou nahrát existující "uložené prostředí" nebo uložit prostředí včetně všech vytvořených materiálů pomocí nástroje Uložit stav světa. Vzhledem k tomu, že veškerý obsah vytvořený ve virtuální realitě je uložen do složky na pevném disku a je přístupný později, je snadné používat obsah (poznámky, videa a obrázky) ve standardním kancelářském softwaru a dalších programech.[26]

2.1.2 Výsledné hodnocení aplikace

COVE-VR je praktický software, který řeší řadu aktuálních problémů souvisejících s procesy. Při interním testování na dvou testovacích skupinách, lze konstatovat, že obě úspěšně splnily úkoly, vytvořily vysvětlující poznámky a relevantní digitální obsah, který usnadňuje asynchronní spolupráci. Odborníci a testovací skupiny zdůrazňovali hodnotu systému při usnadňování jejich komunikace. Například vedoucí uživatel v aplikaci mohl místo rozesílání textových vysvětlivek e-mailem pořídit video ve VR, na němž je zobrazen 3D objekt, a ústně vysvětlit metodu. Kromě toho je možné pořídit snímek součásti z požadovaného úhlu, který může jiný pracovník použít jako vodítko pro vytvoření vektorového obrázku. [26]

2.2. Smart Factory VR

Smart Factory VR je aplikace ve virtuální realitě, která umožňuje uživatelům prozkoumat a ovládat chytrou továrnu. Využívá technologii VR k vytvoření realistického a pohlcujícího zážitku, který uživatelům umožňuje cítit se, jako by se opravdu nacházeli v továrně. Aplikace obsahuje řadu funkcí, které umožňují uživatelům prozkoumat továrnu různými způsoby. Může například uživatele přenášet do různých částí továrny, umožnit jim prohlížet si zařízení a procesy, a dokonce jim umožnit ovládat továrnu. Smart Factory VR má potenciál být použita pro řadu účelů, včetně školení, vzdělávání a spolupráce. Může například být použita k výuce nových zaměstnanců, k poskytnutí studentům pohledu dovnitř továrny a k umožnění odborníkům z různých oblastí spolupracovat na zlepšení výrobních procesů.[27]



Obr. 2-4 Ukázka vytvořené aplikace Smart Factory VR -pohled uživatele B na uživatele A [27]

2.2.1. Hardwarové, softwarové a serverové řešení

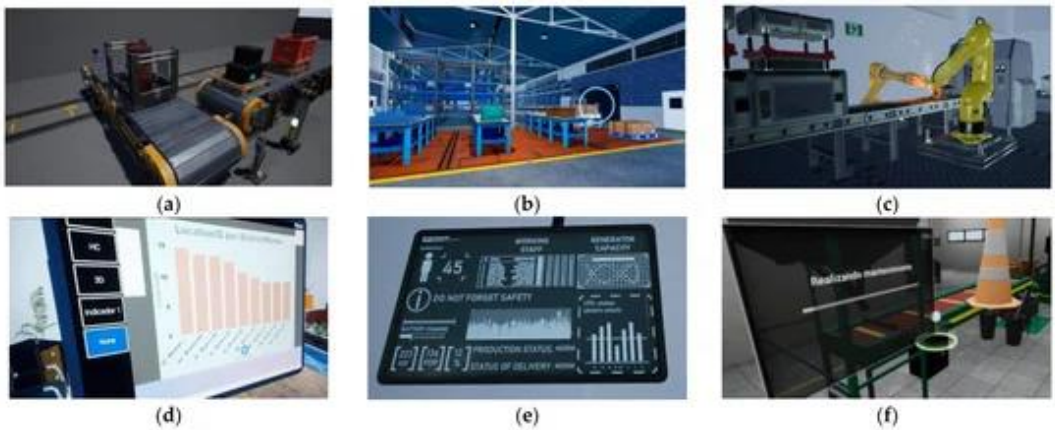
Stejně jako předchozí aplikace tak i Smart Factory VR je založena na robustní hardwarové a softwarové architektuře. Využívá komerční hotové komponenty a funkce, které poskytují spolehlivé a výkonné řešení. Prostředí VR bylo nastaveno pomocí náhlavních souprav Meta Quest 2 a Oculus Rift Sy Rift VR a počítačů se specifikacemi (CPU AMD Ryzen 7-5800H řady Octacore 5000 a GPU NVIDIA GeForce RTX3060 (6 GB-GDDR6), paměť RAM: 16 GB (8 GB DDR4-3200 SO-DIMM *2)). Aplikace Smart Factory VR byla vytvořena v herní systému Unreal Engine 4 (UE4) ve verzi 4.26. Součástí softwaru pro vývoj videoher UE4 je umělá inteligence v reálném čase, zvukové efekty, skriptované akce a vykreslování 2D nebo 3D obrazu. **Hostitel a server:** Pro víceuživatelskou funkcionalitu byly použity dva externí zásuvné moduly integrované do programového jádra: základní modul EOS, který zobrazuje online služby projektu (EOS). Prostřednictvím zásuvného modulu jádra Vivox je projekt propojen se systémem hlasového chatu Vivox, což umožňuje jeho současné používání více uživateli.[27]

2.2.2. Funkce a možnosti využití

Z hlediska uživatelského ovládání aplikace využívá stejné prvky jako předchozí aplikace COVE-VR. Aplikace obsahuje řadu funkcí, které umožňují uživatelům prozkoumat továrnu různými způsoby.

Umožňuje uživatelům procházet továrnou, jako by se opravdu nacházeli v ní. Mohou se pohybovat po virtuálním prostoru volně nebo mohou použít navigační nástroje k přesnému pohybu do požadované oblasti. Dovoluje uživatelům prohlížet si zařízení a procesy. Mohou vidět, jak zařízení fungují a jak jsou procesy integrovány. Uživatelé mohou ovládat továrnu. Mohou například zapínat a vypínat zařízení, spouštět procesy a provádět změny v konfiguraci. [27]

Aplikace má potenciál být použita pro řadu účelů, včetně školení, vzdělávání a spolupráce. Školení může sloužit k výuce nových zaměstnanců. Mohou se naučit o bezpečnosti, operacích a údržbě. Vzdělávání může studentům poskytnout pohled na průmyslové procesy. Mohou se dozvědět o tom, jak fungují chytré továrny a jak jsou integrovány do globálních dodavatelských řetězců. Spolupráce může usnadnit odborníkům z různých oblastí pracovat společně na zlepšení aktuálních řešení továrny. Mohou například spolupracovat na vývoji nových procesů nebo na řešení problémů.



Obr. 2-5 (a) senzory, aktuátory a radiofrekvenční identifikace (RFID) v dopravních prostředcích; (b) realistické zobrazení skladovacího systému pomocí digitálního dvojčete; (c) autonomní roboty spolupracující na opakujících se úkolech; (d) velká data a analýza dat v reálném čase v procesu; (e) rozhraní člověk-stroj s rozhraními a monitory; (f) údržba strojů a zařízení[27]

2.2.3. Výsledné hodnocení

Aplikace Smart Factory VR je cenným nástrojem pro výrobní podniky. Má potenciál být použita pro řadu účelů, včetně školení, vzdělávání a spolupráce.

Aplikace má následující silné stránky: Používá technologii VR, která poskytuje realistický a pohlcující zážitek, což může uživatelům pomoci lépe pochopit fungování továrny a její ovládání. Obsahuje také řadu funkcí, které umožňují uživatelům prozkoumat továrnu různými způsoby, což může být užitečné pro školení nových zaměstnanců, vzdělávání studentů a spolupráci odborníků na chytré továrny. Aplikace je navíc snadno použitelná, protože uživatelé mohou ovládat aplikaci pomocí jednoduchých gest a ovládacích prvků. Nicméně aplikace má také několik slabých stránek: Je stále ve vývoji, což znamená, že může obsahovat chyby nebo nedostatky. Kromě toho vyžaduje speciální vybavení, jako jsou brýle pro VR a počítač s dostatečným výkonem.

Celkově lze Smart Factory VR považovat za slibný nástroj, který má potenciál přinést výrobním podnikům řadu výhod. Aplikace má potenciál zlepšit školení, vzdělávání a spolupráci ve výrobních podnicích.

2.3. ESI Group a aplikace IC.IDO

IC.IDO je softwarový systém společnosti ESI Group, který lze použít k plánování a optimalizaci montážních procesů. Nabízí řadu funkcí, které pomáhají uživatelům zlepšit ergonomii a bezpečnost montážních procesů.

Software lze použít v rámci virtuální reality k vizualizaci a simulaci montážních procesů. Virtuální realita poskytuje realistickou vizualizaci montážních procesů, která může pomoci uživatelům lépe pochopit procesy a identifikovat potenciální problémy. Virtuální simulace umožňuje uživatelům otestovat montážní procesy v reálném čase, což může pomoci identifikovat potenciální problémy s ergonomií, bezpečností nebo výkonností.[28]

2.3.1. Hardwarové, softwarové a serverové řešení

Aplikace IC.IDO je postavena na silné hardwarové a softwarové architektuře, která je totožná s aplikací COVE-VR. Stejně jako předchozí dvě aplikace využívá IC.IDO komerční hotové komponenty, které poskytují spolehlivé a výkonné řešení.

IC.IDO je multiplatformní aplikace, která lze spouštět na široké škále hardwarových zařízení. Pro optimální výkon se doporučuje následující hardwarová specifikace: Procesor Intel Core i5 nebo vyšší, paměť 8 GB RAM nebo více, grafická karta NVIDIA GeForce GTX 1060 nebo vyšší, displej Full HD nebo vyšší a zvukový systém Stereo. [29] Co se týče softwarového řešení, IC.IDO byl vyvinut v herním enginu Unity 3D a je určen pro platformu Microsoft Windows. Pro instalaci IC.IDO je vyžadován systém Windows 10 nebo novější verze. Pokud jde o serverové řešení, IC.IDO lze nasadit na vlastní server nebo využít cloudové řešení od společnosti ESI Group. [28]

2.3.2. Funkce a možnosti využití

Z hlediska uživatelského ovládání aplikace IC.IDO využívá stejné prvky jako předchozí dvě aplikace.

Aplikace IC.IDO nabízí řadu funkcí a možností využití v rámci virtuální reality, které mohou být použity ke zlepšení ergonomie, bezpečnosti a produktivity montážních procesů.

Mezi hlavní funkce aplikace patří vizualizace, simulace, interakce a kolaborace. Vizualizace umožňuje uživatelům vytvářet virtuální modely montážních stanic a sledovat montážní procesy v reálném čase. Simulace umožňuje provádět testy montážních procesů v různých podmínkách a identifikovat potenciální problémy s ergonomií, bezpečností nebo výkonností. Interakce umožňuje uživatelům aktivně zasahovat do virtuálního prostředí, například manipulovat s montážními díly nebo ovládat montážní zařízení. Kolaborace pak podporuje spolupráci mezi různými účastníky, jako jsou inženýři, technici a pracovníci výroby, což umožňuje efektivnější sdílení znalostí a společné řešení problémů.

Co se týče možností využití, aplikace IC.IDO umožňuje analýzu ergonomie montážních procesů, testování nových montážních metod a výcvik zaměstnanců výroby. To přispívá k zvýšení efektivity výrobních procesů, optimalizaci pracovních podmínek a snížení rizika chyb. Pro zajištění optimálního fungování aplikace jsou definovány i doporučené hardwarové specifikace, včetně procesoru, paměti, grafické karty, displeje a zvuku. [28]

2.3.3. Výsledné hodnocení

IC.IDO je komplexní nástroj, který může být cenným přínosem pro výrobní společnosti. Aplikace nabízí řadu funkcí a možností využití, které mohou pomoci zlepšit ergonomii, bezpečnost a produktivitu montážních procesů.

Výhody aplikace zahrnují možnosti využití v různých oblastech výroby, včetně analýzy ergonomie, testování nových procesů a výcviku zaměstnanců. Dále nabízí širokou škálu funkcí, které pomáhají uživatelům lépe porozumět montážním procesům a identifikovat potenciální problémy. Díky využití virtuální reality také poskytuje realistickou a pohlcující vizualizaci montážních procesů.

Mezi nevýhody aplikace patří fakt, že se jedná o komerční software, který vyžaduje placené předplatné. Dále vyžaduje výkonný hardware pro dosažení optimálního výkonu, což může být finančně náročné pro některé uživatele.

3. Shrnutí hlavních možností existujících řešení

Všechny tři výše uvedené aplikace poskytují bohaté funkce, které přinášejí významné výhody pro výrobní podniky, zejména díky využití VR pro realistický a pohlcující uživatelský zážitek. Z tohoto důvody mohou být tyto aplikace užitečné pro výrobní podniky, které chtějí poskytnout svým pracovníkům realistický pohled na jejich výrobní procesy. Aplikace efektivně simulují výrobní procesy a umožňují pracovníkům získávat praktické zkušenosti s novými zařízeními, což zvyšuje jejich odbornost a bezpečnost. Pracovníci mohou efektivně spolupracovat na společných úkolech v reálném čase, což zvyšuje synergii a efektivitu týmu.

3.1. Porovnání hardwarového, softwarového a serverového řešení

Aplikace IC.IDO a Smart Factory VR využívají ke svému provozu HMD a počítač. IC.IDO vyžaduje méně náročný hardware než Smart Factory VR, což je dáno použitím odlišných herních enginů ve vývoji. U platformy COVE-VR nebyly uvedeny minimální požadavky na hardware.

COVE-VR a IC.IDO používají herní engine Unity 3D, zatímco Smart Factory VR používá Unreal Engine 4. Unity 3D je univerzální herní engine, který je vhodný pro vývoj různých typů her a aplikací, včetně VR. Je relativně snadno použitelný a nabízí širokou škálu funkcí a nástrojů. Unity 3D je dobrou volbou pro aplikace, které vyžadují snadné použití a širokou škálu funkcí. Unreal Engine 4 je dobrou volbou pro aplikace, které vyžadují realistickou grafiku a pohlcující zážitek.

Všechny tři aplikace používají virtuální servery pro ukládání dat, sdílení dat a správu uživatelů. Smart Factory VR a COVE-VR používají navíc jeden server pro synchronní spolupráci. IC.IDO lze nasadit na vlastní server nebo použít cloudové řešení od společnosti ESI Group. Tab. 3-1 níže poskytuje srovnání serverových řešení aplikací, zdůrazňuje počet serverů a funkce každé aplikace.

Funkce	SmartFactory VR	COVE-VR	IC.IDO
Typ serveru	Virtuální	Virtuální	Virtuální nebo cloudové
Počet serverů	Dva	Dva	Jeden nebo více
Funkce serveru	Ukládání dat, sdílení dat, správa uživatelů, synchronní spolupráce	Ukládání dat, sdílení dat, správa uživatelů, asynchronní spolupráce	Ukládání dat, sdílení dat, správa uživatelů
Cena serveru	Nutné dodatečné náklady	Nutné dodatečné náklady	Nutné dodatečné náklady nebo je zahrnuta v ceně cloudového řešení

Tab. 3-1 Porovnání serverových řešení aplikací

3.2. Přínosy pro společnost

Všechny tři aplikace nabízejí potenciální přínosy pro společnost včetně snížení výrobních nákladů.

Smart Factory VR, COVE-VR a IC.IDO se zaměřují na zvýšení produktivity a zlepšení bezpečnosti ve výrobě. Aplikace mohou pomoci pracovníkům ve výrobě lépe porozumět výrobním procesům a zefektivnit svou práci a identifikovat a předcházet nebezpečným situacím. IC.IDO se navíc zaměřuje na zlepšení spolupráce, zvýšení flexibility a efektivity učení ve výrobě. Aplikace může pomoci pracovníkům ve výrobě lépe spolupracovat na projektech a řešit problémy, přizpůsobit se změnám v požadavcích na výrobu a lépe porozumět výrobním procesům a připrav se na práci ve výrobě.

Z hlediska uživatelského ovládání aplikace využívají standardní ovládací prvky VR, jako jsou pohybová gesta, ovládání pomocí ovladače a hlasové ovládání. Ovládací prvky v aplikaci jsou intuitivní a snadno se používají.

Při výběru mezi těmito aplikacemi by měly podniky zvážit své specifické potřeby: Smart Factory VR a COVE-VR jsou ideální pro zvýšení produktivity a bezpečnosti, zatímco IC.IDO je lepší volbou pro podporu spolupráce a flexibilního učení.

4. Analýza vlastního řešení

Aplikace pro virtuální kolaborativní trénink průmyslových procesů, která je předmětem dále popisovaného vývoje, se zaměřuje na poskytnutí realistického a interaktivního tréninku pro pracovníky ve výrobě. Aplikace by měla pomoci pracovníkům lépe porozumět procesu, identifikovat a předcházet nebezpečným situacím a zlepšit své dovednosti, a to vše pod dohledem školitele.

4.1. Technické řešení

Na základě bakalářské práce[30], a porovnání existujících řešení bude vlastní řešení realizováno v herním enginu Unity 3D za použití serveru Photon PUN.

Aplikace COVE-VR, Smart Factory VR a IC.IDO využívají HMD s počítačem pro lepší zobrazení prostředí a plynulejší chod aplikace. Toto řešení je však finančně náročné z hlediska pořízení dostatečně výkonného počítače. Z toho důvodu se bude budoucí řešení vyvíjet přímo pro náhlavní soupravu Oculus Quest 2(Obr. 4-1) a to v platformě Android. Tato náhlavní souprava je cenově dostupná, a to za cenu 9 542,- Kč včetně DPH (k datu 30.11. 2023 na stránkách Alza[31]).

Obr. 4-1 Ukázka Oculus Quest 2 [31]

4.2. Funkce a přínosy aplikace

Aplikace bude obsahovat následující funkce. Bude zahrnovat grafické zobrazení, které bude realisticky reprezentovat průmyslový proces. Toto zobrazení bude obsahovat dvě místnosti. V první místnosti budou uživatelé čekat na připojení dalších osob a při naplnění konkrétního počtu osob se všichni přesunou do druhé místnosti, kde budou vykonávat daný proces. Aplikace využije technologii hand tracking pro přirozenou interakci uživatelů s objekty, což zlepší

imersivnost a intuitivnost tréninku. Aplikace také bude obsahovat bude sadu nástrojů, které umožní pracovníkům provádět úkoly v procesu a funkce pro učení pracovníků o procesu.

Aplikace přináší několik klíčových výhod: zlepšení porozumění pracovníků procesům, zvýšení efektivity práce a potenciální růst produktivity. Dále může pomoci pracovníkům identifikovat a předcházet nebezpečným situacím, což by zlepšilo bezpečnost. Nakonec by mohla pomoci výrobním společnostem snížit náklady na školení a certifikaci pracovníků.

Možné rozšíření aplikace zahrnuje specializaci na specifické průmyslové procesy, jako je výroba automobilů či potravin, což umožní cílenější školení.. Také by mohla být přizpůsobena uživateli na základě jeho dovedností nebo zkušeností.

Vlastní řešení VR aplikace pro průmyslový proces má potenciál nabídnout řadu výhod pro společnosti a pracovníky ve výrobě. Aplikace by měla být schopna pomoci pracovníkům lépe porozumět procesům, zlepšit své dovednosti a zvýšit svou bezpečnost.

5. Popis vývoje vlastního řešení

Aplikace VR Multiplayer Training bude vyvinuta pomocí herního enginu Unity 3D, jehož instalace a nastavení je podrobně popsáno v bakalářské práci autora [30, 32–36].

Aplikace využije klíčové assety z Unity Assets Store, včetně Photon Pun 2 pro multiplayer funkce, Photon Voice 2 pro hlasovou komunikaci a Meta XR SDKs pro rozšířené interakce a avatary. Photon Pun 2 a Photon Voice 2 bude využit k připojení uživatele na server, k promítnutí pohybů a změny stavů v rámci multiplayeru a verbální komunikaci. Meta XR Core SDK je balíček s kamerou doporučený pro hardware od společnosti Meta, pro který bude aplikace vyvinuta. Balíček Meta XR Interaction SDK nám poslouží k interakci s objekty, k využití již připravených interakčních objektů od společnosti Meta. Abychom mohli snadno použít jejich prefabrikáty (prefabs), je nutné z daného assetu pomocí manažeru balíčku importovat jeho ukázky, které se importují do projektu ve formě assetu. V poslední řadě asset Meta XR Avatars nám pomůže s grafickým zobrazením uživatele s realistickým designem postav. [37]

5.1. Lobby místnost

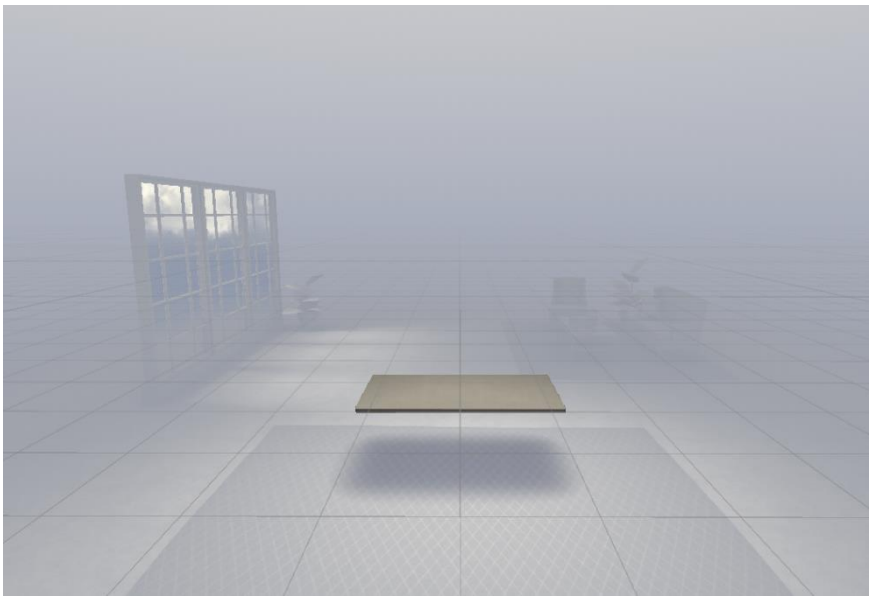
Inicializační fáze aplikace zahrnuje vytvoření Lobby místnosti, která slouží jako vstupní bod pro uživatele přihlašující se k online tréninku. K tomuto účelu byla vytvořena scéna „VRMultiplayer_Lancher“. Samotná místnost nebude připojena k serveru a bude sloužit pouze jako lokální prostředí.

Následně budeme pokračovat grafickým rozšířením scény. Grafické prostředí by nemělo být rušivé, aby se uživatel mohl soustředit na účelnost Lobby místnosti. Součástí Lobby bude také kamera, která umožní uživatelům interakci s prostředím.

Funkcionalita Lobby místnosti umožňuje uživatelům vytvářet, připojovat se k online tréninkům a získávat informace o dostupnosti místností. Uživatel na základě zpětných volání z místností, dostane informace o dalším postupu.

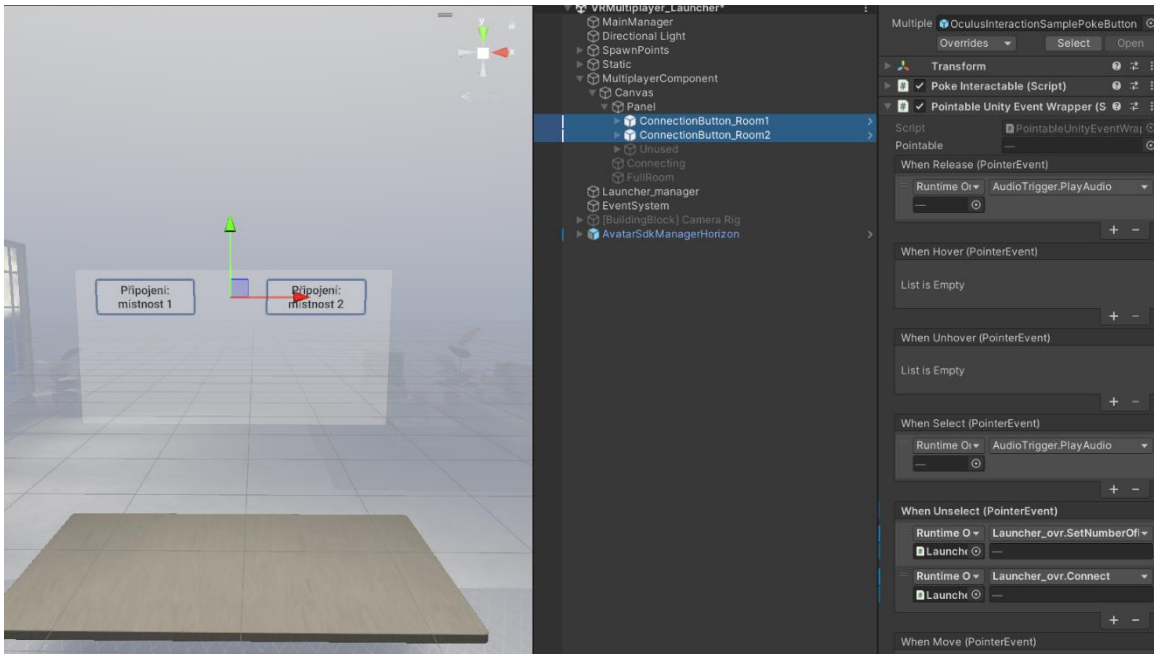
5.1.1. Popis tvorby prostředí

Nejdříve scéně vytvoříme grafické prostředí. K zobrazení okolního prostředí využijeme prefabrikát „LargeRoom“ z balíčku „Meta XR Interaction SDK OVR Samples“. Prefabrikát zobrazuje nenápadné prostředí, které uživatele navodí příjemný a ničím nerušený pocit (Obr. 5-1). Tím se uživatel bude moci plně soustředit na tlačítka pro připojení do online tréninku, které později vytvoříme.



Obr. 5-1 Prefabrikát „LargeRoom“

Aplikace bude obsahovat tlačítka pro řízení připojení do tréninkových místností, včetně skriptů pro ovládání těchto tlačítek.. Zde nám znovu pomůže balíček „Meta XR Interaction SDK OVR Samples“, který má již tlačítka vytvořené s příslušnými scripty pro jejich ovládání. Aplikujeme tedy prefabrikát „OculusInteractionSamplePokeButton“ do naší scény, a to dvakrát, pro možnost připojení do místností budoucího online tréninku. Tlačítka si pojmenujeme „ConnectionButton_Room1“ a „ConnectionButton_Room2“ pro jejich účelné rozlišení mezi sebou. Také změníme jejich popisek, který se objeví ve scéně. Aby tlačítka dokázala provést akci při jejich stisknutí, je zapotřebí na obě tlačítka přidat skript „PointableUnityEventWrapper“, který nám zpřístupní akce při interakci s tlačítkem (Obr. 5-2).



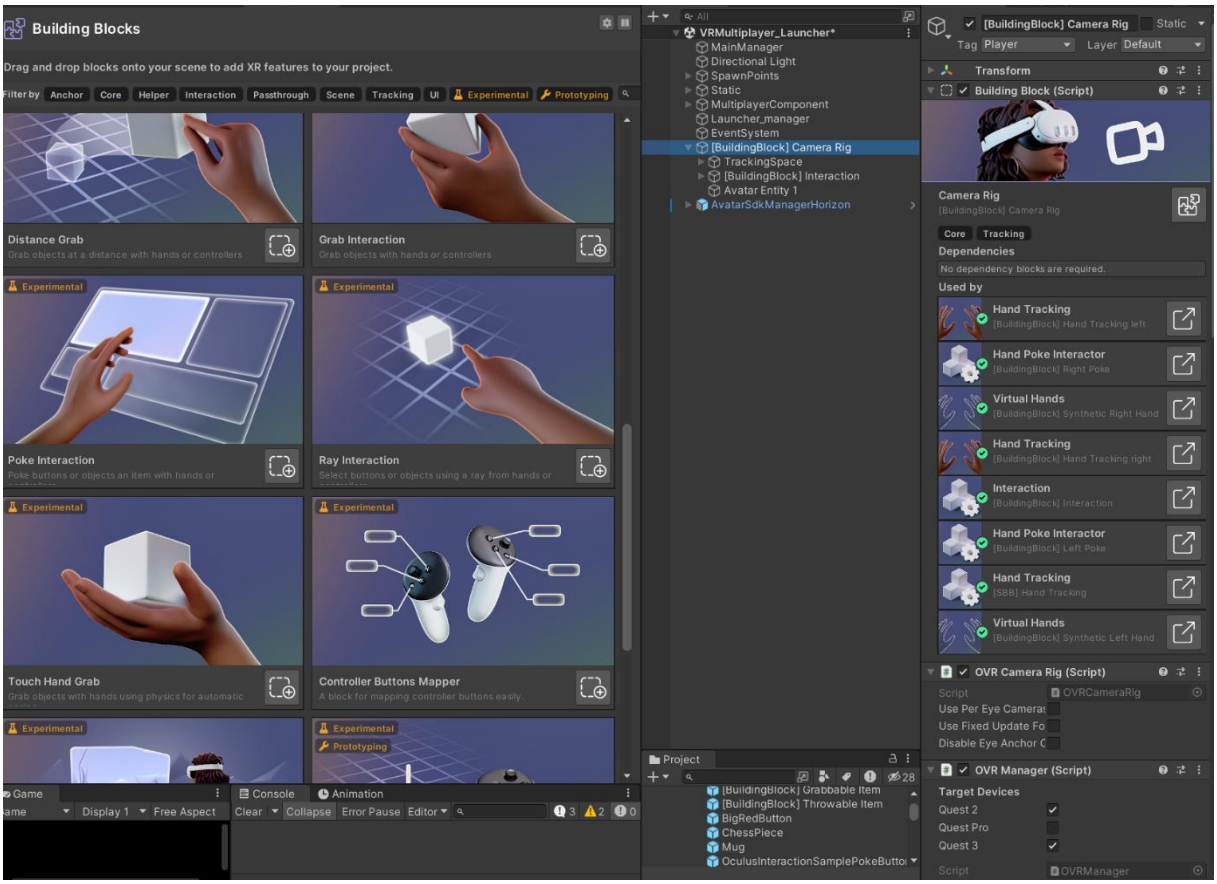
Obr. 5-2 Tlačítka pro připojení do online tréninku

Protože budeme uživatele připojovat do místnosti, je vhodné, aby byl o svém stavu informován. Proto vytvoříme dva další objekty textového pole. První z objektů nazveme

„Connecting“ a do textového pole vepíšeme text „Připojování…“ a druhý objekt nazveme „FullRoom“ a do textového pole vepíšeme text „Místnost je plná. Vyber jinou místnost.“

V neposlední řadě naše grafické zobrazení je potřebné přidat pro uživatele samotnou kameru s prvky, pomocí kterých bude moci interagovat s okolím. Protože využíváme nejnovější balíček Meta OVR, zpřístupnil se nám nástroj „Building Block“, který slouží ke snadnému vkládání OVR assetů do scény. Pomocí tohoto nástroje vložíme komponenty „Camera Rig“, „Hand Tracking“, „Poke Interaction“ a „Virtual Hands“

Na závěr pro zobrazení Meta Avatara pod uživatelovou kamerou, přidáme prefabrikát do kořenu projektu „AvatarSdkManagerHorizon“ a pod kamerou vytvoříme prázdný „GameObject“ s názvem „Avatar Entity 1“, na který připojíme script „SampleAvatarEntity“. Díky tomu se nám při zapnutí scény vytvoří pod naší kamerou avatar, který sleduje naše pohyby. (Obr. 5-3)



Obr. 5-3 OVR kamera s Meta Avatarem

5.1.2. Programování funkcionality

Zprvu se budeme zabývat správným umístění uživatele v Lobby místnosti. Je žádoucí, aby uživatel při zapnutí aplikace byl umístěn přímo před tlačítka, které mu budou sloužit pro připojení do online tréninku. Proto vytvoříme v kořenu hierarchie rodičovský „GameObject“ s názvem „SpawnPoints“, který bude obsahovat místa, kde se uživatel na začátku scény bude moci nacházet. Pro místnost Lobby postačí pouze jedno umístění, ale pro přehlednost v hierarchii projektu rodičovský objekt zachováme. Pod rodičovským objektem „SpawnPoints“ vytvoříme nový „GameObject“, který nazveme „Def_spawn“ a vytvoříme pro něj skript

„BasicRecenter_ovr“. Skript bude uživateli nastavovat polohu a pomocí čtyřúhelníkového objektu zajistí odtemnění kamery při přesunu.

Nejdříve si nadefinujeme proměnné, které budeme využívat. Pro přehlednost budeme vytvářet regiony, a typ nadpisů „[Header()]“, které nám umožní lepší orientaci ve skriptu. Pomocí regionu můžeme sbalit konkrétní regiony kódu a ponechat jen jeho název. Pomocí nadpisů „[Header()]“ přiřadíme nadpisy nad proměnnou, které se promítnou v sekci „Inspector“ v aplikaci Unity 3D. Kompletní script lze nalézt v [Příloha 1.].

```
#region Proměnné
// Kamera pro transformaci místa
private Transform m_CameraRig;

[Header("Teleport")]
// Přiřadit rodičovský objekt [BuildingBlock] Camera Rig
public OVRManager m_OVRManager;

// Čas do teleportace
public float resetPositionTime;

// Po spuštění
public bool playOnAwake;

// Výška kamery
public float height;

[Header("Fade screen")]
// Objekt k rendrování
public Renderer rend;

// Čas trvání
public float fadeDuration;

// Barva přechodu
public Color fadeColor;
    #endregion
```

Proměnná „m_CameraRig“ je privátního typu, což znamená, že ji nemůžeme ručně přiřadit v editoru ani je volat v jiných skriptech i když máme referenci na objekt. Budeme jí následně přiřazovat na startu aplikace. Protože budeme potřebovat pouze její polohu, zvolili jsme přímou komponentu „Transform“.

Proměnná „m_OVRManager“ je veřejného typu, a proto ji můžeme přiřadit v editoru ručně. Protože s ní budeme pracovat na základě jejího skriptu „OVRManager“ přiřadíme jí rovnou tento typ.

Následují proměnné „ResetPositionTime“, která nám bude oddalovat v reálných číslech spuštění skriptu, bool „playOnAwake“, na kterém si zvolíme, zda se má skript pouštět na startu scény, nebo ne, reálné číslo „height“, které nám definuje výšku kamery, vykreslovač „rend“, který zpřístupní odtemnění kamery, reálné číslo „fadeDuration“, které nastavuje čas odtmavění kamery a barva „fadeColor“, která určuje zbarvení zatmavěné kamery.

Nyní se můžeme zabývat samotnými metodami pro realizaci pohybu kamery ve scéně.

```
#region MonoBehaviour Callbacks
void Start()
```

```
{
    m_CameraRig = m_OVRManager.transform;

    if (playOnAwake == true)
    {
        Invoke("ResetVRPosition", resetPositionTime);
    }
}
#endregion
```

Volání při načtení scény zajišťuje metoda „void Start()“. V této metodě přiřadíme k privátní proměnné „m_CameraRig“ proměnnou „m_OVRManager“ s komponentou „transform“. Přiřazená komponenta musí být stejného typu, jako je definovaná proměnná, které přiřazujeme. Pokračujeme funkcí „když“, která nám při splnění vnitřní podmínky „(pokud je „playOnAwake“ nastavena na hodnotu „true“)“ provede vnitřní část kódu. Uvnitř funkce voláme metodu „ResetVRPosition“ s časovým zpožděním, které udává proměnná „resetPositionTime“.

```
#region Teleport
//Veřejná metoda, pro volání teleportu se zatmavením obrazu
public void ResetVRPosition()
{
    ResetVRPosition_meth();
    if (rend != null)
        FadeOut();
}

//Nastavení tvrdé pozice a rotace pouze při zavolání metedy
private void ResetVRPosition_meth()
{
    m_OVRManager.trackingOriginType = OVRManager.TrackingOrigin.EyeLevel;
    m_CameraRig.transform.parent = this.gameObject.transform;
    m_CameraRig.transform.localPosition = new Vector3(0f, height, 0f);
    m_CameraRig.transform.localEulerAngles = Vector3.zero;
}
#endregion
```

Metoda „ResetVRPosition“ je veřejného typu. Díky tomu ji můžeme volat i z jiných skriptů, pokud budeme mít referenci na daný objekt, na kterém skript existuje. V metodě voláme metodu „ResetVRPosition_meth()“. Metoda zajišťuje nastavení typu „EyeLevel“ na scriptu „OVRManager“ a posun kamery pod objekt se skriptem a nastavení kamery lokálních pozic daného objektu. Ve čtvrtém řádku metody nastavujeme pomocí proměnné „height“ výšku kamery. Dále v metodě „ResetVRPosition()“ voláme metodu „FadeOut“, která při přesunu roztemní kameru. Metoda roztemnění bude vysvětlena níže.

```
#region FadeScreen
public void FadeOut()
{
    Fade(1, 0);
}

public void Fade(float alphaIn, float alphaOut)
{
    StartCoroutine(FadeRoutine(alphaIn, alphaOut));
}

public IEnumerator FadeRoutine(float alphaIn, float alphaOut)
{
```

```
    float timer = 0;
    while (timer <= fadeDuration)
    {
        Color newColor = fadeColor;
        newColor.a = Mathf.Lerp(alphaIn, alphaOut, timer / fadeDuration);

        rend.material.SetColor("_Color", newColor);

        timer += Time.deltaTime;
        yield return null;
    }

    if (alphaIn > 0)
    {
        rend.gameObject.GetComponent<MeshRenderer>().enabled = false;
    }
    Color newColor2 = fadeColor;
    newColor2.a = alphaOut;

    rend.material.SetColor("_Color", newColor2);
}
#endregion
```

Metoda „FadeOut()“ využívá metodu „Fade(float alphaIn, float alphaOut), která požaduje dvě vstupní hodnoty v ní již zmíněné. Tyto dvě hodnoty nastavují přechod transparentnosti vykreslovaného materiálu. Přechod se provádí na základě korutiny „FadeRoutine(float alphaIn, float alphaOut)“. Korutina je závislá na čase, který je předem definován proměnnou „fadeDuration“.

Nyní přejdeme k hlavnímu scriptu pro naší Lobby místnost, který se bude zabývat připojením, nebo vytvořením serverové místnosti pro online kolaborativní trénink. Vytvoříme „GameObject“ v kořenu hierarchie a k němu vytvoříme skript s názvem „Launcher_ovr“. Kromě příkazu „[Header()]“, zde budeme používat i další pomůcky, která nám pomohou se ve skriptu lépe zorientovat a to proto, že skript má potenciál být rozsáhlí a tím i méně přehledný. Příkaz „[Tooltip()]“ nám při najetí myší na proměnnou v inspektoru vypíše pole textu, které do nej v píšeme. Zatím co příkaz „[SerializeField]“ nám zpřístupní zobrazení privátní proměnné v inspektoru, jako by to byla proměnná typu veřejné, avšak proměnnou nezpřístupní v jiných skriptech při referenci na objekt. Celý skript lze nalézt v [Příloha 2.].

```
#region Private Serializable Fields
[Header("Nastavení pro tvorbu místnosti")]
[Tooltip("Maximální počet hráčů na jednu místnost. Pokud je místnost plná,
nemohou se připojit nový hráči, a proto bude vytvořena nová místnost.")]
[SerializeField]
private byte maxPlayersPerRoom;

[Header("UI zobrazení při přihlášení")]
[Tooltip("UI Label informující uživatele o probíhajícím připojení")]
[SerializeField]
private GameObject progressLabel;

[Tooltip("UI Panel, který umožňuje uživateli připojit se a hrát")]
[SerializeField]
private GameObject controlPanel;

[Tooltip("UI Panel, který informuje uživate, že je místnost plná")]
[SerializeField]
private GameObject fullRoom;
```

```
#endregion
```

Proměnná „maxPlayersPerRoom“ nám určí maximální počet uživatelů v místnosti. V našem případě bude tvořen průmyslový proces pro dva uživatele, ale procesů časem může být více, a proto si později můžeme dle požadavků měnit počet uživatelů na místnost.

Pokud se uživatel bude přihlašovat do místnosti dáme mu o tom informaci pomocí objektu „progressLabel“. Dále je vhodné, aby při přihlašování do místnosti uživatel již nemohl znovu stisknout tlačítka, a proto vytvoříme proměnnou „controlPanel“, ve které bude přiřazen rodičovský objekt tlačítek. Protože hodláme zjišťovat obsazenost místnosti, vytvoříme proměnnou „fullRoom“, která dá uživateli informaci, pokud bude místnost již zaplněna uživateli.

```
#region Private Fields
/// <summary>
/// Toto je číslo verze tohoto klienta. Uživatelé jsou od sebe odděleni pomocí
gameVersion (což umožňuje provádět změny).
/// </summary>
string gameVersion = "1";

/// <summary>
/// Sledujte aktuální proces.
/// </summary>
bool isConnecting;

string targetRoom;
#endregion
```

Proměnná „gameVersion“ nám zajistí, že pokud se v budoucnu aplikace vylepší a její verze bude zvýšena, nebudou moci uživatele s jinou verzí být společně v jedné místnosti. Bool „isConnecting“ je proměnná sledující aktuální proces připojení. Protože připojení je asynchronní a je založeno na několika zpětných volání ze serveru Photon, musíme tedy sledovat, abychom správně upravili chování, když obdržíme zpětné volání od serveru Photon. Typicky se používá pro zpětné volání v metodě „OnConnectedToMaster()“, která bude popsána níže.

```
#region MonoBehaviourPunCallbacks CallBacks

public override void OnConnectedToMaster()
{
    Debug.Log("PUN Basics Tutorial/Launcher: OnConnectedToMaster() byl zavolán
pomocí PUN");

    if (isConnecting)
    {
        // #Critical: Pokus o připojení k potenciální existující místnosti
        PhotonNetwork.JoinRoom(targetRoom);
        isConnecting = false;
    }
}

public override void OnDisconnected(DisconnectCause cause)
{
    progressLabel.SetActive(false);
    controlPanel.SetActive(true);
```

```
    Debug.LogWarningFormat("PUN Basics Tutorial/Launcher: OnDisconnected() byl
zavolán pomocí PUN s řešením {0}", cause);
}

public override void OnJoinRoomFailed(short returnCode, string message)
{
    if(returnCode == ErrorCode.GameFull)
    {
        Debug.LogError("Místnost je plná.");
        StartCoroutine(GameFull_cor());
    }
    else
    {
        Debug.LogError("Vytvářím místnost s maximálním počtem: " +
this.maxPlayersPerRoom);

        RoomOptions roomOptions = new RoomOptions { MaxPlayers =
maxPlayersPerRoom };

        // #Critical: Vytvoříme novou místnost.
        PhotonNetwork.CreateRoom(targetRoom, roomOptions);
    }

}

public override void OnJoinedRoom()
{
    Debug.Log("PUN Basics Tutorial/Launcher: OnJoinedRoom() zavolán pomocí PUN.
Nyní je tento klient v místnosti.");

    // #Critical: Načítáme pouze v případě, že jsme první hráč
    if (PhotonNetwork.CurrentRoom.PlayerCount == 1)
    {
        Debug.LogError("Načetli jsme: " + targetRoom);

        // #Critical: Načtení úrovně místnosti
        PhotonNetwork.LoadLevel("VRMultiplayer_room2");
    }
}
#endregion

#region Coroutines
IEnumerator GameFull_cor()
{
    progressLabel.SetActive(false);
    fullRoom.SetActive(true);
    yield return new WaitForSeconds(3f);
    fullRoom.SetActive(false);
    controlPanel.SetActive(true);
}
#endregion
```

V této části skriptu již používáme veřejné metody typu „override“. Metody již v balíčku Photon existují, ale nemají pro nás dostačující funkce. Proto je musíme přepsat. Pomocí „override“ využijeme jejich název a volání v ostatních skriptech serveru Photon, ale doplníme si zároveň naší požadovanou funkcionalitu.

„OnConnectedToMaster()“ je metoda, která je volaná za pomocí PUN. Zde již využijeme proměnnou „isConnecting“. Nechceme nic dělat, pokud se nepokoušíme připojit k místnosti. Tento případ, kdy je „isConnecting“ hodnoty „false“, je typicky při ztrátě nebo ukončení online

scény, kdy se při načítání této úrovně zavolá OnConnectedToMaster, v takovém případě nechceme dělat nic. Pokud se k místnosti připojujeme voláme metodu „JoinRoom(targetRoom)“, která se nás pokusí připojit do místnosti s konkrétním názvem.

Pokud by se vyskytla chyba v připojení nebo bychom od místnosti úmyslně odpojili zavolá se metoda „OnDisconnected(DisconnectCause cause)“. Kromě toho, že nám metoda do konzole pomocí příkazu „Debug.LogWarningFormat()“ vypíše příčinu odpojení, je také žádoucí abychom uživateli znovu umožnili vybrat připojení do místností. Proto vypínáme v metodě objekt „progressLabel“, který nás informuje o připojování a zapínáme objekt „controlPanel“, pod kterým máme tlačítka.

V případě že by připojení do místnosti selhalo, volá se metoda „OnJoinRoomFaild(short returnCode, string message)“. V této metodě zjišťujeme dva stavy. Místnost je plná uživatelů, a proto pomocí korutiny „GameFull_cor()“ dáme dočasnou informaci uživateli, na 3 vteřiny, o stavu místnosti a vyzveme ho k připojení do jiné místnosti. Nebo místnost neexistuje, a tak vytváříme místnost pomocí příkazu „CreateRoom(targetRoom, roomOptions)“ s námi předdefinovaným názvem a počtem uživatelů, který jsme si zvolili.

Pokud se připojení zdařilo a jsme prvním uživatele připojujícím do místnosti, tedy tvůrcem místnosti, načteme náš průmyslový trénink, který bude později okomentován. Prozatím stačí, že název scény, který využijeme je „VRMultiplayer_room2“

```
#region MonoBehaviour CallBacks
void Awake()
{
    // #Critical: Uživatelé automaticky synchronizují svou úroveň.
    PhotonNetwork.AutomaticallySyncScene = true;
}

void Start()
{
    progressLabel.SetActive(false);
    controlPanel.SetActive(true);
}
#endregion
```

Metoda „Awake()“ se na rozdíl od metody „Start() volá hned po inicializaci objektu, ještě před startem scény, zatím co „Start()“ se volá až po metodě „Awake()“ před prvním vykreslením snímku a je závislá na stavu objektu, na kterém je skript navázán.

V metodě „Awake()“ zajistíme pomocí „AutomaticallySyncScene = true“, že můžeme použít funkci PhotonNetwork.LoadLevel() na hlavním klientovi a všichni klienti ve stejné místnosti automaticky synchronizují svou úroveň.

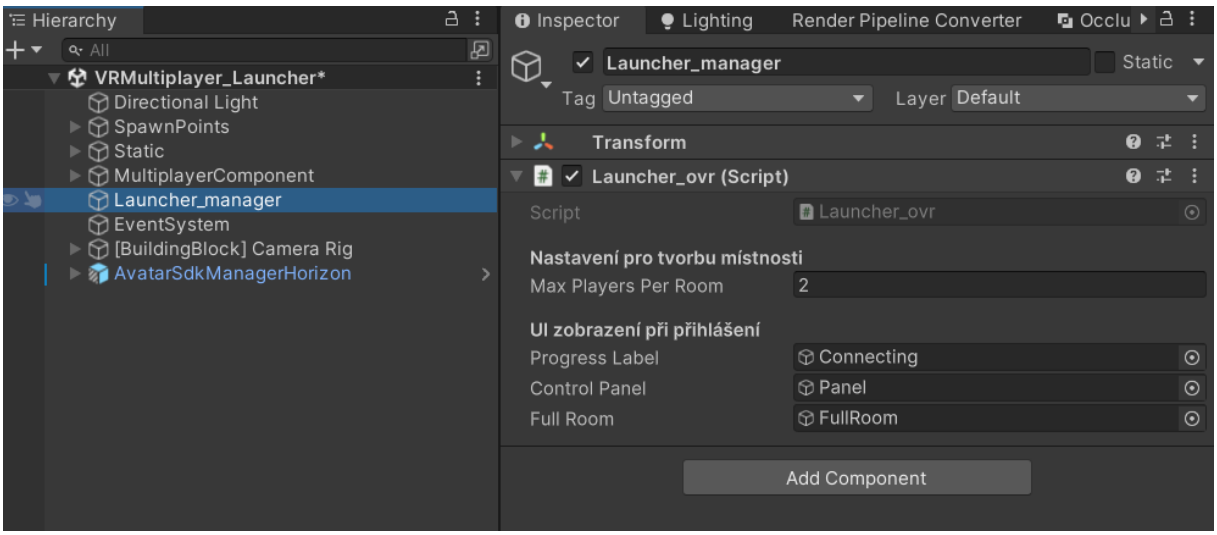
V metodě „Start()“ se ujišťujeme, že jsou zobrazené tlačítka pro připojení do tréninku a skrytý text pro informaci stavu.

```
#region Public Methods
public void Connect(string room)
{
    progressLabel.SetActive(true);
    controlPanel.SetActive(false);

    targetRoom = room;
```

```
    if (PhotonNetwork.IsConnected)
    {
        PhotonNetwork.JoinRoom(targetRoom);
    }
    else
    {
        // sledování snahy připojit se k místnosti, protože když se vrátíme ze
hry, dostaneme zpětné volání, že jsme připojeni, takže potřebujeme vědět, co máme
dělat potom
        isConnecting = PhotonNetwork.ConnectUsingSettings();
        PhotonNetwork.GameVersion = this.gameVersion;
    }
}

public void SetNumberOfPlayers(int count)
{
    maxPlayersPerRoom = (byte)count;
}
    #endregion
```

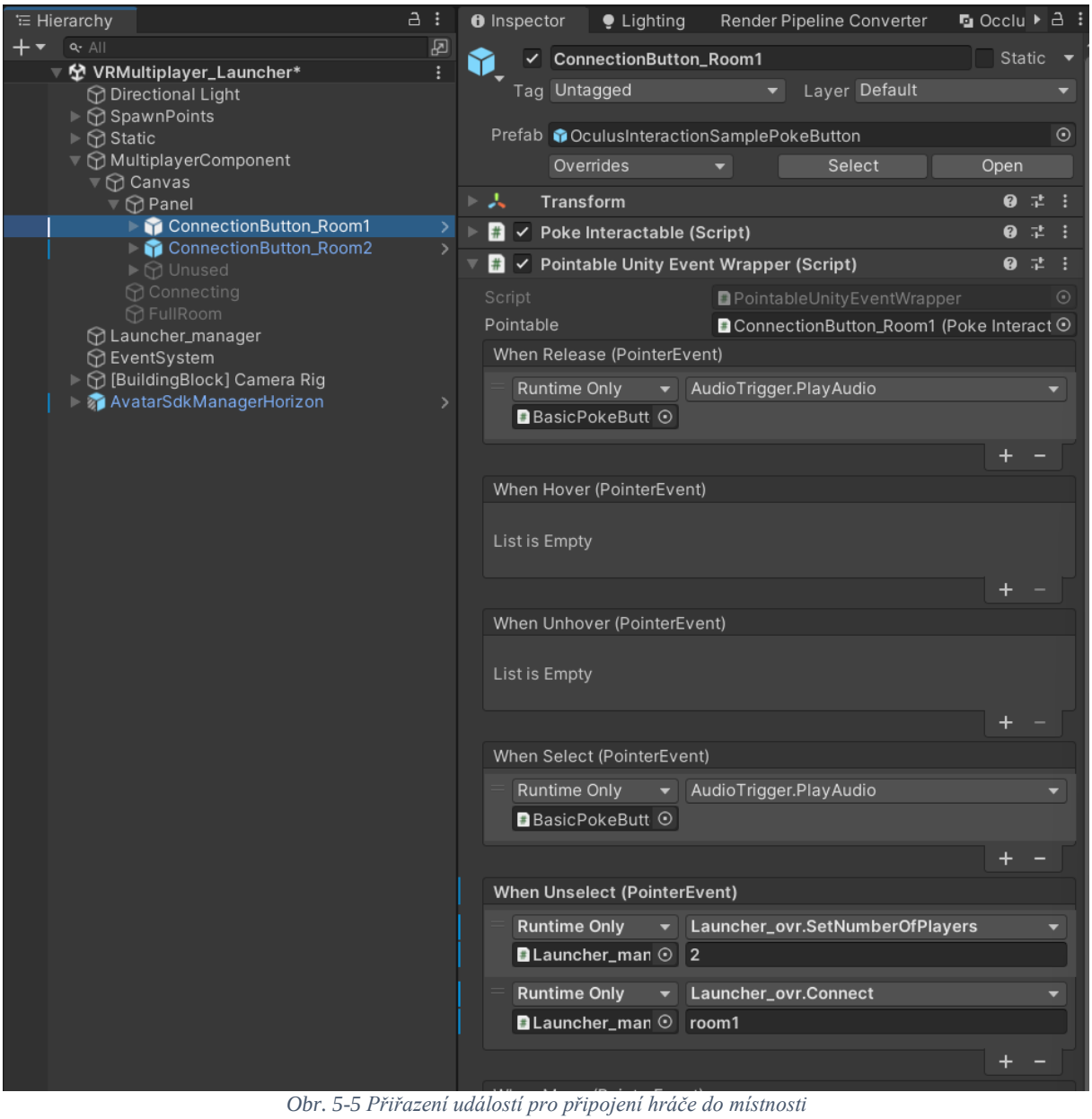
Pomocí metody „Connect(string room)“ spouštíme proces připojení do místnosti. Pokud jsme již připojeni, pokusíme se připojit k vybrané místnosti. Pokud ještě nejsme připojení, připojíme tuto instanci aplikace k síti Photon Cloud Network. Protože budeme chtít metodu použít v „Unity Eventu“, který umí operovat pouze s jednou lokální proměnnou a bylo by žádoucí do metody zahrnout i nastavení počtu osob v místnosti, rozdělili jsme tuto metodu na dvě. Druhá (pomocná) metoda, která se bude volat před metodou „Connect(string room)“, „SetNumberOfPlayers(int count)“ nastaví počet uživatelů dle našich požadavku do proměnné ve skriptu.

Nyní máme skript kompletní a můžeme dosadit objekty do proměnných. Hodnotu „maxPlayersPerRoom“ v základu nastavíme na hodnotu „2“. Hodnota se ale bude měnit dle naší situace podle volání metody „SetNumberOfPlayers(int count). Dále do „progressLabel“ přiřadíme objekt „Connecting“, „controlPanel“ přiřadíme objekt „Panel“ a do „fullRoom“ objekt „FullRoom“. (Obr. 5-4)



Obr. 5-4 Skript "Launcher_ovr

Teď již můžeme volat metody na konkrétních tlačítkách. Přejdeme do tlačítka „ConnectionButton_Room1“ do skriptu „PointableUnityEventWrapper“ a v sekci „When Unselect (PointerEvent)“ přidáme dvě události. Do obou událostí přiřadíme objekt „Launcher_manager“. Na první události nastavíme volání metody „Launcher_ovr.SetNumberOfPlayers“ a do volného pole vepíšeme číslovku „2“. Tím nastavíme maximální počet uživatelů ve vytvářené místnosti. V druhé události nastavíme volání metody „Launcher_ovr.Connect“ a do pole dopíšeme hodnotu „room1“ (Obr. 5-5). Touto událostí zajistíme připojení uživatele do místnosti, nebo její vytvoření. Podobný postup provedeme s objektem „ConnectionButton_Room2“ s tím rozdílem, že maximální počet uživatelů bude nastaven na hodnotu „1“ a do druhé události dopíšeme název místnosti „room2“.



Obr. 5-5 Přiřazení událostí pro připojení hráče do místnosti

Ze skriptu „Launcher_ovr“ je vidět, že vytvářená místnost je vždy ze scény „VRMultiplayer_room2“, ale pro server vytváříme dvě místnosti s jiným názvem. Tím zajistíme dva různé tréninky.

5.2. Místnost průmyslového tréninku

Vytvoříme místnost pro průmyslový trénink, do které se uživatelé dostanou z Lobby místnosti pomocí tlačítek. Vytvoříme scénu s názvem „VRMultiplayer_room2“, kterou jsme již zmínili v předchozí kapitole. Místnost bude připojena na server a bude sloužit, jak pro zaměstnance, tak pro školitele.

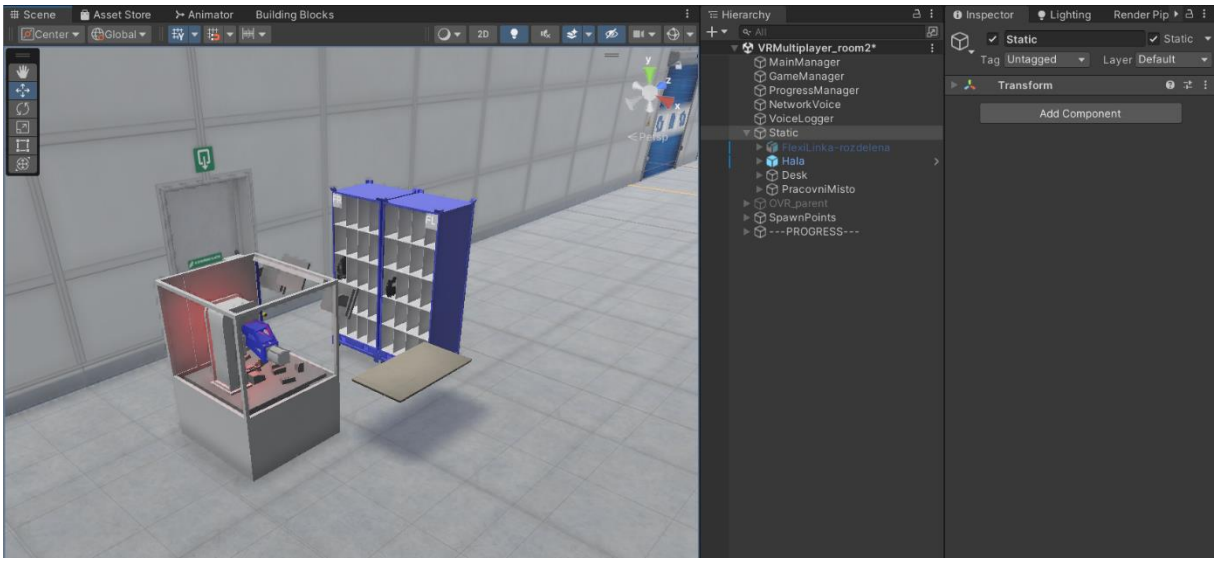
Grafické prostředí bude zobrazovat výrobní halu ve které jsou umístěné poličky, stroj, odkladový stůl a díly k manipulaci. I zde musí být přidaná kamera pro funkčnost uživatele. Kamera bude sloužit pro lokální vizualizaci, zatímco síťový avatar, který bude synchronizovat pohyby a komunikaci mezi uživateli, zajistí realistickou interakci v místnosti. Tento avatar bude generován na začátku připojení do místnosti. Bude obsahovat multiplayerovou verbální komunikaci a bude mít synchronizované pohyby se serverem tak, aby se uživatelé mohli dorozumět i pohyby.

Na závěr celé aplikace se vytvoří funkcionalita. Implementujeme systém pro správné rozmístění uživatelů ve virtuálním prostoru, aby se zabránilo překrývání modelů. Dále implementujeme synchronizaci objektů se serverem, což zahrnuje transformaci a vzdálené metody pro zajištění koherentního sdílení stavu mezi všemi připojenými uživateli. . Uživatelé budou moci interagovat s objekty společně. Implementujeme manažerský skript, který bude koordinovat aktivní prvky tréninkové místnosti, řídil interakce a zpracovával data mezi klienty a serverem. Dále budeme řešit rozmístění uživatelů.

5.2.1. Popis tvorby prostředí

Nejdříve, jako v předchozí scéně, vytvoříme grafické prostředí. Modely byly poskytnuty firmou CIE Group s. r. o. a to ve formátu .fbx. Import modelů se dá jednoduše provést přetažením daného modelu do Unity 3D do okna „Project“ do složky „Assets“. Poté se již vloží model do scény.

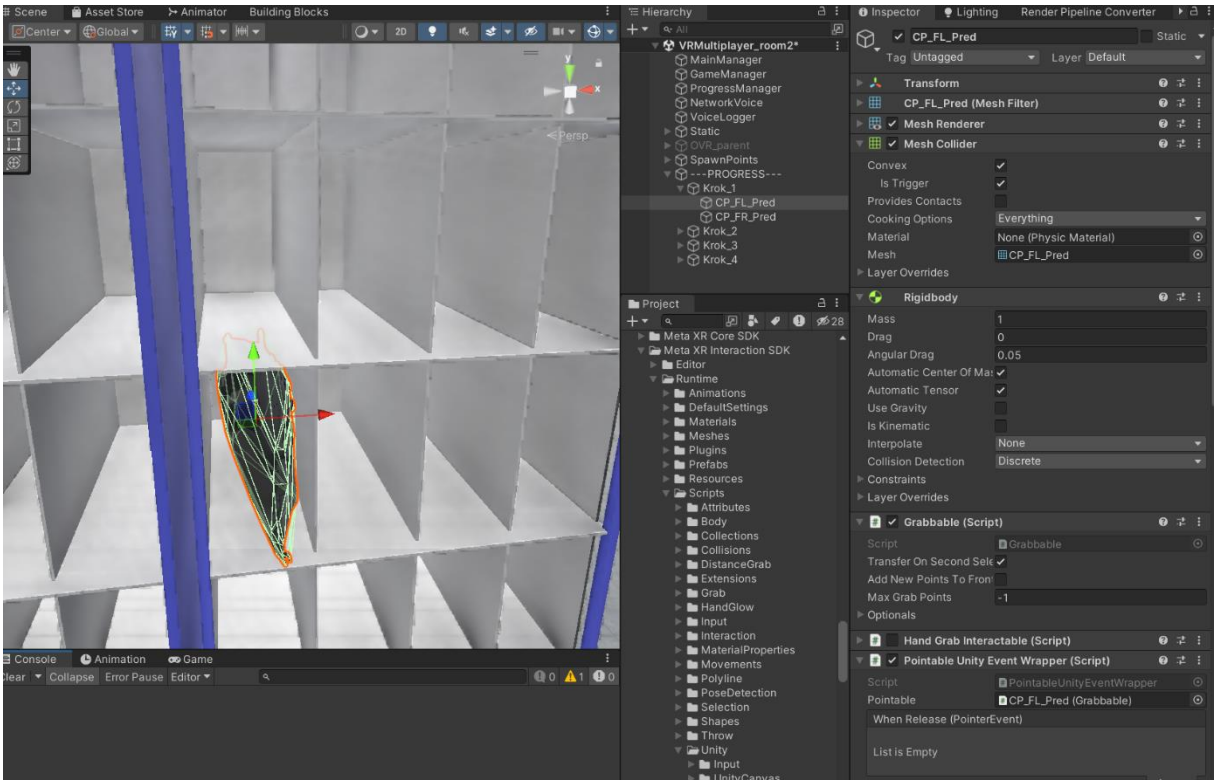
Pro lepší organizaci scény vytvoříme GameObject pojmenovaný „Static“, do kterého zařadíme všechny statické objekty, jako jsou konstrukční prvky haly, stoly a další nábytek, které nebudou vyžadovat interakce s uživateli: : hala, stůl, strojové části bez interakce a poličky. Z důvodu dobré optimalizace na všech těchto objektech nastavíme v záložce „Inspector“ položku „Static“ na hodnotu „true“. (Obr. 5-6)



Obr. 5-6 Grafické prostředí haly

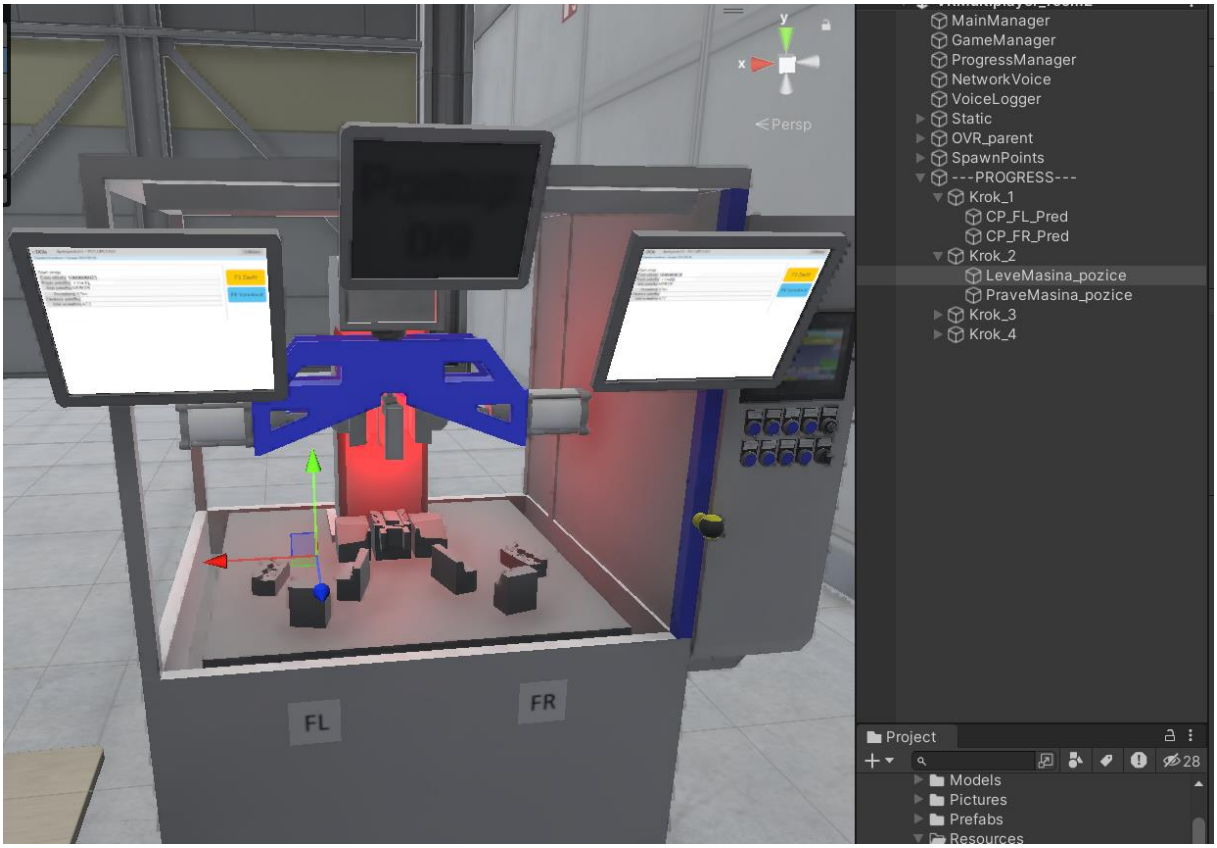
Nyní se můžeme zabývat interaktivními objekty. Založíme hierarchickou strukturu v Unity pod hlavním GameObjectem „---PROGRESS---“ pro lepší správu kroků aplikace. Pod tímto objektem vytvoříme čtyři další GameObjecty pojmenované „Krok_1“ až „Krok_4“, každý reprezentující jednotlivé fáze průmyslového procesu.

V „Krok_1“ umístíme díly dveří určené k manipulaci, které budou uskladněny na příslušných poličkách, připravené k montáži. Oby se objekty dali vzít do ruky, přiřadíme jim komponentu „MeshCollider“, kde nastavíme proměnné „Convex“ a „Is Trigger“ na hodnotu „true“ a komponentu „Rigidbody“, kde nastavíme proměnnou „Use Gravity“ na hodnotu „false“. Využijeme také skripty balíčku „Meta XR Interaction SDK“ a to konkrétně „HandGrabInteractable“ a „Grabbable“, kde nastavíme proměnnou „Transfer On Second Selection“ na hodnotu „true“. Tím zpřístupníme předávání objektu z jedné ruky do druhé. Protože budeme chtít provádět akce při uchopení objektu přiřadíme objektu skript z již zmíněného balíčku se jménem „PointableUnityEventWrapper“, do kterého ručně v proměnné „Pointable“ přiřadíme objekt, na kterém se skript nachází. (Obr. 5-7)



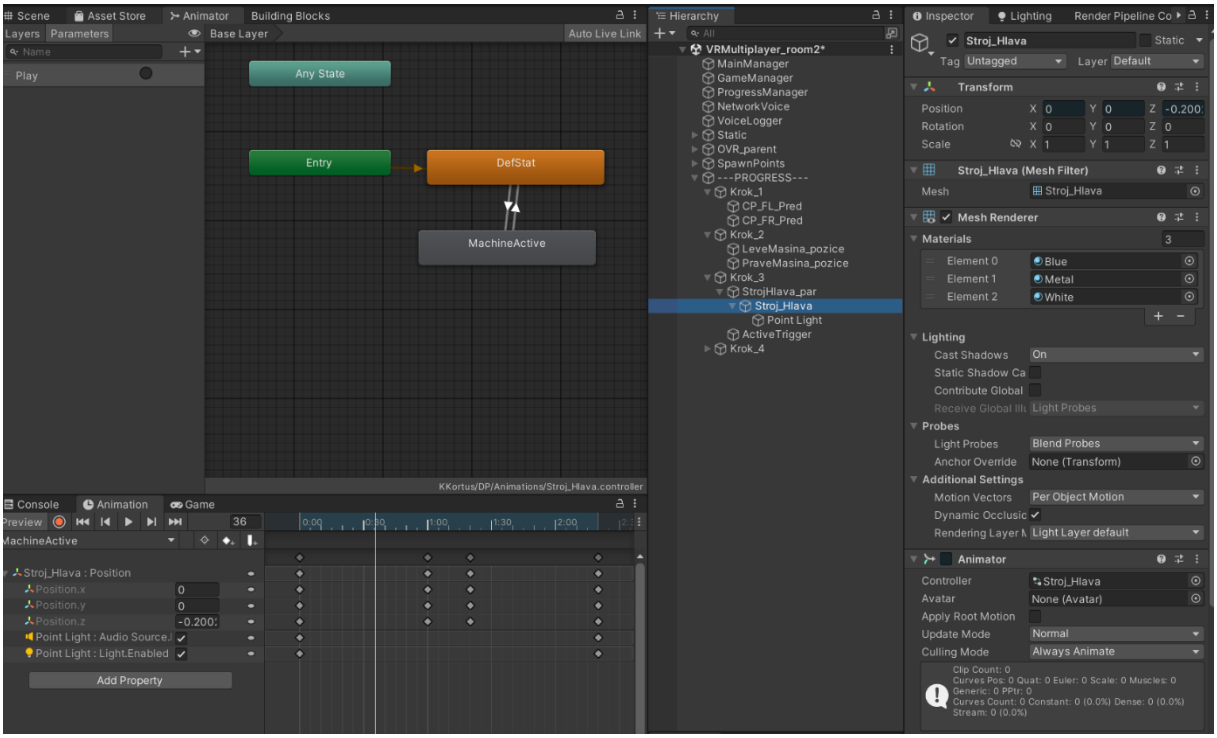
Obr. 5-7 Nastavení uchopení dílu

V objektu „Krok_2“ vytvoříme dva prázdné objekty, které nastavíme na pozici ve stroji, aby když pod ně umístíme konkrétní díly a nastavíme jejich pozici a rotaci na nulu, budou správně založeny ve stroji. (Obr. 5-8)



Obr. 5-8 Pozice založení dílu dveří

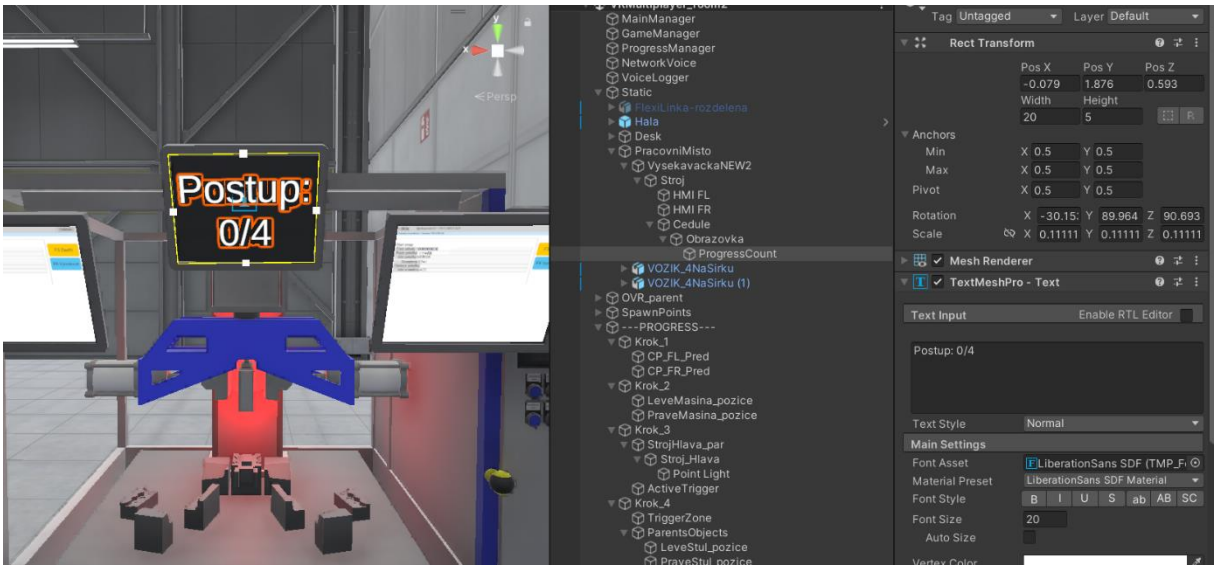
„Krok_3“ bude obsahovat animaci a aktivaci stroje. Do tohoto kroku vytvoříme prázdný objekt „GameObject“ s názvem „StrojHlava_par“, který bude sloužit jako rodičovský objekt pro hlavu stroje, a proto pod něj hlavu stroje přiřadíme. Činíme tak z důvodu udržení pozice animace. Pokud bychom objekt s animací neměli pod rodičovským objektem a s objektem s animací pohnuli na jinou pozici, animace by nemusela správně fungovat. Pro strojní hlavu vytvoříme reálné červené světlo a přiřadíme mu zvuk, který dodá na důvěryhodnosti procesu. Na objekt „Stroj_Hlava“ připojíme komponentu „Animator“ a vytvoříme animaci. Animace stroje obsahuje pohyb hlavy stroje dolů, setrvání v dané pozici a pohyb nahoru. Na začátku animace zapínáme světlo a zvuk a na konci je vypínáme. Protože nechceme, aby animace proběhla hned ze začátku scény, ale až kdy bude vyžádána, vytvoříme parametr typu „Trigger“ s názvem „Play“. Následně vytvoříme graf animace, který po přijetí spouštěče „Play“ provede animaci a vrátí se zpět do původního stavu (Obr. 5-9). Dále vytvoříme pod objektem „Krok_3“ objekt „GameObject“ s názvem „ActiveTrigger“, na kterém později provedeme funkcionalitu spouštění pomocí skriptu. Prozatím na tento objekt přiřadíme komponenty „BoxCollider“ s proměnnou „Is Trigger“ nastavenou na hodnotu „true“ a „Rigidbody“ s proměnnou „Use Gravity“ nastavenou na hodnotu „false“, protože je k akci s objektem budeme potřebovat. Prozatím komponentu „BoxCollider“ vypneme, protože ji budeme zapínat až v době kroku.



Obr. 5-9 Animace strojní hlavy

V posledním kroku „Krok_4“ vytvoříme znovu dva prázdné objekty, které budou umístěny tak, aby pokud dveře budou kopírovat jejich polohu byly správně odložené na stůl. Také pod tímto krokem vytvoříme prázdný objekt, který nám zpřístupní v pozdější funkcionalitě odložení dveří ke kontrole. Tento objekt nazveme „TriggerZone“ a přiřadíme mu komponentu „BoxCollider“ s proměnnou „Is Trigger“ nastavenou na hodnotu „true“, kterou vypneme, z důvodu, který byl zmíněn ve třetím kroku.

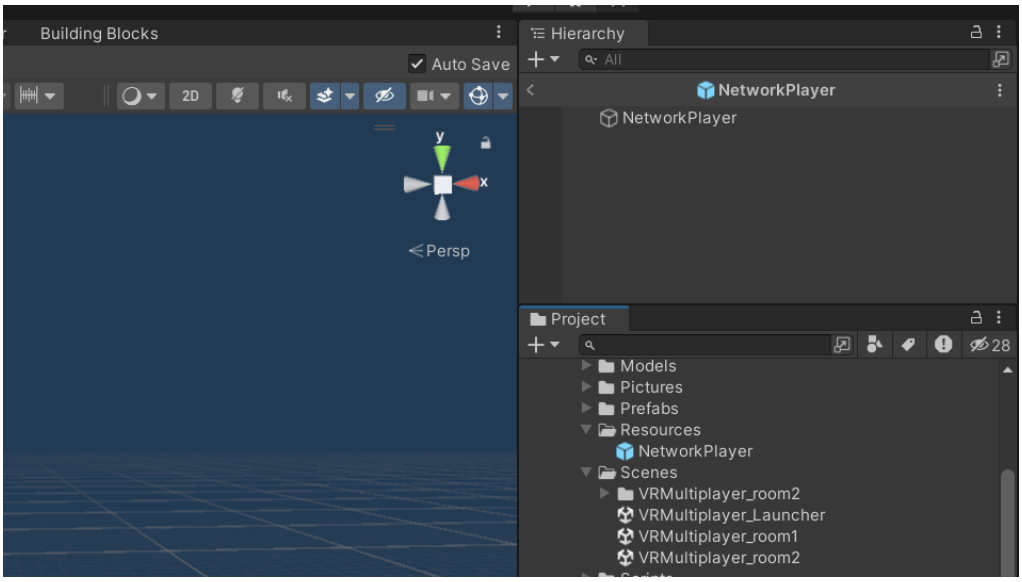
Protože je žádoucí, aby uživatelé věděli, ve které fázi procesu se nacházejí, připravíme si objekt, který zobrazí postup procesu. Pod objektem stroje vytvoříme objekt „TextMeshPro“ s názvem „ProgressCount“, do kterého prozatím vypíšeme text „Postup: 0/4“ a umístíme ho na viditelné místo na stroji. Text se v pozdější funkcionalitě bude aktualizovat na konkrétní krok. (Obr. 5-10)



Obr. 5-10 Výpis postupu procesu

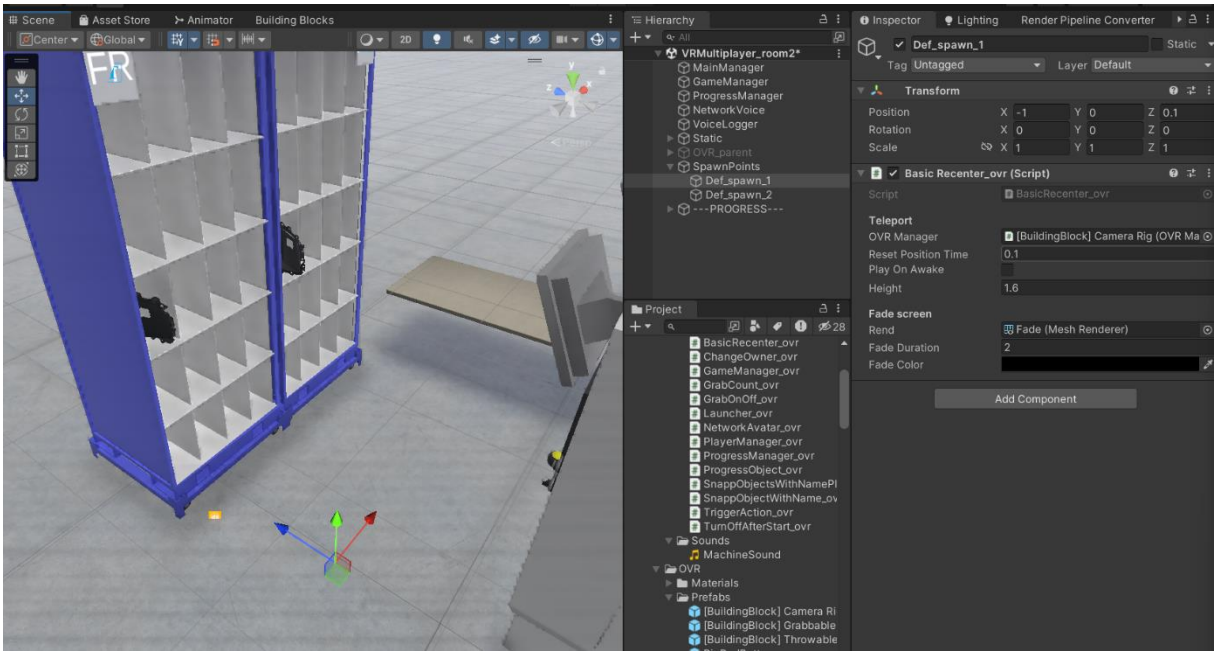
Bude pokračovat vložením OVR kamery do scény stejným způsobem jako tomu bylo předtím v kapitole 5.1.1. Zde ale není nutné využívat balíček „Poke Interaction“ a tak ho vynecháme. Pro promítnutí avatara již nemusíme importovat objekt „AvatarSdkManagerHorizon“, protože se z předchozí scény přenáší pomocí instance.

Pro vytvoření našeho síťového avatara prozatím vytvoříme prázdný objekt s názvem „NetworkPlayer“. Pro tento objekt si vytvoříme v okně „Project“ složku, která se musí jmenovat „Resources“, aby nám bylo umožněné načítání objektu za běhu scény.
Objekt „NetworkPlayer“ do této složky přesuneme a ze scény ho vymažeme. Pokud na něj ve složce dvakrát poklepeme levým tlačítkem myši, zobrazí se okno, kde je možné náš prefabrikát editovat. Prozatím ho necháme prázdný, protože se jím budeme zabývat při programování funkcionality. (Obr. 5-11)



Obr. 5-11 Okno prefabrikátu "NetworkPlayer"

Vytvoříme si také umístění uživatelů ve scéně. Protože víme, že maximální počet uživatelů ve scéně je nastaven na hodnotu dva, vytvoříme objekt „GameObject“ s názvem „SpawnPoints“ do kterého vytvoříme dva objekty „GameObject“ s názvy „Def_spawn_1“ a „Def_spawn_2“ a rozmístíme je ve scéně tak, aby vyhovovali rozložení uživatelů. Na tyto objekty přidáme skript, který jsme si již víše rozebrali, s názvem „BasicRecenter_ovr“. Do proměnné „OVR Manager“ přiřadíme objekt „[BuildingBlock] Camera Rig“, proměnnou „Play On Awake“ necháme nastavenou na hodnotu „false“, proměnnou „Height“ nastavíme na hodnotu „1.6“ pro zobrazení kamery v požadované výši a do proměnné „Rend“ přiřadíme objekt „Fade“, který se nachází pod „CenterEyeAnchor“ (Obr. 5-12). Tím máme připravené rozmístění uživatelů, které později v tvorbě funkcionality scény využijeme.

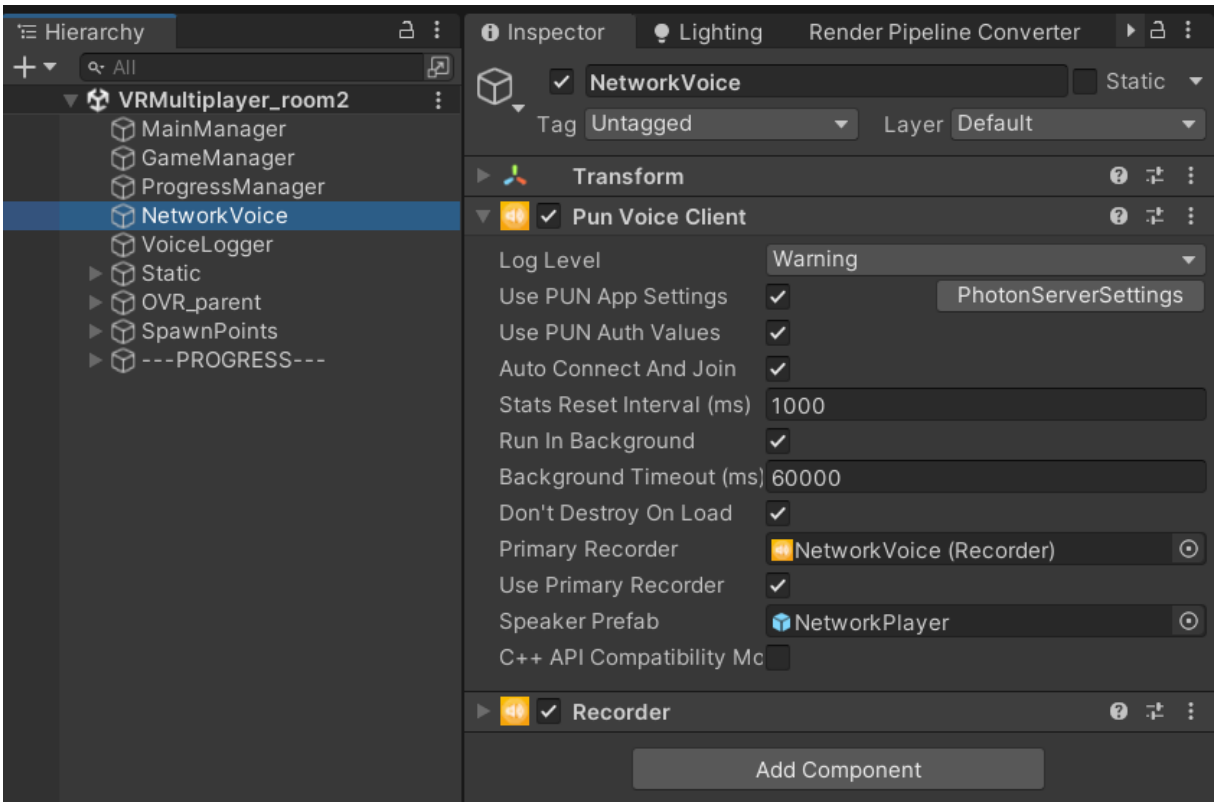


Obr. 5-12 Objekty pro rozmístění uživatelů

5.2.2. Photon Voice a Photon View

V této kapitole se budeme zabývat verbální komunikací uživatelů a promítnutí interaktivních objektů na server.

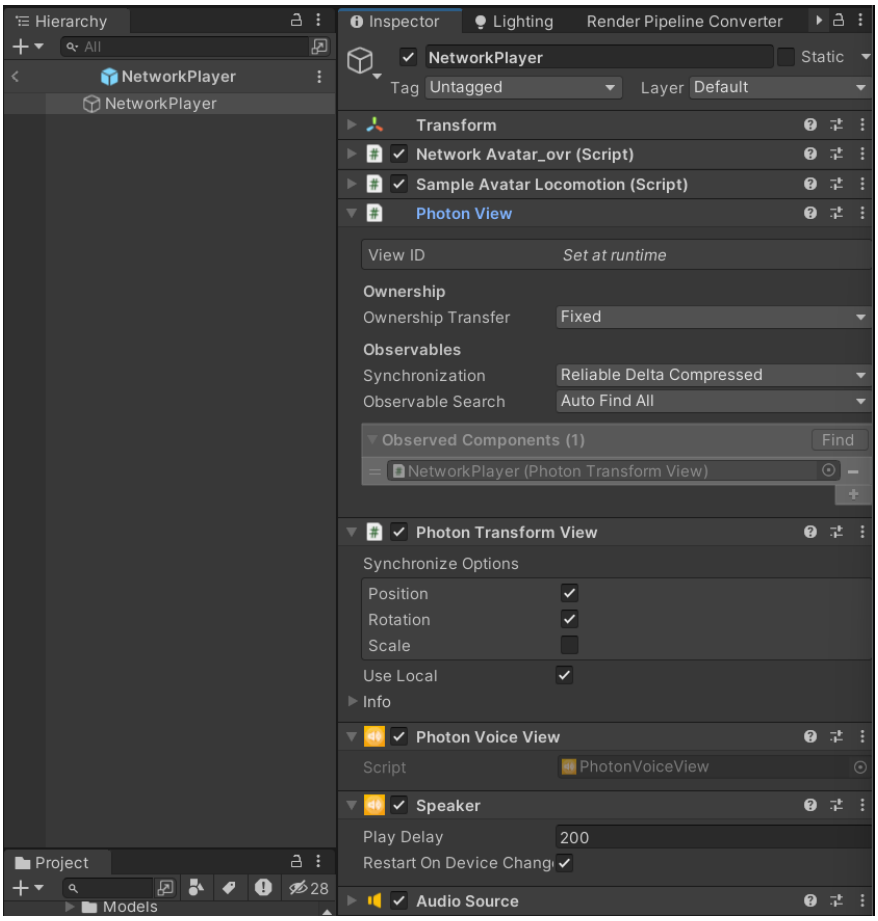
Za účelem zajištění hlasové komunikace mezi uživateli vytvoříme GameObject „NetworkVoice“. Na tento objekt aplikujeme komponenty „Pun Voice Client“ pro správu hlasové komunikace a „Recorder“ pro záznam uživatelského hlasu. V komponentě „Pun Voice Client“ přiřadíme proměnné „Primary Recorder“ objekt, na kterém je skript použit. Proměnné „Speaker Prefab“ přiřadíme prefabrikát již vytvořeného síťového uživatele „NetworkPlayer“. Dále vytvoříme objekt „GameObject“ s názvem „VoiceLogger“ a přiřadíme mu komponentu „Voice Logger“. (Obr. 5-13)



Obr. 5-13 Objekt "NetworkVoice" s komponenty

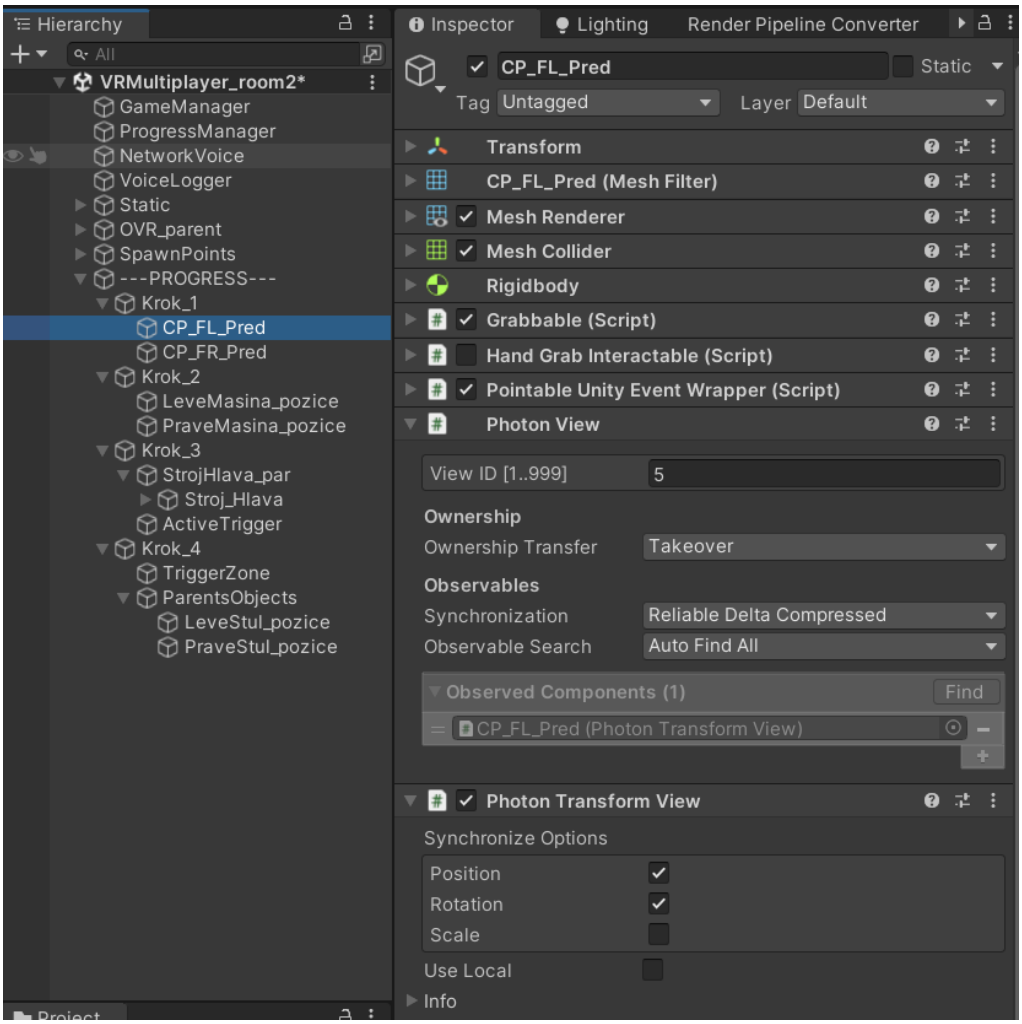
Nyní můžeme přejít na našeho síťového uživatele „NetworkPlayer“, který se nachází ve složce „Resources“. Přiřadíme na něj komponenty „Photon Voice View“, „Speaker“ a „AudioSource“. Tím jsme zajistili verbální komunikaci.

Dále implementujeme synchronizaci polohy a stavu objektů mezi všemi uživateli v síti pomocí komponenty „Photon View“, což umožňuje real-time aktualizace a interakce v distribuovaném virtuálním prostředí. Zůstaneme na síťovém uživateli a přiřadíme mu komponentu „PhotonView“, na které ponecháme nastavení proměnné „Ownership Transfer“ na hodnotu „Fixed“. Pokud je nastavená hodnota „Fixed“, nelze při průběhu scény změnit vlastníka objektu. Protože v budoucnu bude náš síťový uživatel obsahovat avatara, promítneme jeho pohyby a rotaci přiřazením komponenty „Photon Transform View“, kde nastavíme proměnné „Position“ a „Rotation“ na hodnotu „true“. (Obr. 5-14)



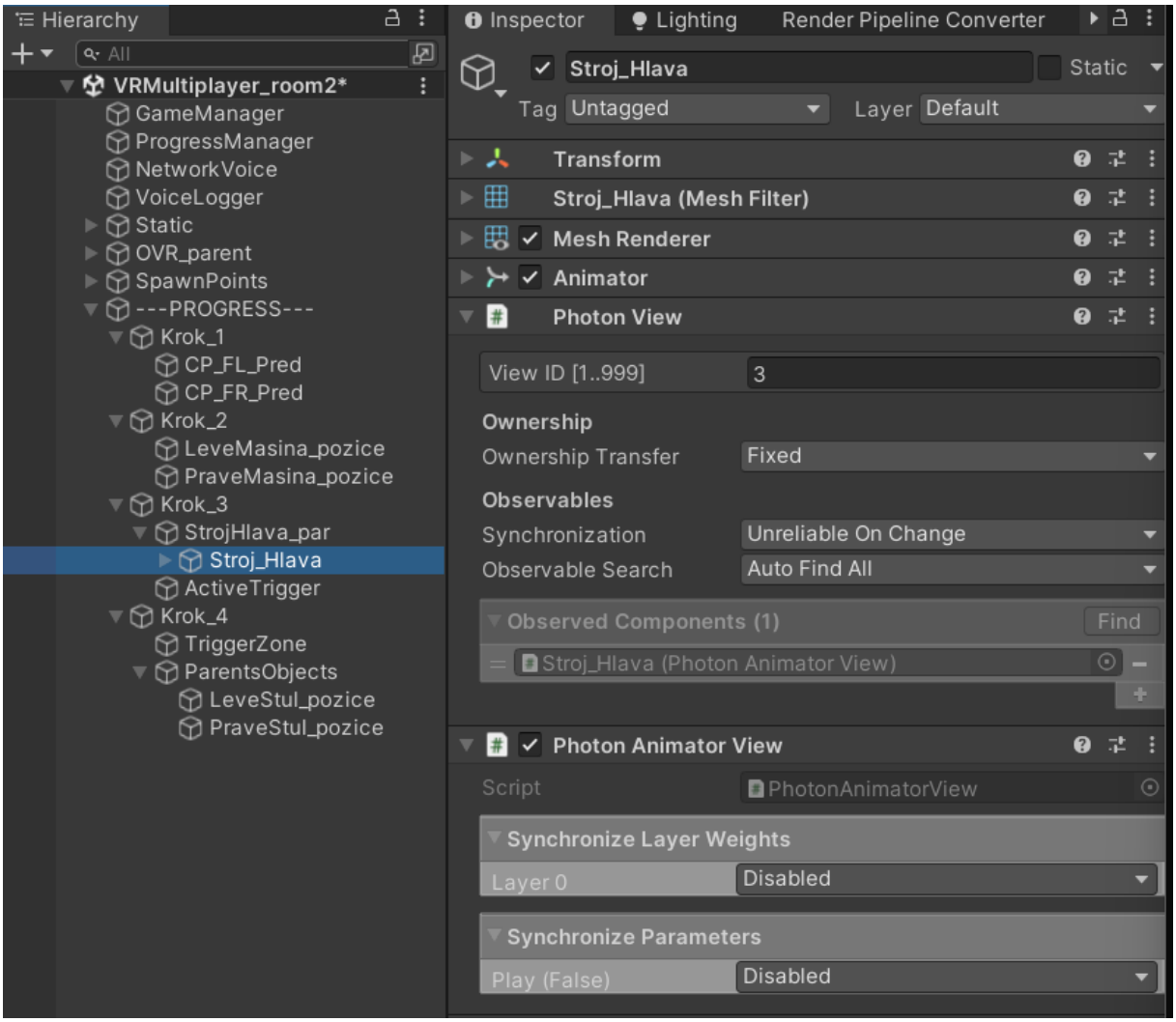
Obr. 5-14 Nastavení síťového uživatele pro komunikaci a promítnutí pohybů na server

Přejdeme zpět do scény do objektu „---PROGRESS---“. V kroku jedna přidáme na obou dílech komponenty „Photon View“ a „Photon Transform View“. Na komponentě „Photon View“ nastavíme proměnnou „Ownership Transfer“ na hodnotu „Takeover“, protože budeme potřebovat předávat vlastnictví objektu mezi síťovými uživateli. V komponentě „Photon Transform View“ nastavíme na proměnných „Position“ a „Rotation“ hodnotu „true“ a proměnnou „Use Local“ na hodnotu „false“ kvůli práci mezi uživateli. (Obr. 5-15)



Obr. 5-15 Nastavení dílů pro server

Protože krok dvě slouží pouze pro nastavení pozic dílů, který sami promítají svou pozici a rotaci a bude fungovat lokálně, nemusíme ho zahrnovat do serveru. Proto se přesuneme na krok tři, kde synchronizujeme „Stroj_Hlava“. Na objekt přiřadíme komponentu „Photon View“. Protože s objektem pohybujeme pomocí animace, nebude nám zde stačit použít komponentu „Photon Transform View“, proto použijeme komponentu „Photon Animator View“, která nám synchronizuje naší animaci se serverem (Obr. 5-16). Krok čtyři můžeme znovu vynechat, protože slouží pouze k umístění dílů stejně jako v kroku dvě.



Obr. 5-16 Synchronizace animace strojní hlavy se serverem

Protože ve funkcionalitě aplikace budeme chtít řídit náš proces napříč serverem, vytvoříme si objekt „GameObject“ s názvem „ProgressManager“ v kořenu scény. Na objekt navážeme komponentu „Photon View“ v základním nastavení. Nyní máme připravenou aplikaci k řešení její funkcionality.

5.2.3. Programování funkcionality

Upravíme avatar pro síťové použití vytvořením nového skriptu „NetworkAvatar_ovr“, který rozšíří funkcionality základního skriptu „SampleAvatarEntity“ z balíčku „Meta Avatars SDK“. Tento nový skript umožní efektivní synchronizaci stavů a pohybů avatara mezi všemi uživateli v síti. Skript vložíme na objekt „NetworkAvatar“ ve složce „Resources“. Metody a proměnné ze skriptu „SampleAvatarEntity“ nebudou okomentovány, protože tvorba avatara není předmětem diplomové práce, okomentovány budou pouze úpravy ve skriptu oproti původnímu. Celý skript si lze prohlédnout v [Příloha 3.].

```
void ConfigureAvatarEntity()
{
    m_photonView = GetComponent<PhotonView>();
    if (m_photonView.IsMine)
    {
        SetIsLocal(true);
```

```
        _creationInfo.features =
Oculus.Avatar2.CAPI.ovrAvatar2EntityFeatures.Preset_Default;
        SampleInputManager sampleInputManager =
OvrAvatarManager.Instance.gameObject.GetComponent<SampleInputManager>();
        SetBodyTracking(sampleInputManager);
        gameObject.name = "MyAvatar";
        StartCoroutine(Loadhands_cor("Player"));
    }
    else
    {
        SetIsLocal(false);
        _creationInfo.features =
Oculus.Avatar2.CAPI.ovrAvatar2EntityFeatures.Preset_Remote;
        SampleInputManager sampleInputManager =
OvrAvatarManager.Instance.gameObject.GetComponent<SampleInputManager>();
        SetBodyTracking(sampleInputManager);
        gameObject.name = "OtherAvatar";
    }
}
```

V metodě „ConfigureAvatarEntity()“, se musíme rozhodnout zda je avatar generován pro nás nebo je generován pro zobrazení dalšího uživatele. Toto zabezpečení provedeme kontrolou komponenty „Photon View“, kterou jsme na objekt dříve přidali, příkazem „IsMine“. Pokud je podmínka splněna, nastavujeme na avatarovi lokální stav ve scéně, pojmenujeme ho „MyAvatar“ a zavoláme korutinu „LoadHands_cor(‘‘Play‘‘)“, která bude později okomentována bude sloužit k generování kolizí pod ruce avatara. Pokud podmínka splněna není, nastavujeme na avatarovi globální stav a pojmenujeme ho „OtherAvatar“. Korutinu zde nevoláme, protože druhý uživatel ve své aplikaci bude mít již kolize přiřazeny na základě stejného skriptu.

```
IEnumerator Loadhands_cor(string _tag)
{
    yield return new WaitForSeconds(1f);
    GameObject leftHand = this.transform.GetChild(2).gameObject;
    leftHand.tag = _tag;
    leftHand.AddComponent<BoxCollider>().size = new Vector3(0.1f, 0.1f, 0.1f);
    leftHand.GetComponent<BoxCollider>().center = new Vector3(0.1f, 0f, 0f);
    GameObject rightHand = this.transform.GetChild(3).gameObject;
    rightHand.tag = _tag;
    rightHand.AddComponent<BoxCollider>().size = new Vector3(0.1f, 0.1f, 0.1f);
    rightHand.GetComponent<BoxCollider>().center = new Vector3(-0.1f, 0f, 0f);
}
```

Korutina, kterou voláme v metodě „ConfigureAvatarEntity()“ pokud je proměnná „IsMine“ rovna hodnotě „true“ zajišťuje přiřazení kolizí pod ruce uživatele pro interakci s objekty ve scéně, které k tomu následně připravíme. Interakce s objekty bude prováděna na základě tagu, který je nastaven při volání korutiny na hodnotu „Player“.

Nyní si vytvoříme skript s názvem „GameManager_ovr“, který bude řídit instanci avatara a řídit umístění uživatelovi kamery včetně avatara. Také si zde pojistíme načtení Lobby místnosti, pokud uživatel náhodně ztratí připojení k serveru. Vytvoříme si v kořenu hierarchie objekt „GameObject“ s názvem „GameManager“ a připojíme na něj tento skript. Celý skript je k dispozici v [Příloha 4.].

```
#region Public Fields
public static GameManager_ovr Instance;
```

```
public BasicRecenter_ovr[] spawnPoints;
#endregion

#region Private Fields
[Header("Inicializace hráče")]
[SerializeField] ulong m_userId;
#endregion
```

Vytvoříme proměnnou „Instance“, která je statického typu. Objekt, který bude do této instance přiřazen nebude při načtení jiné scény zničen a zachová své aktuální proměnné. Činíme tak z důvodu zachování informacích o uživateli při opětovném připojení. Protože máme více než jedno umístění uživatele ve scéně, uděláme z proměnné „spawnPoints“ díky hranatým závorkám pole objektů. Tak budeme moci přiřadit neomezený počet umístění do této proměnné.

Generování avatara komunikuje s Oculus platformou, proto zde vytvoříme proměnnou „m_userId“ ke které se ve skriptu bude přiřazovat ID uživatele.

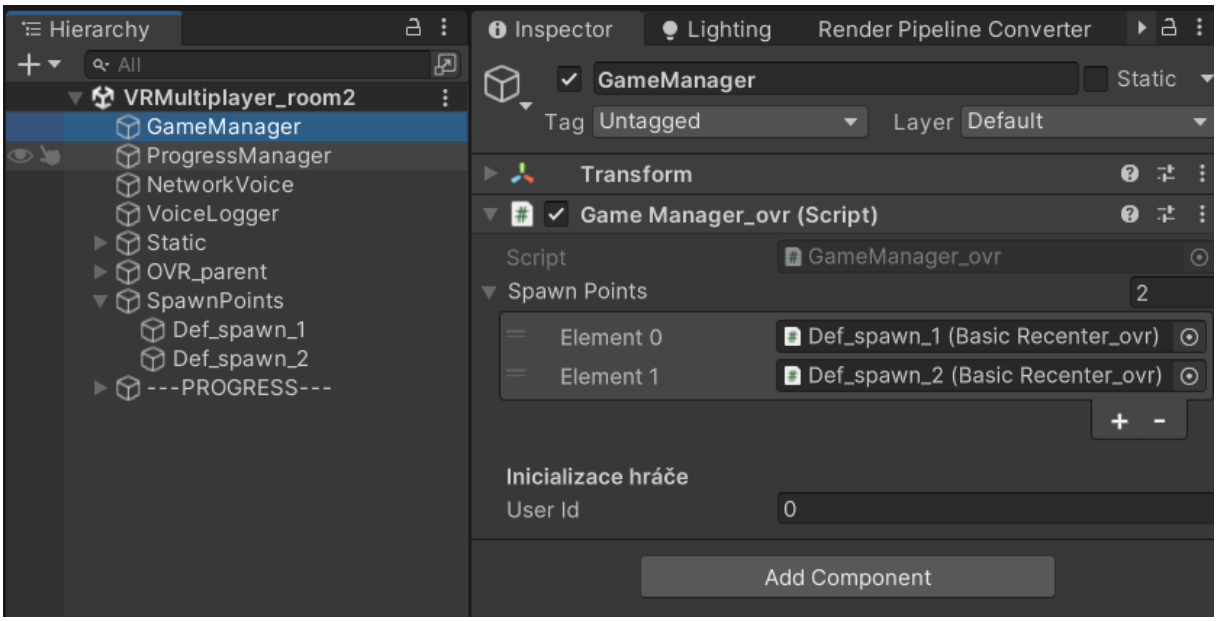
```
#region MonoBehaviour CallBacks
private void Awake()
{
    if(Instance == null)
    {
        Instance = this;
    }
    else
    {
        Destroy(this.gameObject);
    }

}
private void Start()
{
    StartCoroutine(SetUserIdFromLoggedInUser());
    StartCoroutine(InstantiateNetworkedAvatarOnceInRoom());

    if (!PhotonNetwork.IsConnected)
    {
        SceneManager.LoadScene("VRMultiplayer_Launcher");

        return;
    }

    spawnPoints[PhotonNetwork.CurrentRoom.PlayerCount - 1].ResetVRPosition();

    Debug.LogError("vypisuji spawnpoint: " +
spawnPoints[PhotonNetwork.CurrentRoom.PlayerCount - 1]);
}
#endregion
#region Public Methods
public void InstantiateNetworkedAvatar()
{
    Int64 userId = Convert.ToInt64(m_userId);
    object[] objects = new object[1] { userId };
    PhotonNetwork.Instantiate("NetworkPlayer",
spawnPoints[PhotonNetwork.CurrentRoom.PlayerCount-1].transform.position,
Quaternion.identity, 0, objects);
}
#endregion

#region Coroutines
```

```
IEnumerator SetUserIdFromLoggedInUser()…

IEnumerator InstantiateNetworkedAvatarOnceInRoom()
{
    while (PhotonNetwork.InRoom == false)
    {
        Debug.Log("Waiting to be in room before intantiating avatar");
        yield return null;
    }

    InstantiateNetworkedAvatar();
}
#endregion
```

V metodě „Awake()“ na začátku scény tvoříme instanci daného skriptu, pokud však již není vytvořena. Pokud by instance byla vytvořena, starou instanci ponecháme a novou zničíme. Tím zabezpečujeme uchování hodnot.

Metoda „Start()“ proběhne až po předchozí metodě. V ní pomocí korutiny „SetUserIdFromLoggedInUser()“ nastavíme ID uživatele. Tato korutina byla již součástí balíčku Meta, a proto jsme ji pouze využili. Dále voláme korutinu „InstantiateNetworkedAvatarOnceInRoom“, ve které probíhá čekání připojení do místnosti a následně vytvoření avatara na základě veřejné metody „InstantiateNetworkedAvatar()“, která bude vysvětlena později. Dále se ujišťujeme, zda při procesu připojení do místnosti, připojení neselhalo. Pokud ano načte se nám zpět Lobby místnost a průběh skriptu se přeruší. Na závěr metody „Start()“ voláme proměnnou „spawnPoints“ na základě počtu uživatelů v místnosti. Odečítáme od počtu uživatelů hodnotu jedna z důvodu, že pole má první hodnotu na pozici nula. Z konkrétního proměnné „spawnPoints“ se zavolá metoda „ResetVRPosition()“, která nastavuje pozici naší kamery. Je žádoucí, aby se na stejné pozici vytvořil instanci i avatar a to nám popisuje metoda „InstantiateNetworkAvatar()“. Kromě použití ID uživatele, metoda pomocí příkazu „PhotonNetwor.Instatiate“ vytváří instanci objektu s názvem „NetworkPlayer“, který se nachází ve složce „Resources“. Dále zde nastavujeme totožnou pozici a rotaci umístění uživatele podle proměnné „spawnPoints“.

Vrátíme se zpět do scény, kde na objektu „GameManager“ v komponentě „GameManager_ovr“ nastavíme velikost pole „Spawn Points“ na hodnotu „2“. Přiřadíme zde naše dva objekty pro rozmístění uživatelů. (Obr. 5-17)



Obr. 5-17 Nastavení skriptu "GameManager_ovr"

Nyní si připravíme skript pro řízení procesu. Vytvoříme skript s názvem „ProgressManager_ovr“ a přiřadíme ho do scény na objekt „ProgressManager“. Účelem tohoto skriptu bude automatizovat úlohy, které se budou vykonávat pro každý krok. Celý skript je k dispozici v [Příloha 5.].

```
[Serializable]
public class ProgressList
{
    public string _name;
    public UnityEvent _nextAction;
    public float _delayTime;
    public UnityEvent _delayAction;
    public int _count;
    [HideInInspector]
    public int _progressCount;
    public bool _done;
}
```

Vytvoříme si veřejnou serializovatelnou třídu, která nám poslouží ke všem krokům. Proměnná „_name“ nám umožní přehlednost pole díky jeho pojmenování. Ke každému kroku přiřadíme akci, která se bude dít při splnění kroku („_nextAction“) a akci („_delayAction“), která se bude dít se zpožděním o hodnotu „_delayTime“. Proměnná „_count“ zajišťuje nutný počet splnění dané podmínky ke spuštění akcí Tato proměnná se bude porovnávat s proměnou „_progressCount“, která je skryta v inspektoru. Bool „_done“ slouží pro ověření, zda byl úkol splněn a k tomu, aby se akce neprovedly víckrát než jednou.

```
#region Public variables
public List<ProgressList> _progressList = new List<ProgressList>();

[Tooltip("Aktuální progres aplikace")]
public int _progress = 0;
public int _totalProgress;

public TextMeshPro _textVypis;
#endregion
```

Přiřadíme třídu do proměnné „_progressList“, díky tomu budeme moct využívat její proměnné. Dále vytvoříme dvě celočíselné proměnné „_progress“ a „_totalProgress“, které nám poslouží k informaci, ve které části procesu se nacházíme. Výpis budeme provádět do proměnné „_textVypis“.

```
#region Public Methods
public void NextStep(int num)
{
    photonView.RPC(nameof(NextStep_rpc), RpcTarget.All, num);
}
#endregion

#region RPC Methods
[PunRPC]
public void NextStep_rpc(int num)
{
    if(_totalProgress > _progress)
    {
        _progressList[num]._progressCount++;

        if (!_progressList[num]._done && _progressList[num]._progressCount ==
_progressList[num]._count)
        {
            _progressList[num]._nextAction?.Invoke();
            StartCoroutine(DelayAction_cor(_progressList[num]._delayTime,
_progressList[num]._delayAction));

            _progressList[num]._done = true;
            _progress++;

            _textVypis.text = "Postup: " + _progress + "/" + _progressList.Count;
        }
    }
}
#endregion

#region Coroutines
IEnumerator DelayAction_cor(float time, UnityEvent action)
{
    yield return new WaitForSeconds(time);
    action?.Invoke();
}
#endregion
```

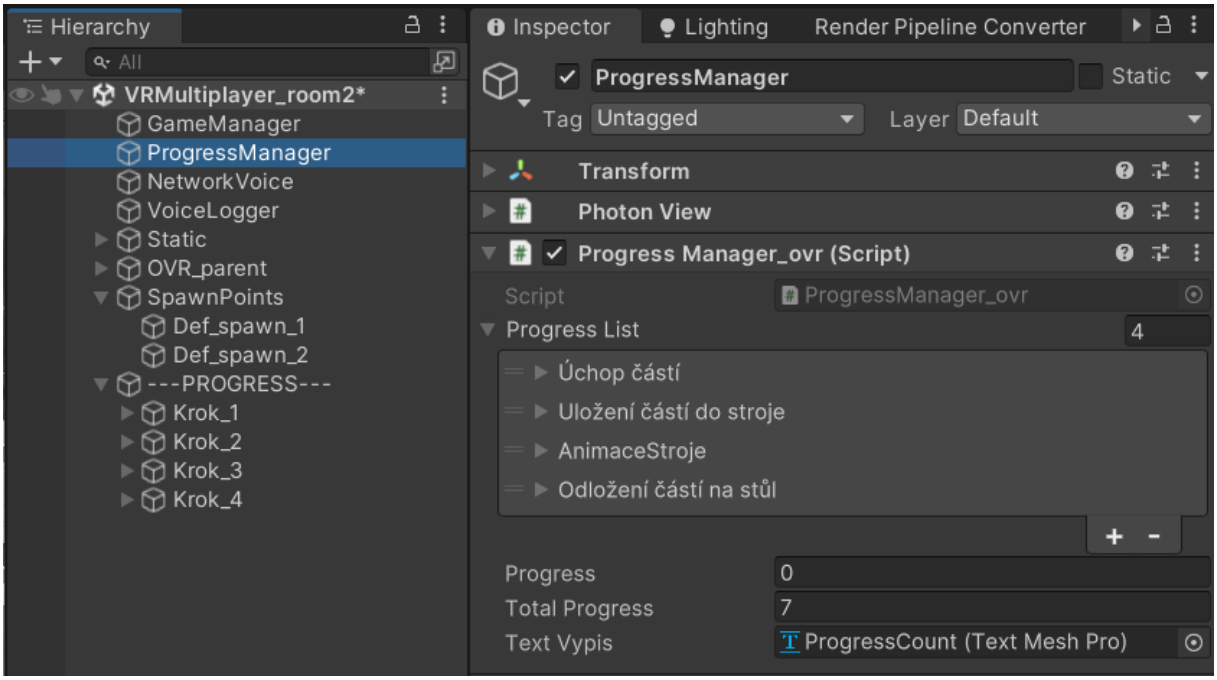
Při volání veřejné metody „NextStep(int num)“, docílíme volání RPC metody „NextStep_rpc“ pro všechny uživatele s konkrétním krokem podle lokální proměnné „num“.

PunRPC je funkce ve Photon PUN 2, která umožňuje volání funkcí na objektech v síti na dálku. To znamená, že klient může volat funkce na objektech, které jsou vlastněny a provozovány jinými klienty nebo serverem.

Metoda „nextStep_rpc(int num)“ se provádí pouze tehdy pokud celkové množství kroků větší než dosavadní počet splněných kroků. Tím pojistíme, že se pod dokončení procesu již v tomto skriptu nic nestane. Pokud je tato podmínka splněna, zvyšujeme na konkrétním kroku hodnotu „_progressCount“ o jedna. Následuje další podmínka, která kontroluje, zda krok již nebyl splněn, zároveň zda se nutný počet splnění kroku rovná s již udělaným počtem. Pokud je i tato podmínka splněna volá se akce kroku „_nextAction“. Otazník v této části skriptu zajistí, že pokud nebyla této proměnné přiřazena žádná akce, proměnná se nezavolá. Dále pokračujeme

voláním zpožděné akce „_delayAction“ s konkrétním zpožděním „_delayTime“ pomocí korutiny „DelayAction_cor(float time, UnityEvent action)“. Protože již všechny akce byly splněny, řekneme že krok byl splněn a přičteme hodnotu jedna k proměnné „_progress“. Na závěr aktualizujeme výpis postupu do proměnné „_textVypis“ za pomocí textového pole, proměnné „_progress“, která určuje aktuální postup a počet hodnot v listu „_progressList“.

Přesuneme se zpět do scény a na objektu „ProgressManager“ v komponentě „ProgressManager_ovr“ nastavíme proměnnou listu „Progress List“ na hodnotu čtyři, podle počtu kroků v procesu. Pojmenujeme konkrétní položky v listu do proměnné „Name“ podle (Obr. 5-18). Do proměnné „Text Vypis“ přiřadíme objekt „ProgressCount“. Později doplníme tento skript o akce v konkrétních krocích.



Obr. 5-18 Nastavení listu ve skriptu "ProgressManager_ovr“

Nyní budeme pokračovat skripty pro konkrétní kroky. Nejdříve zajistíme, aby uživatelé mohli začít vykonávat proces až ve chvíli, kdy jsou všichni účastníci přítomní. K tomuto účelu vytvoříme skript s názvem „ActiveOnStart_ovr“. Celý skript je k dispozici v [Příloha 6.].

```
public UnityEvent afterAllInRoom;

void Update()
{
    if (PhotonNetwork.CurrentRoom.PlayerCount ==
PhotonNetwork.CurrentRoom.MaxPlayers)
    {
        afterAllInRoom.Invoke();
        Destroy(this);
    }
}
```

Vytvoříme veřejnou proměnnou typu „UnityEvent“ s názvem „afterAllInRoom“. Tato akce bude sloužit k vykonání námi definovaných akcí po splnění podmínky. Využijeme zde metodu „Update ()“, která se volá v každém framu aplikace, dokud je objekt, na kterém je připojená, aktivní. V této metodě vytvoříme podmínku, která kontroluje, zda aktuální počet

uživatelů v místnosti odpovídá maximálnímu počtu uživatelů v místnosti. Pokud je tato podmínka splněna provede se akce „afterAllInRoom“ a skript se odstraní. Skript vložíme na objekty dílů dveří.

Pro kontrolu chycení objektů a kontrolu vlastníka vytvoříme skript s názvem „GrabOnOff_ovr“. Celý skript je k dispozici v [Příloha 7.].

```
#region Public variables
public bool _activeGrab = true;

[HideInInspector]
public bool _inHand;
#endregion

#region Private variables
private HandGrabInteractable interactable;

PhotonView _photonView;
#endregion
```

Proměnná „_activeGrab“ slouží k zpřístupnění úchopu. Proměnná „_inHand“ nám poslouží ke kontrole stavu úchopu objektu. Mezi privátní proměnné vytvoříme proměnnou „interactable“, která zpřístupní skript z balíčku Meta. Dále budeme využívat komponentu v proměnné „_photonView“.

```
#region MonoBehaviour Callbacks
private void Start()
{
    interactable = GetComponent<HandGrabInteractable>();
    _photonView = GetComponent<PhotonView>();
}

private void Update()
{
    if (PhotonNetwork.CurrentRoom.PlayerCount ==
PhotonNetwork.CurrentRoom.MaxPlayers && _activeGrab)
    {
        interactable.enabled = _photonView.IsMine;
    }
}
#endregion
```

Nejdříve pomocí metody „Start()“ načteme do privátních proměnných komponenty z daného objektu. Následně v metodě „Update()“ provedeme kontrolu počtu uživatelů v konkrétní místnosti oproti maximálnímu počtu uživatelů na místnost a zda je naše proměnná „_activeGrab“ nastavena na hodnotu „true“. Pokud je tato podmínka splněna, udržuje komponenta pod proměnnou „interactable“ stav vlastníka objektu.

```
#region Public Methods
public void OnGrab()
{
    // Volá funkci RPC na serveru pro aktualizaci uchopení.
    photonView.RPC("UpdateGrabCount", RpcTarget.All, true);
}

public void OffGrab()
{
```

```
    // Volá funkci RPC na serveru pro aktualizaci uchopení.
    photonView.RPC("UpdateGrabCount", RpcTarget.All, false);
}
public void GrabOn_meth()
{
    _activeGrab = true;
}
public void GrabOff_meth()
{
    _activeGrab = false;
    interactable.enabled = false;
}
#endregion

#region RPC Methods
[PunRPC]
public void UpdateGrabCount(bool take)
{
    _inHand = take;
}
#endregion
```

Pokračovat budeme metodami „GrabOn_meth()“ a „GrabOff_meth()“, které budou udržovat lokální stav uchycení objektu. Následně pomocí volání vzdálené metody „UpdateGrabCount(bool take)“ pomocí metod „OnGrab()“ a „OffGrab()“ zajistíme aktuální stav úchopů v rámci serveru. Skript přidáme na objekty dílů dveří.

U objektů dílů musíme zajistit předávání mezi uživateli. Toho docílíme změnou vlastníka na komponentě „PhotonView“. Pro tento účel vytvoříme skript s názvem „ChangeOwner_ovr“. Celý skript je k dispozici v [Příloha 8.]

```
#region Private variables
PhotonView _photonView;
private GrabOnOff_ovr _grabbers;
#endregion

#region Monobehaviour Callbacks
private void Start()
{
    _photonView = GetComponent<PhotonView>();
    _grabbers = GetComponent<GrabOnOff_ovr>();
}

public void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player") && this.enabled)
    {
        ChangeOnMe();
    }
}
#endregion

#region Public methods
public void ChangeOnMe()
{
    if (_photonView != null && !_photonView.IsMine && !_grabbers._inHand)
    {
        _photonView.TransferOwnership(PhotonNetwork.LocalPlayer);
    }
}
```

```
#endregion
```

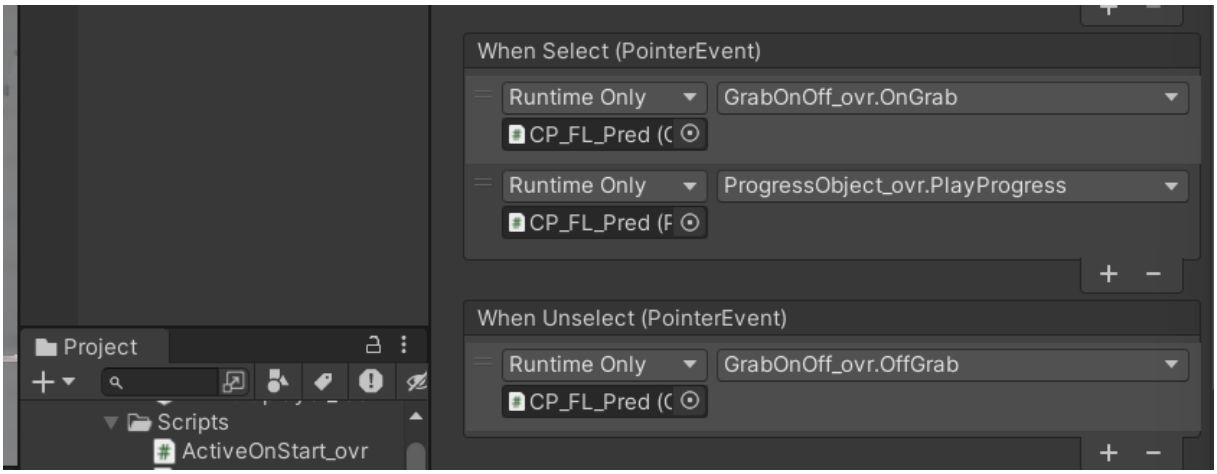
Vytvoříme privátní proměnné „_photonView“ a „_grabbers“, ke kterým budou přiřazeny komponenty objektu v metodě „Start()“. Následně v metodě „OnTriggerEnter(Collider other)“, která se provádí při vstupu jiné kolize do objektu vytvoříme metodu, která vstup omezuje pouze na kolize s tagem „Players“. Zároveň řešíme, zda je skript na objektu aktivní. Pokud je podmínka splněna, zavolá se metoda „ChangeOnMe()“, která obsahuje podmínku kde proměnná „_photonView“ nesmí být prázdná, objekt musí mít jiného vlastníka a zároveň nesmí být uchopen v ruce. Pokud je tato podmínka splněna funkce „TransferOwnership(PhotonNetwork.LocalPlayer)“ změní vlastníka objektu na lokálního uživatele. Skript vložíme na objekty dílů dveří.

Na závěr si pro objekty dveří vytvoříme pomocný skript s názvem „ProgressObject“, který spouštět konkrétní krok při zavolání metody. Celý skript je v [Příloha 9.].

```
#region Public variables
public int progressCount;
#endregion

#region Private variables
private ProgressManager_ovr _progressManager;

private bool oneShot;
#endregion

#region MonoBehaviour Callbacks
private void Start()
{
    _progressManager = FindAnyObjectByType<ProgressManager_ovr>();
}
#endregion

#region Public Methods
public void PlayProgress ()
{
    if(!oneShot)
    {
        oneShot = true;
        _progressManager.NextStep(progressCount);
    }
}
#endregion
```

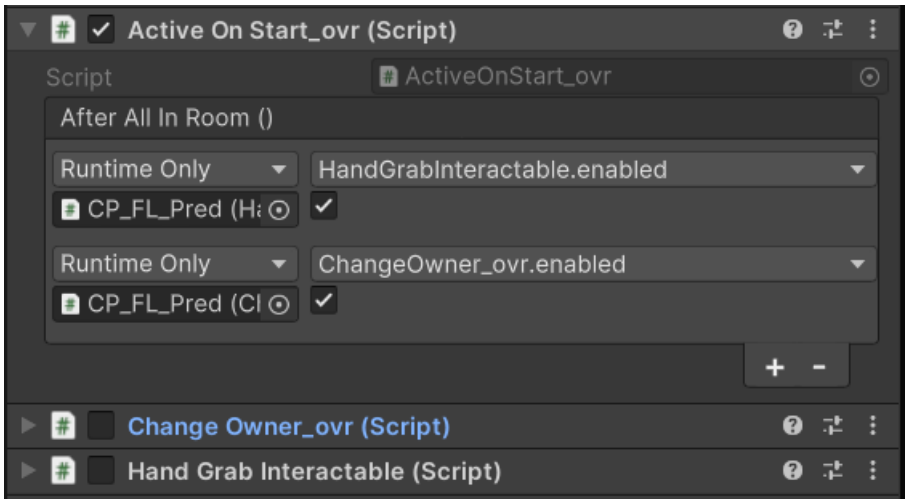
Vytvoříme veřejnou proměnnou „_progressCount“, která nám udá konkrétní krok pro spuštění. Dále privátní proměnnou „_progressManager“, ke které na startu aplikace přiřadíme objekt s komponentou „ProgressManager_ovr“ a bool „_oneShot“. Metoda „PlayProgress()“ Zajišťuje při splnění podmínky volání metody „NextStep()“ ze skriptu „ProgressManager_ovr“ ve scéně s parametrem „_progressCount“. Bool nám zařídí, že se metoda provede pouze jednou. Tento skript také vložíme na díly dveří.

Přesuneme se zpět do scény a provedeme nastavení skriptů, které jsme přiřadili objektům dílů dveří. Ve skriptu „PointableUnityEventWrapper“ přiřadíme v sekci „When Select (PointerEvent)“ dvě komponenty, „OnGrab“ a „PlayProgress“. V sekci „When Unselect (PointerEvent)“ přiřadíme jednu komponentu „OffGrab“. Tím zajistíme volání daných metod při uchopení a upuštění objektu. (Obr. 5-19)



Obr. 5-19 Nastavení dílu dveří - komponenta Pointable Unity Event Wrapper

Pokračujeme nastavením komponenty „Active On Start_ovr“, kde v poli „After On Start_ovr“ přidáme dvě události. První událost bude zapínat komponentu „HandGrabInteractable“ a druhá bude zapínat komponentu „ChangeOwner_ovr“. Protože tyto komponenty chceme zapnout pomocí tohoto skriptu na začátku scény, tak je ze základu na objektech dílů vypneme (Obr. 5-20). Nakonec na komponentě „ProgressObject_ovr“ nastavíme hodnotu „ProgressCount“ na nulu, protože budeme chtít volat první krok.



Obr. 5-20 Nastavení dílu dveří - komponenta Active On Start_ovr

Protože práce na dílech dveří již skončila, můžeme upravit první krok procesu v objektu „ProgressManager“. Nakonec uchopení dílů zajistí celou část kroku jedna, jediné, co musíme nastavit v prvním kroku ve skriptu „ProgressManager_ovr“ je proměnná „Count“ na hodnotu dva, protože máme dva díly dveří, a proto musí být uchopeny oba díly pro splnění celého kroku. Můžeme se přesunout na krok dvě.

Ve druhém kroku potřebujeme zajistit umístění dveří do stroje. K tomuto účelu si vytvoříme skript s názvem „SnappObjectWithName_ovr“ a přiřadíme ho na objekty v kroku dvě. Celý skript je v [Příloha 10.].

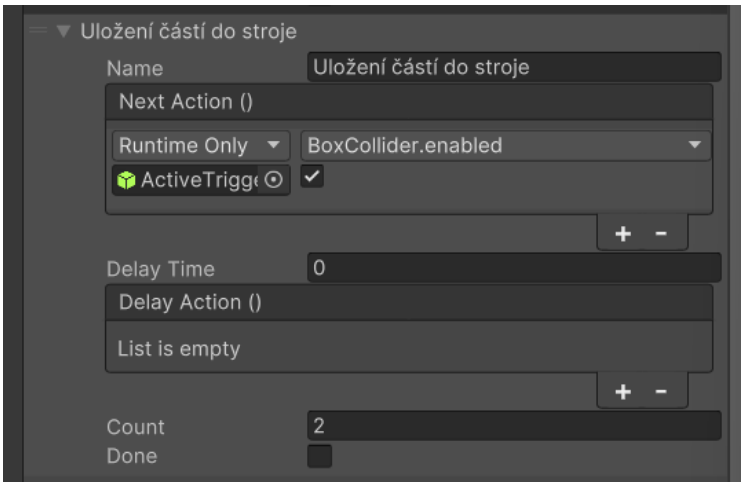
```
#region Public variables
public string _nameObject;
public UnityEvent _actionEvent;
#endregion
```

```
#region Private variables
bool _activeTF = true;
#endregion
```

Vytvoříme si dvě veřejné proměnné. Textovou proměnnou „_nameObject“, která bude sloužit pro kontrolu vstupu a „_actionEvent“, který se provede při splnění podmínek. Dále si zajistit pouze jedno použití, proto vytvoříme bool „_activeTF“.

```
#region MonoBehaviour Callbacks
private void OnTriggerStay(Collider other)
{
    if (_activeTF && other.name == _nameObject &&
!other.GetComponent<GrabOnOff_ovr>()._inHand)
    {
        _activeTF = false;

        other.GetComponent<GrabOnOff_ovr>().GrabOff_meth();

        other.transform.parent = this.transform;
        other.transform.localEulerAngles = Vector3.zero;
        other.transform.localPosition = Vector3.zero;

        if (other.GetComponent<PhotonView>().IsMine)
        {
            _actionEvent.Invoke();
        }
    }
}
#endregion
```

Metoda „OnTriggerStay(Collider other) probíhá neustále ve chvíli vstupu kolize do objektu. Podmínkou „if()“ ji omezíme pouze pokud je „_activeTF“ hodnota „true“, vstupovaný objekt nese název, který je uložen v proměnné „_nameObject“ a zároveň není objekt již držen v ruce. Pokud je tato podmínka splněna metoda zavolá na vloženém objektu metodu „GrabOff_meth“, která nám znemožní opětovné uchopení objektu. Zároveň vložený objekt přiřadí pod sebe jako dětský objekt a vynuluje mu pozici a rotaci. Dále v řešíme další podmínku, která zjišťuje, zda je objekt vlastní, pokud ano provedeme akci na „_actionEvent“. Nyní na objektu „LeveMasina_pozice“ do proměnné „Name Object“ vepíšeme název dílu dveří „CP_FL_Pred“ a přidáme akci, která bude vlastnit objekt „ProgressManager“ na něm volat metodu „NextStep“ s hodnotou „1“ pro krok dvě. To samé uděláme pro objekt „PraveMasina_pozice“ s to změnou, že do proměnné „Name Object“ vepíšeme název „CP_FR_Pred“. Přesuneme se na objekt „ProgressManager“ a nastavíme krok dvě. Protože vkládáme dvě komponenty, proměnná „Count“ bude nastavena na hodnotu „2“. Nyní již při splnění kroku budeme požadovat akci „NextAction()“. Přidáme jednu akci a do ní přiřadíme objekt z kroku 3 v hierarchii s názvem „ActiveTrigger“ kde zapneme komponentu „BoxCollider“. (Obr. 5-21)



Obr. 5-21 Nastavení kroku dvě na komponentě Progress Manager_ovr

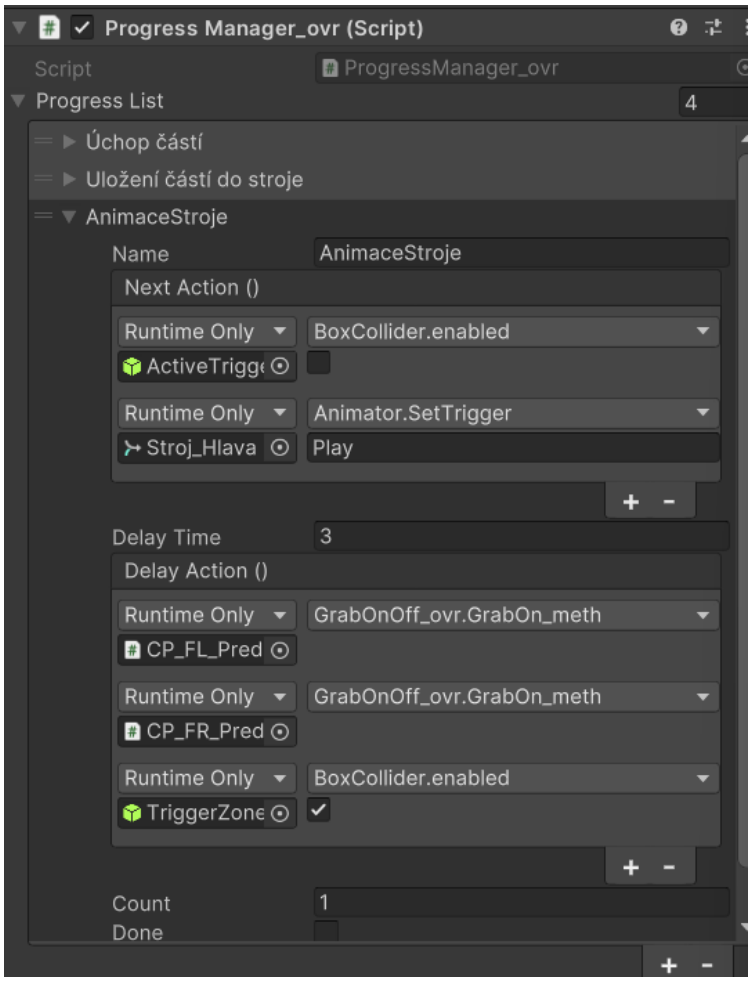
Ve třetím kroku se budeme zabývat prací stroje. Díly máme založené a musíme umět spustit stroj. K tomu si vytvoříme skript s názvem „TriggerAction_ovr“, který při doteku objektu s určitým tagem provede akci. Celý skript je v [Příloha 11.].

```
#region Public variables
public string _tag;
public UnityEvent _action;
public bool _oneShot;
#endregion

#region MonoBehaviout Callbacks
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag(_tag) && _oneShot)
    {
        _oneShot = false;
        _action.Invoke();
    }
}
#endregion
```

Budeme potřebovat tři veřejné proměnné. Proměnnou textového typu „_tag“, akci „_action“ a bool „_oneShot“. Metodu „OnTriggerEnter(Collider other)“ omezíme podmínkou vstupu konkrétního tagu, který se musí rovnat naší proměnné „_tag“ a zajištění že metoda proběhne pouze jednou pomocí proměnné „_oneShot“. Pokud je podmínka splněna, provede se akce „_action“. Skript přiřadíme na objekt „ActiveTrigger“ a nastavíme na něm proměnnou „Tag“ na hodnotu „Player“ a přidáme akci do proměnné „Action“. Do akce vložíme objekt „ProgressManager“ a zavoláme z něj metodu „NextStep“ s hodnotou „2“. Tím spustíme třetí krok procesu.

Přesuneme se zpět na objekt „ProgressManager“ a nastavíme třetí krok. Zde využijeme obě akce. V akci „NextAction“ přidáme dvě události. V jedné bude vložen objekt „ActiveTrigger“, na kterém vypneme komponentu „BoxCollider“ čímž si pojistíme nečinnost objektu. Ve druhé bude vložen objekt „Stroj_Hlava“, na kterém pomocí komponenty „Animator“ a metody „SetTrigger“ nastavíme hodnotu „Play“ a tím se nám spustí animace stroje. Protože animace probíhá okolo 3 vteřin, nastavíme proměnnou „Delay Time“ na hodnotu „3“. Po této době bude volána akce „DelayAction“ do které přidáme tři události. Do první události přiřadíme díl dveří „CP_FL_Pred“ a zavoláme z komponenty „GrabOnOff“ metodu „GrabOnMeth“. Do druhé události přiřadíme díl dveří „CP_FR_Pred“ a zavoláme stejnou metodu jako u předešlého dílu.

Tím po animaci stroje zpřístupníme uchopení dílů. Do třetí události vložíme objekt „TriggerZone“ a zapneme komponentu „BoxCollider“, která nám zpřístupní možnost dokončit proces. Protože tento krok vyžaduje pouze zmáčknutí jednoho tlačítka, proměnná „Count“ bude nastavena na hodnotu „1“. (Obr. 5-22)



Obr. 5-22 Komponenta Progress Manager_ovr – nastavení čtvrtého kroku

Dostáváme se k poslednímu, čtvrtému, kroku. Potřebujeme vyjmout díly ze stroje a odložit je na stůl. Přejdeme na komponentu „TriggerZone“ a vytvoříme zde skript „SnappObjectsWithNamePlus_ovr“. Celý skript je v [Příloha 12.].

```
[Serializable]
 public class SnappObjects
 {
     public string _nameObject;
     public Transform _parObj;
     public bool _activeTF;
     public UnityEvent _actionEvent;
 }
```

Vytvoříme si serializovanou veřejnou třídu „SnappObjects“, která bude vlastnit název objektu „_nameObject“, rodičovský objekt. kam se objekt s názvem bude přesouvat, aktivní stav kroku uchycení „_activeTF“ a akci při uchycení „_ActionEvent“.

```
#region Public variables
public List<SnappObjects> _snappObjects = new List<SnappObjects>();
```

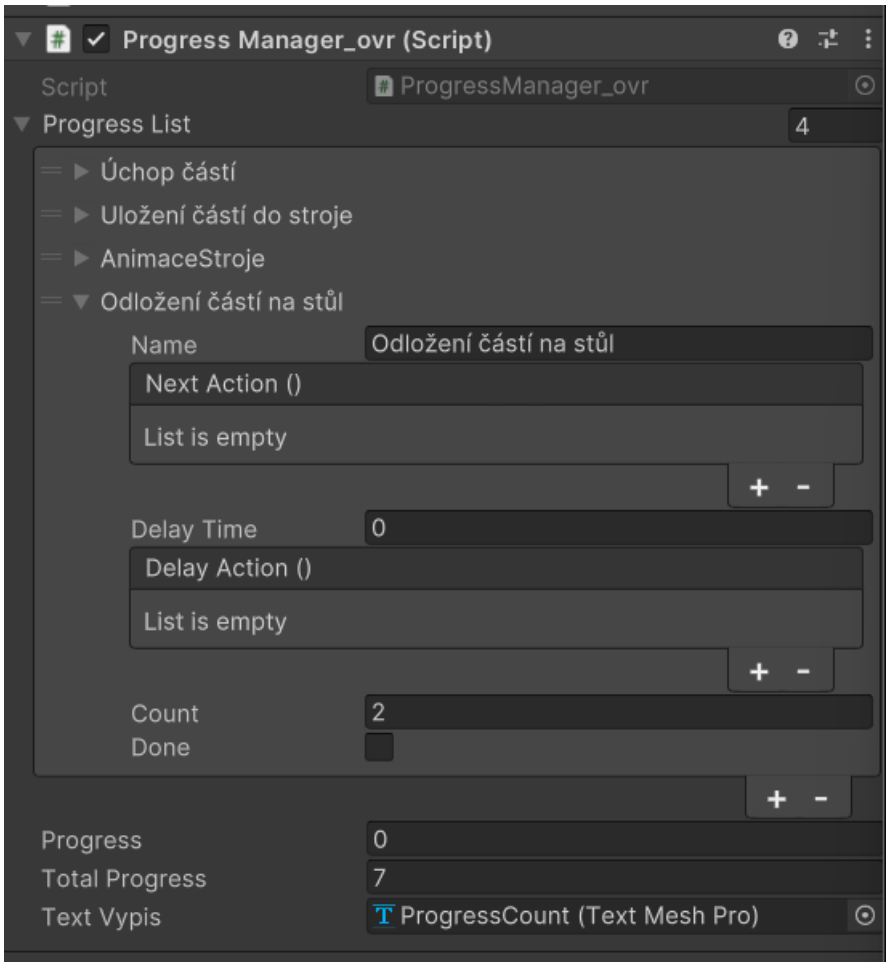
```
#endregion

#region MonoBehaviour Callbacks
private void OnTriggerStay(Collider other)
{
    for(int i = 0; i < _snappObjects.Count; i++)
    {
        if (_snappObjects[i]._activeTF && other.name ==
_snappObjects[i]._nameObject)
        {
            if (!other.GetComponent<GrabOnOff_ovr>()._inHand)
            {
                _snappObjects[i]._activeTF = false;

                other.GetComponent<GrabOnOff_ovr>().GrabOff_meth();

                other.transform.parent = _snappObjects[i]._parObj;
                other.transform.localEulerAngles = Vector3.zero;
                other.transform.localPosition = Vector3.zero;

                if(other.GetComponent<PhotonView>().IsMine)
                {
                    _snappObjects[i]._actionEvent.Invoke();
                }
            }
        }
    }
}
#endregion
```

Vytvoříme proměnnou typu „List<>“, která bude zobrazovat třídu, kterou jsme si předtím vytvořili. Do metody „OnTriggerStay(Collider other)“ přidáme cyklus s krokem „for“, který bude procházet list „_snappObjects“. Pod cyklus vložíme podmínku „if“, kdy objekt v kroku cyklu musí být aktivní pomocí proměnné „_activeTF“ a zároveň vstupovaná kolize musí nést název, která je vložený v proměnné kroku „_nameObject“. Pokud je tato podmínka splněná, kontrolujeme doplňující podmínkou, zda je objekt puštěn z ruky. Při splnění i této podmínky nastavujeme nastavíme proměnnou kroku „_activeTF“ na hodnotu „false“. Tím neumožníme její opětovné splnění. Dále na vloženém objektu pomocí metody „GrabOff_meth“ znemožníme opětovné uchopení, objektu přiřadíme nový rodičovský objekt a to proměnnou „_parObj“ a nastavíme mu hodnoty pozice a rotace na nulu. V závěru kontrolujeme, zda jsme vlastníky objektu a pokud a provedeme akci v daním kroku „_actionEvent“.

Nyní můžeme nastavit na objektu „TriggerZone“ komponentu „SnappObjectsWithNamePlus_ovr“. Proměnné listu „SnappObjects“ nastavíme index na hodnotu „2“. V první položce v listu nastavíme proměnnou „Name Object“ na hodnotu „CP_FL_Pred“, proměnné „Par Obj“ přiřadíme objekt „LeveStul_pozice“, proměnnou „Active TF“ necháme nastavenou na hodnotu „true“ a přidáme akci do proměnné „ActionEvent()“, kde přiřadíme objekt „ProgressManager“ a zavoláme na něm pomocí komponenty „ProgressManager_ovr“ metodu „NextStep“ s hodnotou „3“. Tím volá krok 4. Pro druhou položku listu nastavíme proměnnou „Name Object“ na hodnotu „CP_FR_Pred“, proměnnou „Par Obj“ přiřadíme objekt „PraveStul_pozice“ a dále pokračujeme stejně jako u první položky listu.

Na závěr celé funkcionality se přesuneme zpět na objekt „ProgressManager“, kde v komponentě nastavíme poslední krok, a to krok čtvrtý. Protože ke splnění kroku postačí odložení dílů na stůl, nastavíme pouze proměnnou „Count“ na hodnotu „2“. Nyní můžeme

sečíst hodnoty „Count“ v celém listu „Progress List“ a výsledek „7“ vepsat do proměnné „Total Progress“. (Obr. 5-23)



Obr. 5-23 Komponenta Progress Manager_ovr – nastavení čtvrtého kroku

Tímto jsme dokončili funkcionalitu celkové aplikace a můžeme se přesunout do další kapitoly, kde funkcionalitu popíšeme z uživatelského pohledu.

6. Popis funkcionality vlastního řešení

Tato kapitola se bude zabývat popisem celé aplikace z uživatelského pohledu. Budou zde zmíněny možné kombinace interakcí s objekty v procesu a vysvětlení jejich ošetření proti chybě pro klidný chod aplikace.

Nejdříve se budeme zabývat Lobby místností a následně přejdeme do místnosti s kolaborativním průmyslovým tréninkem.

6.1. Lobby místnost

Uživatel se automaticky zobrazí v Lobby místnosti po spuštění aplikace. Místnost není připojena k serveru, a proto se v ní uživatel nachází sám. Místnost komunikuje se serverem pouze v případě, pokud se uživatel snaží připojit nebo vytvořit virtuální průmyslový trénink. V takovém případě místnost získá ze serveru potřebné informace a uživatele připojí do místnosti nebo ji vytvoří.

Po vstupu do Lobby místnosti si uživatel ihned všimne, že jeho virtuální ruce jsou pevně spojeny s kamerou, což umožňuje přirozenou interakci s uživatelským rozhraním. Umístění uživatele je optimalizováno pro bezprostřední interakci, kde jeho pohled přirozeně padá na uživatelské rozhraní s tlačítky pro připojení do tréninkových místností.

Uživatel má možnost se připojit do místnosti 1 nebo do místnosti 2. Volba více místností je umožněna z důvodu možnosti zaplnění jedné místnosti. Po stisknutí tlačítka se kontrolní panel s tlačítky vypne a zobrazí se text „Připojování…“. Pokud je místnost plná, zobrazí se text „Místnost je plná. Vyber jinou místnost.“. Po třech vteřinách text zmizí a zobrazí se uživateli znovu kontrolní panel s tlačítky, pro výběr místností. Pokud místnost není zaplněna, uživatel se připojí do místnosti s kolaborativním průmyslovým tréninkem.

6.2. Místnost s kolaborativním průmyslovým tréninkem

Tato místnost je již připojena na server. Při příchodu uživatele do místnosti s kolaborativním tréninkem je uživateli vygenerován nový avatar, z důvodu snazší implementace do jiné scény a přiřazení jiných interakčních pomůcek. Uživatel je umístěn v takové pozici, ze které bude moci vykonat celý proces.

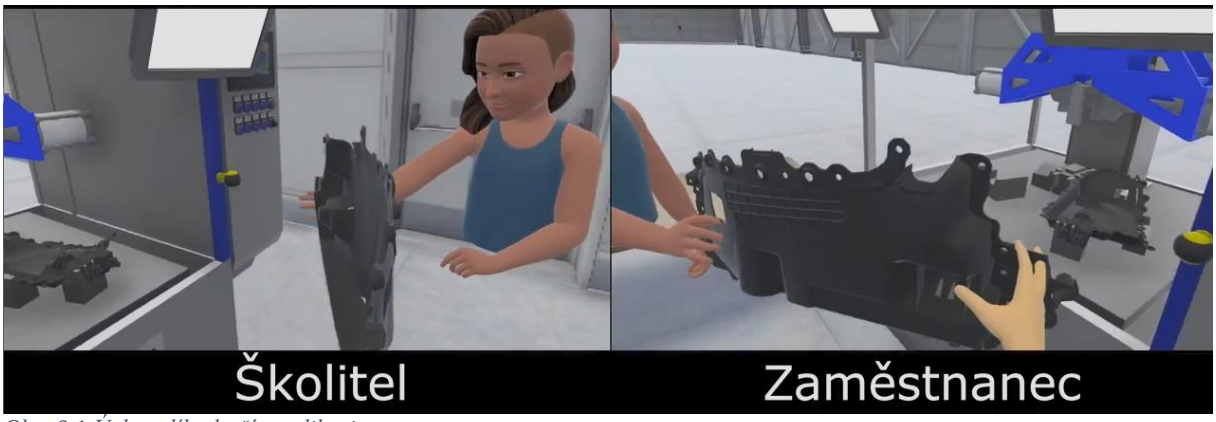
Při vstupu do tréninkové místnosti jsou všechny interaktivní objekty zpočátku neaktivní a aktivují se až po naplnění místnosti potřebným počtem uživatelů, aby se zamezilo předčasným neautorizovaným interakcím. Objekty se zpřístupní při naplnění místnosti, a to v našem případě, když se připojí druhý uživatel. Toto zabezpečení je udělané proto, pokud se jako první připojí zaměstnanec, který má být pod dohledem školitele, neprováděl činnosti sám. Při připojení druhého uživatele se zobrazí avatar druhého uživatele vedle nás. Jako druhý připojený uživatel okamžitě po vstupu do místnosti uvidíme prvního uživatele, což nám umožní společně zahájit tréninkový proces.

Uživatelé mohou komunikovat přímo v aplikaci, buď gestikulací svých avatárů nebo prostřednictvím integrované verbální komunikace, kde 3D audio efekt umožňuje vnímání polohy a směru hlasu ostatních uživatelů. Na avatarech je totiž nastavený 3D výstup zvuku,

takže pokud avatar druhého uživatele stojí za námi a mluví, uslyšíme ho, že nám mluví za zády. Toto nastavení napomáhá k lepší orientaci kolaborativního tréninku.

Proces je tvořen tak, aby uživatelé měli jen minimální nápovědu, která spočívá v možnosti udělat pouze konkrétní krok postupu. Pokud uživatel například bude chtít před založením dílů spustit stroj, nepůjde mu to. Pokud zaměstnanec nebude vědět, jak má pokračovat, měl by se poradit se školitelem. Informaci o postupu je vidět v tabulce na stroji.

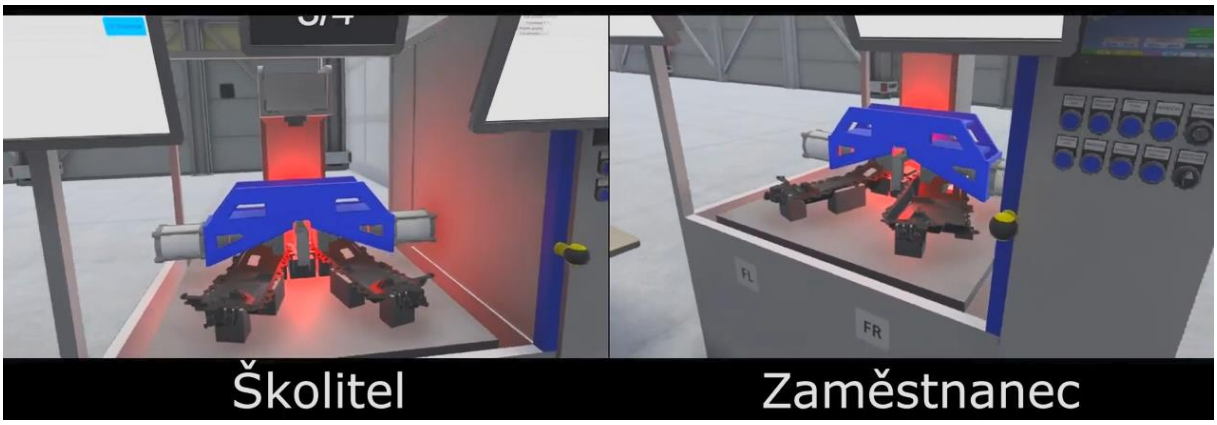
Uživatelé začínají proces tím, že vezmou do rukou oba díly dveří. S dveřmi mohou interagovat oba uživatelé. Interakce druhého uživatele s konkrétními dveřmi, pokud je první drží v ruce, je znemožněna. Skripty na úchopy ve VR od společnosti Meta nejsou připraveny na předávání objektů v rámci Photon. Pokud však díl dveří uživatel upustí, zpřístupní se možnost získat nového vlastníka a uchopit tento předmět druhým uživatelem, nebo znovu prvním. (Obr. 6-1)



Obr. 6-1 Úchop dílu dveří v aplikaci

Dalším krokem procesu je založení dílů dveří do stroje. Oba díly mají své specifické místo ve stroji, které je rozlišeno podle nálepek „FR“ a „FL“. Díly může vložit kdokoliv z uživatelů. Po vložení dílu, se díl nastaví na pozici ve stroji a znemožní se uživatelům úchop daného dílu. Pokud díl založí, není žádoucí, aby s ním někdo znovu hýbal. Díl je již připraven pro práci se strojem.

V neposlední řadě procesu je spuštění stroje. Stroj lze spustit po založení obou dílů do stroje. Stroj může spustit kterýkoliv uživatel, a to tlačítkem na pravé straně stroje. Po zmáčknutí tlačítka se spustí činnost stroje, kdy se v hlavě stroje rozsvítí červené světlo, zapne se zvuk stroje a hlava stroje sjede dolů. Zde chvíli setrvá a znovu se vrací do své původní polohy. Vypne se červené světlo i zvuk. Po vykonání animace stroje se znovu zpřístupní úchopy dílů dveří. (Obr. 6-2)



Obr. 6-2 Spuštění stroje v aplikaci

Na závěr vezme kterýkoliv uživatel díly dveří a odloží jej na odkládací stůl. Prostor pro odložení je rozmístěn po celém stole, proto stačí pustit díl dveří v prostoru stolu a díl dveří se již sám přiřadí na svou správnou pozici na stole a znemožní se mu další uchopení. Tím celý proces končí, a uživatelé mohou diskutovat nad procesem nebo se rozloučit a vypnout aplikaci.

Závěr

Tato diplomová práce představila komplexní přístup k vývoji a implementaci aplikace pro virtuální kolaborativní trénink průmyslových procesů. Cílem práce bylo prozkoumat možnosti virtuální reality (VR) jako nástroje pro zlepšení průmyslového vzdělávání a tréninku. Výsledky výzkumu ukazují, že využití VR v průmyslovém tréninku nabízí významné přínosy včetně zvýšení bezpečnosti a snížení nákladů na školení.

Aplikace využívá VR k posílení pocitu přítomnosti a imerze, což je zásadní pro zlepšení komunikace a spolupráce na dálku. Tato technologie umožňuje realistické zobrazení průmyslových procesů a interakci s objekty, což napomáhá vzdělávání a tréninku pracovníků. Inspirací pro diplomní projekt byly aplikace jako COVE-VR, zaměřená na spolupráci; Smart Factory VR, která se soustředí na zvýšení produktivity a bezpečnosti; a IC.IDO, orientovaná na učení a kreativní procesy. Vyvinutá VR aplikace kombinuje tyto prvky s důrazem na intuitivní interakci a komunikaci, využívající server Photon PUN pro bezproblémovou synchronizaci a kolaboraci.

Během vývoje aplikace byl kladen velký důraz na uživatelskou přívětivost, interaktivitu a realističnost simulace. Testování aplikace s koncovými uživateli prokázalo její efektivitu ve zlepšování pochopení průmyslového procesu a ve zvyšování angažovanosti uživatelů. Aplikace byla schopna účinně simulovat průmyslový scénář, což umožnilo uživatelům bezpečně experimentovat a učit se z chyb bez rizika zranění nebo poškození majetku. Při testování aplikace byly také zjištěny omezení aplikace. Pro kolaborativní trénink musí být brýle připojeny ke kvalitnímu internetu, proto se doporučuje připojovat se na sítě s 5 GHz pásmem (vzhledem k požadavku na vyšší datový tok). I přes kvalitní internet, není synchronizace avatarů kvalitní, a proto by tato problematika mohla být řešena v rámci dalších prací. Připojení, interakce s objekty a komunikace mezi uživateli nevykázala žádné chyby. Pro aplikaci je nutné mít kolem sebe volný prostor o rozložení dvou metrů čtverečních. Pro plynulý chod je vhodné mít výkonnější hardware a tím jsou Meta Quest 3, kde jejich momentální cena je 20 116,- Kč včetně daně[38]. V aplikaci může dojít k cybersickness.[39]

Aplikace je navržena jako univerzální šablona, která může být použita pro různé průmyslové procesy a slouží jako otevřené řešení pro Západočeskou univerzitu v Plzni, Fakultu strojní. Vzhledem k tomu, že je aplikace univerzální, poskytuje robustní základ pro další vývoj a adaptaci pro specifické potřeby různých průmyslových sektorů.

V závěru lze říct, že virtuální kolaborativní trénink je slibnou cestou pro průmyslové vzdělávání a má potenciál revolucionizovat způsob, jakým organizace přistupují k tréninku svých zaměstnanců. Pro budoucí výzkum by bylo vhodné zaměřit se na rozšíření funkcionality aplikace, zahrnutí pokročilých technologií jako je umělá inteligence pro personalizaci učebních plánů a větší integraci s existujícími průmyslovými systémy a procesy.

Bibliografie

[1] GUO, Ziyue, Dong ZHOU, Qidi ZHOU, Xin ZHANG, Jie GENG, Shengkui ZENG, Chuan LV a Aimin HAO. Applications of virtual reality in maintenance during the industrial product lifecycle: A systematic review. *Journal of Manufacturing Systems* [online]. 2020, **56**, 525–538. ISSN 02786125. Dostupné z: doi:10.1016/j.jmsy.2020.07.007

[2] FRANK, Alejandro Germán, Lucas Santos DALENOGARE a Néstor Fabián AYALA. Industry 4.0 technologies: Implementation patterns in manufacturing companies. *International Journal of Production Economics* [online]. 2019, **210**, 15–26. ISSN 09255273. Dostupné z: doi:10.1016/j.ijpe.2019.01.004

[3] SILVESTRI, Luca, Antonio FORCINA, Vito INTRONA, Annalisa SANTOLAMAZZA a Vittorio CESAROTTI. Maintenance transformation through Industry 4.0 technologies: A systematic literature review. *Computers in Industry* [online]. 2020, **123**, 103335. ISSN 01663615. Dostupné z: doi:10.1016/j.compind.2020.103335

[4] LEYER, Michael, Banu AYSOLMAZ, Ross BROWN, Selen TÜRKAY a Hajo A. REIJERS. Process training for industrial organisations using 3D environments: An empirical analysis. *Computers in Industry* [online]. 2021, **124**, 103346. ISSN 01663615. Dostupné z: doi:10.1016/j.compind.2020.103346

[5] WEN, Jing a Masoud GHEISARI. Using virtual reality to facilitate communication in the AEC domain: a systematic review. *Construction Innovation* [online]. 2020, **20**(3), 509–542. ISSN 1471-4175. Dostupné z: doi:10.1108/CI-11-2019-0122

[6] BERG, Leif P. a Judy M. VANCE. An Industry Case Study: Investigating Early Design Decision Making in Virtual Reality. *Journal of Computing and Information Science in Engineering* [online]. 2017, **17**(1). ISSN 1530-9827. Dostupné z: doi:10.1115/1.4034267

[7] NARASIMHA, Shraddhaa, Emma DIXON, Jeffrey W. BERTRAND a Kapil CHALIL MADATHIL. An empirical study to investigate the efficacy of collaborative immersive virtual reality systems for designing information architecture of software systems. *Applied Ergonomics* [online]. 2019, **80**, 175–186. ISSN 00036870. Dostupné z: doi:10.1016/j.apergo.2019.05.009

[8] SCHINA, Laura, Mariangela LAZOI, Roberto LOMBARDO a Angelo CORALLO. Virtual Reality for Product Development in Manufacturing Industries. In: [online]. 2016, s. 198–207. Dostupné z: doi:10.1007/978-3-319-40621-3_15

[9] WOLFARTSBERGER, Josef, Jan ZENISEK a Norbert WILD. Supporting Teamwork in Industrial Virtual Reality Applications. *Procedia Manufacturing* [online]. 2020, **42**, 2–7. ISSN 23519789. Dostupné z: doi:10.1016/j.promfg.2020.02.016

[10] BUROVA, Alisa, John MÄKELÄ, Jaakko HAKULINEN, Tuuli KESKINEN, Hanna HEINONEN, Sanni SILTANEN a Markku TURUNEN. Utilizing VR and Gaze Tracking to Develop AR Solutions for Industrial Maintenance. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* [online]. New York, NY, USA: ACM, 2020, s. 1–13. ISBN 9781450367080. Dostupné z: doi:10.1145/3313831.3376405

[11] SCHWARZ, Stephanie, Georg REGAL, Marina KEMPF a Raimund SCHATZ. Learning Success in Immersive Virtual Reality Training Environments: Practical Evidence from Automotive Assembly. In: *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society* [online]. New York, NY, USA: ACM, 2020, s. 1–11. ISBN 9781450375795. Dostupné z: doi:10.1145/3419249.3420182

[12] WOLFARTSBERGER, Josef, Jan ZENISEK, Christoph SIEVI a Mathias SILMBROTH. A virtual reality supported 3D environment for engineering design review. In: *2017 23rd International Conference on Virtual System & Multimedia (VSMM)* [online]. B.m.: IEEE, 2017, s. 1–8. ISBN 978-1-5386-4494-2. Dostupné z: doi:10.1109/VSMM.2017.8346288

[13] SILTANEN, Sanni a Hanna HEINONEN. Scalable and responsive information for industrial maintenance work. In: *Proceedings of the 23rd International Conference on Academic Mindtrek* [online]. New York, NY, USA: ACM, 2020, s. 100–109. ISBN 9781450377744. Dostupné z: doi:10.1145/3377290.3377296

[14] HE, Zhenyi, Ruofei DU a Ken PERLIN. CollaboVR: A Reconfigurable Framework for Creative Collaboration in Virtual Reality. In: *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)* [online]. B.m.: IEEE, 2020, s. 542–554. ISBN 978-1-7281-8508-8. Dostupné z: doi:10.1109/ISMAR50242.2020.00082

[15] PIDEL, Catlin a Philipp ACKERMANN. Collaboration in Virtual and Augmented Reality: A Systematic Overview. In: [online]. 2020, s. 141–156. Dostupné z: doi:10.1007/978-3-030-58465-8_10

[16] MÜTTERLEIN, J., S. JELSCH a T HESS. Specifics of collaboration in virtual reality: how immersion drives the intention to collaborate. *PACIS*. 2018.

[17] LEE, Yongjae a Byounghyun YOO. XR collaboration beyond virtual reality: work in the real world. *Journal of Computational Design and Engineering* [online]. 2021, **8**(2), 756–772. ISSN 2288-5048. Dostupné z: doi:10.1093/jcde/qwab012

[18] WOHLGENANNT, Isabell, Alexander SIMONS a Stefan STIEGLITZ. Virtual Reality. *Business & Information Systems Engineering* [online]. 2020, **62**(5), 455–461. ISSN 2363-7005. Dostupné z: doi:10.1007/s12599-020-00658-9

[19] AIRBUS. *Virtual reality with real benefits* [online]. 27. září 2017 [vid. 2023-11-29]. Dostupné z: https://www.airbus.com/en/newsroom/news/2017-09-virtual-reality-with-real-benefits

[20] SIEMENS a HUBERTUS BREUER. *Working with virtual hands* [online]. březen 2021 [vid. 2023-11-29]. Dostupné z: https://www.siemens.com/global/en/company/stories/research-technologies/digitaltwin/virtual-hands.html

[21] FORD. *Bez tužky a papíru* [online]. 15. listopad 2019 [vid. 2023-11-29]. Dostupné z: https://www.fordmedia.cz/documents/bez-tuzky-a-papiru-diky-360-degrees-skicovani-mohou-designeri-navrhovat-interier-z-pohledu-ridice-368252

[22] BROOKES, Jack, Matthew WARBURTON, Mshari ALGHADIER, Mark MON-WILLIAMS a Faisal MUSHTAQ. Studying human behavior with virtual reality: The Unity Experiment Framework. *Behavior Research Methods* [online]. 2020, **52**(2). ISSN 15543528. Dostupné z: doi:10.3758/s13428-019-01242-0

[23] PARK, Sang Min a Young Gab KIM. A Metaverse: Taxonomy, Components, Applications, and Open Challenges. *IEEE Access* [online]. 2022, **10**. ISSN 21693536. Dostupné z: doi:10.1109/ACCESS.2021.3140175

[24] FILLATREAU, P., J.-Y. FOURQUET, R. LE BOLLOC'H, S. CAILHOL, A. DATAS a B. PUEL. Using virtual reality and 3D industrial numerical models for immersive interactive checklists. *Computers in Industry* [online]. 2013, **64**(9), 1253–1262. ISSN 01663615. Dostupné z: doi:10.1016/j.compind.2013.03.018

[25] AR + VR in der Produktion: Kostenloses Online-Seminar zu Virtual- und Augmented-Reality. *https://niedersachsen.digital/ar-und-vr-in-der-produktion/* [online]. 22. září 2020 [vid. 2023-11-30]. Dostupné z: https://niedersachsen.digital/ar-und-vr-in-der-produktion/

[26] BUROVA, Alisa, John MÄKELÄ, Hanna HEINONEN, Paulina Becerril PALMA, Jaakko HAKULINEN, Viveka OPAS, Sanni SILTANEN, Roope RAISAMO a Markku TURUNEN. Asynchronous industrial collaboration: How virtual reality and virtual tools aid the process of maintenance method development and documentation creation. *Computers in Industry* [online]. 2022, **140**. ISSN 01663615. Dostupné z: doi:10.1016/j.compind.2022.103663

[27] ALPALA, Luis Omar, Darío J. QUIROGA-PARRA, Juan Carlos TORRES a Diego H. PELUFFO-ORDÓÑEZ. Smart Factory Using Virtual Reality and Online Multi-User: Towards a Metaverse for Experimental Frameworks. *Applied Sciences* [online]. 2022, **12**(12), 6258. ISSN 2076-3417. Dostupné z: doi:10.3390/app12126258

[28] ESI GROUP. *IC.IDO Virtual Reality Engineering Software* [online]. 2023 [vid. 2023-11-30]. Dostupné z: https://www.esi-group.com/products/ic-ido

[29] ESI GROUP. *IC.IDO 16.0 - Hardware Requirements* [online]. 2022 [vid. 2023-11-30]. Dostupné z: https://myesi.esi-group.com/downloads/software-documentation/ic.ido-16.0-hardware-requirements-zip-0?destination=node/11056%3Fcid%3D11056

[30] KORTUS, K. Možnosti virtuální kolaborace. *ZČU* [online]. 27. květen 2022 [vid. 2023-11-30]. Dostupné z: https://dspace5.zcu.cz/bitstream/11025/49477/1/BP_Virtualni_Kolaborace_KKortus.pdf

[31] ALZA. Alza - Oculus Quest 2. *https://www.alza.cz/search.htm?exps=oculus%20quest%202* [online]. 2023 [vid. 2023-11-30]. Dostupné z: https://www.alza.cz/search.htm?exps=oculus%20quest%202

[32] OKITA, A. *Learning C# Programming with Unity 3D*. second. B.m.: Routledge, 2019. ISBN ISBN-13: 978-1138336810.

[33] SUNG, K. a G. SMITH. *Basic Math for Game Development with Unity 3D: A Beginner's Guide to Mathematical Foundations*. B.m.: Apress, 2019. ISBN ISBN 978-1484254424.

[34] LINOWES, J. *Unity 2020 Virtual Reality Projects: Learn VR development by building immersive applications and games with Unity 2019.4 and later versions*. 3rd vyd. B.m.: Packt Publishing, 2020. ISBN ISBN 978-1839217333.

[35] LAVALLE, S. M. Virtual Reality, Cambridge University Press 2023. *http://lavalle.pl/vr/* [online]. 2023 [vid. 2023-11-30]. Dostupné z: http://lavalle.pl/vr/

[36] UNITY. Oficiální Unity3D návody. *https://learn.unity.com/* [online]. 2023 [vid. 2023-11-30]. Dostupné z: https://learn.unity.com/

[37] PHOTON. Photon Pun Server. *https://www.photonengine.com/fusion/pricing* [online]. 2023 [vid. 2023-11-30]. Dostupné z: https://www.photonengine.com/fusion/pricing

[38] ALZA. *Alza - Meta Quest 3* [online]. 2024 [vid. 2024-05-04]. Dostupné z: https://www.alza.cz/gaming/meta-quest-3-512-gb-d7962774.htm?gclid=Cj0KCQjwq86wBhDiARIsAJhuphmy6KE7ZaoYSCTlKlRqzX4Q0chXueT8lqKCOVmNNFX8MtG3dszqwT0aAo2oEALw_wcB&kampan=adwacc_prislusenstvi-pro-it-tv_pla_all_ad-top_alza-dny_c_9062897___OC002b2z_678211603437_~155324879016~

[39] VAN DER MEER, Nesse, Vivian VAN DER WERF, Willem-Paul BRINKMAN a Marcus SPECHT. Virtual reality and collaborative learning: a systematic literature review. *Frontiers in Virtual Reality* [online]. 2023, **4**. ISSN 2673-4192. Dostupné z: doi:10.3389/frvir.2023.1159905

Přílohy

Tato kapitola označuje seznam příloh daného objektu.

Příloha 1. – Kód skriptu BasicRecenter_ovr.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace Com.KKortus.VRM_dp
{
    public class BasicRecenter_ovr : MonoBehaviour
    {
        #region Proměnné
        // Kamera pro transformaci místa
        private Transform m_CameraRig;

        [Header("Teleport")]
        // Přiřadit rodičovský objekt [BuildingBlock] Camera Rig
        public OVRManager m_OVRManager;

        // Čas do teleportace
        public float resetPositionTime = 0.1f;

        // Po spuštění
        public bool playOnAwake;

        // Výška kamery
        public float height;

        [Header("Fade screen")]
        // Objekt k rendrování
        public Renderer rend;

        // Čas trvání
        public float fadeDuration = 2f;

        // Barva přechodu
        public Color fadeColor;
        #endregion

        #region MonoBehaviour Callbacks
        void Start()
        {
            m_CameraRig = m_OVRManager.transform;

            if (playOnAwake == true)
            {
                Invoke("ResetVRPosition", resetPositionTime);
            }
        }
        #endregion

        #region Teleport
        //Veřejná metoda, pro volání teleportu se zatmavením obrazu
        public void ResetVRPosition()
        {
            ResetVRPosition_meth();
            if (rend != null)
```

```
                FadeOut();
        }

        //Nastavení tvrdé pozice a rotace pouze při spuštění aplikace; při reloadu
scény zůstane tak jak se postavil
        private void ResetVRPosition_meth()
        {
            m_OVRManager.trackingOriginType = OVRManager.TrackingOrigin.EyeLevel;
            m_CameraRig.transform.parent = this.gameObject.transform;
            m_CameraRig.transform.localPosition = new Vector3(0f, height, 0f);
            m_CameraRig.transform.localEulerAngles = Vector3.zero;
        }
        #endregion

        #region FadeScreen
        public void FadeOut()
        {
            Fade(1, 0);
        }

        public void FadeIn()
        {
            rend.gameObject.GetComponent<MeshRenderer>().enabled = true;
            Fade(0, 1);
        }

        public void Fade(float alphaIn, float alphaOut)
        {
            StartCoroutine(FadeRoutine(alphaIn, alphaOut));
        }

        public IEnumerator FadeRoutine(float alphaIn, float alphaOut)
        {
            float timer = 0;
            while (timer <= fadeDuration)
            {
                Color newColor = fadeColor;
                newColor.a = Mathf.Lerp(alphaIn, alphaOut, timer / fadeDuration);

                rend.material.SetColor("_Color", newColor);

                timer += Time.deltaTime;
                yield return null;
            }

            if (alphaIn > 0)
            {
                rend.gameObject.GetComponent<MeshRenderer>().enabled = false;
            }
            Color newColor2 = fadeColor;
            newColor2.a = alphaOut;

            rend.material.SetColor("_Color", newColor2);
        }
        #endregion
    }
}
```

Příloha 2. – Kód skriptu Launcher_ovr.cs

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;

using Photon.Pun;
using Photon.Realtime;
using UnityEngine.UI;
using System;
using Oculus.Platform;
using Unity.VisualScripting;

namespace Com.KKortus.VRM_dp
{
    public class Launcher_ovr : MonoBehaviourPunCallbacks
    {
        #region Private Serializable Fields
        [Header("Nastavení pro tvorbu místnosti")]
        [Tooltip("Maximální počet hráčů na jednu místnost. Pokud je místnost plná, nemohou se připojit nový hráči, a proto bude vytvořena nová místnost.")]
        [SerializeField]
        private byte maxPlayersPerRoom;

        [Header("UI zobrazení při přihlášení")]
        [Tooltip("UI Label informující uživatele o probíhajícím připojení")]
        [SerializeField]
        private GameObject progressLabel;
        [Tooltip("UI Panel, který umožňuje uživateli připojit se a hrát")]
        [SerializeField]
        private GameObject controlPanel;
        [Tooltip("UI Panel, který informuje uživate, že je místnost plná")]
        [SerializeField]
        private GameObject fullRoom;
        #endregion

        #region Private Fields
        /// <summary>
        /// Toto je číslo verze tohoto klienta. Uživatelé jsou od sebe odděleni pomocí gameVersion (což umožňuje provádět změny).
        /// </summary>
        string gameVersion = "1";

        /// <summary>
        /// Sledujte aktuální proces. Protože připojení je asynchronní a je založeno na několika zpětných volání z Photon,
        /// musíme sledovat, abychom správně upravili chování, když obdržíme zpětné volání od Photonu.
        /// Typicky se používá pro zpětné volání OnConnectedToMaster().
        /// </summary>
        bool isConnecting;

        string targetRoom;
        #endregion

        #region MonoBehaviourPunCallbacks CallBacks

        public override void OnConnectedToMaster()
        {
            Debug.Log("PUN Basics Tutorial/Launcher: OnConnectedToMaster() byl zavolán pomocí PUN");

            ///<summary>
            ///Nechceme nic dělat, pokud se nepokoušíme připojit k místnosti.
```

```
            /// Tento případ, kdy je isConnecting false, je typicky při ztrátě nebo
ukončení hry kdy se při načítání této úrovně zavolá OnConnectedToMaster, v takovém
případě se volá
            /// nechceme nic dělat.
            ///</summary>
            if (isConnecting)
            {
                // #Critical: Nejdříve se snažíme připojit k potenciální existující
místnosti. Pokud ano, je to dobře, jinak se nám vrití volání OnJoinRandomFaild()
                PhotonNetwork.JoinRoom(targetRoom);
                isConnecting = false;
            }
        }

        public override void OnDisconnected(DisconnectCause cause)
        {
            progressLabel.SetActive(false);
            controlPanel.SetActive(true);

            Debug.LogWarningFormat("PUN Basics Tutorial/Launcher: OnDisconnected()
byl zavolán pomocí PUN s řešením {0}", cause);
        }

        public override void OnJoinRoomFailed(short returnCode, string message)
        {
            if(returnCode == ErrorCode.GameFull)
            {
                Debug.LogError("Místnost je plná.");
                StartCoroutine(GameFull_cor());
            }
            else
            {
                Debug.LogError("Vytvářím místnost s maximálním počtem: " +
this.maxPlayersPerRoom);

                RoomOptions roomOptions = new RoomOptions { MaxPlayers =
maxPlayersPerRoom };

                // #Critical: Nepodařilo se nám připojit do konkrétní místnosti,
nejspíš žádná nexistuje. Vytvoříme novou místnost.
                PhotonNetwork.CreateRoom(targetRoom, roomOptions);
            }

        }

        public override void OnJoinedRoom()
        {
            Debug.Log("PUN Basics Tutorial/Launcher: OnJoinedRoom() zavolán pomocí
PUN. Nyní je tento klient v místnosti.");

            // #Critical: Načítáme pouze v případě, že jsme první hráč, jinak se
spoléháme na 'PhotonNetwork.AutomaticallySyncScene' pro synchronizaci naší
instance scény.
            if (PhotonNetwork.CurrentRoom.PlayerCount == 1)
            {
                Debug.LogError("Načetli jsme: " + targetRoom);

                // #Critical: Načtení úrovně místnosti
                PhotonNetwork.LoadLevel("VRMultiplayer_room2");
            }
        }
        #endregion
```

```
        #region MonoBehaviour CallBacks
        void Awake()
        {
            //    #Critical:    tím    zajistíme,    že    můžeme    použít    funkci
PhotonNetwork.LoadLevel()  na  hlavním  klientovi  a  všichni  klienti  ve  stejné
místnosti automaticky synchronizují svou úroveň.
            PhotonNetwork.AutomaticallySyncScene = true;
        }

        void Start()
        {
            progressLabel.SetActive(false);
            controlPanel.SetActive(true);
        }
        #endregion

        #region Public Methods
        /// <summary>
        /// Spuštění procesu připojení.
        /// - Pokud jsme již připojeni, pokusíme se připojit k vybrané místnosti.
        /// - Pokud ještě nejsme připojení, připojíme tuto instanci aplikace k síti
Photon Cloud Network.
        /// </summary>
        public void Connect(string room)
        {
            progressLabel.SetActive(true);
            controlPanel.SetActive(false);

            targetRoom = room;

            if (PhotonNetwork.IsConnected)
            {
                PhotonNetwork.JoinRoom(targetRoom);
            }
            else
            {
                // sledování snahy připojit se k místnosti, protože když se vrátíme
ze hry, dostaneme zpětné volání, že jsme připojeni, takže potřebujeme vědět, co
máme dělat potom
                isConnecting = PhotonNetwork.ConnectUsingSettings();
                PhotonNetwork.GameVersion = this.gameVersion;
            }
        }

        public void SetNumberOfPlayers(int count)
        {
            maxPlayersPerRoom = (byte)count;
        }
        #endregion

        #region Coroutines
        IEnumerator GameFull_cor()
        {
            progressLabel.SetActive(false);
            fullRoom.SetActive(true);
            yield return new WaitForSeconds(3f);
            fullRoom.SetActive(false);
            controlPanel.SetActive(true);
        }
        #endregion
    }
}
```

Příloha 3. – Kód skriptu NetworkAvatar_ovr.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Oculus.Avatar2;
using System;

using Oculus.Platform;

namespace Com.KKortus.VRM_dp
{
    public class NetworkAvatar_ovr : OvrAvatarEntity
    {
        [SerializeField] int m_avatarToUseInZipFolder = 2;
        PhotonView m_photonView;
        List<byte[]> m_streamedDataList = new List<byte[]>();
        int m_maxBytesToLog = 15;
        [SerializeField] ulong m_instantiationData;
        float m_cycleStartTime = 0;
        float m_intervalToSendData = 0.08f;

        [System.Serializable]
        private struct AssetData
        {
            public AssetSource source;
            public string path;
        }

        [Header("Assets")]
        [Tooltip("Asset paths to load, and whether each asset comes from a preloaded zip file or directly from StreamingAssets. See Preset Asset settings on OvrAvatarManager for how this maps to the real file name.")]
        [SerializeField]
        private List<AssetData> _assets = new List<AssetData> { new AssetData { source = AssetSource.Zip, path = "0" } };
        public GameObject ovrCamera;

        [Tooltip("Adds an underscore between the path and the postfix.")]
        [SerializeField]
        private bool _underscorePostfix = true;
        protected bool HasLocalAvatarConfigured => _assets.Count > 0;
        [Tooltip("Filename Postfix (WARNING: Typically the postfix is Platform specific, such as \"_rift.glb\")")]
        [SerializeField]
        private string _overridePostfix = String.Empty;

        protected override void Awake()
        {
            ConfigureAvatarEntity();
            base.Awake();
        }

        private void Start()
        {
            m_instantiationData = GetUserIdFromPhotonInstantiationData();
            _userId = m_instantiationData;

            int playerCount = PhotonNetwork.PlayerList.Length;
            if (_assets.Count > 0)
            {
```

```
                AssetData assetData = _assets[0];
                assetData.path = playerCount.ToString();
                _assets[0] = assetData;
            }

            StartCoroutine(TryToLoadUser());
        }

        void ConfigureAvatarEntity()
        {
            m_photonView = GetComponent<PhotonView>();
            if (m_photonView.IsMine)
            {
                SetIsLocal(true);
                _creationInfo.features                                          =
Oculus.Avatar2.CAPI.ovrAvatar2EntityFeatures.Preset_Default;
                SampleInputManager              sampleInputManager              =
OvrAvatarManager.Instance.gameObject.GetComponent<SampleInputManager>();
                SetBodyTracking(sampleInputManager);
                gameObject.name = "MyAvatar";
                StartCoroutine(Loadhands_cor("Player"));
            }
            else
            {
                SetIsLocal(false);
                _creationInfo.features                                          =
Oculus.Avatar2.CAPI.ovrAvatar2EntityFeatures.Preset_Remote;
                SampleInputManager              sampleInputManager              =
OvrAvatarManager.Instance.gameObject.GetComponent<SampleInputManager>();
                SetBodyTracking(sampleInputManager);
                gameObject.name = "OtherAvatar";
            }
        }

        IEnumerator TryToLoadUser()
        {
            var                             hasAvatarRequest                    =
OvrAvatarManager.Instance.UserHasAvatarAsync(_userId);
            while (hasAvatarRequest.IsCompleted == false)
            {
                yield return null;
            }
            LoadLocalAvatar();
        }

        private void LoadLocalAvatar()
        {
            if (!HasLocalAvatarConfigured)
            {
                OvrAvatarLog.LogInfo("No local avatar asset configured");
                return;
            }

            // Zip asset paths are relative to the inside of the zip.
            // Zips can be loaded from the OvrAvatarManager at startup or by calling
OvrAvatarManager.Instance.AddZipSource
            // Assets can also be loaded individually from Streaming assets
            foreach (var asset in _assets)
            {
                bool isFromZip = (asset.source == AssetSource.Zip);

                string assetPostfix = GetAssetPostfix(isFromZip);
```

```
                var assetPath = $"{asset.path}{assetPostfix}";
                LoadAssets(new[] { assetPath }, asset.source);
            }
        }
        private string GetAssetPostfix(bool isFromZip)
        {
            string assetPostfix = (_underscorePostfix ? "_" : "")
                                  +
OvrAvatarManager.Instance.GetPlatformGLBPostfix(_creationInfo.renderFilters.quali
ty, isFromZip)
                                  +
OvrAvatarManager.Instance.GetPlatformGLBVersion(_creationInfo.renderFilters.quali
ty, isFromZip)
                                  +
OvrAvatarManager.Instance.GetPlatformGLBExtension(isFromZip);
            if (!String.IsNullOrEmpty(_overridePostfix))
            {
                assetPostfix = _overridePostfix;
            }

            return assetPostfix;
        }

        private void LateUpdate()
        {
            float elapsedTime = Time.time - m_cycleStartTime;
            if (elapsedTime > m_intervalToSendData)
            {
                RecordAndSendStreamDataIfMine();
                m_cycleStartTime = Time.time;
            }

        }

        void RecordAndSendStreamDataIfMine()
        {
            if (m_photonView.IsMine)
            {
                byte[] bytes = RecordStreamData(activeStreamLod);
                m_photonView.RPC("RecieveStreamData", RpcTarget.Others, bytes);
            }
        }

        [PunRPC]
        public void RecieveStreamData(byte[] bytes)
        {
            m_streamedDataList.Add(bytes);
        }

        void LogFirstFewBytesOf(byte[] bytes)
        {
            for (int i = 0; i < m_maxBytesToLog; i++)
            {
                string  bytesString  =  Convert.ToString(bytes[i],  2).PadLeft(8,
'0');
            }
        }

        private void Update()
        {
            if (m_streamedDataList.Count > 0)
            {
                if (IsLocal == false)
```

```
                {
                    byte[] firstBytesInList = m_streamedDataList[0];
                    if (firstBytesInList != null)
                    {
                        ApplyStreamData(firstBytesInList);
                    }
                    m_streamedDataList.RemoveAt(0);
                }
            }
        }

        ulong GetUserIdFromPhotonInstantiationData()
        {
            PhotonView photonView = GetComponent<PhotonView>();
            object[] instantiationData = photonView.InstantiationData;
            Int64 data_as_int = (Int64)instantiationData[0];
            return Convert.ToUInt64(data_as_int);
        }

        IEnumerator Loadhands_cor(string _tag)
        {
            yield return new WaitForSeconds(1f);
            GameObject leftHand = this.transform.GetChild(2).gameObject;
            leftHand.tag = _tag;
            leftHand.AddComponent<BoxCollider>().size = new Vector3(0.1f, 0.1f,
0.1f);
            leftHand.GetComponent<BoxCollider>().center = new Vector3(0.1f, 0f,
0f);
            GameObject rightHand = this.transform.GetChild(3).gameObject;
            rightHand.tag = _tag;
            rightHand.AddComponent<BoxCollider>().size = new Vector3(0.1f, 0.1f,
0.1f);
            rightHand.GetComponent<BoxCollider>().center = new Vector3(-0.1f, 0f,
0f);
        }
    }
}
```

Příloha 4. – Kód skriptu GameManager_ovr.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using UnityEngine.SceneManagement;

using Photon.Pun;
using Photon.Realtime;
using UnityEngine.UI;
using System;
using Oculus.Platform;

namespace Com.KKortus.VRM_dp
{
    public class GameManager_ovr : MonoBehaviourPunCallbacks
    {
        #region Public Fields
        public static GameManager_ovr Instance;

        public BasicRecenter_ovr[] spawnPoints;
        #endregion
```

```
        #region Private Fields

        [Header("Inicializace hráče")]
        [SerializeField] ulong m_userId;
        #endregion

        #region MonoBehaviour CallBacks
        private void Awake()
        {
            if(Instance == null)
            {
                Instance = this;
            }
            else
            {
                Destroy(this.gameObject);
            }

        }
        private void Start()
        {
            StartCoroutine(SetUserIdFromLoggedInUser());
            StartCoroutine(InstantiateNetworkedAvatarOnceInRoom());

            if (!PhotonNetwork.IsConnected)
            {
                SceneManager.LoadScene("VRMultiplayer_Launcher");

                return;
            }

            spawnPoints[PhotonNetwork.CurrentRoom.PlayerCount                    -
1].ResetVRPosition();

            Debug.LogError("vypisuji          spawnpoint:          "          +
spawnPoints[PhotonNetwork.CurrentRoom.PlayerCount - 1]);
        }

        #endregion

        #region Photon Callbacks
        /// <summary>
        /// Zavolá se pokud lokální hráč opustí místnost. Musíme načíst scénu
Launcher.
        /// </summary>
        public override void OnLeftRoom()
        {
            SceneManager.LoadScene(0);
        }

        #endregion

        #region Public Methods
        public void InstantiateNetworkedAvatar()
        {
            Int64 userId = Convert.ToInt64(m_userId);
            object[] objects = new object[1] { userId };
            PhotonNetwork.Instantiate("NetworkPlayer",
spawnPoints[PhotonNetwork.CurrentRoom.PlayerCount-1].transform.position,
Quaternion.identity, 0, objects);
        }
        #endregion
```

```
        #region Coroutines
        IEnumerator SetUserIdFromLoggedInUser()
        {
            if (OvrPlatformInit.status == OvrPlatformInitStatus.NotStarted)
            {
                OvrPlatformInit.InitializeOvrPlatform();
            }

            while (OvrPlatformInit.status != OvrPlatformInitStatus.Succeeded)
            {
                if (OvrPlatformInit.status == OvrPlatformInitStatus.Failed)
                {
                    Debug.LogError("OVR Platform failed to initialise");
                    yield break;
                }
                yield return null;
            }

            Users.GetLoggedInUser().OnComplete(message =>
            {
                if (message.IsError)
                {
                    Debug.LogError("Getting Logged in user error " +
message.GetError());
                }
                else
                {
                    m_userId = message.Data.ID;
                }
            });
        }
        IEnumerator InstantiateNetworkedAvatarOnceInRoom()
        {
            while (PhotonNetwork.InRoom == false)
            {
                Debug.Log("Waiting to be in room before intantiating avatar");
                yield return null;
            }

            InstantiateNetworkedAvatar();
        }
        #endregion
    }
}
```

Příloha 5. – Kód skriptu ProgressManager_ovr.cs

```
using Photon.Pun;
using Photon.Realtime;
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.Events;

namespace Com.KKortus.VRM_dp
{
    [Serializable]
```

```
    public class ProgressList
    {
        public string _name;
        public UnityEvent _nextAction;
        public float _delayTime;
        public UnityEvent _delayAction;
        public int _count;
        [HideInInspector]
        public int _progressCount;
        public bool _done;
    }
    public class ProgressManager_ovr : MonoBehaviourPunCallbacks
    {
        #region Public variables
        public List<ProgressList> _progressList = new List<ProgressList>();

        [Tooltip("Aktuální progres aplikace")]
        public int _progress = 0;
        public int _totalProgress;

        public TextMeshPro _textVypis;
        #endregion

        #region Public Methods
        public void NextStep(int num)
        {
            photonView.RPC(nameof(NextStep_rpc), RpcTarget.All, num);
        }
        #endregion

        #region RPC Methods
        [PunRPC]
        public void NextStep_rpc(int num)
        {
            if(_totalProgress > _progress)
            {
                _progressList[num]._progressCount++;

                if (!_progressList[num]._done && _progressList[num]._progressCount
== _progressList[num]._count)
                {
                    _progressList[num]._nextAction?.Invoke();
                    StartCoroutine(DelayAction_cor(_progressList[num]._delayTime,
_progressList[num]._delayAction));

                    _progressList[num]._done = true;
                    _progress++;

                    _textVypis.text = "Postup: " + _progress + "/" +
_progressList.Count;
                }
            }
        }
        #endregion

        #region Coroutines
        IEnumerator DelayAction_cor(float time, UnityEvent action)
        {
            yield return new WaitForSeconds(time);
            action?.Invoke();
        }
        #endregion
    }
```

```
}
```

Příloha 6. – Kód skriptu ActiveOnStart_ovr.cs

```
using Photon.Pun;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

namespace Com.KKortus.VRM_dp
{
    public class ActiveOnStart_ovr : MonoBehaviourPunCallbacks
    {
        public UnityEvent afterAllInRoom;

        void Update()
        {
            if              (PhotonNetwork.CurrentRoom.PlayerCount              ==
PhotonNetwork.CurrentRoom.MaxPlayers)
            {
                afterAllInRoom.Invoke();
                Destroy(this);
            }
        }
    }
}
```

Příloha 7. – Kód skriptu GrabOnOff_ovr.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using TMPro;
using Oculus.Interaction.HandGrab;

namespace Com.KKortus.VRM_dp
{
    public class GrabOnOff_ovr : MonoBehaviourPunCallbacks
    {
        #region Public variables
        public bool _activeGrab = true;

        [HideInInspector]
        public bool _inHand;
        #endregion

        #region Private variables
        private HandGrabInteractable interactable;

        PhotonView _photonView;
        #endregion

        #region MonoBehaviour Callbacks
        private void Start()
        {
            interactable = GetComponent<HandGrabInteractable>();
            _photonView = GetComponent<PhotonView>();
```

```
        }

        private void Update()
        {
            if              (PhotonNetwork.CurrentRoom.PlayerCount              ==
PhotonNetwork.CurrentRoom.MaxPlayers && _activeGrab)
            {
                interactable.enabled = _photonView.IsMine;
            }
        }
        #endregion

        #region Public Methods
        public void OnGrab()
        {
            // Volá funkci RPC na serveru pro aktualizaci uchopení.
            photonView.RPC("UpdateGrabCount", RpcTarget.All, true);
        }

        public void OffGrab()
        {
            // Volá funkci RPC na serveru pro aktualizaci uchopení.
            photonView.RPC("UpdateGrabCount", RpcTarget.All, false);
        }
        public void GrabOn_meth()
        {
            _activeGrab = true;
        }
        public void GrabOff_meth()
        {
            _activeGrab = false;
            interactable.enabled = false;
        }
        #endregion

        #region RPC Methods
        [PunRPC]
        public void UpdateGrabCount(bool take)
        {
            _inHand = take;
        }
        #endregion
    }
}
```

Příloha 8. – Kód skriptu ChangeOwner_ovr.cs

```
using Photon.Pun;
using Photon.Realtime;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace Com.KKortus.VRM_dp
{
    public class ChangeOwner_ovr : MonoBehaviourPunCallbacks
    {
        #region Private variables
        PhotonView _photonView;
        private GrabOnOff_ovr _grabbers;
        #endregion
```

```
        #region Monobehaviour Callbacks
        private void Start()
        {
            _photonView = GetComponent<PhotonView>();
            _grabbers = GetComponent<GrabOnOff_ovr>();
        }

        public void OnTriggerEnter(Collider other)
        {
            if (other.CompareTag("Player") && this.enabled)
            {
                ChangeOnMe();
            }
        }
        #endregion

        #region Public methods
        public void ChangeOnMe()
        {
            if (_photonView != null && !_photonView.IsMine && !_grabbers._inHand)
            {
                _photonView.TransferOwnership(PhotonNetwork.LocalPlayer);
            }
        }
        #endregion
    }
}
```

Příloha 9. – Kód skriptu ProgressObject_ovr.cs

```
using Com.KKortus.VRM_dp;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ProgressObject_ovr : MonoBehaviour
{
    #region Public variables
    public int progressCount;
    #endregion

    #region Private variables
    private ProgressManager_ovr _progressManager;

    private bool oneShot;
    #endregion

    #region MonoBehaviour Callbacks
    private void Start()
    {
        _progressManager = FindAnyObjectByType<ProgressManager_ovr>();
    }
    #endregion

    #region Public Methods
    public void PlayProgress ()
    {
        if(!oneShot)
        {
            oneShot = true;
```

```
            _progressManager.NextStep(progressCount);
        }
    }
    #endregion
}
```

Příloha 10. – Kód skriptu SnappObjectWithName_ovr.cs

```
using Com.KKortus.VRM_dp;
using Oculus.Interaction.HandGrab;
using Photon.Pun;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

namespace Com.KKortus.VRM_dp
{
    public class SnappObjectWithName_ovr : MonoBehaviour
    {
        #region Public variables
        public string _nameObject;
        public UnityEvent _actionEvent;
        #endregion

        #region Private variables
        bool _activeTF = true;
        #endregion

        #region MonoBehaviour Callbacks
        private void OnTriggerStay(Collider other)
        {
            if (_activeTF && other.name == _nameObject &&
!other.GetComponent<GrabOnOff_ovr>()._inHand)
            {
                _activeTF = false;

                other.GetComponent<GrabOnOff_ovr>().GrabOff_meth();

                other.transform.parent = this.transform;
                other.transform.localEulerAngles = Vector3.zero;
                other.transform.localPosition = Vector3.zero;

                if (other.GetComponent<PhotonView>().IsMine)
                {
                    _actionEvent.Invoke();
                }
            }
        }
        #endregion
    }
}
```

Příloha 11. – Kód skriptu TriggerAction_ovr.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
```

```
namespace Com.KKortus.VRM_dp
{
    public class TriggerAction_ovr : MonoBehaviour
    {
        #region Public variables
        public string _tag;
        public UnityEvent _action;
        public bool _oneShot;
        #endregion

        #region MonoBehaviout Callbacks
        private void OnTriggerEnter(Collider other)
        {
            if (other.CompareTag(_tag) && _oneShot)
            {
                _oneShot = false;
                _action.Invoke();
            }
        }
        #endregion
    }
}
```

Příloha 12. – Kód skriptu SnappObjectsWithNamePlus_ovr.cs

```
using Oculus.Interaction.HandGrab;
using Photon.Pun;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

namespace Com.KKortus.VRM_dp
{
    [Serializable]
    public class SnappObjects
    {
        public string _nameObject;
        public Transform _parObj;
        public bool _activeTF;
        public UnityEvent _actionEvent;
    }

    public class SnappObjectsWithNamePlus_ovr : MonoBehaviour
    {
        #region Public variables
        public List<SnappObjects> _snappObjects = new List<SnappObjects>();
        #endregion

        #region MonoBehaviour Callbacks
        private void OnTriggerStay(Collider other)
        {
            for(int i = 0; i < _snappObjects.Count; i++)
            {
                if (_snappObjects[i]._activeTF && other.name ==
_snappObjects[i]._nameObject)
                {
                    if (!other.GetComponent<GrabOnOff_ovr>()._inHand)
                    {
```

```
                        _snappObjects[i]._activeTF = false;

                        other.GetComponent<GrabOnOff_ovr>().GrabOff_meth();

                        other.transform.parent = _snappObjects[i]._parObj;
                        other.transform.localEulerAngles = Vector3.zero;
                        other.transform.localPosition = Vector3.zero;

                        if(other.GetComponent<PhotonView>().IsMine)
                        {
                            _snappObjects[i]._actionEvent.Invoke();
                        }
                    }
                }
            }
        }
        #endregion
    }
}
```