ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA STROJNÍ

Studijní program: N0715A270012 Průmyslové inženýrství a management
Studijní specializace: N0715A270012S00 Průmyslové inženýrství a management

DIPLOMOVÁ PRÁCE

Tvorba modulárního interaktivního digitálního dvojčete výrobního úseku pro virtuální realitu

Autor: Bc. Matěj DVOŘÁK
Vedoucí práce: doc. Ing. Petr Hořejší, Ph.D.

Akademický rok 2021/2022

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta strojní
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Matěj DVOŘÁK**
Osobní číslo:	**S21N0019P**
Studijní program:	**N0715A270012 Průmyslové inženýrství a management**
Téma práce:	**Tvorba modulárního interaktivního digitálního dvojčete výrobního úseku pro virtuální realitu**
Zadávající katedra:	**Katedra průmyslového inženýrství a managementu**

Zásady pro vypracování

1. Úvod
2. Analýza současného stavu
3. Výrobní úseky vytvořené ze stavebnice Fishertechnik
4. Výběr vhodné technologie pro digitalizaci
5. Popis tvorby interaktivních digitálních dvojčat
6. Popis verze pro virtuální realitu
7. Závěr

Rozsah diplomové práce: **50 – 70 stran**
Rozsah grafických prací: **0**
Forma zpracování diplomové práce: **tištěná**

Seznam doporučené literatury:

1. OKITA, A. *Learning C# Programming with Unity 3D.* Second edition, Boca Raton, FL. USA: Routledge, 2019. 690 p. ISBN-13: 978-1138336810.
2. SUNG, K., SMITH, G. *Basic Math for Game Development with Unity 3D: A Beginner's Guide to Mathematical Foundations.* Bothel, WA. USA: Apress, 2019. 424 p. ISBN 978-1484254424.
3. LINOWES, J. *Unity Virtual Reality Projects: Learn Virtual Reality by developing more than 10 engaging projects with Unity 2018.* 2nd Edition, Birmingham, UK: Packt Publishing, 2018. 492 p. ISBN 978-1788478809.
4. LaVALLE, S. M. *Virtual Reality.* Cambridge University Press, 2020. 390 p., dostupné zdarma 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. Bc. Miroslav Malaga**
Katedra průmyslového inženýrství a managementu

Datum zadání diplomové práce: **20. září 2021**
Termín odevzdání diplomové práce: **27. května 2022**

L.S.

Doc. Ing. Milan Edl, Ph.D.
děkan

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

V Plzni dne 20. září 2021

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 chtěla poděkovat vedoucímu mé diplomové práce doc. Ing. Petru Hořejšímu, Ph.D. za cenné rady, ochotu a čas, který mi věnoval během zpracování diplomové práce. Dále bych rád poděkoval Ing. Bc. Miroslavu Malagovi za poskytnutá data a za pomoc při tvorbě digitálního dvojčete.

ANOTAČNÍ LIST DIPLOMOVÉ PRÁCE

AUTOR	**Příjmení** Dvořák	**Jméno** Matěj	
STUDIJNÍ OBOR	Průmyslové inženýrství a management (PIMB)		
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	Tvorba modulárního interaktivního digitálního dvojčete výrobního úseku pro virtuální realitu		

FAKULTA	strojní	**KATEDRA**	KPV	**ROK ODEVZD.**	2022

POČET STRAN (A4 a ekvivalentů A4)

CELKEM	80	**TEXTOVÁ ČÁST**	68	**GRAFICKÁ ČÁST**	12

STRUČNÝ POPIS	Diplomová práce se zabývá tvorbou modulárního interaktivního digitálního dvojčete vybraného výrobního úseku stavebnice Fischertechnik pro virtuální realitu. První část práce se zaměřuje na teoretický popis pojmů jako jsou Průmysl 4.0, digitální dvojče a softwary vhodné pro tvorbu výstupu této práce. V další části práce je dle zvolených kritérií vybrán nejvhodnější software. Následuje popis tvorby interaktivních digitálních dvojčat ve smyslu 3D modelování v programu Blender a programování jednotlivých částí výrobního úseku v softwaru Unity 3D. Na závěr je popsáno převedení výstupu práce do virtuální reality. Výsledek této práce bude sloužit pro pedagogicko-vědecké účely.
KLÍČOVÁ SLOVA	Digitální dvojče, digitální stín, virtuální realita, Fischertechnik, sériový port, COM port, 3D model, Unity 3D, Blender

SUMMARY OF DIPLOMA SHEET

AUTHOR	**Surname** Dvořák	**Name** Matěj	
FIELD OF STUDY	Průmyslové inženýrství a management (PIMB)		
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	Development of a manufacturing section modular interactive digital twin for virtual reality		

FACULTY	Mechanical Engineering		**DEPARTMENT**	Industrial Engineering and Management		**SUBMITTED IN**	2022

NUMBER OF PAGES (A4 and eq. A4)

TOTALLY	80		**TEXT PART**	68		**GRAPHICAL PART**	12

BRIEF DESCRIPTION	The thesis deals with the creation of a modular interactive digital twin of a selected production section from the Fischertechnik construction kit for virtual reality. The first part of the thesis focuses on the theoretical description of concepts such as Industry 4.0, digital twin and software suitable for the creation of the output of this thesis. In the next part of the thesis the most suitable software is selected according to the chosen criteria. This is followed by a description of the creation of interactive digital twins in terms of 3D modelling in Blender and the programming of individual parts of the production section in Unity 3D software. Finally, the transfer of the output of the work into virtual reality is described. The result of this work will be used for pedagogical and scientific purposes.
KEY WORDS	Digital twin, digital shadow, virtual reality, Fischertechnik, serial port, COM port, 3D model, Unity 3D, Blender

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

Přehled použitých zkratek

4IR – Fourth industrial revolution – Čtvrtá průmyslová revoluce

IoT – Internet of things – Internet věcí

IoS – Internet of services – Internet služeb

VR – Virtual reality – Virtuální realita

PLM – Product lifecycle management – Řízení životního cyklu výrobku

CAI - Computer-assisted instruction – Počítačem podporovaná výuka

SW – Software

3D – Three-dimensional – Trojrozměrný

LTS – Long term support – Dlouhodobá podpora

URP – Universal render pipeline

USB – Univerzální sériová sběrnice

PCI – Peripheral component interconnect – Sběrnice používaná pro připojování přídavných karet k základní desce

SPP – Serial port profile – Základní profil používaný v rámci rozhraní bluetooth

TCP – Transmission control protocol – Jeden ze základních protokolů sady protokolů internetu

API – Application programming Interface – Rozhraní pro programování aplikací

HMD – Head mounted display – Náhlavní souprava

FPS – Frames per second – Snímková frekvence

Úvod

V dnešní době, kdy je kladen velký důraz na rozvoj pokročilé výrobní techniky, informačních a komunikačních technologií, se výrobní model přesouvá z digitálního na inteligentní. Podniky musí zvyšovat svou celkovou úroveň industrializace, automatizace a digitalizace, aby dosáhly vyšší efektivity, kompetence a konkurenceschopnosti. V rámci tohoto neustálého rozvoje se čím dál více zmiňuje pojem Průmysl 4.0. Ten ve výrobě představuje propojenou informační síť s lidmi a stroji – v souvislosti s tímto propojením patří mezi klíčové pojmy simulace a digitální dvojčata, tedy virtuální modely fyzického systému, které se mohou používat k zobrazování současného stavu podniku nebo k plánování a simulování nastávajících změn. Jelikož komunikace mezi takovýmto virtuálním modelem a jeho reálným fyzickým protějškem probíhá většinou obousměrně, otevírá svět digitálních dvojčat nespočet možností pro rozvoj podniků.

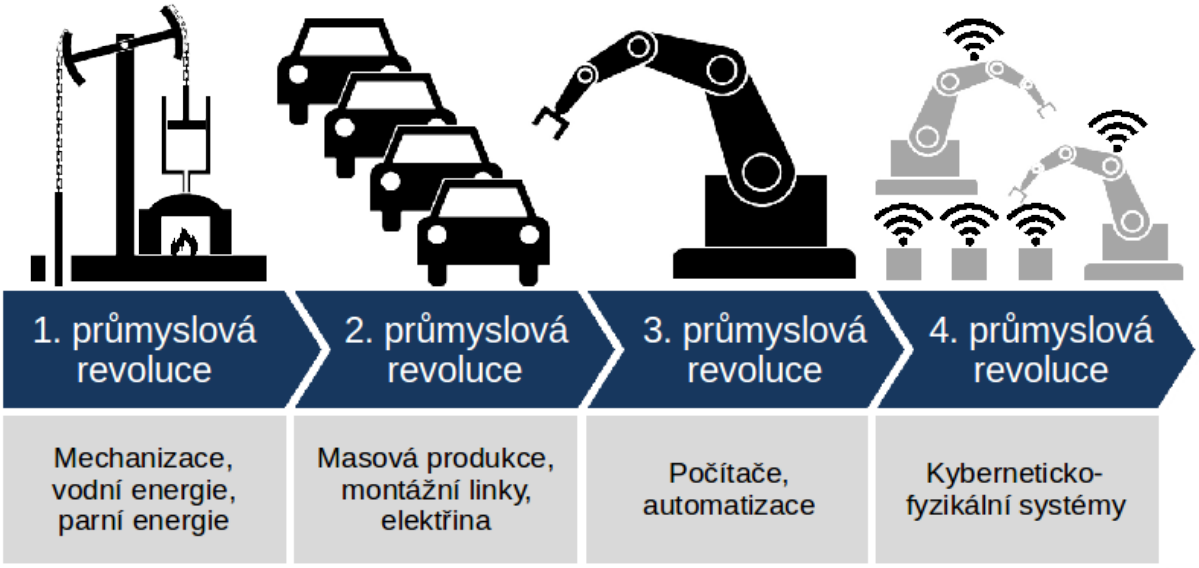
Autor této práce čerpá ze zkušeností nabraných při práci na projektech, které se věnovaly digitalizaci a tvorbě interaktivních 3D modelů v propojení s virtuální realitou. Většina těchto projektů již byla úspěšně implementována do reálných podniků.

Hlavním cílem této práce je vytvoření aplikace ve virtuální realitě, která má za úkol komunikovat skrze sériové porty s výrobním úsekem vytvořeným ze stavebnice Fischertechnik. Uživatel by s touto aplikací měl být schopen sledovat reálné dění na pracovištích a neměla by také chybět možnost interakce s jednotlivými 3D modely. Tato aplikace bude sloužit i jako učební pomůcka při výuce na Západočeské univerzitě v Plzni.

1. Průmysl 4.0

V poslední době je Průmysl 4.0 jedním z hojně diskutovaných trendů nejen ve strojním inženýrství. Podniky musejí stále více digitalizovat své prostředí, aby udržely krok s ostatními a trendy Průmyslu 4.0 jsou jim při tom obrovskou podporou.

Průmysl jako takový je považován za část ekonomiky, která vyrábí hmotné statky, jež jsou vysoce mechanizované a automatizované. Od počátku industrializace vedly technologické skoky ke změnám paradigmat, které se dnes ex post nazývají "průmyslové revoluce": v oblasti mechanizace (tzv. 1. průmyslová revoluce), intenzivního využívání elektrické energie (tzv. 2. průmyslová revoluce) a rozsáhlé digitalizace (tzv. 3. průmyslová revoluce). Na základě pokročilé digitalizace v rámci továren se zdá, že kombinace internetových technologií a technologií orientovaných na budoucnost v oblasti "inteligentních" objektů povede k nové zásadní změně paradigmatu průmyslové výroby (viz Obr. 1-1). [1]



Obr. 1-1 Průmyslová revoluce [7]

1.1 Charakteristika

Cílem Průmyslu 4.0 je dosáhnout vyšší úrovně automatizace, efektivity a produktivity. Mezi nejvýraznější charakteristiky tohoto konceptu se řadí digitalizace, optimalizace, personalizace výroby, interakce člověk-stroj, služby s přidanou hodnotou a další. Tyto charakteristiky nejenže silně korelují s internetovými technologiemi a pokročilými algoritmy, ale také naznačují, že Průmysl 4.0 je průmyslový proces přidávání hodnoty a řízení znalostí.

Ke zvýšení produktivity používá Průmysl 4.0 technologie jako internet věcí (IoT) a služeb (IoS), kybernetické fyzické systémy (CPS), průmyslovou automatizaci, nepřetržitou konektivitu, kybernetickou bezpečnost, inteligentní robotiku, PLM[1], sémantické technologie,

[1] Informační platforma zahrnující výrobní, technická a marketingová data o daném výrobku

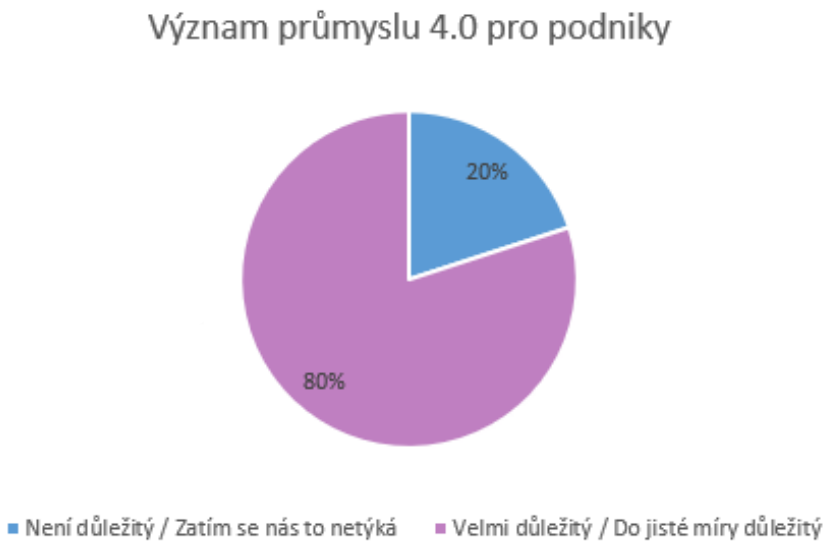
průmyslová big data[2] a počítačové vidění (Obr. 1-2). Tyto uvedené technologie mohou ovlivnit způsob výroby výrobků a také vnímání hodnoty výrobku zákazníky. Navržené výrobky mají jedinečnou elektronickou identifikaci, aby bylo možné sledovat životní cyklus výrobku. To umožní dostatečný sběr údajů o jeho používání. Podniky mohou díky tomu lépe porozumět spotřebě jednotlivých výrobků a tím je mohou přizpůsobit specifickým požadavkům zákazníka. Kromě toho budou vazby mezi stroji, zařízeními a prvky dodavatelského řetězce s pomocí společně sdílených informací poskytovat možnost rychle měnit priority zakázek (podle požadavků zákazníků nebo požadavků na údržbu), sledovat a řídit výkonnost montážních linek, sledovat dodávky a také zlepšovat logistické trasy. [3]

Obr. 1-2 Technologie Průmyslu 4.0 [5]

1.2 Výzkum a metodika

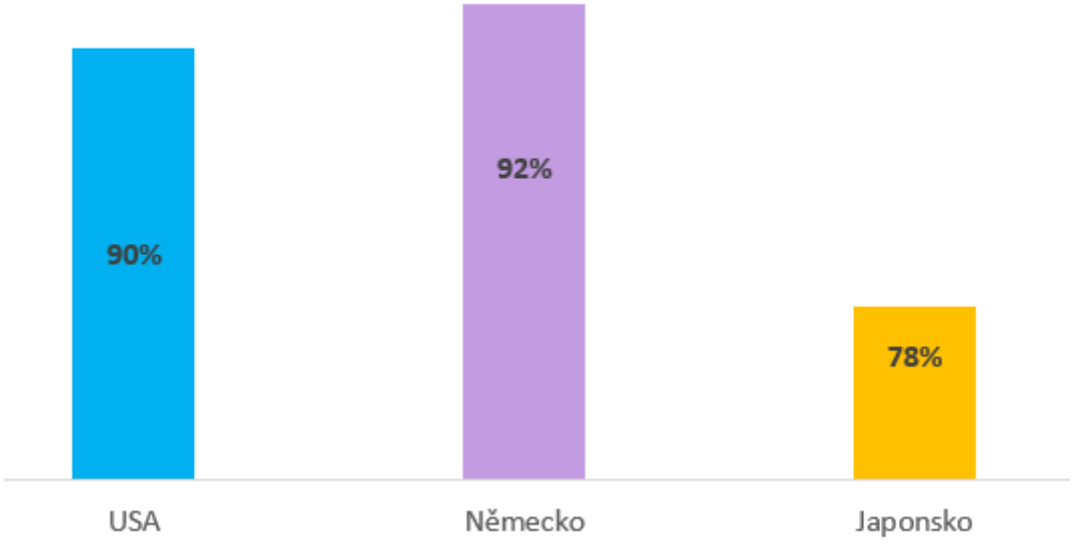
Předpoklady, které jsou uvedeny v kapitole 1.1, jednoznačně dokládají příchod 4. průmyslové revoluce (4IR). Již nyní je koncept Průmyslu 4.0 nevyhnutelným budoucím fungováním většiny podniků. Na druhé straně je velmi zajímavou otázkou poznání praktického aspektu vnímání tohoto trendu a možnosti uplatnění jeho nástrojů v činnosti podniků. Výzkum provedený v článku [6] vychází ze sekundárních dat, získaných z mnoha odborných zpráv a studií, realizovaných státními institucemi, konsorcii či podniky.

[2] Soubory dat příliš velké na to, aby je dokázaly zachytit, zpracovat nebo spravovat běžně používané softwary



Obr. 1-3 Význam průmyslu 4.0 pro podniky [3]

Většina dotazovaných společností si je vědoma významu Průmyslu 4.0. Jak je vidět na Obr. 1-3, pouze pro 20 % respondentů není 4IR důležitá a dosud neovlivnila jejich provoz. Podniky, pro které je Průmysl 4.0 pro jejich fungování zásadní, mohou vývoj a aplikaci nových technologií vnímat jako příležitost nebo hrozbu pro svou činnost a postavení na trhu. Na Obr. 1-4 je znázorněn přístup k této problematice ve Spojených státech, Německu a Japonsku. Většina podniků ve všech třech zemích považuje Průmysl 4.0 za příležitost, nikoliv za hrozbu. V USA a Německu takový názor vyjádřilo více než 90 % dotázaných. Podnikatelé z Japonska jsou ve vztahu ke 4. průmyslové revoluci o něco méně nakloněni optimismu, nicméně i mezi nimi většina uznává její pozitivní rozměr (viz Obr. 1-4). [3]



Obr. 1-4 Průmysl 4.0 jako příležitost, ne jako riziko [3]

2. Digitální dvojče

Jak je již jasné z kapitoly 1, digitalizace nekončí na úrovni továren. Chytré továrny vyrábějí chytré výrobky. Miniaturizace a pokles cen umožňují integraci informačních, komunikačních a senzorových technologií i do těch nejmenších výrobků. Výrobky začínají být schopny vnímat svůj vlastní stav i stav svého okolí. Ve spojení se schopností zpracovávat a komunikovat tato data umožňují vytvářet digitální dvojčata.

Od vzniku konceptu Johna Vickerse a Dr. Michaela Grievese [2] se mnoho autorů pokoušelo definovat pojem digitální dvojče, počínaje leteckým průmyslem se zaměřením na stavební mechaniku, materiálové vědy a dlouhodobé předpovědi výkonnosti leteckých a kosmických plavidel. S nástupem Průmyslu 4.0 se pozornost přesunula na výrobu a chytré výrobky. V této souvislosti může digitální dvojče pomoci při zajišťování kontinuity informací v průběhu celého životního cyklu výrobku, při virtuálním uvádění (výrobních) systémů do provozu a při podpoře rozhodování a předvídání chování systému ve fázi vývoje výrobku i ve všech následujících fázích životního cyklu na základě počítačem podporovaných simulací.

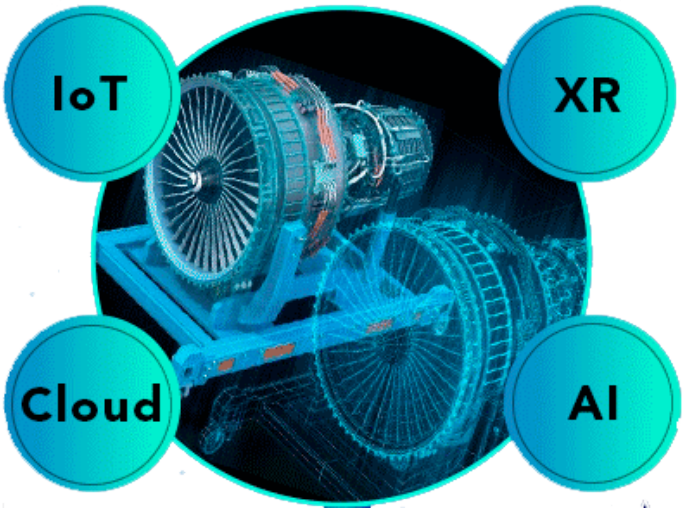
Pro účely této diplomové práce je digitální dvojče (Obr. 2-1) chápáno jako komplexní digitální reprezentace jednotlivého produktu. Zahrnuje vlastnosti, stav a chování reálného objektu prostřednictvím modelů a dat. Digitální dvojče je soubor realistických modelů, které mohou simulovat jeho skutečné chování v nasazeném prostředí a je vyvíjeno souběžně s jeho fyzickým dvojčetem a zůstává jeho virtuálním protějškem po celou dobu životního cyklu výrobku. Kromě toho lze digitální dvojče rozdělit do tří podkategorií, dle stupně integrace – konkrétně dle odlišného stupně toku dat a informací, který může nastat mezi fyzickou částí a její digitální kopií. Těmito podkategoriemi jsou digitální model, digitální stín a digitální dvojče. [4]

Obr. 2-1 Digitální dvojče [8]

Oblasti použití

Aplikace digitálního dvojčete se dnes hojně využívají v mnoha průmyslových odvětvích. Díky přesným virtuálním reprezentacím objektů a simulacím provozních procesů umožňují tyto aplikace různým podnikům rychle identifikovat oblasti pro inovace a zlepšit obchodní procesy a výkonnost. Mezi příklady digitálních dvojčat se řadí Product Digital Twin, Component

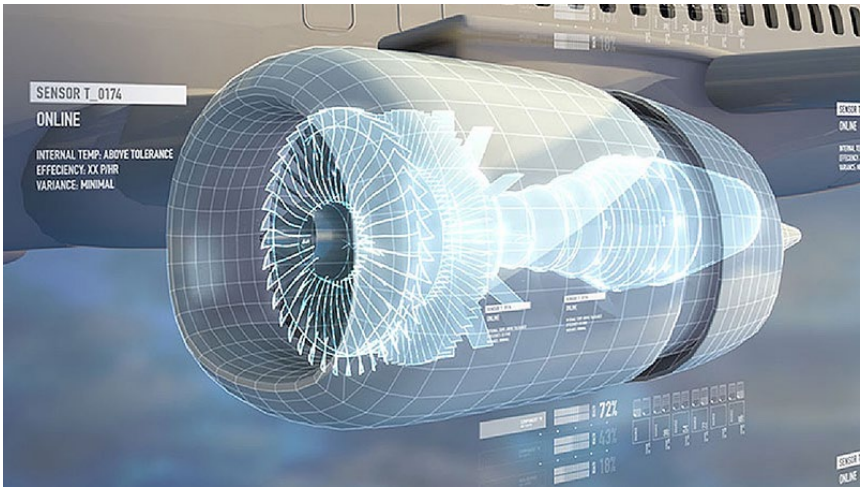
Digital Twin a Performance Digital Twin, které se používají pro vizualizaci objektů a procesů, ať už jednoduchých, nebo složitějších. Aplikace digitálního dvojčete zahrnuje čtyři technologie, které umožňují vytvářet digitální reprezentaci, shromažďovat a ukládat data v reálném čase a na základě získaných informací poskytovat cenné poznatky. Mezi tyto technologie patří internet věcí (IoT), rozšířená a virtuální realita, cloud a umělá inteligence (Obr. 2-2). V závislosti na typu aplikace může být konkrétní technologie využita ve větší či menší míře. [9]



Obr. 2-2 Technologie využívané digitálním dvojčetem [9]

2.1 Letectví a kosmonautika

Mezi různými průmyslovými odvětvími, kde se digitální dvojče používá, jsou aplikace tohoto konceptu v letectví a kosmonautice poměrně vyspělé. Jedním z důvodů je skutečnost, že letectví a kosmonautika jsou nejranější průmyslovou oblastí, která digitální dvojče používá. Tuto technologii poprvé zavedla NASA a následně ji aplikovaly velké společnosti, jako jsou Northrop Grumman, Airbus, Boeing a General Electric Company (GE). Vycházeje z potřeby předvídat poruchy a navrhovat strategie údržby letadla ve vesmíru, NASA vytvořila model digitálního dvojčete, který odráží stav letadla v reálném čase (Obr. 2-3). Tento model dokáže optimalizovat výkon letadla, předpovídat potenciální poruchy a podporovat inženýry, aby na dálku rozpoznali poruchy a nabídli účinná řešení díky napájení daty v reálném čase. Nedávno NASA také vynaložila úsilí na zajištění bezpečnosti posádky v letadle, u kterého došlo k poruše. [10]



Obr. 2-3 Digitální dvojče v letectví [11]

2.2 Automobilový průmysl

Automobilový průmysl zahrnuje různé fáze celého životního cyklu vozidla, jako je konstrukce, výroba a údržba. Vzhledem k tomu, že konstrukce automobilů jsou stále složitější, jsou požadavky na vysoce přesnou kontrolu a údržbu stále vyšší. V důsledku toho mnoho společností přijalo v automobilovém průmyslu koncept digitálního dvojčete (Obr. 2-4). Softwarová společnost Bsquare, která se zabývá IoT, vytvořila modely digitálního dvojčete pro motory nákladních automobilů a další díly, aby vytvořila scénáře oprav pro výrobce nákladních automobilů PACCAR. Za účelem podpory údržby nákladních vozidel je vytvořen model motoru za konkrétních podmínek s využitím dat shromážděných v reálném čase pomocí senzorů a informací v systému. Na základě toho lze zkrátit dobu údržby až o 20 %. Společnost GE se domnívá, že digitální vlákno a digitální dvojče by v budoucnu mohly být využity ke sledování lokomotivy po celou dobu jejího životního cyklu. Konkrétně lze digitálně sledovat způsob, jakým je lokomotiva navržena, konfigurována, konstruována, provozována a udržována. Zejména díky tomu, že lze v reálném čase shromažďovat údaje o stavu jednotlivých komponent a změnách souvisejících proměnných, mohou inženýři optimalizovat výkon lokomotivy od úspory paliva až po neplánované prostoje. Společnost International Business Machines Corporation (IBM) se domnívá, že digitální dvojče přinese během několika příštích let vozidlům obrovskou hodnotu. Ještě před výrobou skutečného vozidla mohou inženýři vytvářet různé modely a ty simulovat. Díky tomu se dozví, jak bude vozidlo fungovat za různých podmínek, jak bude komunikovat s různými řidiči atd. Mohou také předem zjistit, co se děje, a zjistit, kdy, proč a jak k problému došlo, a tím snížit náklady a rizika neplánovaných odstávek. [10]

Obr. 2-4 Digitální dvojče v automobilovém průmyslu [12]

2.3 Energetický průmysl

Vzhledem k tomu, že digitální dvojče může zvýšit spolehlivost a pomáhá monitorovat mnoho zařízení na výrobu energie, zavedlo tuto technologii do elektroenergetiky mnoho společností. Společnost GE například vybudovala digitální větrnou farmu, která nově definuje budoucnost větrné energie (viz Obr. 2-5). Na základě průběžně shromažďovaných dat v reálném čase (počasí, hlášení komponent, servisní zprávy) je pro každou větrnou turbínu vytvořen systém digitálního dvojčete, který optimalizuje strategii údržby zařízení, zlepšuje spolehlivost a zvyšuje roční produkci energie. Očekává se, že zvýší účinnost až o 20 %. Společnost Siemens se zase pokusila vytvořit model digitálního dvojčete pro skutečnou energetickou síť ve Finsku. Digitální síť se používá pro plánování, provoz a údržbu zařízení, a přináší tak několik výhod, jako je převedení většiny manuálních prací na simulaci na automatizované práce, zlepšení využití dat, standardizace rozhraní pro data a další. [10]

Obr. 2-5 Digitální dvojče v energetickém průmyslu [13]

2.4 Strojírenský průmysl

Díky digitálním dvojčatům mohou společnosti simulovat a ověřovat každý krok vývoje, aby identifikovaly problémy a případné poruchy ještě před výrobou finálního produktu. Například dvojče, které je vyobrazené na Obr. 2-6, pomáhá vývojovým týmům pochopit, jak může případná změna ve výrobním procesu ovlivnit výsledky výroby, a podle toho jim umožní upravit výrobní operace tak, aby bylo dosaženo cílených zlepšení. Díky tomu mohou výrobci zlepšit provozní procesy, a tím snížit celkové náklady na výrobu. [9]

Obr. 2-6 Digitální dvojče ve strojírenském průmyslu [14]

3. Fischertechnik jako digitální dvojče

Pro splnění hlavního cíle finální práce – vytvoření interaktivního modelu digitálního dvojčete ve virtuální realitě – byly vybrána stavebnice Fischertechnik, jejíž úseky poslouží jako model pro digitální dvojče automatizované výrobní linky. Tato podkapitola obsahuje popis této stavebnice a dále rozebírá několik přístupů, při kterých byla použita právě pro tvorbu digitálního dvojčete.

Fischertechnik Training Factory Industry 4.0 (Obr. 3-1) od společnosti Fischertechnik GmbH je fyzický simulační model, který v malém měřítku a s moderními digitálními prvky kopíruje standardní průmyslový výrobní závod. Vzhledem k tomu, že tento model je plně automatizovaný a digitalizovaný a je integrován s mnoha funkcemi Průmyslu 4.0 používanými ve skutečných průmyslových závodech, lze automatizovaný průmyslový proces modelu simulovat, pochopit a aplikovat v malém měřítku předtím, než bude implementován ve větším měřítku, a také otestovat funkce Průmyslu 4.0 před širším nasazením. Fischertechnik Training Factory 4.0 je modulární, vysoce flexibilní model a také výkonný školicí a simulační model. [10]

Obr. 3-1 Simulační model Fischertechnik [10]

Podívejme se nyní na příklady prací, jejichž cílem bylo vytvoření plnohodnotného digitálního dvojčete, vycházející z fyzického modelu tvořeného právě stavebnicí Fischertechnik.

V práci [10] se autor věnuje metodám implementace digitálního dvojčete v korelaci s PLM. Hlavním konceptem je implementace digitálního dvojčete právě pro model Fischertechnik Training Factory 4.0. Hlavním cílem jeho práce je porovnání a posouzení konceptů digitálního

dvojčete ve třech fázích životního cyklu výrobku na základě funkčního, uživatelského, fyzického a datového aspektu. [10]

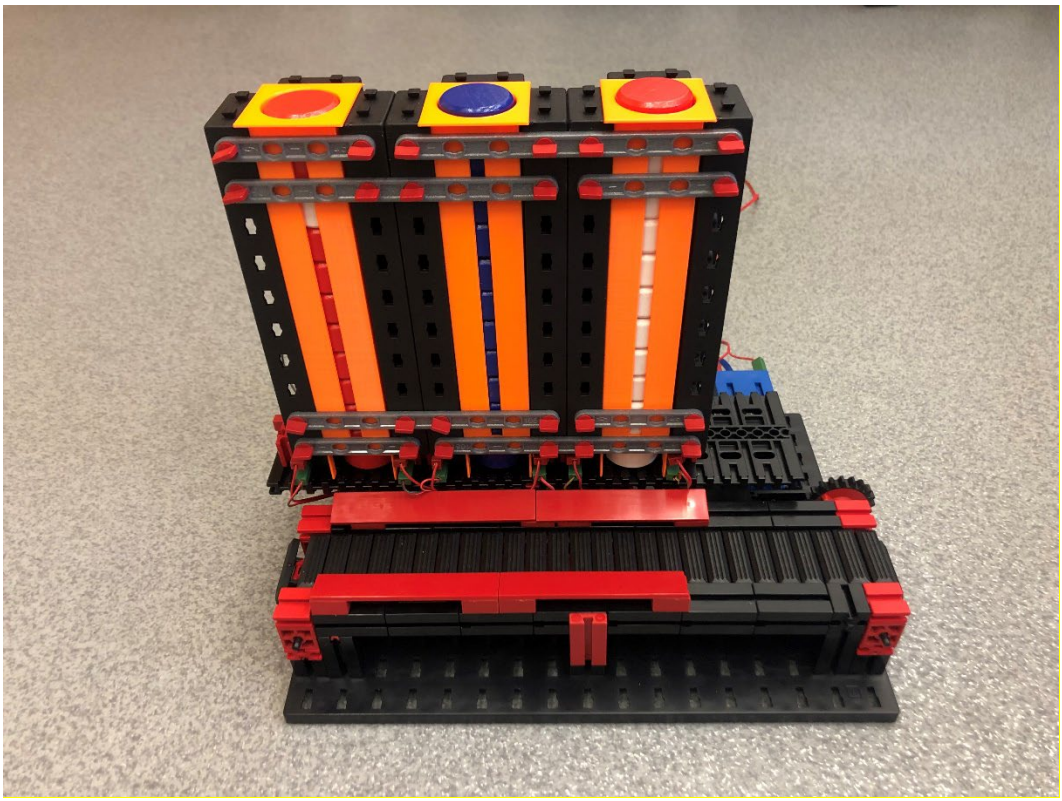
V [15] se autorka věnuje tvorbě 3D simulačního modelu v softwaru Plant Simulation, který slouží jako digitální dvojče pro model Fischertechnik. Modely této práce jsou ale odděleny a jediným spojovacím článkem je uživatel, bez kterého není možné propojit ovládací rozhraní obou systémů tak, aby běžely současně a model v softwaru Plant Simulation tak mohl být brán jako plnohodnotné digitální dvojče. [15]

Autor [16] se ve své práci zabývá propojením simulačního modelu Fischertechnik s jeho digitálním dvojčetem, vytvořeným také v softwaru Plant Simulation. Pro analýzu tohoto dvojčete jsou navrženy tři scénáře, které mají dokázat potenciál zvolených nástrojů pro účely vzdělávání. Kvůli nedostatečné znalosti programovacího jazyka C++ je výsledkem práce vytvoření 2 příruček, které usnadňují práci při využití obou těchto nástrojů. [16]

4. Vytvořené výrobní úseky

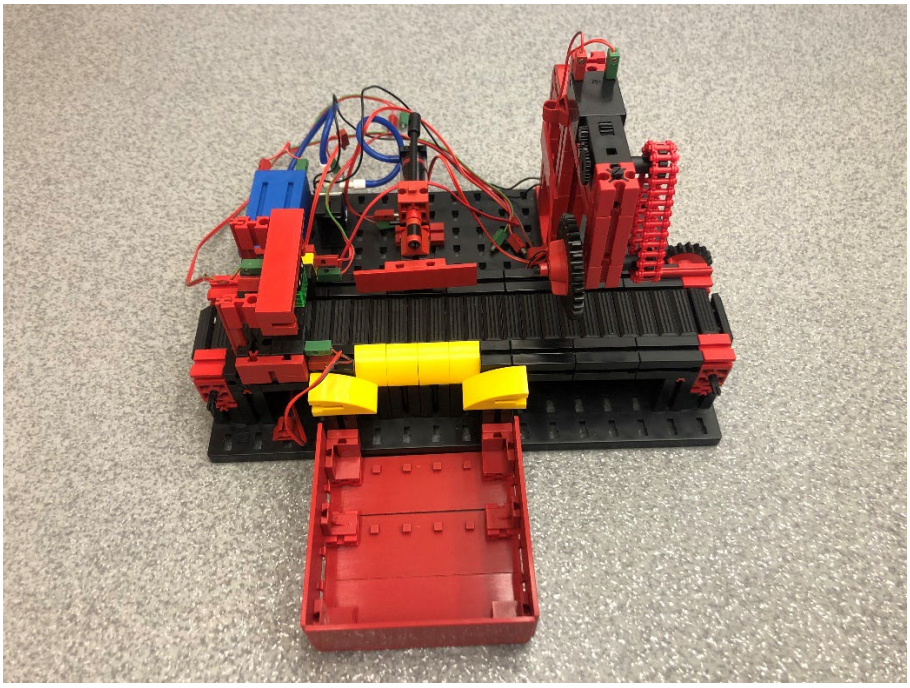
V návaznosti na předchozí kapitolu se tato kapitola věnuje již vybraným výrobním úsekům modelu Fischertechnik, které simulují automatizovanou výrobní linku. Z těchto úseků bude dále vytvořeno interaktivní digitální dvojče ve virtuální realitě. Všechny zde zmiňované modely byly poskytnuty Ing. Bc. Miroslavem Malagou.

Prvním výrobním úsekem je zásobník, který je vidět na Obr. 4-1. Tento zásobník simuluje sklad s polotovary, který je na začátku automatizované výrobní linky. Prostřednictvím podavačů je možné volit, který polotovar bude vpuštěn do výroby.



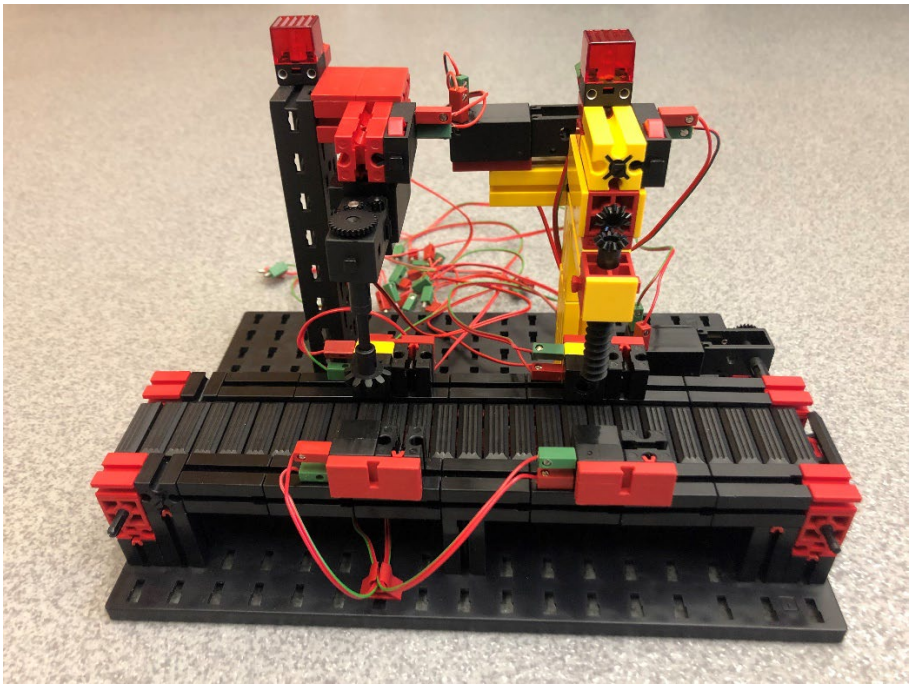
Obr. 4-1 Zásobník

Na sklad navazuje druhý úsek, který je složený ze dvou částí. V první části je polotovar načten laserovým měřícím senzorem, který rozhodne, zda se jedná o zmetek. Pokud ano, součást je vytlačena do paletky určené pro zmetky. V opačném případě putuje dál výrobou až na pracoviště pily, kde je simulováno nařezání součásti na požadovanou délku (viz Obr. 4-2).



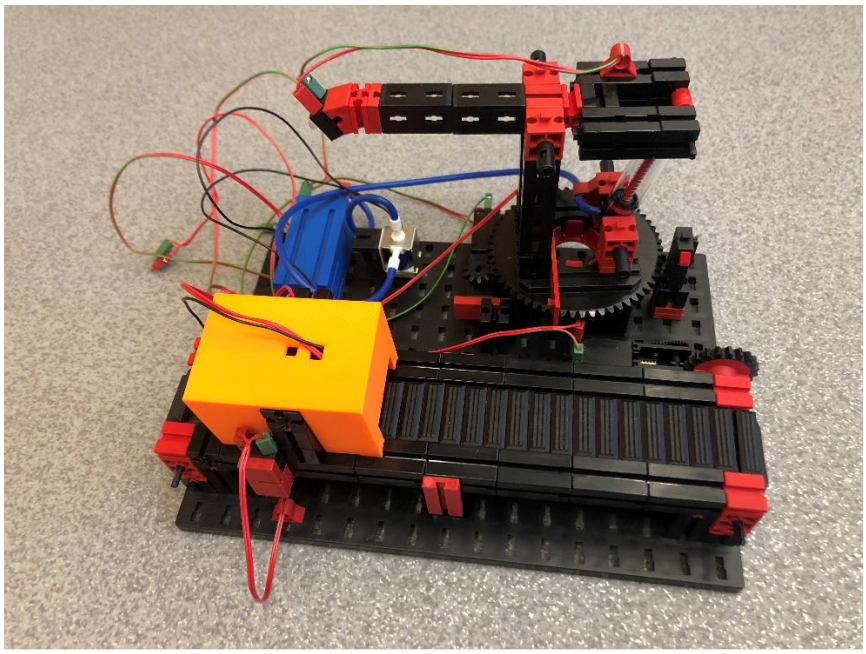
Obr. 4-2 Laserový senzor a pracoviště pily

Třetí výrobní úsek je tvořen dvěma pracovišti, a to frézkou a vrtačkou. Obě tyto pracoviště mají ještě zakomponovaná světelná čidla, která v případě rozsvícení značí poruchu daného stroje (Obr. 4-3).



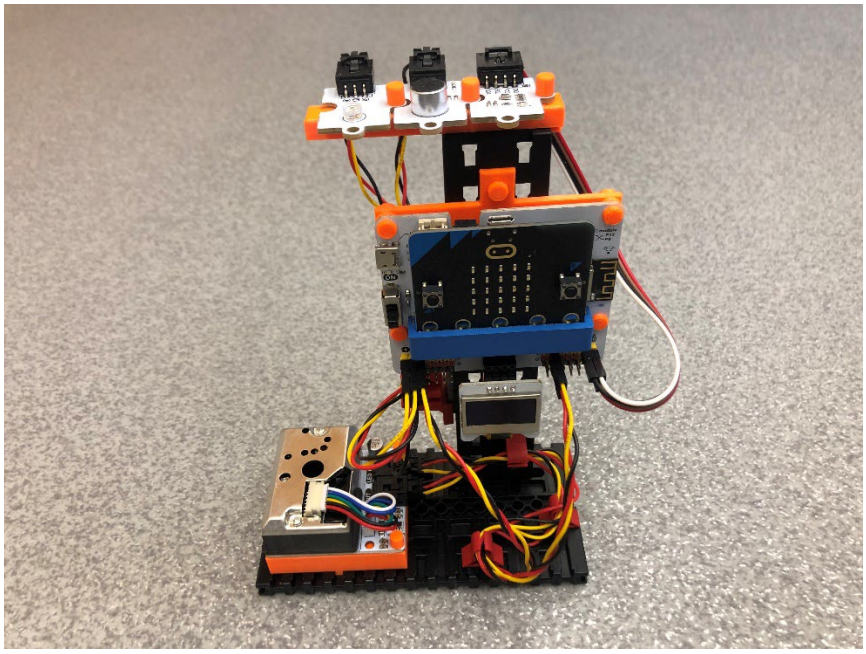
Obr. 4-3 Frézka a vrtačka

Posledním výrobním úsekem (Obr. 4-4) je složené pracoviště, které se skládá ze senzoru barev a svářečky. Obrobek je nejprve otestován senzorem barev, který vysílá červené světlo. Díky tomu, jakou intenzitou je toto světlo odráženo od různě barevných povrchů, dokáže senzor určit typ obrobku a předat tuto informaci do svářečky, která provede finální proces a již hotový výrobek pokračuje do palety.



Obr. 4-4 Senzor barev a svářečka

Součástí této automatizované výrobní linky je ještě senzorová stanice (Obr. 4-5), která komunikuje se všemi jednotlivými výrobními úseky a obsahuje senzory, které jsou schopné detekovat otřesy, měřit úroveň hluku, teplotu vzduchu, vlhkost vzduchu a další.



Obr. 4-5 Senzorová stanice

5. Technologie pro digitalizaci

Jelikož hlavním cílem celé práce je vytvoření interaktivního modulárního digitálního dvojčete ve virtuální realitě, je důležité zvolit vhodné softwary s adekvátním vývojářským prostředím, které tuto tvorbu umožňují. Tato kapitola popisuje softwary umožňující vytvoření interaktivního prostředí a jeho následné převedení do VR.

5.1 Software pro tvorbu 3D modelů – Blender

3D modely jsou základem každé aplikace, která se zaměřuje na virtuální realitu. Jelikož pro zobrazení výstupu této práce bude použit VR headset[3] Oculus Quest 2, jehož výkon zajišťuje kvůli platformě Android mobilní procesor, je nutné, aby 3D modely byly co nejvíce optimalizované. Z toho důvodu je důležité zvolit takový modelovací nástroj, ve kterém je možné tvořit takzvané low – poly modely. Ty zpravidla obsahují menší počet polygonů, které jsou při běhu aplikace vykreslovány pomocí zobrazovacího zařízení (v tomto případě VR headset). Čím nižší počet těchto polygonů, tím rychlejší je chod aplikace.

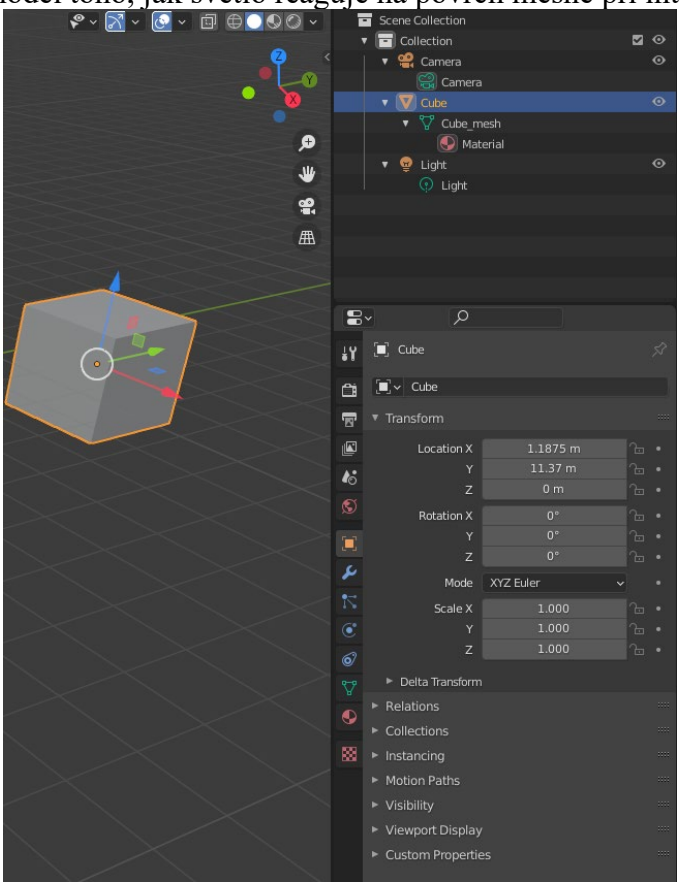
Tomuto kritériu jednoznačně odpovídá SW Blender, což je bezplatný a open source[4] balíček sloužící k 3D tvorbě. Podporuje celou řadu 3D technologií, jako je modelování, rigging, animaci, simulaci, rendering, kompozici a sledování pohybu, střih videa a tvorbu her. Pokročilí uživatelé využívají API Blenderu pro skriptování v jazyce Python, aby si mohli aplikaci přizpůsobit a psát specializované nástroje [29]. Blender se primárně používá v aplikacích počítačové grafiky. Je tedy možné například vytvářet fotorealistické scény, přičemž možnosti Blenderu zahrnují komplexní osvětlení, hloubku ostrosti, volumetrické efekty (mlha) a fyzikálně přesné odrazy a lomy. Používá se však také ve vědeckých a technických aplikacích k vytváření syntetických obrazů, především pro aplikace počítačového vidění a strojového učení. Pro splnění cílů této práce byl použit pro vymodelování jednotlivých části výrobních úseků stavebnice Fischertechnik a pro jejich otexturování, tedy nastavení dílčích materiálů a jejich přiřazení k daným dílům. Klíčové části datového modelu programu Blender (Obr. 5-1) jsou:

- **Scéna** – 3D prostor, do kterého lze umístit objekty.
- **Objekt** – Abstraktní "obal", který obsahuje další data (meshe[5], kamery, světla atd.). Klíčovými vlastnostmi objektu jsou poloha a orientace v prostoru a také data v něm obsažená.
- **Mesh** – Sada umístění vrcholů a matice spojitosti, která popisuje, jak jsou vrcholy spojeny do hran a ploch.
- **Kamera** – Určuje úhel pohledu na scénu a způsob jejího vykreslení. Lze zadat parametry, jako je ohnisková vzdálenost a clona objektivu.
- **Světlo** – Slouží jako zdroj světla pro scénu. Světlo se dá v Blenderu využít v několika typech, například jako reflektor nebo bodové světlo. Může také poskytovat mesh s

[3] Brýle sloužící pro zobrazování aplikací ve virtuální realitě.
[4] Počítačový software s otevřeným zdrojovým kódem.
[5] Kolekce bodů, stěn a hran ve 3D kartézské soustavě souřadnic.

vyzařovacím materiálem, což je užitečné pro simulaci plošných světel, jako jsou okna nebo sluneční světlo zasahující do celé scény.

- **Materiál** – model toho, jak světlo reaguje na povrch meshe při interakci s ním.



Obr. 5-1 Klíčové části datového modelu v prostředí Blenderu

5.2 Software pro tvorbu digitálního dvojčete

V dnešní době existuje několik programů, které podporují využití vhodného enginu pro virtuální realitu. Engine lze chápat jako jádro virtuálního světa, kolem kterého se následně skládají 3D modely. Na nich se generují různé algoritmy, aplikuje fyzika, pravidla a způsob vykreslování. Díky správně nakonfigurovanému Enginu je možné ve VR využívat dynamické objekty, fyzikální síly a obecně vytvářet interakci ve všech různých formách. Mezi známé programy patří Unity 3D, Godot engine, Unreal Engine a SimLab Soft. Cílem této kapitoly je představit zmíněné programy, jejich výhody, nevýhody a na závěr je porovnat a vybrat nejlepší variantu pro realizaci tohoto projektu. Rozhodování bude podle stanovených kritérií jako jsou cena, funkce, výstup, zkušenosti, softwarové možnosti a komunita. Jedním z hlavních kritérií je také možnost vytvořený obsah přizpůsobit a modifikovat dle vlastních potřeb. Tyto programy nabízejí možnost vytvářet libovolná individuální řešení jednotlivých projektů a přizpůsobovat jim obsah aplikace.

5.2.1 Unity 3D

Unity je multiplatformní herní engine vyvinutý společností Unity Technologies. Byl použit pro vývoj počítačových her pro PC, konzole, mobily a web. Aplikace Unity je kompletním SW pro

tvorbu 3D nebo 2D prostředí, vytváření animací a interakce s modely. Poskytuje propojení s několika platformami, kde Unity 3D nabízí základní platformy jako Windows, IOS, WebGL a díky spolupráci s Android studiem i Android aplikace. Spolupráce s Android studiem je na vysoké úrovni, co dokazuje i fakt, že v posledních letech se tento engine začal více zaměřovat na mobilní aplikace a podobně menší projekty. V posledních letech se tento SW začíná rozšiřovat i do jiných odvětví než jen herních. Na svých stránkách nabízejí podporu z hlediska automotivu, architektury a filmů. Samotné Unity je velmi komplexní, co se týká spolupráce s jinými programy (např. Pixys, Blender, CAD SW atd.). Z hlediska různých modulů je součástí Unity Asset Store, který je online knihovnou jednotlivých komponent a řešení pro snadnější tvorbu. [17]

Obr. 5-2 Prostředí Unity 3D [18]

Na Obr. 5-2 je prostředí layoutu Unity, kde hlavní část rozložení tvoří *scene*, které ukazuje reálný vzhled VR včetně stínů, barev a grafiky. Kromě grafického prostředí pro tvorbu a interakci s jednotlivými modely podporuje také tvorbu skriptů především v jazyce C# a UnityScript (podobná syntaxe jako Javascript). [17]

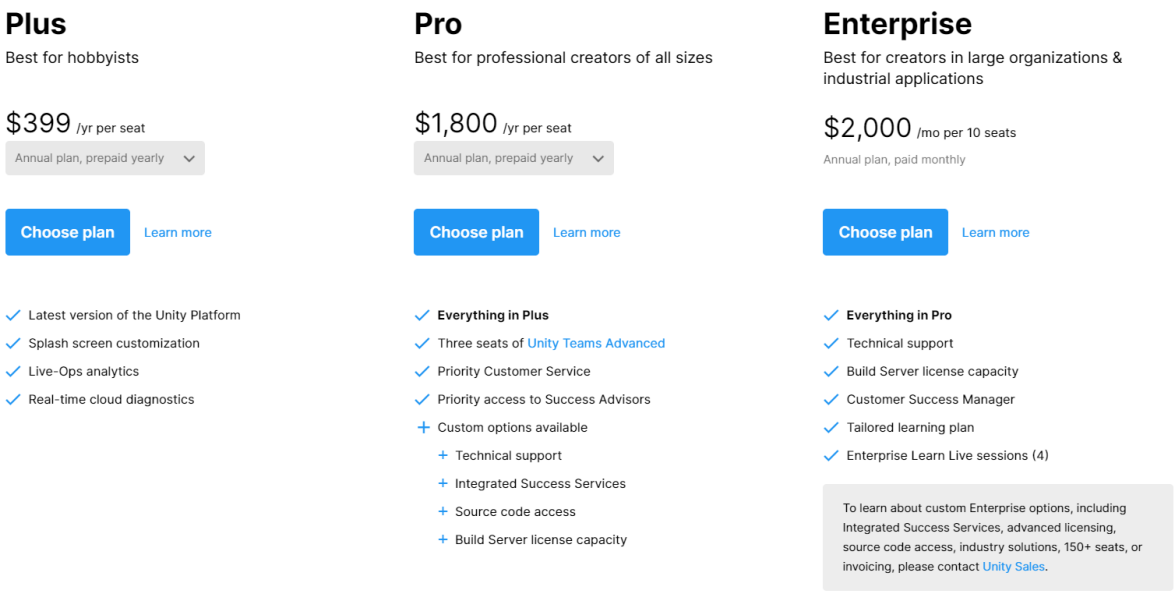
Možnosti platby za využívání Unity jsou rozděleny podle toho, jakou verzi uživatel využívá. Dostupné varianty jsou Personal, Plus, Pro a Enterprise. Verze se volí dle obratu firmy za rok:

- Personal – obrat za rok < 100 000$
- Plus – obrat za rok < 200 000$
- Pro – obrat za rok > 200 000$
- Enterprise – obrat za rok > 200 000$ – k této verzi jsou dodávána další zvýhodnění

Pro využívání Unity3D musí uživatel platit částky zmíněné níže:

- Personal – Free
- Plus - 40$/ měsíc
- Pro – 150$/ měsíc
- Enterprise – 2000/měsíc

Na Obr. 5-3 jsou vidět jednotlivé placené licence Unity. Následně také podmínky a cena, jež jsou zásadní pro jejich zakoupení. [19]



Obr. 5-3 Možnosti platby za SW Unity 3D [19]

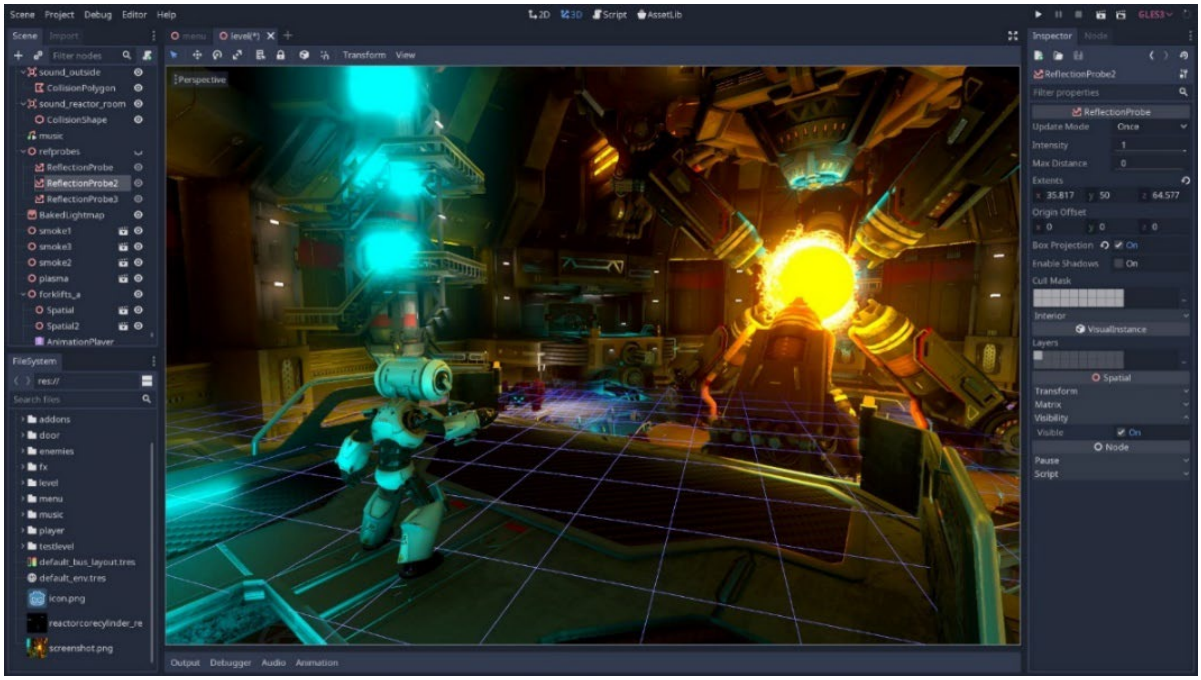
5.2.2 Godot Engine

Godot, jehož prostředí je možné vidět na Obr. 5-4, je 2D i 3D multiplatformní open source herní engine pod licencí MIT (Massachusetts Institute of Technology). Vývoj tohoto SW začal v roce 2007 a v únoru 2014 byl poprvé uvolněn pro veřejnost. Vývojové prostředí funguje na platformách na Windows, Mac OS a Linux (oba 32 a 64 bit) a může vytvářet hry cílené na PC, konzole, mobily a weby. Godot je členem Software Freedom Conservancy, což je nezisková charita, která pomáhá propagovat, vylepšovat, rozvíjet a bránit projekty svobodného a otevřeného softwaru (FLOSS). Godot Engine je bezplatný a otevřený software vydaný na základě permisivní licence MIT. Tato licence poskytuje uživatelům řadu svobod:

- Libovolné používání
- Základní funkce Godot Engine a změna jeho nastavení
- Distribuce různých aplikací vytvořených v Godot Engine, lze i komerčně a pod jinou licencí v rámci podmínek Godot Engine

Jediným omezením je, že se musí sdílet a šířit oznámení o autorských právech a licenční prohlášení programu Godot Engine, v momentu distribuce projektu. Lze využít jinou licenci, ale je důležité v dokumentaci zmínit, z které licence tento produkt pochází. Jako programovací jazyk se využívá buď C# nebo vlastní skriptovací jazyk GD Script. Jedná se o vysokoúrovňový a dynamicky programovací jazyk, který je podobný Pythonu. Rozdíl je v tom, že GD Script má striktní deklarování proměnných a optimalizovanou architekturu na bázi scén. Godot je dodáván s editorem kódu, která nabízí možnost automatického odsazování, zvýraznění syntaxe a také nápovědy. Je také vybaven ladicí funkcí s možností nastavit animační program. Godot má také svůj vlastní vestavěný fyzikální engine pro 2D i 3D mód, který podporuje detekci

kolizí, tuhá tělesa, statické tělo, postavy, vozidla, raycasting[6] a klouby. V současné době mezi podporované platformy patří Windows, OS X, Linux, FreeBSD, Android, iOS, BlackBerry 10, HTML5 a další. K dispozici je také podpora ve vývoji pro Runtime Windows. [20]



Obr. 5-4 Prostředí Godot Engine [21]

5.2.3 Unreal Engine

Jedná se původně o herní engine, který byl vytvořen firmou Epic Games. Jeho první verze z roku 1998 byla použita ve hře Unreal. Od té doby byl Unreal Engine několikrát vylepšen a doplněn, aby mohl být použit v několika desítkách novějších herních titulů. Jádro Unreal Enginu, napsané v programovacím jazyku C++, podporuje mnoho různých platforem, jako jsou Microsoft Windows, Linux, Mac OS, Mac OS X na PC. Díky hernímu zaměření a kladeným nárokům na grafickou kvalitu a podobnost se skutečností, se jedná o editor, který obsahuje fotorealistické vykreslování a dokáže provádět velmi reálnou fyziku na daných 3D modelech (Obr. 5-5). Unreal se skládá z několika komponent, které společně fungují při řízení hry a zajištují vysokou kvalitu z několika pohledů. Součástí je zvukový, fyzikální a grafický engine spolu s možností online modulu. Mezi jednotlivými moduly jsou vytvořené algoritmy, které zajištují fungování a komptabilitu s VR. Unreal Engine je zdarma ke stažení. K platbě dochází, jakmile aplikace začne vydělávat peníze. Uživatel zaplatí 5 % z hrubého příjmu, pokud výdělek za čtvrtletí přesáhne 3000 dolarů. [22, 23]

[6] Technika vykreslování používaná v počítačové grafice a výpočetní geometrii. Dokáže vytvořit trojrozměrnou perspektivu ve dvourozměrné mapě.

Obr. 5-5 Prostředí Unreal Engine [24]

5.2.4 Simlab Soft

Společnost Simlab Soft byla založena v roce 2007 a soustředí se na vývoj 3D SW produktů, které se dají využít v průmyslu. Mezi jednu z jejich největší předností patří široká škála 3D importů a exportů používaných v mnoha komerčních produktech a vývoji. Společnost spolupracuje s několika významnými distributory CAD programů, které se využívají pro vytváření 3D modelů v průmyslové praxi. Díky této spolupráci se engine Simlab Composer stává velmi zajímavým pro používání s průmyslovými 3D modely. [25]

Simlab Composer je kompletní, snadný a na funkce bohatě řešený SW, který umožní vytvářet 3D prostředí pro virtuální realitu. Obsahuje všechny nástroje, které jsou potřeba k importu modelů, vytváření dynamické vizualizace, vykreslování, vytváření jednoduchých scén ve VR s následnou interakcí. Co se týká výstupu, podporuje Simlab platformy Windows a Mac OS, kam lze uložit danou scénu v několika formátech (např. 3D PDF, FBX, DWG, 3D MAX a další), a také mobilní zařízení. Z hlediska dostupnosti nabízí distributor celkem 5 typů SW, které se liší dle nabízených funkcí a ceny (Tab. 5-1). Společnost nabízí dvě ceny, kde první se týká pronájmu licence na rok a druhá trvalého zakoupení SW. Pro větší přehled jsou znázorněny v. [25–27]

Typ SW	Cena/rok	Cena	Funkce
LITE	Zdarma	Zdarma	Vytvoření scény, vykreslování
PRO	79$	199$	Import/Export, 3D PDF, Animace, vytváření textur, sdílení v cloudu
VR	199$	499$	Vytváření VR, Interaktivní trénink, VR 360°
MECHANICAL	299$	799$	Import/Export+, CAD VR, Mechanické simulace
ULTIMATE	599$	1499$	Podpora souborů JT, Katalogy VR, Automatizace (Kódování a vizuální skriptování)

Tab. 5-1 Přehled licencí Simlab [27]

Z hlediska splnění cílů tohoto projektu by bylo potřeba zakoupit minimálně licenci VR, která nabízí vytvoření VR prostředí pro hardware jako jsou HTC Vive, Oculus Quest, Desktop nebo mobilní zařízení (Obr. 5-6). Modul pro interaktivní trénink by mohl posloužit pro potřebnou interakci s objekty.

Obr. 5-6 Prostředí Simlab Composer [28]

6. SW porovnání

Výběr optimálního SW byl proveden podle zvolených kritérií. Jedná se o nadefinované podmínky, které pomohou vybrat optimální SW pro následnou tvorbu interaktivního modelu. Hodnocení jednotlivých kritérií bylo zvoleno pomocí bodové škály 1-5, kde 1 bod značí z pohledu daného kritéria nejhorší variantu, 5 nejlepší. Nejvyšší bodový součet signalizuje software, který je nejvíce vhodný pro tvorbu interaktivního digitálního dvojčete. Pro správný výběr kritérií měla vliv i získaná data, kterými jsou vytvořené výrobní úseky Fischertechnik z kapitoly 4. Zvolená kritéria jsou:

- Cena
- Dostupnost assetů
- Zkušenost autora
- Podpora komunity
- Programovací jazyk

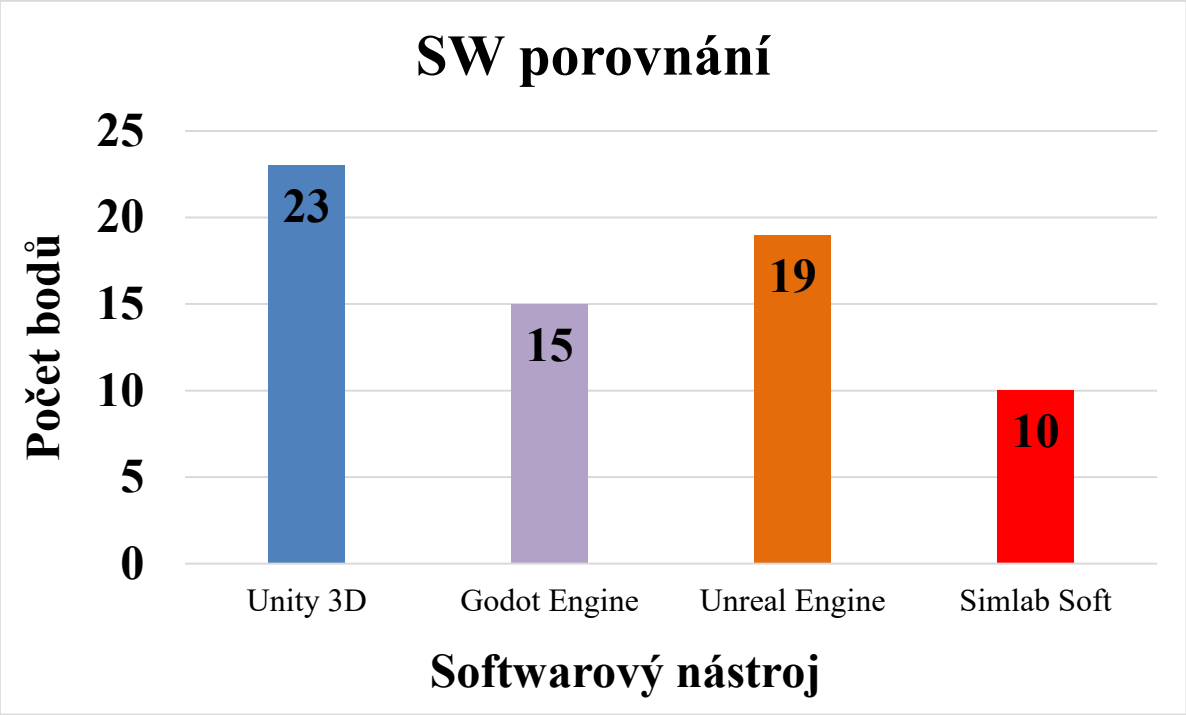
Z pohledu tohoto projektu se jedná o nově používaný nástroj (digitální dvojče ve virtuální realitě), který je třeba otestovat. Proto byla jako první kritérium zvolena cena – při bodovém hodnocení je nejvyšší bodový stav udělován těm softwarům, které nabízejí osobní licenci zdarma. To nesplňuje pouze software Simlab Soft, u kterého je nutné o osobní licenci žádat a čekat na validaci a na licenční klíč. Dalším kritériem je dostupnost assetů. Assety obsahují již předdefinované třídy, objekty, skripty a další, jež slouží ke zjednodušení a urychlení tvorby výsledných aplikací. Unity dostalo nejvíce bodů z toho důvodu, že je možné stáhnout asset *Serial Port Utility Pro*, který významně usnadní napojení na sériový port a jeho následné čtení dat. Zkušenost autora vychází převážně ze zkušenosti s prostředím jednotlivých softwarů. Kritériem podpora komunity se rozumí podpora ze strany ostatních uživatelů, kteří v daném softwaru narazili na podobné problémy, které mohou nastat při tvorbě digitálního dvojčete, a nabízejí jeho řešení. Posledním kritériem je programovací jazyk, jehož bodové ohodnocení reflektuje zkušenost autora s odpovídající syntaxí.

SOFTWARE → KRITÉRIA ↓	Unity 3D	Godot Engine	Unreal Engine	Simlab Soft
Cena	5	5	5	3
Dostupnost assetů	4	2	3	1
Zkušenost autora	5	1	4	3
Podpora komunity	4	2	4	2
Programovací jazyk	5	5	3	1
Σ	**23**	**15**	**19**	**10**

Tab. 6-1 Bodové hodnocení SW variant dle zvolených kritérií

Tab. 6-1 reprezentuje bodové ohodnocení vybraných softwarových variant. Každá varianta obsahuje bodové ohodnocení dle výše zmíněné bodové škály 1-5. Kritéria byla zkoumána

v rámci splnění cílů této práce a byla bodově hodnocena jak z hlediska reálných informací, daných od výrobců softwaru, tak z pohledu zkušeností autora. Z toho důvodu je bodové hodnocení spíše subjektivní. Pro lepší vizualizaci byl vytvořen sloupcový graf Graf 6-1, který znázorňuje, jak si mezi sebou jednotlivé SW vedly.



Graf 6-1 - Porovnání SW z pohledu kritérií

Z grafu je vidět, že nejlépe vychází varianta Unity 3D s celkovým ziskem 23 bodů. Druhým možným softwarem pro tvorbu digitálního dvojčete je program Unreal Engine, který se dostal na hodnotu 19. Pro Unity ovšem rozhodla lepší zkušenost autora s prostředím a programovací jazyk C#, který je oproti C++ (používaný Unreal Enginem) autorovi více znám. Godot Engine sice podporuje oba zmiňované programovací jazyky, ale ztratil díky tomu, že autor nemá zkušenost s tímto prostředím a hrozilo by prodloužení tvorby finální aplikace. Simlab Soft oproti ostatním ztrácí hlavně na dostupnosti assetů, jelikož jeho knihovna není tak obsáhlá, a na podpoře programovacího jazyka (nepodporuje C# ani C++).

7. Popis tvorby interaktivních digitálních dvojčat

V kapitole 5.1 a 5.2 byly vybrány softwarové nástroje, které se použijí pro tvorbu interaktivního digitálního dvojčete, tedy pro výstup této práce. Tato kapitola obsahuje podrobný popis modelování jednotlivých dílů výrobního úseku v Blenderu a následný vývoj aplikace v programu Unity 3D včetně její komunikace s reálným modelem, který byl vytvořen ze stavebnice Fischertechnik.

7.1 Modelování v softwaru Blender

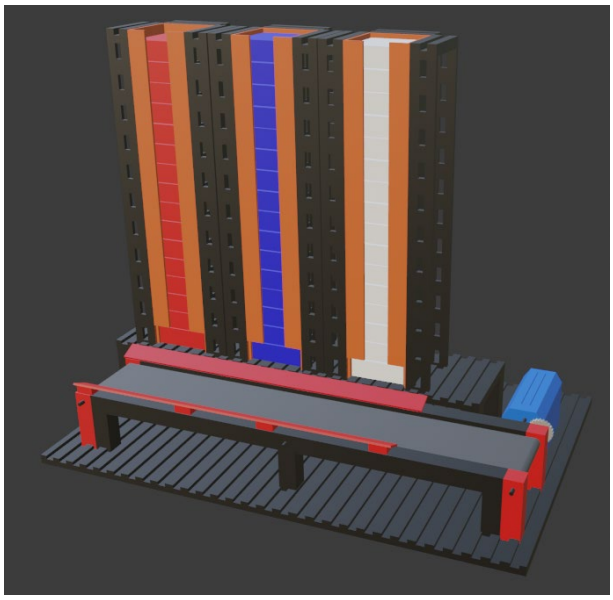
Jak je již zmíněno výše, pro plynulejší chod aplikace je nutné vymodelovat jednotlivé části výrobního úseku tak, aby byl zajištěn co nejplynulejší chod aplikace ve virtuální realitě podporující platformu Android. Z toho důvodu jsou úseky vymodelovány ze základních geometrických útvarů a poskládány po částech do větších funkčních celků. Dále jsou modely zjednodušeny vynecháním veškerých kabelových svazků, které v reálném modelu slouží k propojení jednotlivých pracovišť výrobní linky. Kvůli animování[7], které je nezbytnou součástí tvorby digitálního dvojčete, je nutné u každé části výrobního úseku nadefinovat ty prvky, které budou simulovat pohyb reálného modelu. Části jsou z důvodu jednodušší pozdější manipulace v programu Unity 3D rozděleny na 6 úseků:

Senzorová stanice – Obecně zajišťuje komunikaci s ostatními částmi výrobního úseku zobrazuje data, jako je hladina otřesů na pracovišti, úroveň hluku, vlhkost a teplota vzduchu. Jelikož reálný model této stanice je sice zakomponován do výrobního úseku, ale nezobrazuje žádná data, bylo v 3D modelu ubráno několik tlačítek, které na tvorbu digitálního dvojčete nemají žádný vliv. Pro možnost budoucího napojení na reálná data byl ponechán panel uprostřed modelu (viz Obr. 7-1).

[7] Vytváření zdánlivě se pohybujících věcí ve vybraném softwaru.

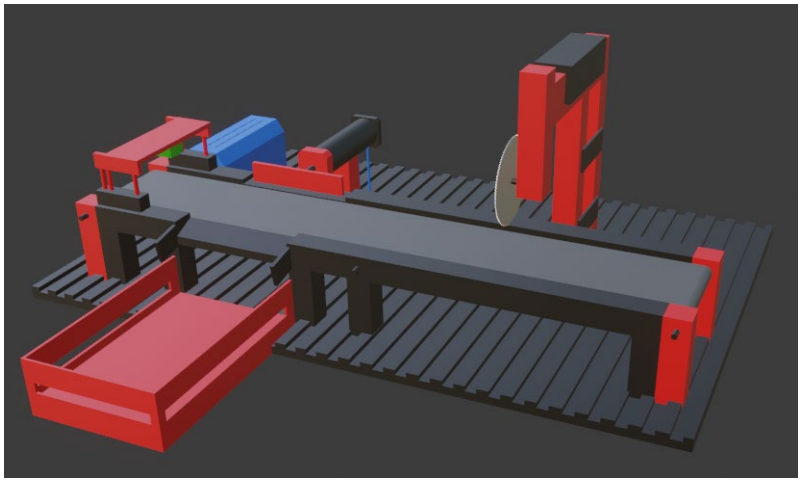
Obr. 7-1 Senzorová stanice – 3D model

Sklad polotovarů – Sklad obsahující tři druhy polotovarů (pro zjednodušení jsou použity stejné geometrické tvary odlišené barvou – červená, modrá, bílá) v regálech, ze kterých je poté materiál vyskladňován na pásový dopravník, který je také součástí tohoto 3D modelu (viz Obr. 7-2). Pro pozdější animaci je nutné rozdělit polotovary na dílčí části, aby se mohly po pásovém dopravníku pohybovat jednotlivě.



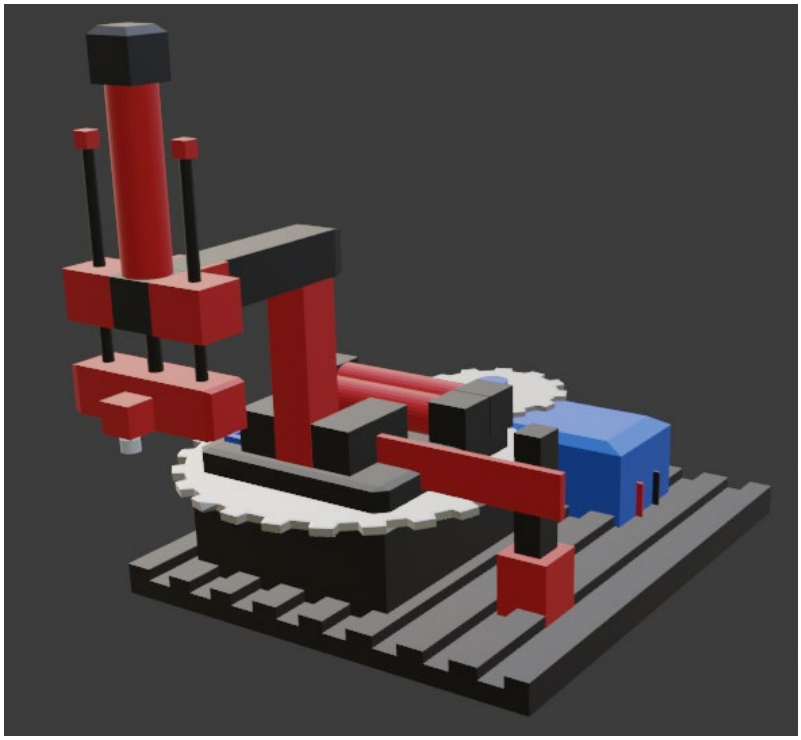
Obr. 7-2 Sklad polotovarů – 3D model

Pracoviště s laserovým senzorem a pilou – V tomto úseku (Obr. 7-3) se nejprve polotovar dostane pod laserový senzor, který na základě vad v materiálu rozhodne, zda se jedná o zmetek, nebo může polotovar pokračovat po pásovém dopravníku dále. Pokud senzor vyhodnotí polotovar jako zmetek, odsune ho do paletky určené pro vadné kusy. V opačném případě putuje na pracoviště pily, kde je simulováno nařezání na menší části a obrobek pokračuje dále k rotačnímu rameni. Pro animaci je nezbytné oddělit rameno s posuvníkem, které odsouvá zmetky do paletky a dále kotouč pily.



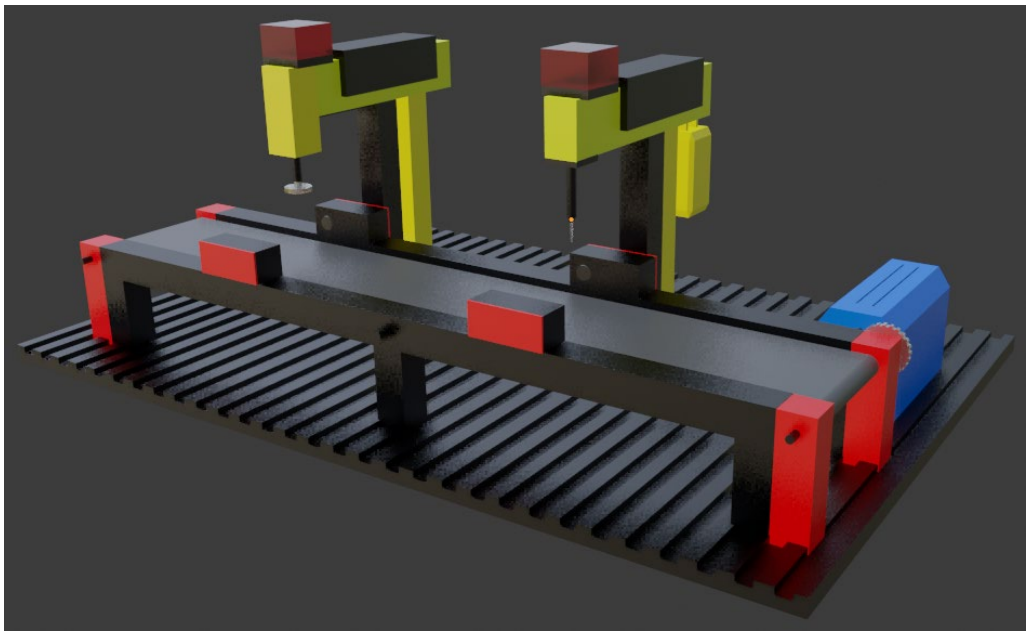
Obr. 7-3 Pracoviště s laserovým senzorem a pilou – 3D model

Rotační rameno – Rameno sloužící pro přesun opracovaného polotovaru mezi dvěma výrobními úseky, a to mezi pracovištěm pily a frézkou. Pro zanimování pohybu ramene jsou rozděleny všechny jeho části, které vystupují z většího ozubeného kola (viz Obr. 7-4).



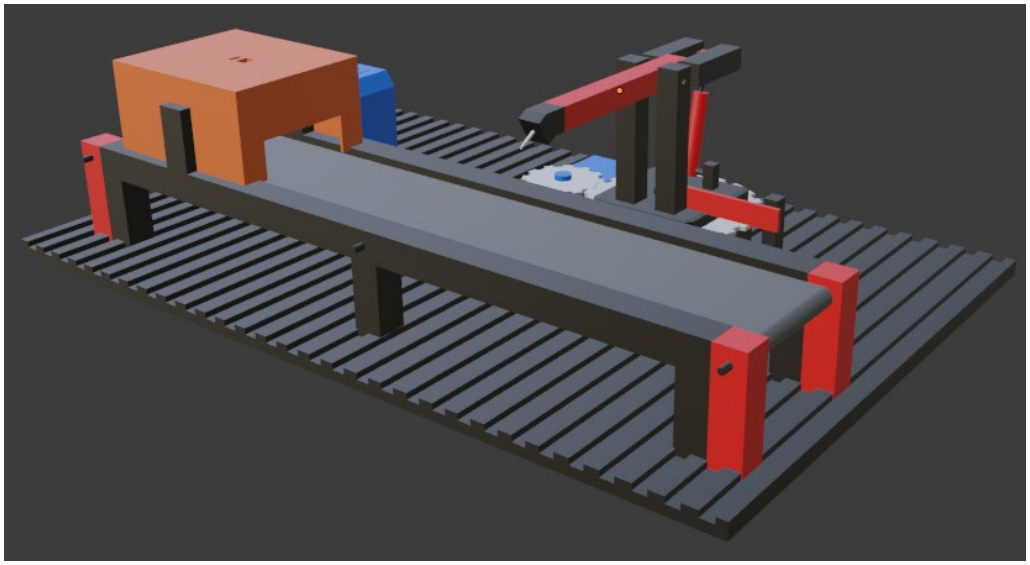
Obr. 7-4 Rotační rameno – 3D model

Frézka a vrtačka – Zde (Obr. 7-5) se polotovar dostává po pásovém dopravníku nejprve na pracoviště frézky, kde je obroben a dále putuje k vrtačce. Obě tyto pracoviště mají na svých stanovištích světelné senzory pro znázornění poruchy na zařízení. V této části je pro pozdější rozpohybování odděleno jak rameno s frézou, tak s vrtačkou.



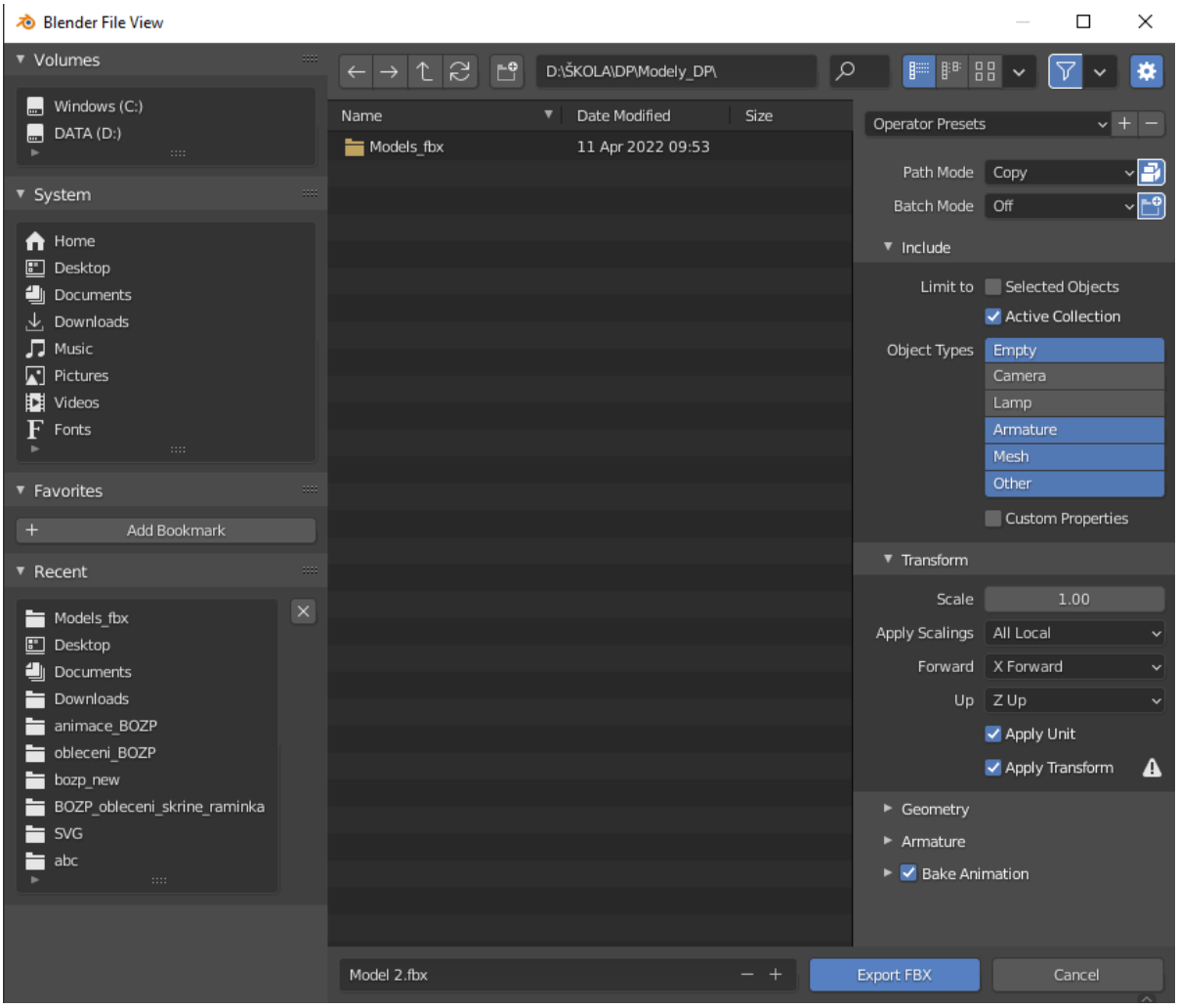
Obr. 7-5 Frézka a vrtačka – 3D model

RGB senzor a svařovací stanice – V posledním výrobním úseku se obrobek dostává do RGB senzoru, ve kterém je díky prostupnosti daného materiálu zjištěno, jaký typ obrobku se zpracovává a s touto informací putuje dále do svařovací stanice, kde probíhá finální obrobení součásti a její následný přesun do palety pro hotové výrobky (není součástí 3D modelu). Zde je pro provedení animace odděleno pouze rameno svařovací stanice, které rotuje pouze kolem osy y (Obr. 7-6).



Obr. 7-6 RGB senzor a svařovací stanice – 3D model

Po vymodelování každé části jsou výrobní úseky otexturovány – každé části je přiřazen určitý typ materiálu. Dalším krokem je vyexportování hotového modelu spolu s texturami ve formátu *.fbx*, který je nejvhodnější při importu do softwaru Unity 3D. Během exportu je nutné nastavit několik parametrů, aby modely obsahovaly vše potřebné, jako jsou meshe a textury. Pro zachování textur se v kolonce *Path Mode* nastaví hodnota na *Copy*. Dále se v kolonce *Object Types* označí vše kromě hodnot typu *Camera* a *Lamp* (tyto objekty budou nastavovány až v rámci prostředí Unity 3D). Posledním krokem je kontrola hodnoty *Transform* (ideální nastavení na hodnotu 1), zkontrolování nastavení os *Forward* a *Up* (závislé na typu modelování) a zaškrtnutí možností *Apply Unit* a *Apply Transform* (viz Obr. 7-7).



Obr. 7-7 Nastavení exportu v SW Blender

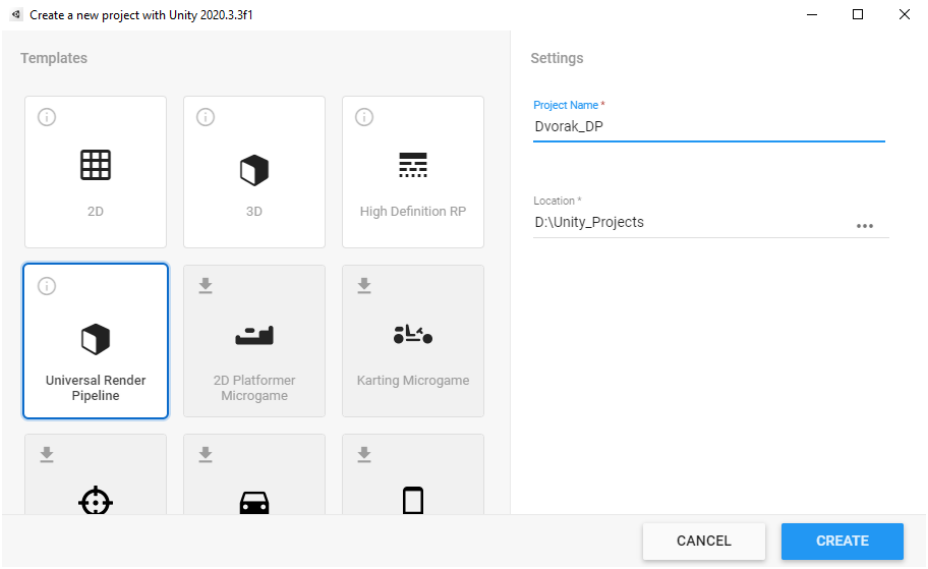
Po kliknutí na *Export FBX* jsou již všechny modely připraveny pro nahrání do SW Unity 3D. Modely mají v sobě zakomponované textury v podobě materiálů a jsou rozděleny tak, aby se s nimi mohlo ve VR interagovat a daly se co nejvěrohodněji rozpohybovat.

7.2 Tvorba interaktivního digitálního dvojčete v prostředí Unity 3D

Unity 3D je multiplatformní herní engine sloužící pro tvorbu her a aplikací pro PC, konzole, mobily a web. Pro tvorbu interaktivního digitálního dvojčete je nejprve prostředí vytvářeno pro platformu Windows, kde probíhá veškerý import modelů, tvorba interakcí a animací, skriptování v jazyce C# a další. Poslední fází je transformace aplikace na žádanou platformu Android a nastavení projektu z hlediska optimalizace výkonu.

7.2.1 Založení a základní nastavení projektu

Prvním krokem je založení projektu přes samostatnou aplikaci Unity Hub, která umožňuje lepší správu nad projekty vyvíjené SW Unity 3D. Projekt je zakládán ve verzi 2020.3.3f. Tato verze má oproti ostatním výhodu v tom, že je společností Unity Technologies vedena jako verze s dlouhodobou podporou aktualizací a údržbou. Díky tomu lze projekt udržovat na této verzi delší dobu bez nutnosti upgradu na verzi vyšší. Projekt je následně vytvořen v šabloně URP (viz Obr. 7-8).



Obr. 7-8 Unity – založení URP projektu

URP je předpřipravená skriptovatelná vykreslovací pipeline, kterou lze rychle a snadno přizpůsobit a která umožňuje vytvářet optimalizovanou grafiku pro širokou škálu platforem. Tato šablona obsahuje ukázkovou scénu, která obsahuje příklady konfigurace nastavení osvětlení, materiálů, shaderů[8] a efektů následného zpracování v URP. Dále obsahuje několik před konfigurovaných prostředků URP, které umožňují rychle přepínat mezi úrovněmi kvality grafiky, a předvolby, které byly optimalizovány pro použití na jakékoliv platformě.

Ve vytvořené *SampleScene* se smažou všechny objekty kromě objektu *MainCamera*, který je prozatím ponechán pro verzi na platformu Windows. Do okna *Project* jsou naimportovány všechny vymodelované výrobní úseky stavebnice Fischertechnik (formát *.fbx*) spolu s modelem

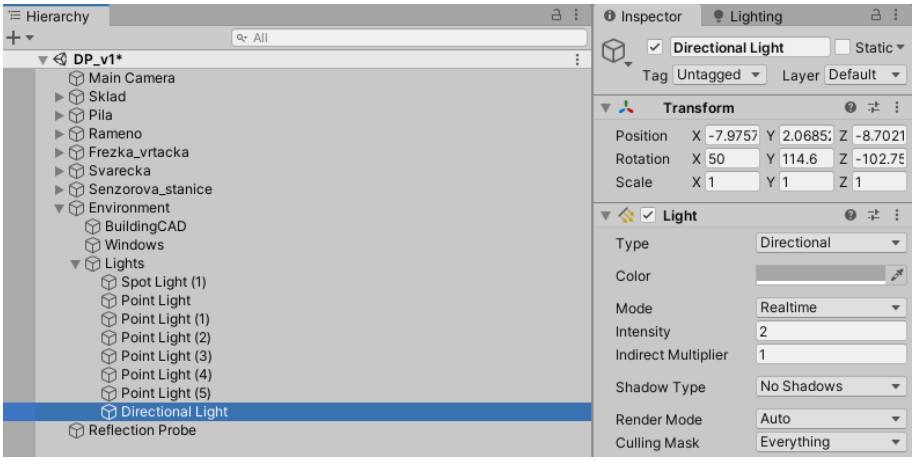
[8] Počítačový program sloužící k řízení jednotlivých částí programovatelného grafického řetězce grafické karty.

Building, což je model prostředí, které bylo autorem této práce vytvořeno již pro dřívější projekty.

Aby mohl digitální model kopírovat reálné chování výrobní linky, je zapotřebí modely rozestavit dle skutečného modelu. Prostředí spolu s připravenou výrobní linkou je vidět na Obr. 7-9.

Obr. 7-9 Prostředí s výrobní linkou

Prostředí *Building* již obsahuje zapečená světla[9] v podobě jednoho reflektoru *Spot Light* a šesti bodových světel formou *Point Light*. Pro lepší design celého prostředí je ve scéně obsažen jeden objekt typu *Reflection Probe,* což je odrazová sonda sloužící reflexním materiálům (v tomto případě oknům) k lepšímu vykreslování odrazu světla. Všechny zmíněné objekty jsou již uloženy v meshi celého prostředí, avšak nemají vliv na roszestavené modely výrobního úseku. Z toho důvodu je do scény přidáno jedno realtimové světlo *Directional Light*, jehož nastavení je vidět na Obr. 7-10.



Obr. 7-10 Nastavení *Directional Light*

[9] Vypočtení osvětlení statické geometrie a následné uložení tohoto výpočtu do textur materiálů

7.2.2 Propojení digitálního dvojčete s reálným modelem

Tato kapitola popisuje napojení na reálný model. Stavebnice Fischertechnik, která slouží jako vzor pro simulaci vymodelovaného prostředí, je propojena s PC pomocí USB. Po jejím spuštění začne díky USB propojení vypisovat očekávaná data na COM port počítače. Tato data jsou ve formě bytů. Tabulka Tab. 7-1 Tabulka akcí jednotlivých výrobních úseků, zpracovaná Ing. Bc. Miroslavem Malagou, znázorňuje rozdělení výrobní linky na jednotlivé pracoviště spolu s tím, jaké datové hodnoty jsou posílány na COM port v závislosti na provedené akce reálného modelu.

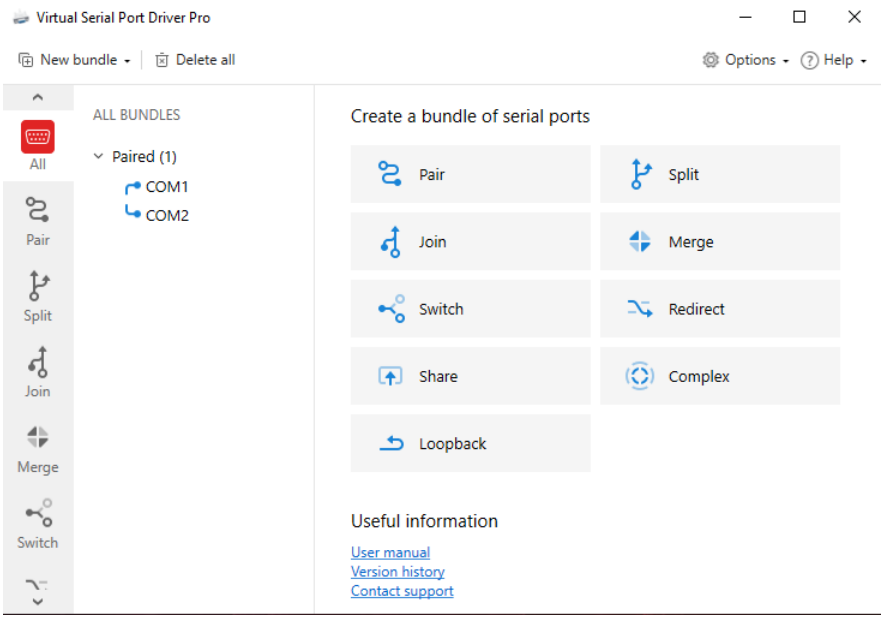
VÝROBEK		
Typ	**Kód**	**Textový popisek**
Bily	1	Bílý obrobek
Cerveny	2	Červený obrobek
Modry	3	Modrý obrobek
Zmetek	4	Obrobek je vyhodnocený jako zmetek
NeniZmetek	5	Obrobek není vyhodnocený jako zmetek
SKLAD		
Akce	**Kód**	**Textový popisek**
Slot1VyskladneniStart	10	Začátek operace vyskladnění materiálu ze skladu, slotu č. 1. Zahájení pohybu pístu pro vyskladnění.
Slot1Vyskladneni	11	Vyskladnění materiálu ze skladu, slotu č. 1 na dopravníkový pás. Dosažení krajní vyskladňovací polohy pístu.
Slot1VyskladneniKonec	12	Ukončení operace vyskladnění materiálu ze skladu, slotu č. 1.
Slot2VyskladneniStart	13	Začátek operace vyskladnění materiálu ze skladu, slotu č. 2. Zahájení pohybu pístu pro vyskladnění.
Slot2Vyskladneni	14	Vyskladnění materiálu ze skladu, slotu č. 2 na dopravníkový pás. Dosažení krajní vyskladňovací polohy pístu.
Slot2VyskladneniKonec	15	Ukončení operace vyskladnění materiálu ze skladu, slotu č. 2.
Slot3VyskladneniStart	16	Začátek operace vyskladnění materiálu ze skladu, slotu č. 3. Impuls pro zahájení pohybu pístu pro vyskladnění.
Slot3Vyskladneni	17	Vyskladnění materiálu ze skladu, slotu č. 3 na dopravníkový pás. Dosažení krajní vyskladňovací polohy pístu.
Slot3VyskladneniKonec	18	Ukončení operace vyskladnění materiálu ze skladu, slotu č. 3.
VsechnySlotyPrazdne	19	Informace, že všechny sloty skladu jsou prázdné.
KONTROLA		
Akce	**Kód**	**Textový popisek**
RentgenovaKontrolaStart	30	Zahájení rentgenové kontroly obrobku.

MaterialJeZmetek	4	Informace, že materiál je zmetek.
MaterialNeniZmetek	5	Informace, že materiál není zmetek.
RentgenovaKontrolaKonec	31	Ukončení rentgenové kontroly výrobku.
ZmetekVyrazovaniStart	32	Zahájení fyzického vyřazení obrobku ze systému, protože je zmetek.
ZmetekVyrazovaniKonec	33	Ukončení fyzického vyřazení obrobku ze systému, protože je zmetek.
RezaniStart	34	Zahájení operace řezání.
RezaniKonec	35	Ukončení operace řezání.
MANIPULAČNÍ ROBOT		
Akce	**Kód**	**Textový popisek**
ManipulacníRobotStart	51	Zahájení práce manipulačního robota.
UchopeníVyrobku	52	Uchopení obrobku manipulačním robotem.
UmisteniVyrobku	53	Umístění obrobku na cílovou pozici robotem.
ManipulacniRobotKonec	54	Ukončení práce manipulačního robota.
OBRÁBĚNÍ		
Akce	**Kód**	**Textový popisek**
FrezovaniOpravaStart	71	Výskyt chyby na frézce, kterou je potřeba odstranit uživatelem.
FrezovaniOpravaKonec	72	Ukončení opravy, resp. odstranění chyby na frézce uživatelem.
FrezovaniStart	73	Zahájení operace frézování.
FrezovaniKonec	74	Ukončení operace frézování.
VrtaniOpravaStart	75	Výskyt chyby na vrtačce, kterou je potřeba odstranit uživatelem.
VrtaniOpravaKonec	76	Ukončení opravy, resp. odstranění chyby na vrtačce uživatelem.
VrtaniStart	77	Zahájení operace vrtání.
VrtaniKonec	78	Ukončení operace vrtání.
SVAŘOVÁNÍ		
Akce	**Kód**	**Textový popisek**
PecStart	90	Začátek ohřevu obrobku.
BilyVyrobek	1	Informace, že obrobek je bílý, resp. že je určený pro bílý finální výrobek.
CervenyVyrobek	2	Informace, že obrobek je červený, resp. že je určený pro červený finální výrobek.
ModryVyrobek	3	Informace, že obrobek je modrý, resp. že je určený pro modrý finální výrobek.
PecKonec	91	Ukončení ohřevu obrobku.
SvarovaniStart	92	Zahájení operace svařování.
SvarovaniKonec	93	Ukončení operace svařování.
VyrazeniHotovehoVyrobkuZeSystemu	100	Vyřazení hotového výrobku z výrobního systému.

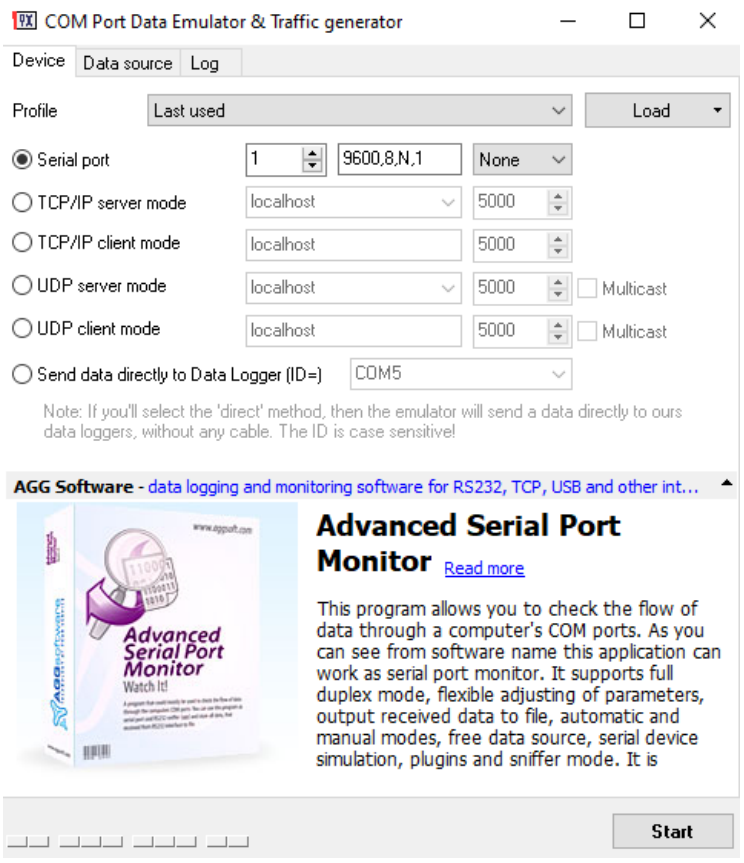
Tab. 7-1 Tabulka akcí jednotlivých výrobních úseků

7.2.3 Virtuální propojení COM portů

Pro testovací účely interaktivního digitálního dvojčete jsou staženy a nainstalovány programy Virtual Seriál Port Driver Pro (Obr. 7-11) a COM Port Data Emulator & Traffic generator (Obr. 7-12).



Obr. 7-11 Virtual Seriál Port Driver Pro



Obr. 7-12 COM Port Data Emulator & Traffic generator

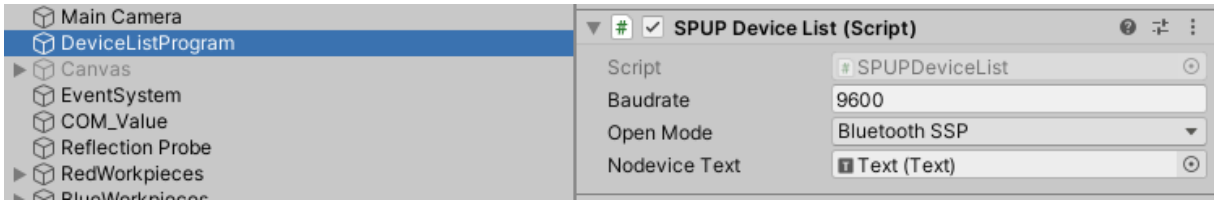
Prvním z nich je vytvořeno připojení mezi virtuálními COM porty *COM1* a *COM2*. Druhý software, COM Port Data Emulator & Traffic generator, poté umožní posílání předem nadefinovaných dat na zvolený port. Pro urychlení vývoje finální aplikace jsou proto posílány celočíselné datové typy na COM1, který tento výstup zrcadlí na COM2. Ten se v softwaru Unity 3D použije pro načítání celých čísel (integer) a zobrazování daných stavů digitálního dvojčete tak, aby věrohodně kopírovalo stav reálného modelu.

7.2.4 Napojení na COM port v SW Unity 3D

Pro napojení na COM2, jehož vytvoření je zmíněno v kapitole 7.2.3, je do softwaru Unity 3D naimportován předdefinovaný asset[10] Serial Port Utility Pro, mezi jehož hlavní funkce patří:

- Snadná implementace komunikace mezi počítači v Unity prostřednictvím sériových portů.
- Implementace komunikace mezi počítačem a mikro kontrolerem (Arduino, Ftdi, Microchip, Cypress, Silicon Labs atd) prostřednictvím sériových portů.
- Multiplatformní použití s operačními systémy Windows, Mac a Android.
- Funkci řízenou událostmi pro příjem dat.
- Detekce chyby při fyzickém odpojení zařízení.
- Podpora fyzického rozhraní USB, PCI a další.
- Podpora Bluetooth SPP (Virtuální COM Port).
- Podpora emulátoru sériového portu TCP (režim serveru a režim klienta). [30]

Do scény je v dalším kroku vložen objekt *DeviceListProgram,* který již obsahuje komponentu ve formě skriptu s názvem *SPUP Device List.* Důležitým nastavením pro čtení z virtuálního COM portu je nastavit položku *Baudrate* na stejnou hodnotu, jako je nastavena u COM1 na Obr. 7-13, tedy 9600. Jelikož jsou pro testovací účely COM porty nastaveny jako virtuální, hodnota *Open Mode* se nastaví na *Bluetooth SSP*, čímž je docíleno načítání všech dostupných COM portů nacházející se v místním počítači (Obr. 7-14).



Obr. 7-13 Nastavení objektu *DeviceListProgram*

Po spuštění *Play Modu* v Editoru se automaticky načtou a vygenerují všechny COM porty, které jsou v počítači dostupné. Na Obr. 7-14 lze vidět COM porty 1, 2, 3, 4, 15 a 16. Hlavním cílem je napojení na virtuálně vytvořený COM2, který přijímá data z COM1 skrze virtuální propojení.

[10] Element, který můžete použít ve hře nebo projektu. Tento element může pocházet ze souboru vytvořeného mimo Unity, například 3D modelu, zvukového souboru, obrázku nebo jiného typu souboru, který Unity podporuje.

Obr. 7-14 Načtené COM porty

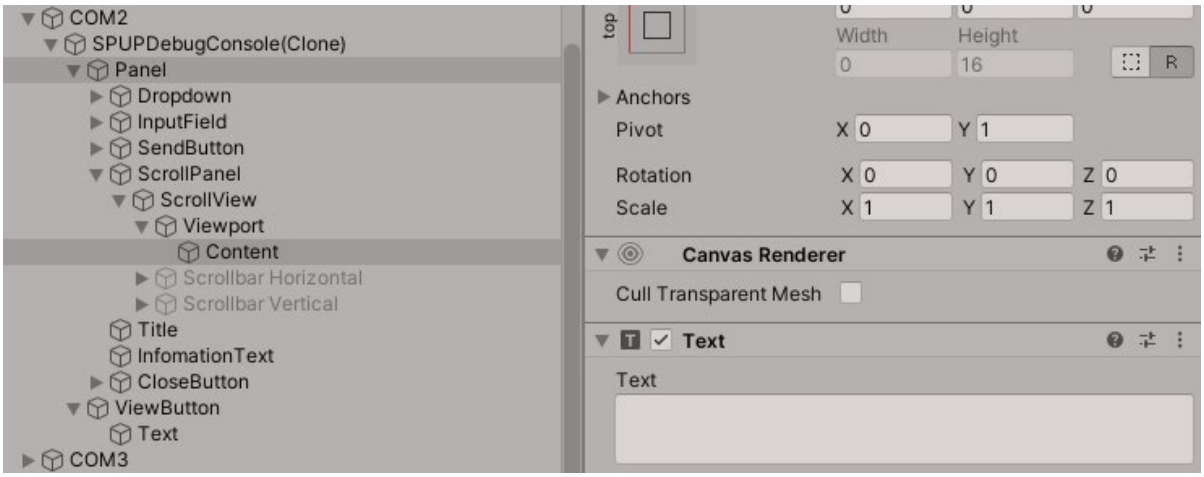
Každému z těchto COM portů je assetem Serial Port Utility Pro přiřazen skript *SerialPort Utility Pro,* jehož kód musí být následně upraven.

```
    public void SerialDebugAddString(string message, bool
send_direction)

        {

            if(message.Length > 128) message =
message.Remove(128); //max 128

            string dir = send_direction ? " (SEND)" : "";

            string debug = message;

            //string debug = string.Format("[{0}{1}] {2}\n",
System.DateTime.Now.ToString("MM/dd HH:mm:ss"), dir, message); //
Odebrání data a času na výpisu

            SerialDebugString = message; // Příjem dat

            if
(SerialDebugString.Split("\n".ToCharArray()).Length >=
SerialDebugStringMAX)

            {

                SerialDebugString =
SerialDebugString.Remove(SerialDebugString.LastIndexOf('\n'));

            }

        }
```

V kódu je zakomentován příkaz *System.DateTime.Now.ToString*, který defaultně vypisuje nejenom přijímaná data na COM portu, ale i datum a čas přijetí, což pro účely této aplikace není potřebné. Místo toho je přidán příkaz *SerialDebugString = message*, který na výstup v Unity 3D posílá úplně stejná data, jaké COM port přijme.

Pod každým objektem portu je vygenerován objekt *SPUPDebugConsole(Clone)*, jenž obsahuje velmi důležitý skript *Debug Console*. Metoda *ViewButtonClick()* totiž otevírá daný COM port a tím i jeho přístup ke čtení dat z počítače. Hierarchicky jsou objektu

SPUPDebugConsole(Clone) přiřazeny podřazené objekty včetně položky *Content,* jejíž *TextBox* vypisuje přijímané hodnoty z COM portu formou textového řetězce *String* (viz Obr. 7-15).



Obr. 7-15 Hierarchie generovaných objektů pro daný COM port

Asset Serial Port Utility Pro je předdefinovaný tak, že u každého COM portu generuje několik objektů typu *Canvas*[11] a *Button*[12], které by díky uživatelskému rozhraní překrývaly modely a prostředí aplikace. Z toho důvodu je vytvořen skript *Get COM Value*, zobrazený níže.

```
public class GetCOMValue : MonoBehaviour

    {

        public string COMPort = "COM2";

        public GameObject OpenedCOMGO;

        public GameObject SpupDebugConsoleOurCOM;

        public DebugConsole debugConsoleScript;

        GameObject[] SpupConsolesGOs;

        Canvas[] SpupConsolesCanvases;

        public GameObject ContentGO;

        private Text contentText;

        public int ReceivedData; // přijatá data na COM portu ve
formátu string


        public int CheckForIntChange;// int, který kontroluje, zda-
li se změnila hodnota a tudíž nastal nový stav
```

[11] Oblast, ve které se nachází všechny prvky uživatelského rozhraní.
[12] Tlačítko uživatelského rozhraní

```
        public GameObject ManagerGO;
        private WorkpieceFlow workPieceFlowScript;


        void Start()
        {
            OpenedCOMGO = GameObject.Find(COMPort);
            Invoke(nameof(GetCOMComponentAndRecievedData), 0.01f);
            CheckForIntChange = ReceivedData;
            workPieceFlowScript =
ManagerGO.GetComponent<WorkpieceFlow>();
        }


        private void FixedUpdate()
        {
            int.TryParse(contentText.text, out ReceivedData);
            if (CheckForIntChange != ReceivedData && (ReceivedData
!= 11 && ReceivedData != 12 && ReceivedData != 14 && ReceivedData !=
15 && ReceivedData != 17 && ReceivedData != 18 && ReceivedData != 19
&& ReceivedData != 33 && ReceivedData != 52 && ReceivedData != 53 &&
ReceivedData != 54))
            {
                CheckForIntChange = ReceivedData;
                workPieceFlowScript.ListOfID.Add(CheckForIntChange);


                if (CheckForIntChange == 10 || CheckForIntChange ==
13 || CheckForIntChange == 16)
                {
                    workPieceFlowScript.PerformAnimationMain();
                }
            }
        }
```

V úvodu skriptu je deklarován jmenný prostor *namespace SerialPortUtility* z důvodu načítání komponent z assetu Serial Port Utility Pro, který si tuto třídu generuje a připojuje se k ní v každém skriptu. Na 5. řádku je nadefinován *public string COMPort*, jehož hodnotu je možné měnit v závislosti na čísle portu, ke kterému se zrovna reálný model připojuje. Tato hodnota se pak kopíruje v metodě *Start()*, tam je do do proměnné *GameObject OpenedCOMGO* načten právě zvolený COM port. Proměnné *ContentGO, contentText* slouží k načtení textového řetězce z objektu *Content* (přijímá data ze zvoleného COM portu) a v metodě *FixedUpdate()* je pomocí funkce *int.TryParse(contentText.text, ouz ReceivedData)* převeden na celočíselný datový typ vypisovaný pomocí proměnné *int ReceivedData*. Protože se však metoda *FixedUpdate()* ve skriptu volá každý frame, je přidána pojišťovací proměnná *int CheckForIntChange*, která se v metodě *Start()* nastaví na stejnou počáteční hodnotu, jako proměnná *int ReceivedData* a přidáním podmínky *if* do metody *FixedUpdate()* kontroluje, v jaký daný moment se změní hodnota *int ReceivedData*. Druhá část podmínky obsahuje celočíselné hodnoty, které nemají vliv na výsledné animace, a proto jsou díky dané podmínce ignorovány. Ve chvíli splnění zmíněné podmínky se hodnota těchto dvou proměnných (*CheckForIntChange* a *ReceivedData*) znovu synchronizuje a výsledná hodnota je přidána jako poslední prvek do generického listu *ListOfID*, který obsahuje komponenta *workPieceFlowScript*. Ta je načítána z *GameObject ManagerGO* a jejím hlavním úkolem je nadefinování a pouštění jednotlivých animačních stavů skrze generický list tak, aby animace proběhla i při časové prodlevě, kterou může způsobit rozdílná rychlost mezi procesy reálného modelu a jeho digitálního dvojčete.

```
        void GetCOMComponentAndRecievedData()

        {

            SpupDebugConsoleOurCOM =
OpenedCOMGO.transform.GetChild(0).gameObject;

            debugConsoleScript =
SpupDebugConsoleOurCOM.GetComponent<DebugConsole>();

            debugConsoleScript.ViewButtonClick();

            SpupConsolesGOs =
FindGameObjectsByName("SPUPDebugConsole(Clone)");

            SpupConsolesCanvases = new
Canvas[SpupConsolesGOs.Length];

            LoadCanvasesAndTurnThemOff();

            ContentGO = GameObject.Find("Content"); // Najdi
Gameobject s názvem Content, který zobrazuje přijímaná data na
vybraném COM portu

            contentText = ContentGO.GetComponent<Text>();

        }
```

```
        GameObject[] FindGameObjectsByName(string name)
        {
            return
System.Array.FindAll((FindObjectsOfType(typeof(GameObject)) as
GameObject[]), p => p.name == name);
        }
        void LoadCanvasesAndTurnThemOff()
        {
            for (int i = 0; i < SpupConsolesGOs.Length; i++)
            {
                SpupConsolesCanvases[i] =
SpupConsolesGOs[i].GetComponent<Canvas>();
                SpupConsolesCanvases[i].enabled = false;
            }
        }
    }
}
```

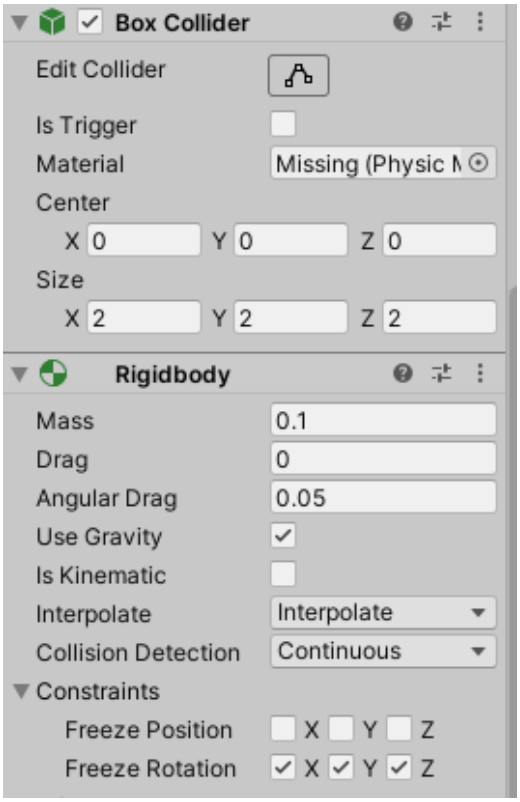
Poslední část skriptu *GetCOMValue* obsahuje metody *GetCOMComponentAndReceivedData(), FindGameObjectsByName()* a *LoadCanvasesAndTurnThemOff()*. Poslední dvě jmenované nejprve načtou do pole *GameObject[]* všechny objekty ze scény, které nesou název *SPUPDebugConsole(Clone)*. Pro každý tento objekt je do druhého generovaného pole *SpupConsolesCanvases[]* načtena komponenta *Canvas*, která je následně deaktivována. Tím je docíleno toho, že se po spuštění aplikace vypne celé uživatelské rozhraní pro všechny COM porty, které by mohly překrývat vytvořenou scénu obsahující digitální dvojče.

Metoda *GetCOMComponentAndRecievedData()* načítá ze zvoleného COM portu podřazený objekt *GameObject SpupDebugConsoleOurCOM*, z něhož je dále příkazy *SpupDebugConsoleOurCOM.GetComponent<DebugConsole>()* a *debugConsoleScript.ViewButtonClick()* načtena již zmiňovaná komponenta *DebugConsole*, na které je volána metoda *ViewButtonClick()* pro otevření COM portu na startu aplikace.

7.2.5 Animace modelů výrobního úseku

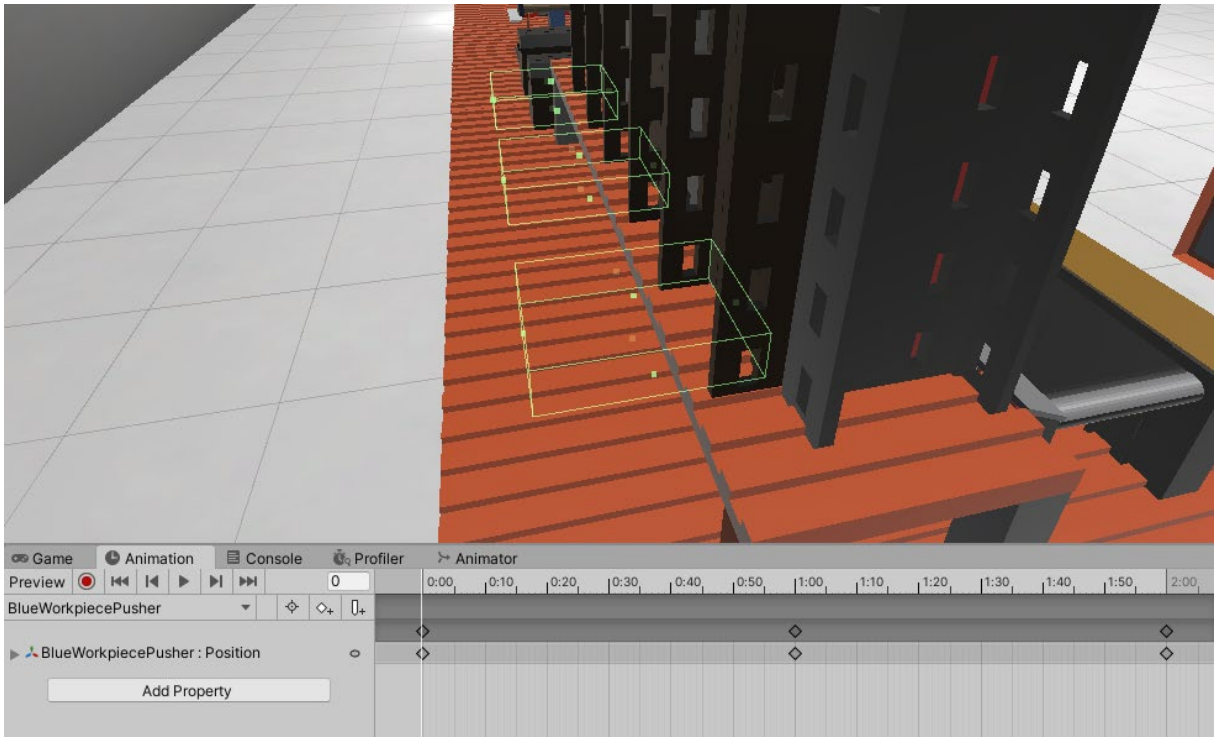
V této části je již dvojče napojeno na reálný model výrobního úseku stavebnice Fischertechnik. Posledním krokem je tedy nastavení objektů polotovarů, částí jednotlivých pracovišť a pásového dopravníku.

Na všechny polotovary, které jsou rozřazeny dle barvy, jsou přidány komponenty *Rigidbody* a *BoxCollider*. Na komponentě *Rigidbody* je přenastavena hodnota *Mass* na 0.1 a zapnuta gravitace zaškrtnutím checkboxu *Use Gravity*. Díky tomu se jednotlivé části budou chovat jako fyzické objekty v reálném světě s vlastní vahou 0.1 kg a budou kolidovat s ostatními komponentami typu *Collider*. Pro vyšší plynulost aplikace a sjednocení animací je dále v kolonce *Interpolate* nastavena hodnota na *Interpolate* a jsou zamčeny rotace objektu ve všech osách (viz Obr. 7-16).



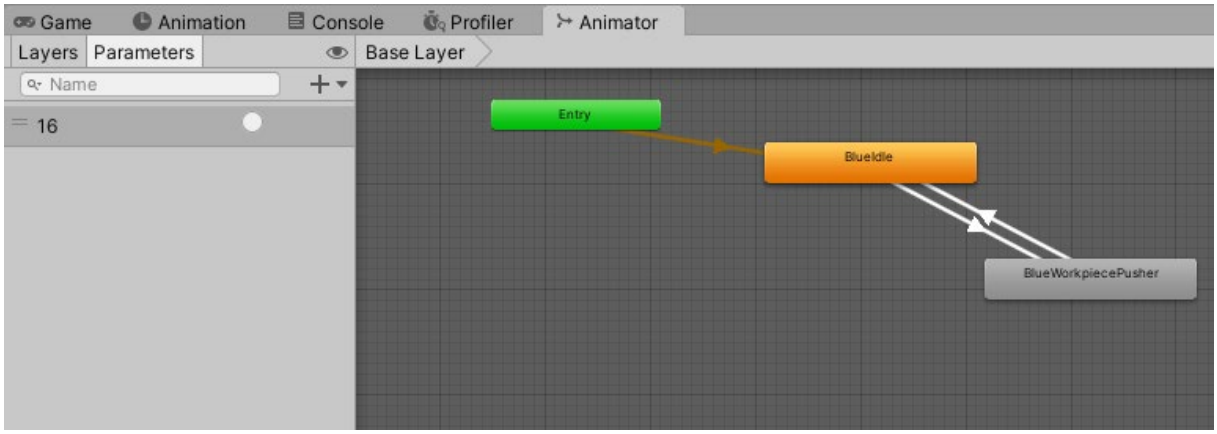
Obr. 7-16 Nastavení komponent *Box Collider* a *Rigidbody*

Ohledně animací se jako první vygenerují tři prázdné objekty, kterým se přiřadí komponenta *Box Collider*. Pozice těchto objektů je nastavena tak, aby sloužily jako písty, které vysouvají polotovary na dopravníkový pás. Dalším bodem je přídání komponenty *RigidBody*, avšak nyní už se zaškrtnutím pole *IsKinematic* – zamezí pádu objektu na zem po spuštění aplikace. Na těchto objektech je poté vytvořena animace posunu v ose x, ve které vysouvají vždy jeden kus polotovaru ze skladu (Obr. 7-17).



Obr. 7-17 Umístění a animace pístů pro vyskladnění

Na takto vytvořených animacích je zapotřebí umístit *Triggery*, které přepínají mezi jednotlivými animačními stavy. Po přepnutí do okna *Animator* se na všech objektech typu *WorkpiecePusher* přidá *Trigger* s jedinečným ID (dle Tab. 7-1) v závislosti na typu polotovaru – *10, 13* a *16.* V tomto případě daný *Trigger* přepíná mezi takzvaným *IdleState*, ve kterém je píst nehybný, a stavem, kdy se pohybuje v ose x za účelem vyskladnění materiálu (Obr. 7-18).

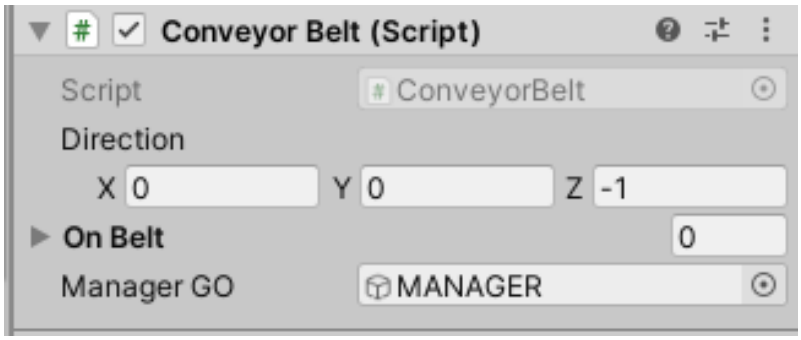


Obr. 7-18 *Animator* a *Trigger* pro písty

Dalším krokem je vytvoření pohybu pásového dopravníku a zajištění toho, aby polotovar nepropadával skrz. Dopravníku je přiřazena komponenta *Box Collider*, která zajistí kontakt mezi polotovarem a dopravníkem. Pro rozpohybování materiálu po pásu je vytvořen skript *ConveyorBelt*.

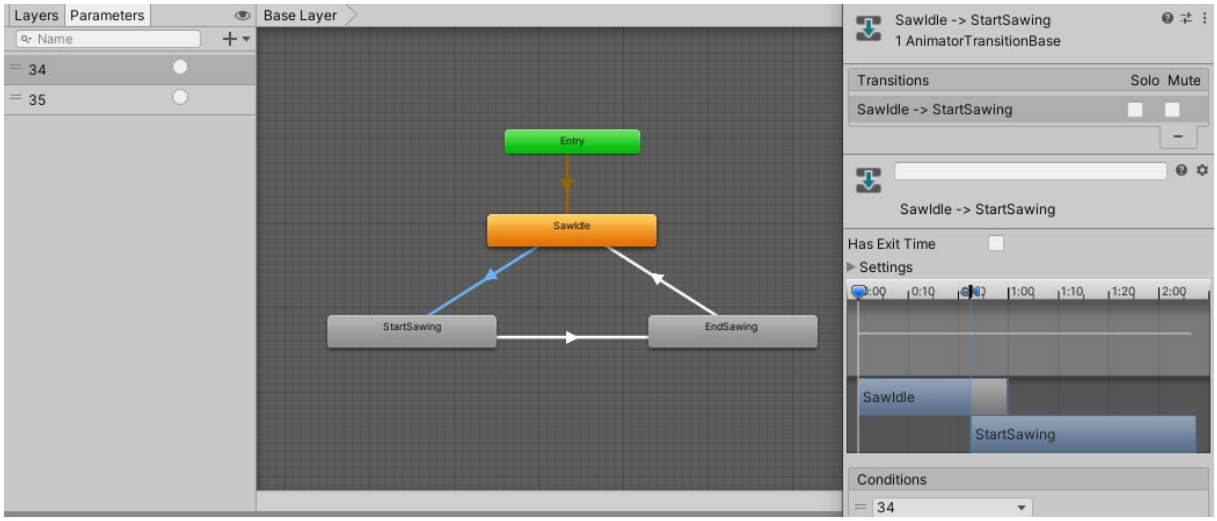
```
public class ConveyorBelt : MonoBehaviour
{
    public Vector3 direction;
    public List<GameObject> onBelt;
    public GameObject ManagerGO;
    private WorkpieceFlow workpieceFlowScript;

    // Start is called before the first frame update
    void Start()
    {
        workpieceFlowScript =
ManagerGO.GetComponent<WorkpieceFlow>();
    }

    // Update is called once per frame
    void Update()
    {
        for(int i = 0; i <= onBelt.Count -1; i++)
        {
            onBelt[i].GetComponent<Rigidbody>().velocity =
workpieceFlowScript.BeltSpeed * direction;
        }
    }
    // When something collides with the belt
    private void OnCollisionEnter(Collision collision)
    {
        onBelt.Add(collision.gameObject);
    }
    // When something leaves the belt
    private void OnCollisionExit(Collision collision)
    {
        onBelt.Remove(collision.gameObject);
    }
}
```

Skript přistupuje k centrálně řízené proměnné *BeltSpeed*, která je načítána z objektu *ManagerGO,* ze skriptu *WorkpieceFlow*. Metody *OnCollisionEnter()* a *OnCollisionExit()* generují kolizi se všemi objekty, které obsahují *Collider* a dotknou se pásu v průběhu aplikace. V metodě *Update*, která se provádí v každém framu[13] aplikace při jejím spuštění, probíhá kontrola rychlosti pásu skrze výše zmíněnou proměnnou *BeltSpeed*, násobena vektorem *Direction*. Díky tomuto vektoru je v editoru možné zvolit směr pohybu polotovaru po dopravníkovém pásu (viz Obr. 7-19).



Obr. 7-19 - *ConveyorBelt*

Po rozpohybování polotovarů následuje tvorba animací jednotlivých pracovišť. Postup tvorby animace závisí na tom, co přesně se na reálném pracovišti děje. Pro příklad slouží popis tvorby animace a animačních stavů na pracovišti pily. Opět je nejprve vytvořen animační klip *SawIdle*, který je nastaven jako výchozí a pila v tomto stavu čeká na příjezd polotovaru. V závislosti na Tab. 7-1 jsou vytvořeny další dva animační klipy – *StartSawing* a *EndSawing*, které zajišťují start a konec procesu řezání. Na Obr. 7-20 je vidět napojení jednotlivých animačních klipů spolu s podmínkami, které musí být splněny, aby se docílilo spuštění a ukončení operace řezání.



Obr. 7-20 Vytvořený *Animator* pro pracoviště pily

Takto jsou vytvořeny animace pro všechna zbylá pracoviště – Laserový senzor, manipulační robot, frézka, vrtačka, senzor barev a svářečka. U všech pracovišť jsou nastaveny podmínky

[13] Snímek spuštěné aplikace

typu *Trigger*. Ty jsou dále napojeny odpovídajícím animačním stavům. Následuje úprava skriptu *WorkpieceFlow*.

```
public class WorkpieceFlow : MonoBehaviour
{
    public float BeltSpeed = 0;
    public int AnimID = 0;

    public GameObject RedWorkpiecePusher;
    private Animator RWAnim;
    public GameObject BlueWorkpiecePusher;
    private Animator BWAnim;
    public GameObject WhiteWorkpiecePusher;
    private Animator WWAnim;

    public List<int> ListOfID;
    void Start()
    {
        RWAnim = RedWorkpiecePusher.GetComponent<Animator>();
        BWAnim = BlueWorkpiecePusher.GetComponent<Animator>();
        WWAnim = WhiteWorkpiecePusher.GetComponent<Animator>();
    }
    public void PerformAnimationMain()
    {
        if (ListOfID.Count == 0)
        {
            Debug.Log("Buffer animací prázdný, opakování animace za
3s");
            Invoke(nameof(PerformAnimationMain), 3);
        }
        else
        {
            AnimID = ListOfID[0];
            ListOfID.RemoveAt(0);
            SetAnimationStateMain();
        }
    }
```

```
    public void PerformAnimationWasterPusher()
    {
        AnimID = ListOfID[0];
        if (AnimID == 32)
        {
            BeltSpeed = 0;
            beltGearsRotScript.gearsRotationBool = false;
            wasterPusherAnim.SetTrigger("32");
            ListOfID.RemoveAt(0);
        }
        else
        {
            Debug.Log("Materiál není zmetek, pokračuje na pracoviště
pily");
        }
    }
    public void SetAnimationStateMain()
    {
        if (AnimID == 10)
        {
            WWAnim.SetTrigger("10");
        }
        else if (AnimID == 13)
        {
            RWAnim.SetTrigger("13");
        }
        else if (AnimID == 16)
        {
            BWAnim.SetTrigger("16");
        }
    public void SetBeltSpeed()
    {
        BeltSpeed = 1.5f;
        beltGearsRotScript.gearsRotationBool = true;
    }
}
```

Skript nejprve obsahuje deklaraci proměnných typu *GameObject* a *Animator*, které slouží k načtení vytvořených animací na daných pracovištích. Dále se v něm vyskytuje rychlost dopravníkového pásu *BeltSpeed*, identifikátor animace *AnimID*, jehož hodnota je posílána z reálného modelu skrze propojený sériový port a generický list *List<int> ListOfID*, který přijímaná data ukládá ve formou integer do fronty za sebe a zastupuje funkci bufferu[14], díky kterému se provedou všechny animační stavy, aniž by byl nějaký přeskočen. Metoda *PerformAnimationMain()* kontroluje, zda generický list obsahuje nějaký prvek s animací. Pokud ne, upozorní uživatele, že je list animací prázdný a metodu opakuje za několik sekund znovu. Pokud nalezne nultý prvek fronty s celočíselnou hodnotou, uloží tuto hodnotu do proměnné *AnimID*, vymaže hodnotu z fronty, aby se daná animace neprovedla vícekrát, a následně spustí metodu *SetAnimationStateMain()*. Ta již porovná hodnotu *AnimID* se svými nadefinovanými podmínkami a pustí animační stav přepnutím příslušného parametru *Trigger*. Metoda *PerformAnimationWasterPusher()* provádí obdobnou činnost, ovšem pouze pro stanoviště, na kterém se rozhoduje, zda je polotovar zmetek, nebo ne. Tato metoda je vytvořena z toho důvodu, že toto stanoviště jako jediné nezastavuje polotovar při každé kolizi, avšak pouze tehdy, jedná-li se o zmetek. Metoda *SetBeltSpeed()* je volána podmínkami, které označují ukončení práci na pracovišti. Materiál tak po dokončení obrobení může pokračovat na další pracoviště.

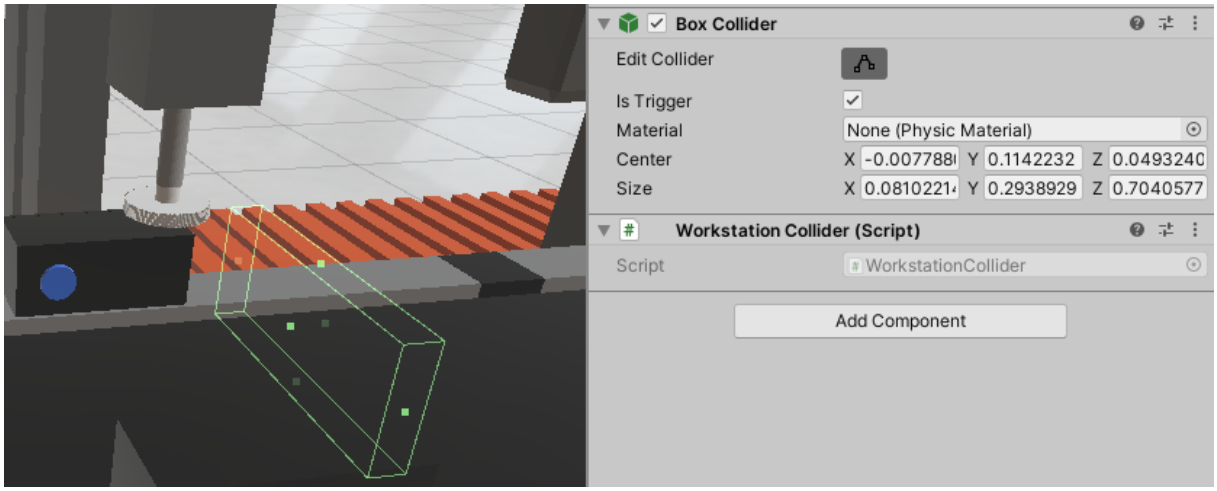
Posledním skriptem, který zajišťuje správnou fluktuaci výrobku výrobní linkou, je *WorkStationCollider*.

```
public class WorkstationCollider : MonoBehaviour
{
    private GameObject ManagerGO;
    private WorkpieceFlow workPieceFlowScript;
    private bool firstTrigger = false;
    private BeltGearsRotation beltGearsRotScript;

    // Start is called before the first frame update
    void Awake()
    {
        ManagerGO = GameObject.Find("MANAGER");
        workPieceFlowScript =
ManagerGO.GetComponent<WorkpieceFlow>();
        beltGearsRotScript =
ManagerGO.GetComponent<BeltGearsRotation>();
    }

    private void OnTriggerEnter(Collider other)
```

[14] Paměť určena pro dočasné uchování dat

```
{
    if (other.tag == "Workpiece" && !firstTrigger)
    {
        firstTrigger = true;
        workPieceFlowScript.BeltSpeed = 0;
        beltGearsRotScript.gearsRotationBool = false;
        workPieceFlowScript.PerformAnimationMain();
    }
}
private void OnTriggerExit(Collider other)
{
    if (other.tag == "Workpiece" && firstTrigger)
    {
        firstTrigger = false;
    }}}
```

Ten je ve formě komponenty obsažen na všech pracovištích, kde se výrobek zastavuje a probíhá animační stav. Jeho úkolem je při kolizi s objektem *Workpiece* prvek zastavit, z objektu *ManagerGO* zastavit pásový dopravník přes proměnnou *BeltSpeed* a provést metodu *PerformAnimationMain()*, která je načítána ze skriptu *WorkpieceFlow*. Při přesunu výrobku z pracoviště se tato zóna vrátí pomocí booleanu *firstTrigger* do původního stavu tak, aby při příjezdu dalšího materiálu proběhlo vše od začátku. Takto nadefinovaná zóna ve formě *BoxCollider* je viditelná na Obr. 7-21.



Obr. 7-21 - Zóna pro kontrolu animace

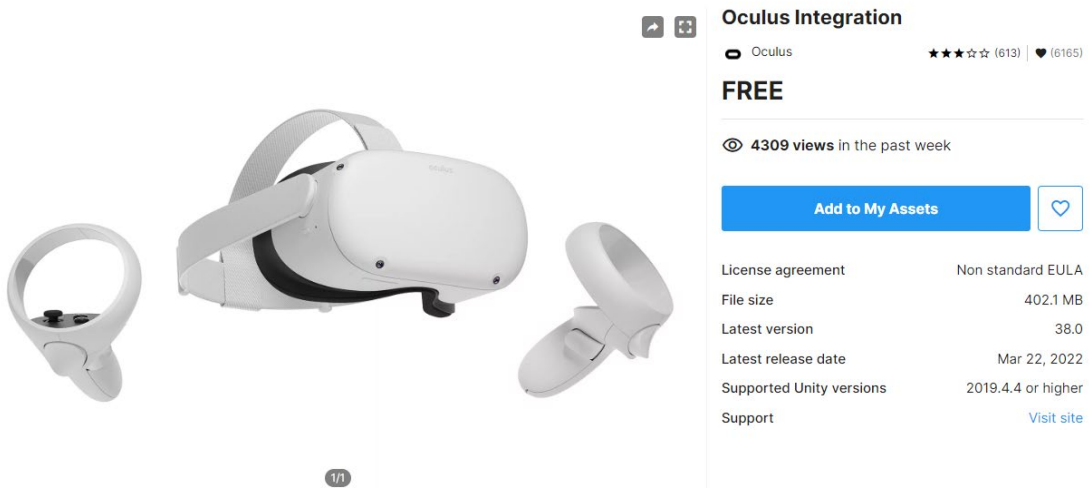
Pro hezčí vizualizaci výsledné aplikace jsou v posledním kroku vytvořeny zvuky pro umocnění pocitu ve virtuální realitě, díle animace nástrojů jednotlivých pracovišť, rotace ozubených kol, simulující pohyb dopravníkového pásu a efekty pro pracoviště laserového senzoru, senzoru barev a svářečky.

8. Popis verze pro virtuální realitu

Projekt byl v předchozí kapitole otestován na platformě Windows. V této části je sepsáno jeho převedení do platformy Android spolu s kompletním přepnutím do verze pro virtuální realitu a následnou optimalizací projektu pro mobilní VR headset Oculus Quest 2.

8.1 Import balíčků a nastavení kamery pro VR

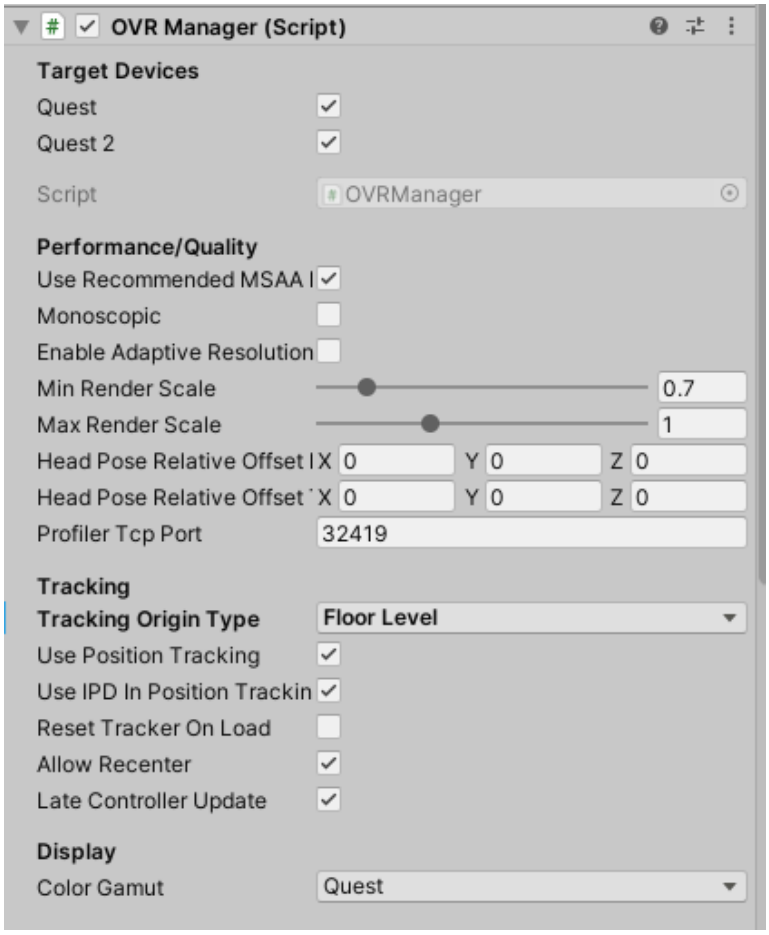
Pro převedení projektu do režimu kompatibility s VR headsetem Oculus Quest 2 se nejprve na stránkách www.assetstore.unity.com naimportuje asset Oculus Integration (Obr. 8-1).



Obr. 8-1 Import Oculus Integration [31]

V softwaru Unity 3D se zaškrtne *SelectAll,* tím se do projektu stáhne a vloží všechna data z assetu. Dalším krokem je instalace balíčků *XR Plugin Management* a *Oculus XR Plugin*. K tomu je nutné v Unity vybrat v horní liště okno *Window -> Package Manager*, ve kterém se v horní části v položce *Packages* zvolí *Unity Registry*. Vypsáním názvů balíčků zmíněných výše a následným kliknutím na *Install* v pravém dolním rohu je instalace hotova.

V projektu se poté vymaže kamera typu *MainCamera* a nahradí se objektem *OVRCameraRig*, který se umístí na podlahu místnosti a na komponentě *OVR Manager* se navolí hodnoty dle Obr. 8-2.

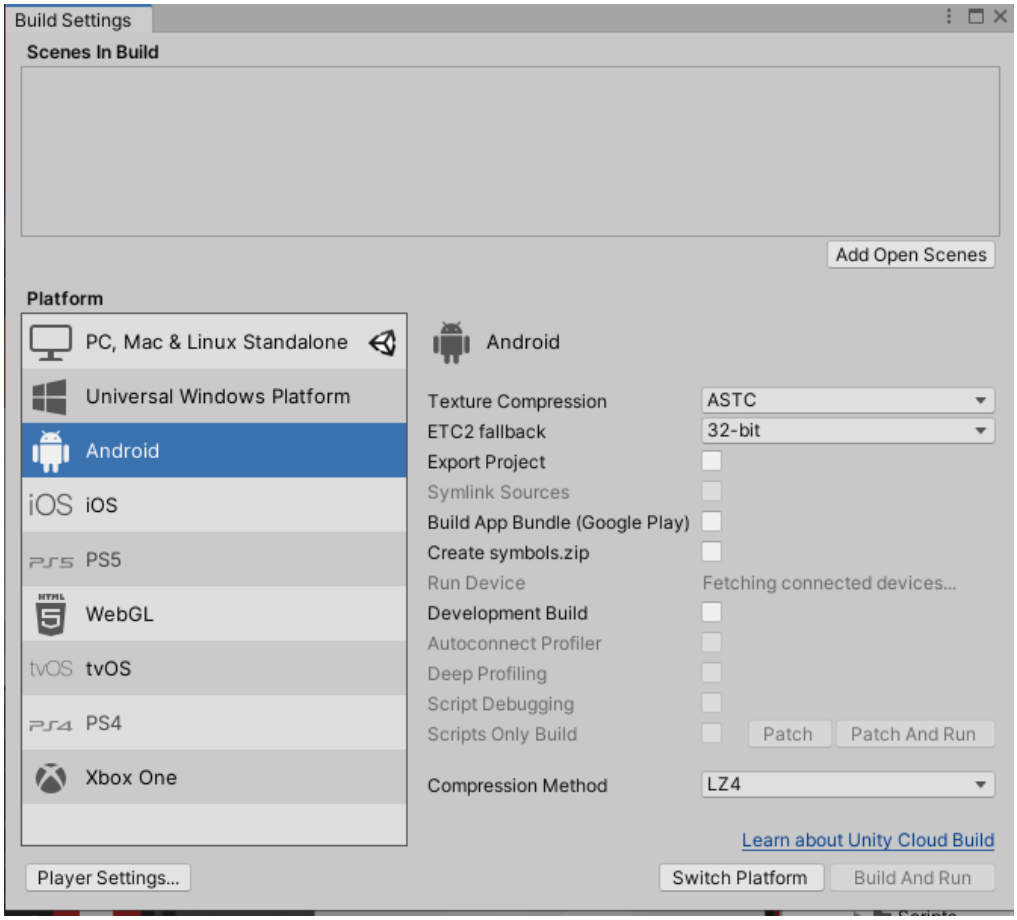


Obr. 8-2 Nastavení OVRCameraRig a jejích komponent

8.2 Přepnutí a optimalizace projektu pro mobilní VR

Důležitým aspektem při převádění projektu do režimu VR je nejenom přepnutí platformy, ale i několika dalších grafických nastavení, které mají obrovský vliv na plynulost výsledné aplikace, zvlášť pokud se jedná o mobilní VR headset.

Nejdříve se projekt přepne na platformu Android, a to vybráním *File -> Build Settings* v horní liště Unity 3D a následným vybráním *Android* a kliknutím na *Switch Platform* v pravém dolním rohu. Po aktualizaci platformy se jako možnost komprese *Texture Compression* vybere volba *ASTC*. Komprese *ASTC* poskytuje nejlepší rovnováhu mezi kvalitou a velikostí příslušného souboru. Bez ohledu na to, co je zde nastaveno, je možné nastavení komprese u jednotlivých textur přepsat. Všechno toto nastavení lze vidět na Obr. 8-3.

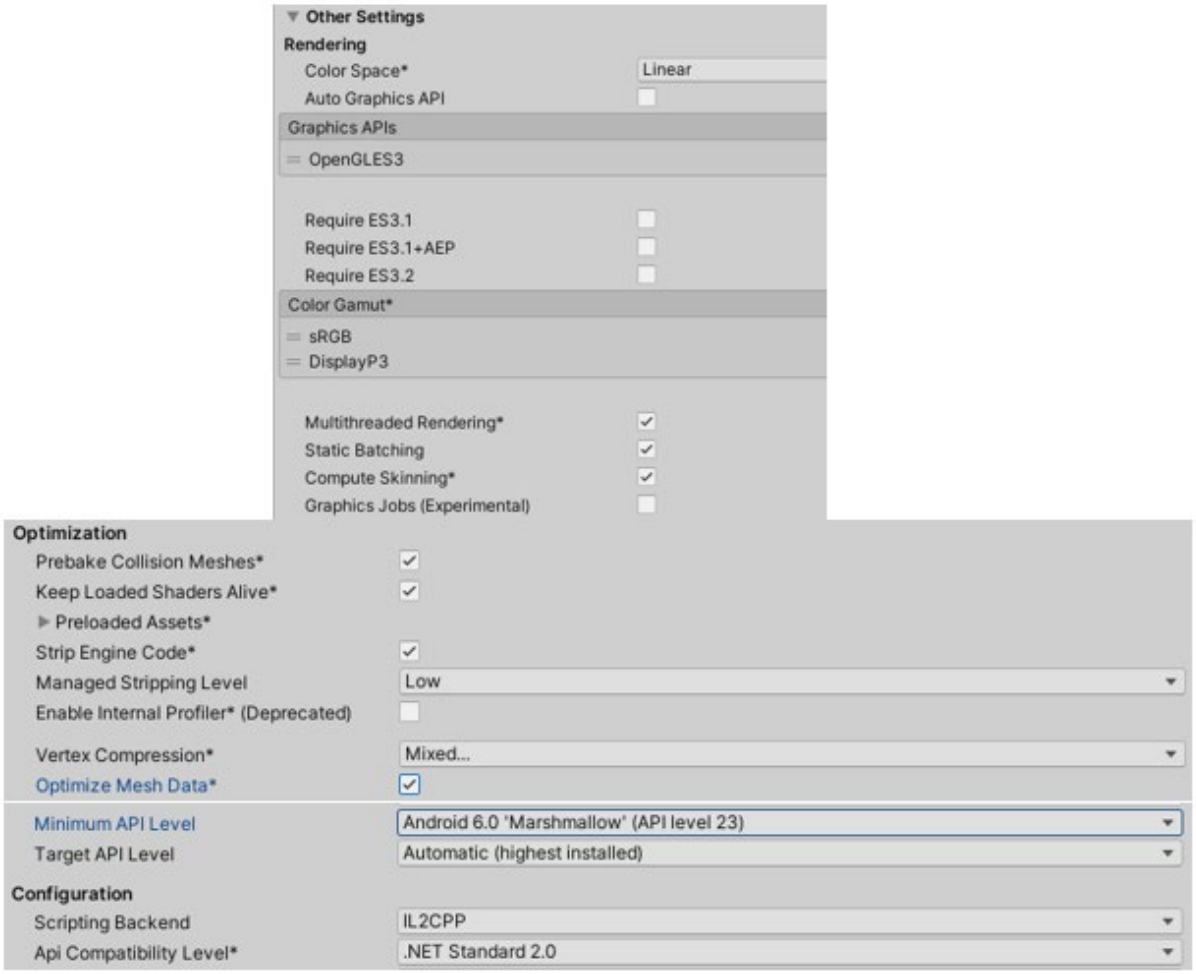


Obr. 8-3 Přepnutí projektu na platformu Android

Z okna *Build Settings* se kliknutím na *Player Settings* a následným výběrem *Other Settings* dostáváme do grafického nastavení celého projektu. Zde je několik základních nastavení ovlivňující výkon finální aplikace (Obr. 8-4):

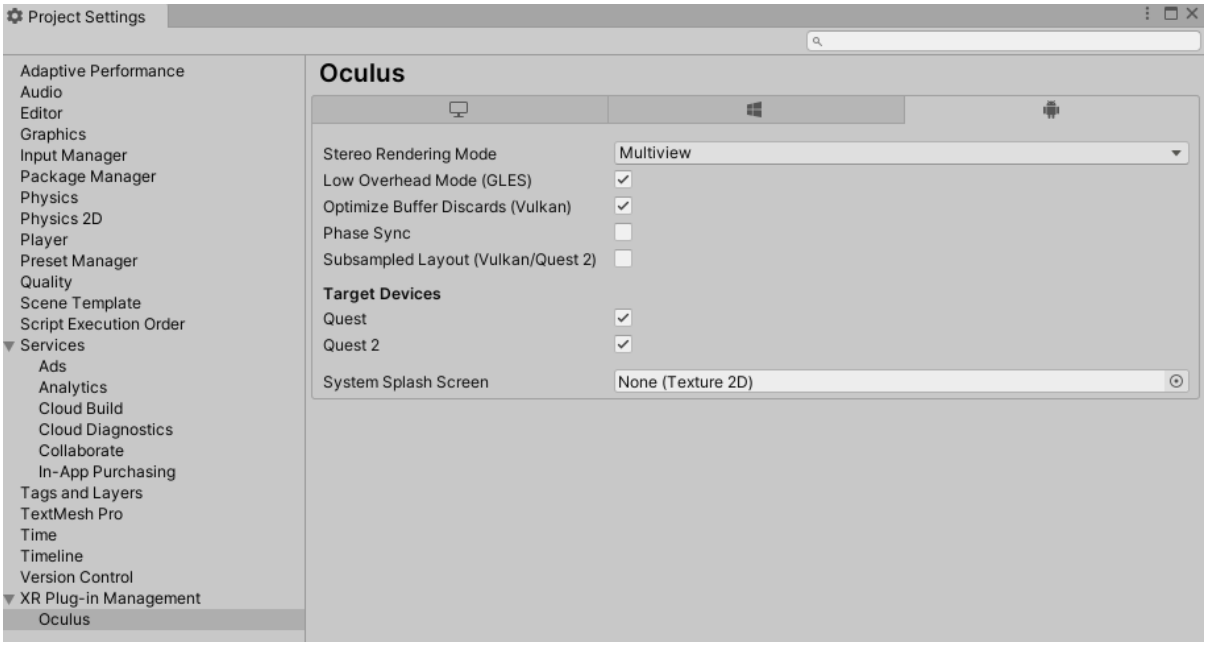
- **Color Space –** Linear – Lineární barevný prostor, který vytváří realističtější osvětlení objektů.
- **Auto Graphics API** – Vypnuté – V panelu *Graphics APIs* se nastaví rozhraní *OpenGLES3*. Je podporováno na všech mobilních zařízeních, na kterých Oculus VR aktuálně funguje. Navíc je *OpenGLES3* potřeba i pro lineární barevný prostor.
- **Multithreaded Rendering** – Zapnuté – Více vláknové vykreslování přesune volání grafického API z hlavního vlákna do samostatného pracovního vlákna.
- **Static Batching** – Zapnuté – Statické dávkování spojí meshe označené jako statické do jednoho velkého za předpokladu, že mají stejný materiál. To umožní nižší počet vykreslovaných objektů na displeji zobrazovacího zařízení a tím i plynulejší chod aplikace.
- **Dynamic Batching –** Zapnuté – Stejná funkce jako *Static Batching*, jen u meshů, které nejsou označené jako statické.
- **Compute Skinning** – Zapnuté – Podporované pro VR aplikace s rozhraním *OpenGLES 3*. *GPU Skinning* je proces, při kterém jsou jednotlivé vrcholy meshe transformovány na základě aktuálního umístění při jejich animaci.
- **Minimum API Level** – Všechna zařízení HMD se systémem Android, která podporují Oculus, by měla být spuštěna na úrovni API 21, Lollipop nebo vyšší.

- **Scripting Backend** – IL2CPP – Zajišťuje vyšší FPS aplikace, ale prodlužuje dobu buildu, tedy generování finální verze aplikace.
- **Prebake Collision Meshes** – Zapnuté – Pokud je metoda *Prebake* aktivní, Unity generuje meshe obsažené v aplikaci již při buildu a tím urychluje její načítání.
- **Keep Loaded Shaders Alive** – Zapnuté – Zapnutím této možnosti se při spuštění načtou všechny shadery. Prodlužuje se počáteční doba načítání aplikace, ale řeší se tím případné chyby v kompilaci shaderů při běhu aplikace.
- **Optimize Mesh Data** – Zapnuté – Odstraňuje veškerá mesh data, které nejsou materiálem používána a tím snižuje výslednou velikost aplikace.



Obr. 8-4 ***Other Settings***

Pro poslední nastavení projektu na VR se v okně *Project Settings* rozklikne možnost *XR Plug-in Management* a zvolí se *Oculus* (Obr. 8-5). V novém okně se nastavení mění pouze pro platformu Android. Hodnota *Stereo Rendering Mode* je jednou z nejdůležitějších grafických možností při optimalizaci projektu pro VR a musí být nastavena na *Multiview*. Během tohoto typu renderování (na rozdíl od typu *Multipass*) se veškerý obsah aplikace vykresluje pouze pro levé oko a do druhého se pouze duplikuje s odhadovanou hodnotou pozice všech objektových vrcholů. Oproti *Multipass* nastavení tedy zvyšuje výkon až dvojnásobně.



Obr. 8-5 ***XR Plug-in Management – Oculus***

Tímto je projekt převeden do virtuální reality a připraven na napojení digitálního dvojčete a reálného modelu skrze HMD Oculus Quest 2.

9. Závěr

Tato diplomová práce se skládá z teoretické a praktické části. Pro vysvětlení pojmů Průmysl 4.0 a digitální dvojče byla použita literatura z odborných publikací. Dále byla popsána obecně stavebnice Fischertechnik, jež v této práci slouží pro simulaci reálného modelu, spolu s jednotlivými výrobními úseky, které model obsahuje. Následoval popis a výběr vhodné technologie pro digitalizaci. Softwary byly porovnány jak z pohledu informací udávaných výrobcem, tak z pohledu zkušeností autora této práce. V softwarech Blender a následně Unity 3D byla vytvořena aplikace, která byla následně převedena pro použití ve virtuální realitě.

Hlavním praktickým výstupem této práce je aplikace, která ve virtuální realitě simuluje procesy reálného modelu stavebnice Fischertechnik, a to včetně komunikace s modelem v reálném světě. Z pohledu definice digitálního dvojčete, tedy dle míry integrace, spadá uvedená aplikace do druhé podkategorie, tedy digitálního stínu. Výstup bude sloužit i jako pomůcka při výuce na Katedře průmyslového inženýrství a managementu na Západočeské univerzitě v Plzni. Dalším využitím je propojení a využití výstupů z této práce Ing. Bc. Miroslavem Malagou ve své dizertační práci a pro další vědecké účely (články, publikace, a další). V neposlední řadě bude aplikace obecně používána i pro prezentaci katedry.

Lze konstatovat, že bylo vytvořeno modulární interaktivní digitální dvojčete pro virtuální realitu. Modularita je zajištěna díky jednoznačnému oddělení dílčích výrobních úseků v aplikaci, s tím, že každý tento úsek funguje jako „black box“ se svými vstupy a výstupy. Nebude tedy výrazně problematické tuto aplikaci dále rozšiřovat, či měnit pořadí strojů ve výrobě. Pro další vývoj této aplikace by bylo vhodné zvýšit míru integrace a posunout tak finální aplikaci do první integrační podkategorie. Z toho důvodu je dalším logickým krokem vylepšení komunikace mezi reálným a digitálním modelem, a to nahrazením komunikace přes sériové porty cloudovým serverem. Další podmínkou pro to, aby se zvýšila míra integrace digitálního dvojčete, je vytvoření ovládacího panelu uvnitř digitálního modelu, pomocí kterého bude možné ovlivňovat chování reálného modelu skrze posílané příkazy.

Při rozšiřování reálného modelu je zapotřebí vymodelovat nové výrobní úseky stavebnice a napojit je do současného virtuálního modelu, omezení na takováto rozšíření nejsou nijak limitující. Pro zrychlení implementace těchto rozšíření by bylo v budoucnu vhodné vytvořit knihovnu výrobních úseků stavebnice Fischertechnik, která by obsahovala jednotlivé nadefinované 3D modely s vytvořenými animacemi a umožňovala by libovolně upravovat či doplňovat celou výrobní linku.

Autor této práce při tvorbě využil svých zkušeností z řešení několika předchozích projektů pro virtuální realitu a mohl tak zkvalitnit výstup celé práce.

10. Bibliografie

[1] LASI, Heiner, Peter FETTKE, Hans Georg KEMPER, Thomas FELD and Michael HOFFMANN. Industry 4.0. *Business and Information Systems Engineering* [online]. 2014, 6(4). ISSN 18670202. Dostupné z: doi:10.1007/s12599-014-0334-4

[2] HAAG, Sebastian and Reiner ANDERL. Digital twin – Proof of concept. *Manufacturing Letters* [online]. 2018, 15, 64–66 [Citace: 2021-11-30]. ISSN 2213-8463. Dostupné z: doi:10.1016/J.MFGLET.2018.02.006

[3] ŚLUSARCZYK, Beata. Industry 4.0 : are we ready? *Polish Journal of Management Studies* [online]. 2018, Vol. 17, No. 1(1), 232–248 [Citace: 2021-11-30]. ISSN 2081-7452. Dostupné z: doi:10.17512/PJMS.2018.17.1.19

[4] *Průmysl 4.0. Průmysl 4.0* [online]. Technodat 2018 [Citace: 2021-11-30]. Dostupné z: https://www.prumysl-4.cz/

[5] STANTON CHASE. 2017 Global Industrial Survey. *2017 Global Industrial Survey*. 2017.

[6] GRIEVES, Michael. Digital Twin : Manufacturing Excellence through Virtual Factory Replication - A Whitepaper by Dr . Michael Grieves. *White Paper*. 2014, (March).

[7] *Průmysl 4.0: čtvrtá průmyslová revoluce právě probíhá* [online]. [Citace: 2021-11-30]. Dostupné z: https://www.enovation.cz/aktuality/dotace-pro-podnikatele/prumysl-40-ctvrta-prumyslova-revoluce-prave-probiha/

[8] *Digitální dvojče: Vůdčí technologie inteligentního průmyslu | ANASOFT* [online]. [Citace: 2021-11-30]. Dostupné z: https://www.anasoft.com/emans/cz/home/Novinky-blog/Blog/digitalni-dvojce-top-technologie-inteligentniho-prumyslu

[9] *Digital Twin Use Cases and Applications | softengi.com* [online]. [Citace: 2021-12-01]. Dostupné z: https://softengi.com/blog/use-cases-and-applications-of-digital-twin/

[10] TUAN, Mai Anh. Comparison of different digital twin implementation concepts and implementation concept for a Fischertechnik industry 4.0 demonstrator [online]. no date [Citace: 2021-12-01]. Dostupné z: http://epub.vgu.edu.vn/handle/dlibvgu/1077

[11] *Interest grows in digital twins, digital engineering - Aerospace America* [online]. [Citace: 2021-12-01]. Dostupné z: https://aerospaceamerica.aiaa.org/year-in-review/interest-grows-in-digital-twins-digital-engineering/

[12] *How Digital Twin is disrupting Automotive Industry in Real World – Thinking Beyond Tomorrow* [online]. [Citace: 2021-12-01]. Dostupné z: https://sidni.in/2021/07/28/how-digital-twin-is-disrupting-automotive-industry-in-real-world/

[13] *How Digital Twins are Changing the Energy Industry* [online]. [Citace: 2021-12-01]. Dostupné z: https://www.automation.com/en-us/articles/2018/how-digital-twins-are-changing-the-energy-industry

[14] *Digitale tweeling | Sieformatie | Siemens Netherlands* [online]. [Citace: 2021-12-01]. Dostupné z: https://new.siemens.com/nl/nl/bedrijf/stories/sieformatie/digital-twin.html

[15] ŠVELCHOVÁ, Alena. 3D simulační model v Plant Simulation jako digitální dvojče modelu Training Factory Industry 4.0 [online]. 2021 [Citace: 2021-12-02]. Dostupné z: http://dspace5.zcu.cz/handle/11025/44458

[16] LOUWAGIE, Ruben, Sapena SUPERVISOR, Fabio MERCANDETTI and Philipp MORGENTHALER. Industry 4.0 through Fischertechnik and a Digital Twin Author: External expert. no date.

[17] *Unity Real-Time Development Platform | 3D, 2D VR & AR Engine* [online]. [Citace: 2021-12-01]. Dostupné z: https://unity.com/

[18] *Unity Software: WHAT & WHY You Need to Know!* [online]. [Citace: 2021-12-01]. Dostupné z: https://www.zekiah.com/insights/software-development/unity-software-what-why-you-need-to-know

[19] *Powerful 2D, 3D, VR, & AR software for cross-platform development of games and mobile apps.* [online]. [Citace: 2021-12-01]. Dostupné z: https://store.unity.com/#plans-business

[20] *Godot Engine - Features* [online]. [Citace: 2021-12-01]. Dostupné z: https://godotengine.org/features

[21] *GitHub - godotengine/godot: Godot Engine – Multi-platform 2D and 3D game engine* [online]. [Citace: 2021-12-01]. Dostupné z: https://github.com/godotengine/godot

[22] *The most powerful real-time 3D creation tool - Unreal Engine* [online]. [Citace: 2021-12-01]. Dostupné z: https://www.unrealengine.com/en-US/

[23] *Download - Unreal Engine* [online]. [Citace: 2021-12-01]. Dostupné z: https://www.unrealengine.com/en-US/download

[24] *Edddison for Unreal Engine |* [online]. [Citace: 2021-12-01]. Dostupné z: https://www.edddison.com/products/edddison-unreal-engine/

[25] *Simlab Soft - Enabling Interactive VR* [online]. [Citace: 2021-12-01]. Dostupné z: https://www.simlab-soft.com/index.html

[26] *VR Creation and Sharing Software* [online]. [Citace: 2021-12-01]. Dostupné z: https://www.simlab-soft.com/3d-products/simlab-composer-vr-appl.php?utm_source=footer

[27] *Simlab Composer Features* [online]. [Citace: 2021-12-01]. Dostupné z: https://www.simlab-soft.com/3d-products/simlab-composer-main.aspx

[28] *Novinky verze SimLab Composer 10* [online]. [Citace: 2021-12-01]. Dostupné z: https://www.cad.cz/aktuality/77-aktuality/10475-novinky-verze-simlab-composer-10.html

[29] *Home of the Blender project – Free and Open 3D Creation Software.* [online]. [Citace: 2022-04-13]. Dostupné z: https://www.blender.org/

[30] *Serial Port Utility Pro for Unity: Device control becomes easy with Unity.* [online]. [Citace: 2022-04-14]. Dostupné z: https://portutility.com/

[31] *Oculus Integration | Integration | Unity Asset Store* [online]. [Citace: 2022-04-14]. Dostupné z: https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022

Seznam příloh

PŘÍLOHA č. 1

Skript *GetCOMValue*

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

namespace SerialPortUtility
{
    public class GetCOMValue : MonoBehaviour
    {
        public string COMPort = "COM2";

        public GameObject OpenedCOMGO;
        public GameObject SpupDebugConsoleOurCOM;
        public DebugConsole debugConsoleScript;

        GameObject[] SpupConsolesGOs;
        Canvas[] SpupConsolesCanvases;

        public GameObject ContentGO;
        private Text contentText;
        public int ReceivedData; // přijatá data na COM portu ve
formátu string

        public int CheckForIntChange;// int, který kontroluje, zda-
li se změnila hodnota a tudíž nastal nový stav

        public GameObject ManagerGO;
        private WorkpieceFlow workPieceFlowScript;

        // Start is called before the first frame update
        void Start()
        {
            OpenedCOMGO = GameObject.Find(COMPort);
            Invoke(nameof(GetCOMComponentAndRecievedData), 0.01f);
            CheckForIntChange = ReceivedData;
            workPieceFlowScript =
ManagerGO.GetComponent<WorkpieceFlow>();
        }

        private void FixedUpdate()
        {
            int.TryParse(contentText.text, out ReceivedData);
            if (CheckForIntChange != ReceivedData)
            {
                CheckForIntChange = ReceivedData;
                workPieceFlowScript.AnimID = CheckForIntChange;
                workPieceFlowScript.SetAnimationState();
```

```
            }
        }

        // Update is called once per frame
        void GetCOMComponentAndRecievedData()
        {
            SpupDebugConsoleOurCOM =
OpenedCOMGO.transform.GetChild(0).gameObject;
            debugConsoleScript =
SpupDebugConsoleOurCOM.GetComponent<DebugConsole>();
            debugConsoleScript.ViewButtonClick();
            SpupConsolesGOs =
FindGameObjectsByName("SPUPDebugConsole(Clone)");
            SpupConsolesCanvases = new
Canvas[SpupConsolesGOs.Length];
            LoadCanvasesAndTurnThemOff();
            ContentGO = GameObject.Find("Content"); // Najdi
Gameobject s názvem Content, který zobrazuje přijímaná data na
vybraném COM portu
            contentText = ContentGO.GetComponent<Text>();
        }

        GameObject[] FindGameObjectsByName(string name)
        {
            return
System.Array.FindAll((FindObjectsOfType(typeof(GameObject)) as
GameObject[]), p => p.name == name);
        }

        void LoadCanvasesAndTurnThemOff()
        {
            for (int i = 0; i < SpupConsolesGOs.Length; i++)
            {
                SpupConsolesCanvases[i] =
SpupConsolesGOs[i].GetComponent<Canvas>();
                SpupConsolesCanvases[i].enabled = false;
            }
        }
    }
}
```

PŘÍLOHA č. 2

Skript *WorkpieceFlow*

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

public class WorkpieceFlow : MonoBehaviour
{
    public float BeltSpeed = 0;
    public int AnimID = 0;

    public GameObject RedWorkpiecePusher;
    private Animator RWAnim;
    public GameObject BlueWorkpiecePusher;
    private Animator BWAnim;
    public GameObject WhiteWorkpiecePusher;
    private Animator WWAnim;

    public GameObject WasterPusher;
    private Animator wasterPusherAnim;

    public GameObject SawWorkstation;
    private Animator sawWorkstationAnim;

    public GameObject HandlingRobot;
    private Animator handlingRobotAnim;

    public GameObject MillingWorkstation;
    private Animator millingWorkstationAnim;
    public GameObject ErrorLightMill;
    private Renderer errorLightMillMat;
    public GameObject DrillingWorkstation;
    private Animator drillingWorkstationAnim;
    public GameObject ErrorLightDrill;
    private Renderer errorLightDrillMat;

    public GameObject WeldingWorkStation;
    private Animator weldingWorkstationAnim;

    public List<int> ListOfID;


    void Start()
    {
        RWAnim = RedWorkpiecePusher.GetComponent<Animator>();
        BWAnim = BlueWorkpiecePusher.GetComponent<Animator>();
        WWAnim = WhiteWorkpiecePusher.GetComponent<Animator>();

        wasterPusherAnim = WasterPusher.GetComponent<Animator>();
        sawWorkstationAnim =
SawWorkstation.GetComponent<Animator>();
        handlingRobotAnim = HandlingRobot.GetComponent<Animator>();
        millingWorkstationAnim =
MillingWorkstation.GetComponent<Animator>();
        drillingWorkstationAnim =
DrillingWorkstation.GetComponent<Animator>();
```

```
        weldingWorkstationAnim =
WeldingWorkStation.GetComponent<Animator>();

        errorLightMillMat = ErrorLightMill.GetComponent<Renderer>();
        errorLightDrillMat =
ErrorLightDrill.GetComponent<Renderer>();
    }

    public void PerformAnimationMain()
    {
        if (ListOfID.Count == 0)
        {
            Debug.Log("Buffer animací prázdný, opakování animace za
3s");
            Invoke(nameof(PerformAnimationMain), 3);
        }
        else
        {
            AnimID = ListOfID[0];
            ListOfID.RemoveAt(0);
            SetAnimationStateMain();
        }
    }

    public void PerformAnimationWasterPusher()
    {
        AnimID = ListOfID[0];

        if (ListOfID.Count == 0)
        {
            return;
        }

        else if (AnimID == 32)
        {
            BeltSpeed = 0;
            wasterPusherAnim.SetTrigger("32");
            ListOfID.RemoveAt(0);
        }
    }

    public void SetAnimationStateMain()
    {
        if (AnimID == 10)
        {
            WWAnim.SetTrigger("10");
        }

        else if (AnimID == 13)
        {
            RWAnim.SetTrigger("13");
        }

        else if (AnimID == 16)
        {
```

```
        BWAnim.SetTrigger("16");
    }

    else if (AnimID == 30)
    {
        Debug.Log("Zahájení rentgenové kontroly výrobku");
        Invoke(nameof(PerformAnimationMain), 3);
    }

    else if (AnimID == 4)
    {
        Debug.Log("Materiál je zmetek");
        Invoke(nameof(PerformAnimationMain), 3);
    }

    else if (AnimID == 5)
    {
        Debug.Log("Materiál není zmetek");
        Invoke(nameof(PerformAnimationMain), 3);
    }

    else if (AnimID == 31)
    {
        Debug.Log("Ukončení rentgenové kontroly");
        SetBeltSpeed();
    }

    else if (AnimID == 34)
    {
        sawWorkstationAnim.SetTrigger("34");
        Invoke(nameof(PerformAnimationMain), 4);
    }

    else if (AnimID == 35)
    {
        sawWorkstationAnim.SetTrigger("35");
        Invoke(nameof(SetBeltSpeed), 2);
    }

    else if (AnimID == 51)
    {
        handlingRobotAnim.SetTrigger("51");
    }

    else if (AnimID == 71)
    {
        Debug.Log("Výskyt chyby na frézce, odstraní uživatel");
        errorLightMillMat.material.EnableKeyword("_EMISSION");
        Invoke(nameof(PerformAnimationMain), 4);
    }

    else if (AnimID == 72)
    {
        Debug.Log("Chyba na frézce odstraněna");
        errorLightMillMat.material.DisableKeyword("_EMISSION");
```

```
        Invoke(nameof(PerformAnimationMain), 1);
    }

    else if (AnimID == 73)
    {
        millingWorkstationAnim.SetTrigger("73");
        Invoke(nameof(PerformAnimationMain), 3);
    }

    else if (AnimID == 74)
    {
        millingWorkstationAnim.SetTrigger("74");
        Invoke(nameof(SetBeltSpeed), 2);
    }

    else if (AnimID == 75)
    {
        Debug.Log("Výskyt chyby na vrtačce, odstraní uživatel");
        errorLightDrillMat.material.EnableKeyword("_EMISSION");
        Invoke(nameof(PerformAnimationMain), 4);
    }

    else if (AnimID == 76)
    {
        Debug.Log("Chyba na vrtačce odstraněna");
        errorLightDrillMat.material.DisableKeyword("_EMISSION");
        Invoke(nameof(PerformAnimationMain), 1);
    }

    else if (AnimID == 77)
    {
        drillingWorkstationAnim.SetTrigger("77");
        Invoke(nameof(PerformAnimationMain), 3);
    }

    else if (AnimID == 78)
    {
        drillingWorkstationAnim.SetTrigger("78");
        Invoke(nameof(SetBeltSpeed), 2);
    }

    else if (AnimID == 90)
    {
        Debug.Log("Začátek ohřevu výrobku");
        Invoke(nameof(PerformAnimationMain), 3);
    }

    else if (AnimID == 1)
    {
        Debug.Log("Barva obrobku je: BÍLÁ");
        Invoke(nameof(PerformAnimationMain), 1);
    }

    else if (AnimID == 2)
    {
```

```
            Debug.Log("Barva obrobku je: ČERVENÁ");
            Invoke(nameof(PerformAnimationMain), 1);
        }

        else if (AnimID == 3)
        {
            Debug.Log("Barva obrobku je: MODRÁ");
            Invoke(nameof(PerformAnimationMain), 1);
        }

        else if (AnimID == 91)
        {
            Debug.Log("Ukončení ohřevu obrobku");
            SetBeltSpeed();
        }

        else if (AnimID == 92)
        {
            weldingWorkstationAnim.SetTrigger("92");
            Invoke(nameof(PerformAnimationMain), 3);
        }

        else if (AnimID == 93)
        {
            weldingWorkstationAnim.SetTrigger("93");
            Invoke(nameof(SetBeltSpeed), 2);
        }

        else if (AnimID == 100) Debug.Log("Výrobek byl dokončen a
vyřazen z výrobního systému");
    }

    public void SetBeltSpeed()
    {
        BeltSpeed = 1.5f;
    }
}
```

PŘÍLOHA č. 3

Skript *ConveyorBelt*

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

public class ConveyorBelt : MonoBehaviour
{
    public Vector3 direction;
    public List<GameObject> onBelt;
    public GameObject ManagerGO;
    private WorkpieceFlow workpieceFlowScript;

    // Start is called before the first frame update
    void Start()
    {
        workpieceFlowScript =
ManagerGO.GetComponent<WorkpieceFlow>();
    }

    // Update is called once per frame
    void Update()
    {
        for(int i = 0; i <= onBelt.Count -1; i++)
        {
            onBelt[i].GetComponent<Rigidbody>().velocity =
workpieceFlowScript.BeltSpeed * direction;
        }
    }

    // When something collides with the belt
    private void OnCollisionEnter(Collision collision)
    {
        onBelt.Add(collision.gameObject);
    }

    // When something leaves the belt
    private void OnCollisionExit(Collision collision)
    {
        onBelt.Remove(collision.gameObject);
    }
}
```