ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA STROJNÍ

Studijní program: N0715A270012 Průmyslové inženýrství a management

DIPLOMOVÁ PRÁCE

3D Bin Packing Problem s využitím virtuální reality

Autor: **Bc. Michael Pompl**

Vedoucí práce: **Ing. Pavel Raška, Ph.D.**

Akademický rok 2020/2021

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta strojní

Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Michael POMPL**
Osobní číslo:	**S19N0175P**
Studijní program:	**N0715A270012 Průmyslové inženýrství a management**
Studijní obor:	**Průmyslové inženýrství a management**
Téma práce:	**3D Bin Packing Problem s využitím virtuální reality**
Zadávající katedra:	**Katedra průmyslového inženýrství a managementu**

Zásady pro vypracování

1. Úvod
2. 3D Bin Packing Problem – definice problému a algoritmy
3. Virtuální realita – definice, popis vytváření prostředí pro virtuální realitu
4. Vizualizace řešení 3D Bin Packing Problem ve virtuální realitě
5. Zhodnocení a závěr

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

Seznam doporučené literatury:

1. KANNA, S. Rajesh. *Genetic Algorithm for Bin Packing*. India, 2016. ISBN 978-1-52026-739-5.
2. MALEY, Paul. *Virtual & Agumented Reality For Dummies*. United States of America: John Waley & Sons, Inc. 2018. ISBN 978-1-11948-134-8.
3. LINOWES, Jonatan. *Unity Virtual Reality Projects*. United Kingdom: Packt Publishing, 2018. ISBN 978-1-78847-880-9.

Vedoucí diplomové práce: **Ing. Pavel Raška, Ph.D.**
Katedra průmyslového inženýrství a managementu

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

Datum zadání diplomové práce: **21. září 2020**
Termín odevzdání diplomové práce: **28. května 2021**

L.S.

Doc. Ing. Milan Edl, Ph.D.
děkan

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

V Plzni dne 21. září 2020

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

Při této příležitosti bych chtěl poděkovat Ing. Pavlu Raškovi, Ph.D. a doc. Ing. Petru Hořejšímu, Ph.D., za trpělivost, cenné připomínky a rady, které mi v průběhu práce ochotně poskytli. Dále bych chtěl poděkovat Katedře průmyslového inženýrství a managementu za poskytnutí potřebných prostředků k možnosti vypracování práce. Nakonec bych chtěl poděkovat všem, kteří mi radou a cennými zkušenostmi umožnili dokončit tuto práci, a to především mé rodině a přátelům.

ANOTAČNÍ LIST DIPLOMOVÉ PRÁCE

AUTOR	**Příjmení** Bc. Pompl	**Jméno** Michael	
STUDIJNÍ OBOR	N0715A270012 „Průmyslové inženýrství a management“		
VEDOUCÍ PRÁCE	**Příjmení (včetně titulů)** Ing. Raška, Ph.D.	**Jméno** Pavel	
PRACOVIŠTĚ	ZČU - FST - KPV		
DRUH PRÁCE	**DIPLOMOVÁ**	~~**BAKALÁŘSKÁ**~~	**Nehodící se škrtněte**
NÁZEV PRÁCE	3D Bin Packing Problem s využitím virtuální reality		

FAKULTA	strojní		**KATEDRA**	KPV		**ROK ODEVZD.**	2021

POČET STRAN (A4 a ekvivalentů A4)

CELKEM	119		**TEXTOVÁ ČÁST**	119		**GRAFICKÁ ČÁST**	0

STRUČNÝ POPIS (MAX 10 ŘÁDEK) **ZAMĚŘENÍ, TÉMA, CÍL POZNATKY A PŘÍNOSY**	Diplomová práce obsahuje popis problematiky Bin Packing Problem s vizualizací ve virtuální realitě. Teoretická část je zaměřena převážně na vysvětlení, čím se Bin Packing Problem zabývá a jaké jsou metody jeho řešení. V praktické části je tato problematika vizualizována ve virtuálním prostředí pomocí modelu virtuálního logistického komplexu (skladu). V závěrečné části jsou vysvětleny jednotlivé části kódu programu a za jakým účelem byly implementovány.
KLÍČOVÁ SLOVA **ZPRAVIDLA JEDNOSLOVNÉ POJMY, KTERÉ VYSTIHUJÍ PODSTATU PRÁCE**	Bin Packing Problem, aplikace pro VR, BPP algoritmy, virtuální realita, programovací jazyk C#, Unity3D, Oculus Rift

SUMMARY OF MASTER SHEET

AUTHOR	**Surname** Bc. Pompl	**Name** Michael
FIELD OF STUDY	N0715A270012 “Industrial Engineering and Management”	
SUPERVISOR	**Surname (Inclusive of Degrees)** Ing. Raška Ph.D.	**Name** Pavel
INSTITUTION	ZČU - FST - KPV	

TYPE OF WORK	**DIPLOMA**	~~**BACHELOR**~~	**Delete when not applicable**
TITLE OF THE WORK	3D Bin Packing Problem Using Virtual Reality		

FACULTY	Mechanical Engineering	**DEPARTMENT**	Industrial engineering and management	**SUBMITTED IN**	2021

NUMBER OF PAGES (A4 and eq. A4)

TOTALLY	119	**TEXT PART**	119	**GRAPHICAL PART**	0

BRIEF DESCRIPTION **TOPIC, GOAL, RESULTS AND CONTRIBUTIONS**	The thesis contains a description of the Bin Packing Problem with visualization in virtual reality. The theoretical part is mainly focused on explaining what the Bin Packing Problem is and what are the methods of its solution. In the practical part, the problem is visualized in a virtual environment using a model of a virtual logistics complex (warehouse). The final part explains the different parts of the source code and for which purpose they were implemented.
KEY WORDS	Bin Packing Problem, VR application, BPP algorithms, virtual reality, C# programming language, Unity3D, Oculus Rift

Obsah

Přehled použitých zkratek a symbolů

BPP	Bin Packing Problem
3D	Trojrozměrný prostor
2D	Dvojrozměrný prostor
DP	Diplomová práce
OPT	Optimum
WCS	„Worst Case Scenario“ přístup
OnA	Online algoritmus
OfA	Offline algoritmus
CP	„Conner Points (Rohový bod)
B&B	„Branch and Bounds“ metoda
EP	„Extreme Point“ (Extrémní bod)
WB	„Wall Building“ (Algoritmus stavění zdi)
CG	„Column Generation“ (Algoritmus generování sloupů)
TS	„Tree Search“ (algoritmus větvení)
LS	„Local Search“ (algoritmus lokálního vyhledávání)
N	„Neighbour“ funkce hledání souseda
RP	Rastr Point (Rastrový bod)
VR	Virtuální realita
HW	Hardware
HMD	„Head Mounted Display“ (náhlavní brýle pro VR)
„All-in-one“	„Vše v jednom“
UI	„User Interface“ (Uživatelské rozhraní)
ID	Identifikační číslo

Seznam obrázků

Seznam tabulek

Seznam příloh

1 Úvod

V dnešní době žijeme ve světě, kde, se produkty vyrábějí v jedné části světa a spotřebovávají ve druhé. Mnoho společností se účastní přemisťování produktů a zboží na mezinárodním trhu. Logistika v tomto případě hraje zásadní roli a mnoho větších společností, především těch nadnárodních si v rámci sítě svých poboček zřizuje samostatné oddělení zabývající se přepravou svých výrobků či zboží [28].

V úvodu diplomové práce je obsaženo seznámení se s řešením optimalizačního problému týkajícího se BPP a jeho aplikace pro trojrozměrný prostor. Z počátku práce je popsána základní teoretická část týkající se problematiky Bin Packing Problem za účelem zorientování se v problematice. Dále v rešeršní části jsou popsány varianty algoritmů, které mohou být využity pro řešení problému umístění jednotlivých beden (např. ve formě krabic, obalů, palet atd.). V práci je modelována situace, kde ve virtuálním prostředí dělník nakládá kontejnery s předměty do nákladního prostoru, které mají být uloženy na pozice vypočtené pomocí algoritmů řešících 3D Bin Packing problém.

Praktická část této práce se zabývá různými fázemi realizace modelu ve virtuální realitě, který využívá data generovaná pomocí aplikace pro setřídění beden na základě zvolené strategie. Model byl postaven pomocí prostředí Unity3D. Výhodou tohoto prostředí je zobrazení setříděných a nesetříděných beden nejen pomocí monitoru, ale také možnosti zobrazení pomocí náhlavního displeje (pomocí virtuálních brýlí). Uživatel se díky využití VR rychleji zorientuje v systému uspořádání jednotlivých beden, tak dokáže intenzivněji vnímat i rozměry jednotlivých beden. V práci je také popsána vytvořená asistence ve virtuálním modelu, která má uživateli pomáhat při umísťování jednotlivých beden do jednotlivých pozic v prostoru. Uživatel má také možnost jednotlivé bedny uchopit a přemístit je podle svého uvážení. V modelu je implementována základní fyzika, proto je možné otestovat, zda vygenerované uložení beden je možné realizovat i v praxi.

Vytvořená aplikace by měla sloužit, jako podpůrná pomůcka pro zaučování pracovníků ve skladu, jakým způsobem umisťovat předměty do skladovacího prostoru. Aplikaci lze také využít jako studijní pomůcka, kde student po ukončení procesu přemístění a uskladnění beden, může vymýšlet různá zlepšení v oblasti ergonomie, či rozšiřovat funkčnost samotné aplikace ve virtuálním prostředí.

2 Logistika

Pro představu důležitosti logistiky v rámci českého průmyslu jsou uvedeny v Tabulce č.1-1 statistické údaje, které zobrazuje informace o množství přepraveného materiálu od roku 2010 až do roku 2019. Z dat v tabulce je patrné, že se během let objem přepravovaného materiálu a zboží se zvyšuje. Díky tomuto trendu přichází nové výzvy jak pro interní, tak externí podnikovou logistiku, kdy se se zvyšujícím objemem výroby kladou vyšší nároky na logistické procesy.

	2010	2015	2016	2017	2018	2019
Přeprava věcí celkem (tis. tun)	***451 671***	***549 085***	***539 063***	***570 976***	***593 761***	***618 819***
Železniční doprava	82 900	97 280	98 034	96 516	99 307	98 804
Silniční doprava	355 911	438 906	431 889	459 433	479 235	504 099
Vnitrozemská vodní doprava	1 642	1 853	1 779	1 568	1 374	1 735
Letecká doprava	14	6	6	6	5	4
Ropovody	11 205	11 040	7 356	13 453	13 839	14 177
Přepravní výkon celkem (mil. tkm)	***68 495***	***76 613***	***68 172***	***62 936***	***60 327***	***57 888***
Železniční doprava	13 770	15 261	15 619	15 843	16 564	16 180
Silniční doprava	51 832	58 714	50 315	44 274	41 073	39 059
Vnitrozemská vodní doprava	679	585	620	623	554	569
Letecká doprava	22	31	31	32	30	29
Ropovody	2 191	2 023	1 588	2 165	2 107	2 050

Tabulka 2-1 Mezioborové srovnání přepravních výkonů nákladní dopravy [6]

Logistika je jedním z hlavních faktorů, která sice nepřidává hodnotu produktu (pokud se nejedná o logistickou firmu), ale je nedílnou součástí výrobního procesu jako celku. Logistiku zcela eliminovat nelze, a proto je snahou co nejvíce snižovat náklady na ní vynaložené. V případě logistických firem je logistika hlavní pilířem činnosti firmy a tvoří její nezvratnou přidanou hodnotu. Při velkých objemech přepravovaného materiálu nebo zboží, je důležité efektivně pracovat s informacemi a proces nakládky efektivně řídit. Za tímto účelem se v logistických procesech využívá operační výzkum (Operational Research).

Logistické procesy by také měly být v souladu se skladováním jednotlivých předmětů (materiál, produkty, zboží atd.), tzn. při uskladnění v určitých částech podniku by měl brán zřetel na obrat daného předmětu. Musí být řízeno rozmístění předmětů ve skladovacích prostorech, kde předměty s vysokým obratem by se měli nalézat na dostupnějších místech atd. Zde byla nastíněna pouze část podmínek týkající se optimalizace procesů v oblasti logistiky, které by měly být brány v úvahu při rozhodování. Diplomová práce je zaměřena na proces uskladnění, kde se v rámci tohoto procesu rozhoduje o způsobu a místa uložení.

Je samozřejmé, že způsob přepravy se odvíjí od typu přepravovaného předmětu. V oblasti BPP jsou využívány pojmy jako „Bedna“ a „Kontejner“. Bedny zpravidla představují menší krabice, nebo kartony, které výrobce používá k zabalení zboží. Slouží jako ochrana, aby zboží nebylo nijak poškozeno. Kontejner zpravidla představuje ocelový kvádrový zásobník, který je postupně plněn bednami o různých velikostech. Rozměry kontejnerů se liší dle typu kontejneru, který je zapotřebí k uskladnění beden.

Problematika popisovaná v diplomové práci spadá do sekce tzv. „Packing Problem“ neboli problematiky balení. Problematika se zabývá prostorovou optimalizací, stabilitou, pořadím beden pro vykládku [28].

2.1 Bin Packing Problem

Pro uvedení do problematiky řešené v teoretické části diplomové práce je nutné v krátkosti popsat základní podstatu „Bin Packing Problem“ označovaného pomocí zkratky BPP.

Pro uvedení do problematiky řešené v této práci je nutné v krátkosti popsat základní podstatu „Bin Packing Problem“ označovaného pomocí zkratky BPP. Z hlediska matematického popisu BPP lze použít terminologii využívanou při ukládání předmětů do batohu, tzv. „Knapsack Problem“. V tomto problému je definován určitý počet předmětů (*n*) s neomezeným počtem kontejnerů (batohů). Kapacita kontejneru *C* představuje kladné číslo. Předmět je vložen do kontejneru, za podmínky, že předměty vložené do kontejneru nepřesahují jeho kapacitu. Zároveň počet využitých kontejnerů má být co nejmenší. Bednou se míní objekt obdélníkového tvaru. V případě trojrozměrného prostoru se může jednat o kvádr (či krychli). Bedna je definována rozměrem pro každou dimenzi. Při splnění daných podmínek je naložen do kontejneru. Ten je též obdélníkového (2D) či kvádrového tvaru (3D), avšak jeho rozměry jsou větší než rozměry předmětu (jinak by nebylo možné předmět vložit do kontejneru), viz *Obr. 2-1*. Práce je orientována na variantu BPP v trojrozměrném prostoru (3D), kde jsou jednotlivé bedny určeny šířkou w_i, výškou h_i a hloubkou d_i kde

$$i \in I = \{1, \dots, n\} \qquad (1)$$

Řešení je v mnoha případech velmi složité. Běžná řešení takového problému mohou vyústit do dvou scénářů, dle kterých je problém vyřešen. První scénář řešení je pomocí kompletního propočtu všech možných variant řešení optimalizačního problému. Druhým je nekompletní řešení, kde je využit některý např. heuristický algoritmus který zpravidla nalezne suboptimální řešení daného problému. Z tohoto důvodu je usilováno o zjednodušování BPP, nebo je snaha o úpravu jednotlivých heuristických metod vedoucí k efektivnějšímu nalezení řešení dle typu BPP. Zvolení správného uskladnění (kontejneru) je velmi důležité k dosažení dobré kvality logistických procesů. Z tohoto důvodu by se měla každá společnost snažit o využívání vhodně dimenzovaných kontejnerů s co nejmenším volným místem v každém z kontejnerů.

Z hlediska matematického popisu BPP lze použít terminologii využívanou při ukládání předmětů do batohu, tzv. „Knapsack Problem“. V tomto problému je definován určitý počet předmětů (beden) (*n*) s neomezeným počtem kontejnerů (batohů). Kapacita kontejneru *C* představuje kladné číslo. Předmět je vložen do jednoho kontejneru, za podmínky, že předměty vložené do kontejneru nepřesahují jeho kapacitu. Zároveň počet využitých kontejnerů má být co nejmenší.

Jedna z možných formulací BPP na základě problematiky batohu „Knapsack“ lze vyložit jako umístění předmětu do batohu (kontejneru) pomocí hmotnostního limitu batohu.

Matematická definice vypadá následovně [30]:

Minimalizace (kontejnerů): $z = \sum_{i=1}^{n} y_i$ (2)

- x_i je binární proměnná pro vkládaný předmět

Je předmětem:

$$\sum_{i \in I}^{n} w_j x_{ij} \leq C y_i, \ \mathrm{i} \in \mathrm{N} = \{1, \dots, n\} \qquad (3)$$

- C je kapacita každého kontejneru
- w_j je hmotnost předmětu
- y_i je binární proměnná pro kontejner s přiřazeným indexem

$$\sum_{j=1}^{n} x_{ij} = 1\,,\ \mathrm{j} \in N \qquad (4)$$

$$y_i = 0 \text{ nebo } 1, \mathrm{i} \in \mathrm{N} \qquad (5)$$

$$x_{ij} = 0 \text{ nebo } 1, \mathrm{i} \in \mathrm{N}, \mathrm{j} \in \mathrm{N}, \qquad (6)$$

, kde dle (5) y_i= 1 pokud je kontejner použit a dle (6) x_{ij} = 1 pokud je bedna s indexem *j* vložena do kontejneru s indexem *i.*

Je předpokládáno, že *C* je kladné číslo (v tomto případě hmotnostní kapacita) a v případě započtení váhy lze uvést:

$$w_j \leq C, \forall j \in N \qquad (6)$$

Podstatu celého problému lze ilustrovat na nakládání palet na nákladní vůz. Každá paleta je závislá na typu beden, které jsou na ni přepravovány. Paleta tento předmět (bednu) může obsahovat po dobu manipulace, skladování a přepravy. Přepravní bedny jsou běžně vyráběny z kartonu, plastu, kovu nebo dřeva v závislosti na druhu přepravy. V tomto případě může nastat situace, že je nutné naložit několik předmětů (beden) o různých velikostech do předem určených skladovacích objektů (palety, stojany, obaly atd.) ve vymezeném prostoru. Cílem je co nejvíce zmenšit počet kontejnerů (skladovacích jednotek), který je zapotřebí pro uskladnění veškerých předmětů. V rámci společnosti se však problém vyskytuje při skladovacích procesech ale i u dalších odvětvích například při plánování procesů, ve výrobě, expedici, nebo i v administrativní složce. V každém případě se však problém liší svými omezeními [38].

Důležitým faktorem v rámci BPP je počet beden a počet druhů beden, které se mají skladovat z důvodu možnosti uložení veškerých předmětů (beden). Pokud společnost disponuje několika druhy beden, různé druhy beden o různé velikosti umožní vyplnit potřebné prostory a zvyšuje se tím efektivnosti nakládky. Problém tohoto přístupu je v potřebě větší plochy pro skladování různých druhů a s tím spojené náklady [28]. V této práci je počet beden stanoven ze vstupních dat, kapitola **Chyba! Nenalezen zdroj odkazů.**.

Obr. 2-1 Grafická vizualizace BPP pro nakládku auta [22]

Pokud se v řešeném problému vyskytuje více než jeden druh předmětu, je zapotřebí pro jednotlivé druhy stanovit charakteristiku dle jednoho z atributů beden (cena, váha, rozměry atd.). Je možné, že bedny mohou být v reálném prostředí náchylné k poškození během

přepravy, a proto je nutné zavést patřičná omezení v rámci bezpečnosti, než jak je tomu v rámci simulací ve virtuální podobě [38].

Při řešení BPP je snahou splnit jednotlivá omezení pomocí různých faktorů např.[21]:

- Typu palet
- Typu algoritmu
- Metodiky BPP
- Nákladů spojených s typem dopravy
- Časem pro výpočet
- Způsobem rozhodování
- Specifikací přepravovaného předmětu

Náklady spojené s logistikou se mohou týkat například typu přepravy a prostředků k ní určené. Zda se jedná o přepravu pomocí nákladního automobilu, lodě či jiného přepravního prostředku. Záleží také na celkových rozměrech a váze nákladu, podle kterého se volí vhodná přeprava a následně i způsob uložení nákladu. V případě nákladu náchylného na poškození nebo nákladu netradičního tvaru se zvedá šance, že výchozí přepravní podmínky jsou nedostačující a je nutné se uchýlit k použití konkrétnějších metod a přípravků pro zajištění bezproblémové přepravy.

V případě jednoho z faktorů způsobu rozhodování, je logika volby vhodného řešení člověkem zodpovědným za daný logistický proces. Pokud neexistuje v podniku standard pro způsob nakládání beden do skladových prostor, je zapotřebí přítomnost člověka, který má tacitní znalost v rámci daného problému. V této situaci je snahou převést tuto znalost do podoby explicitní, kde systematika rozhodnutí je srozumitelná a je uchována pro další užití. Tacitní znalost navíc nezaručuje, že použitý přístup je přístupem optimálním.

Pro BPP existuje několik variant, které se dělí dle svého zaměření. Jedná se o zaměření, která udávají prioritu (vyšší váhu) některému z omezení. Může se jednat např. o:

- Rozměry beden (dimenze X, Y,Z)
- Jejich tvar
- Celkový počet kontejnerů (palet, krabic pro uskladnění atd.) do kterých mají být uloženy bedny.
- Dimenze, ve které je problematika BPP řešena – jednodimenzionální BPP (1D BPP), dvoudimenzionální BPP (2D BPP), třídimenzionální BPP (3D BPP), Knapsack, Cutting stock, BPP s variabilními rozměry, Dimenzionální BPP

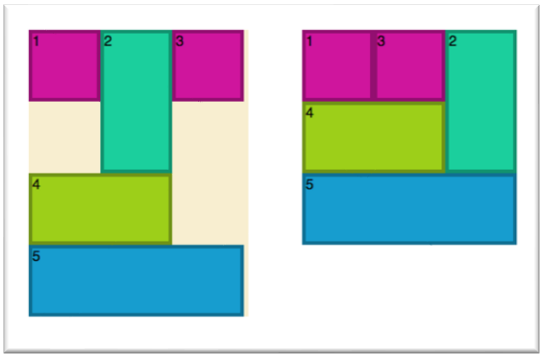
Práce je orientována zejména na dimenzi BPP s definovanými rozměry beden, jakými jsou šířka, výška bedny a její hloubka.

2.1.1 1D BPP

V případě jednodimenzionální varianty, objekty mají pouze jeden „rozměr“, kterým může být cena, šířka bedny, čas apod. Proto se v této variantě optimalizuje pouze jeden parametr bedny. Nejčastěji je optimalizován využitý prostor dle hloubky, nebo šířky bedny, který je vkládán do úložného prostoru s určitými parametry. Cílem je naložit všechny bedny do co nejmenšího počtu kontejnerů (úložných prostor) a co nejmenším plýtváním volným prostorem v kontejneru, přičemž se zkouší přístupy dle různých druhů algoritmů [21].

2.1.2 2D BPP

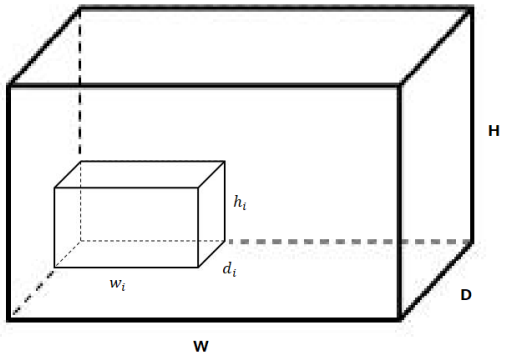
V případě dvoudimenzionální varianty je operováno se dvěma libovolnými „rozměry“. Typickým příklad pro 2D BPP je řešení snižování počtu kontejnerů pomocí různé orientace uložení obdélníků reprezentující bedny s předměty [21].



Obr. 2-2 Příklad 2D BPP na bednách [21]

2.1.3 3D BPP

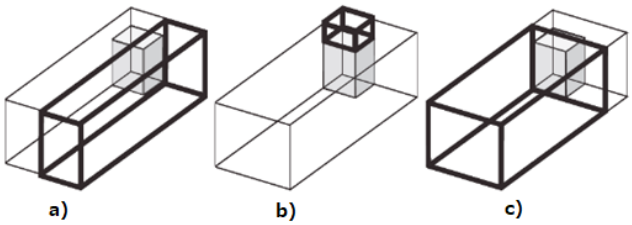
Práce je orientována na variantu 3D BPP. Tato verze zahrnuje optimalizaci zohledňující všechny 3 rozměry – šířku, výšku a hloubku.



Obr. 2-3 Rozměry pro bednu (předmět) a kontejner

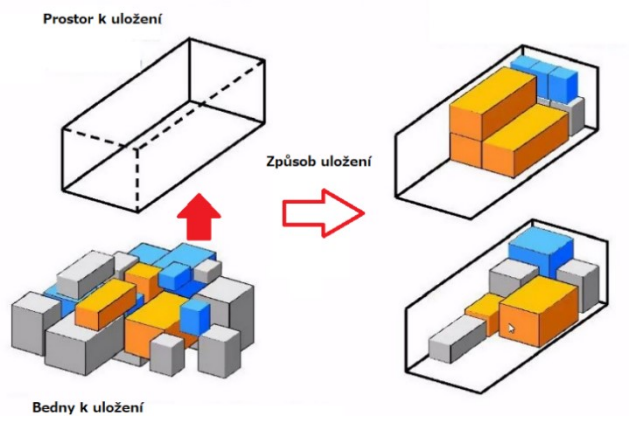
Bedny jsou určeny šířkou w_i, výškou h_i a hloubkou d_i kde i značí pořadové číslo bedny, tj. $i \in I = \{1, \dots, n\}$. Rozměry kontejneru (úložného prostoru) jsou šířka W, výška H a hloubka D. Předpokládá se, že $w_i \ll W, h_i \ll H, d_i \ll D$ kde $i \in I$. Trojrozměrná verze je běžně aplikovatelná pro problematiku nakládání a přepravy předmětů. Základní roli ve zvolené metodice hraje způsob rozdělení prostoru v kontejneru a způsob uložení beden kontejneru („Cutting“ a „Packing“ problémy). Důležitou úlohu také hraje seskupování malých a velkých předmětů vedoucí k upravení metodiky nakládky předmětů.

Na *Obr. 1-4* je též možné vidět možnosti tvorby nově vzniklých míst k uložení ve 3D v daném úložném prostoru (kontejneru) [50]. V této práci jsou brány v potaz pouze předměty o tvaru kvádru, popřípadě krychle v 3D prostoru.



Obr. 2-4 Vznik volných míst v úložném prostoru při vložení předmětu (bedny) [37]

Metodika zvolená pro uložení beden v úložném prostoru (kontejneru) může brát v potaz různé cíle, např. co největší homogenizaci kontejneru, tzn., že pokud jsou v kontejnerech skladovány stejné bedny, je pravděpodobné, že jsou využity kontejnery stejného tvaru a velikosti. Tato metodika rozděluje skupinu použitých kontejnerů na homogenní nebo heterogenní. V případě, že je metodika uložení zaměřena především na dosažení maximálního počtu beden naložených v prostoru kontejneru je důraz kladen především na naplnění kontejneru vhodnými tvary beden, viz *Obr. 2-5*.



Obr. 2-5 Cíl metodiky ukládání předmětů do úložného prostoru

Při této variantě je možné, že některé bedny nemusí být vůbec uloženy a ponechány mimo kontejner. V druhém případě se metodika zaměřuje na snížení celkového počtu kontejnerů nutných k naložení veškerých beden k uložení [21]. Při minimalizaci počtu požitých kontejnerů, nebo uložení nákladu s nejvyšší hodnotou (bodová hodnota dle váhy, ceny atd.), mohou být řešeny následující problematiky [50]:

- Problematika rozdělení jednoho druhu předmětu („Single Stock-Size Cutting Stock Problem“ – SSSCSP) – pokud jsou všechny kontejnery identické a bedny jsou mírně heterogenní.
- Problematika rozměru jedné bedny („Single Bin-Size Bin Packing Problem“ - SBSBBP) – pokud jsou kontejnery stejné a bedny jsou heterogenní.
- Problematika několika rozdělení více druhů předmětů („Multiple Stock-Size Cutting Stock Problem“ - MSSCSP) Pokud jsou kontejnery mírně heterogenní. To stejné platí pro bedny.
- **Problematika BPP s bednami o různých rozměrech** („Multiple Bin-Size Bin Packing Problem“ - MBSBPP) – pokud kontejnery jsou v rozměrech mírně heterogenní a bedny jsou heterogenní.

- Problematika rozdělení zbytkových předmětů („Residual Cutting Stock Problem“ - RCSP) – pokud jsou kontejnery heterogenní a bedny mírně heterogenní.
- Problematika zbytkového BPP („Residual Bin Packing Problem“ - RBPP) – pokud jsou kontejnery a bedny v rozměrech heterogenní.
- Problematika balení identického předmětu („Identical Item Packing Problem“ - IIPP) – pokud existuje jeden kontejner a bedny jsou všechny stejné.
- Problematika umístění jednoho velkého objektu („Single Large Object Placement Problem“ - SLOPP) – pokud existuje jeden kontejner a bedny jsou lehce heterogenní.
- Problematika jednoho batohu - („Single Knapsack Problem“ - SKSP) – pokud existuje jeden kontejner a bedny jsou heterogenní.
- Problematika umístění několika velkých identických objektů („Multiple Identical Large Object Placement Problem“ - MILOPP) – pokud existuje více stejných kontejnerů a bedny jsou mírně heterogenní.
- Problematika umístění více velkých heterogenních předmětů („Multiple Heterogeneous Large Object Placement Problem“ - MHLOPP) – pokud jsou kontejnery mírně nebo více heterogenní a bedny jsou mírně heterogenní.
- Problematika mnohonásobného identického batohu („Multiple Identical Knapsack Problem“ - MIKSP) – pokud existuje několik identických kontejnerů a bedny jsou heterogenní.
- Problematika mnohonásobného heterogenního batohu („Multiple Heterogeneous Knapsack Problem“ - MHKSP) – varianta pracuje s mírně a silně heterogenními kontejnery a s heterogenními bedny.

Všechny uvedené problematiky jsou založeny na trojrozměrnosti kontejnerů. V potaz by měla být brána i problematika, která je založena na 2D rozměru s jednou variabilní složkou v podobě šířky, výšky nebo hloubky kontejneru. Jedná se o problematiku minimalizace při existenci jednoho kontejneru, který popsal [28] jako problematiku otevřené dimenze („Open Dimension Problem“).

Problematika batohu (Knapsack)

Knapsack metodika uložení předmětů, v tomto případě do batohů, je založena na ceně nakládaných předmětů a je úzce spjata s problematikou 3D BPP. Cílem je naložit do jednoho kontejneru takovou skupinu předmětů, která má nejvyšší celkovou hodnotu předmětů. Cena předmětu může být stanovena v různých formách, například jako váha, či objem předmětů. Pokud se zaměřuje na objemovou stránku, je cílem minimalizovat nevyužité místo mezi předměty v kontejneru [12].

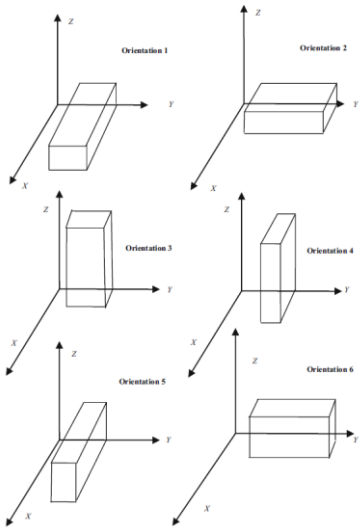
DP je zaměřena na problematiku BPP, jejímž cílem je uložení beden o různých rozměrech do úložného prostoru. Problematika byla vybrána z důvodu vstupu předmětů o nehomogenních rozměrech.

2.1.4 BPP omezení

Metoda pro uložení předmětů (bedny) u BPP se řídí dle toho, jak jsou definována omezení. Pokud jsou rozměry úložných prostor homogenní vzhledem k předmětům (bednám) v nich uložených, je pravděpodobné, že nastane situace, kdy je obtížné najít optimální polohu pro rozložení nákladu v úložném prostoru. V případě zahrnutí omezení, je metoda nazývána BPP s omezeními.

Nejvýznamnější omezeními jsou [21]:

- Rozložení
- Překrývání jednotlivých beden
- Váha
- Únosnost
- Bezpečnost – omezení proti narážení
- Stabilita
- Palety
- Způsob zasílání
- Datum dodávky
- Náklady na přepravu.



Obr. 2-6 Ukázka možnosti rotace předměty [29]

Všechny tyto mantinely podstatnou vahou ztěžují výpočet optimálního umístění, a proto je vynaložena snaha mantinely co do největší míry odstraňovat. Se složitostí omezení, roste i čas potřebný k výpočtu. Pokud je řešen problém v logistice, zejména ve skladu, je nutné dosáhnout přijatelného výsledku v krátkém čase. I přestože určitá metodika dosáhne přijatelného výsledku, nemusí vyhovovat svojí časovou náročností výpočtu. Proto je nutné vybrat vhodnou metodiku BPP pro řešení daného problému. Velký vliv na výsledek má pravidelnost nebo vzor, jakým se bedny ukládají do kontejneru. Jedná se o sekvenci ukládání beden, při které je dosaženo uspokojení požadavků všech omezujících faktorů. Velkou roli při přepravě hrají prostory k uložení (palety, kontejnery, krabice atd.), které jsou využity k snadnému provedení logistického procesu se snadnou manipulací. Dodatečně plní roli při stabilizaci, a navíc při určitých situacích uskladnění lze dle kontejneru či jiného skladovacího prostředku rozeznat typ uskladněných předmětů (specifické předměty uložené ve specifickém kontejneru) [21].

Pro 3D BPP jsou definována 3 základní omezení, které jsou důležité pro podstatu problematiky [50]:

- Předměty mohou být pouze umístěny jejich okraji paralelně k ploše zdi kontejneru
- Všechny bedny musí být umístěny uvnitř kontejneru
- Předměty se nesmí překrývat

3 Algoritmy

Pro BPP může být přijatelné řešení nalezeno pomocí různých typů algoritmů. V případě užití algoritmů se jejich účinnost odhaduje pomocí tzv. „nejhoršího scénáře“ („Worst Case Scenario“ - WCS) vzhledem k minimalizaci počtu úložných prostor (kontejner, paleta atd.) k uskladnění předmětů. Příklad může být daný počet předmětů (beden) *n* a optimum pro použitý seznam předmětů *I* s označením OPT(*n*). Vychází tak rovnice [38]:

$$WCS = \frac{n}{OPT(n)} \tag{7}$$

Algoritmy lze rozdělit podle heuristiky na algoritmy odhadové, které se dělí na „online“ a „offline“ algoritmy. V případě online heuristiky předměty vstupují do výpočtu v daném pořadí. Pro tento typ algoritmu musí být položena otázka, kam předmět uložit v rámci nákladového prostoru a zda vůbec daný předmět je uložen. Offline algoritmy naopak již obsahují informaci ohledně pořadí. Je nutné tedy znát celý seznam předmětů, které vstupují do výpočtu. Odhad dle offline heuristiky je přesnější než online verze, avšak s nevýhodou v podobě delších výpočetních časů a složitosti algoritmů [21].

3.1 Online algoritmy

Online algoritmy (OnA) jsou aplikovány pro problematiky v rámci interaktivních výpočtů. OnA pracují se vstupními daty, která se v čase mohou měnit, a proto jsou data přijímaná po dávkách, na které systém OnA musí reagovat. Vstupní data do OnA nejsou dopředu známa a algoritmus neprovádí žádné odhady ohledně dat, která mají být obdržena. Funkcionalita je založena na přijímání sekvence úkolů, které musí být řešeny v takovém pořadí, v jakém byly přijmuty v rámci dané sekvence. Výkonnost OnA je obvykle vyjadřováno pomocí kompetitivní analýzy („Competetive Analysis“) [10].

3.1.1 Next Fit

Algoritmus posoudí, zda se aktuální předmět (bedna) dá vložit do stejného kontejneru (úložného prostoru), jako předmět předchozí. Pokud se aktuální předmět nevejde do aktuálního kontejneru, je použit („otevřen“) kontejner nový (v případě stojanu nové patro). To znamená, že v této metodice se nelze vracet k předchozím kontejnerům a vkládat do nich nadále předměty, i když v předchozích kontejnerech by předmět uložen být mohl. K použití metody je zapotřebí pouze konstantní množství pracovní paměti. Výsledky tímto přístupem nejsou příliš efektivní [10].

Z pramene [10] byl pro „NextFit“ algoritmus použit pseudokód, ze kterého si lze představit logiku s jakou algoritmus pracuje.

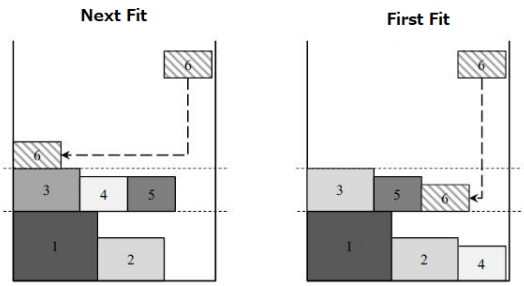
Pseudokód pro algoritmus Next Fit [10]:

- m – celkový počet doposud otevřených kontejnerů
- R – zbývající volné místo v nejaktuálněji otevřeném kontejneru
- x – indexovaný předmět (bedna) (v případě [10] se jedná o váhu předmětu)
- j – index bedny
- σ – výstup ve formě posledního otevřeného kontejneru při ukládání bedny s indexem "*j*"
- n – počet beden k uložení

```
procedure NextFit
    m ← 0
    R ← 0
    while j ≤ n do
        if x_j < R then
            m ← m + 1
            R ← 1 − x_j
        else
            R ← R − x_j
        σ(j) ← m
        j ← j + 1
```

Pseudokód zpočátku deklaruje počáteční hodnoty proměnných „m" a „R". V cyklu, kde je v podmínce kontrolován index bedny vůči celkovému počtu beden k umístění. Pokud je splněna podmínka cyklu, je přistoupeno k testování druhé podmínky, kde je kontrolována bedna s indexem „j" (objem) vůči volnému prostoru v aktuálně otevřeném kontejneru „R". Pokud je splněna tato podmínka je inkrementováno „m" a „R" je zmenšeno o objem vložené i-té bedny „x_j". V případě že podmínka není splněna, je otevřen nový kontejner (bednu nebylo možné uložit). Výsledkem spuštění algoritmu je poslední otevřený kontejner (do kterého byla vložena bedna „x_j")..

V následujícím obrázku (Obr. č. 3-1), je možné vidět systematiku ukládání v rámci dvojrozměrného prostoru v porovnání s variantou „First Fit", která je popsána v následující podkapitole 3.1.2. Z obrázku je patrné, že pokud je cílem využití co nejmenšího prostoru (pater) k ukládání beden, je výhodnější využít variantu „First Fit". Pro 3. dimenzi může být -



Obr. č. 3-1 Ukázka 2D verze způsobu ukládání dle Next Fit a First Fit online algoritmů

3.1.2 First Fit

Algoritmus vkládá předmět do prvního kontejneru, ve kterém je nalezen dostatečný prostor k uložení aktuálního předmětu. Nový kontejner je použit pouze v případě, že v předchozích kontejnerech není pro aktuální předmět místo. Algoritmus si tak vede seznam o otevřených kontejnerech. To také znamená, že algoritmus musí pokaždé při přiřazování předmětu kontejneru začínat kontrolu dostupnosti od prvního kontejneru, který byl otevřen. Pokud existuje více míst, kam lze předmět uložit, je vybráno místo co nejblíže dnu kontejneru [10].

Proměnné algoritmu jsou obdobné jako u algoritmu „Next Fit“ s výjimkou „R“, kde nyní tato proměnná zastupuje mapu zbývajících volných míst ve všech otevřených kontejnerech. Dále byl přidán index “*i*” pro procházení všech doposud otevřených kontejnerů. Navíc uvnitř pseudokódu je deklarována proměnná „flag“ pro kontrolu možnosti uložení bedny „x_j“ do volného prostoru kontejneru „R[*i*]“.

Pseudokód pro přístup „First Fit“[10]:

```
procedure FIRSTFIT
    m ← 0
    R — mapa zbývajících volných míst ve všech otevřených kontejnerech
    while j ≤ n do
        flag ← False
        for i = 1 to m do
            if x_j < R[i] then
                R[i] ← R[i] − x_j
                σ(j) ← i
                flag ← True
                break
        if flag = False then
            m ← m + 1
            R[m] ← 1 − x_j
            σ(j) ← m
        j ← j + 1
```

Prvním krokem v pseudokódu je inicializace proměnných. Pokud je podmínka cyklu splněna (kontrola indexu bedny vůči celkovému počtu beden k uložení), je přiřazena booleanovská hodnota nepravda proměnné „flag“. Následuje procházení všech otevřených kontejnerů pomocí cyklu s krokem „for“ a pokud je u kontejneru s indexem „*i*“ splněna podmínka možnosti uložení bedny do volného prostoru kontejneru, je sníženo „R“ (volné místo) daného kontejneru o objem bedny „x_j“, řešení je přiřazen index posledního otevřeného kontejneru „*i*“ a proměnné „flag“ je přiřazena hodnota „true“ pro vyznačení úspěšného uložení. V případě nemožnosti uložení bedny do žádného z otevřených kontejnerů („flag“ zůstane „false“), je otevřen kontejner nový a bedna je uložena do nového kontejneru (index „m“ je inkrementován) a volné místo „R[m]“ nabývá hodnoty celkového místa kontejneru

mínus objem vložené bedny „x_j“. Řešení (σ_j) nabývá hodnoty „m“. Před koncem cyklu je index bedny „j“ inkrementován.

3.1.3 Best Fit

Best Fit algoritmus operuje na podobném principu algoritmu First Fit. Rozdílné chování algoritmu je dáno v zachování úložných prostor (kontejnerů, pater ve stojanu atd.), kde algoritmus setřídí tyto prostory v sestupném pořadí dle součtu velikostí aktuálně vložených předmětů v daném prostoru [20].

Proměnné uvedené v pseudokódu pro algoritmus „Best Fit“ jsou obdobné s proměnnými pseudokódu pro algoritmus „First Fit“. Rozdíl je ve využití proměnné „*ind*“, která slouží pro kontrolu volného místa v kontejnerech a případně na základě této proměnné je otevřen nový kontejner. Pseudokód pro přístup „Best Fit“ [10]:

```
procedure BestFit
    m ← 0
    R — : mapa zbývajících volných míst ve všech otevřených kontejnerech
    while j ≤ n do
        ind ← −1
        for i = 1 to m do
            if x_j < R[i] then
                if ind = −1 or R[i] < R[ind] then
                    ind ← i
        if ind = −1 then
            m ← ind ← m + 1
            R[m] ← 1
        σ(j) ← ind
        R[ind] ← R[ind] − x_j
        j ← j + 1
```

Na začátku pseudokódu jsou deklarovány proměnné včetně inicializace. Následuje kontrola podmínky cyklu, kde je porovnána hodnota indexu aktuální bedny vůči celkovému počtu beden k umístění. Za předpokladu splnění podmínky je deklarována proměnná „*ind*“ a je ji přidělena počáteční hodnota. Následuje procházení seznamu otevřených kontejnerů, kde je kontrolováno, zda je možné aktuální bednu k umístění (x_j) uložit do volného místa kontejneru s indexem „*i*“. Následuje kontrola podmínky, kde je kontrolována hodnota proměnné „*ind*“ na počáteční hodnotu, nebo zda je volné místo „R[*i*]“ menší než volné místo „R[*ind*]“. Pokud jedna z podmínek je splněna, je přiřazena hodnota indexu „*i*“ proměnné „*ind*“ (znamená možnost vložení do jedné z otevřených beden). Pseudokód pokračuje podmínkou pro kontrolu hodnoty proměnné „*ind*“. V případě nezměněné hodnoty proměnné (proměnná je stále přiřazena počáteční hodnota), je celkový počet kontejnerů inkrementován a tato hodnota je také přiřazena do proměnné „*ind*“. Volnému místu v posledním kontejneru („R[m]“) je přidělena hodnota a výsledné proměnné σ(j) je přiřazena hodnota proměnné

„*ind*“ (číslo celkového počtu otevřených kontejnerů). Nakonec je sníženo volné místo nově otevřeného kontejneru o objem bedny „x_j“ s následnou inkrementací indexu „j“.

3.1.4 Worst Fit

Algoritmus přiřadí předmět prvnímu nejprázdnějšímu kontejneru, který již obsahuje nějaké předměty (je otevřen). Nový kontejner je využit pouze tehdy, pokud předmět nelze vložit do žádného z používaných kontejnerů. V případě shody volného místa v nejprázdnějších kontejnerech je předmět vložen do dříve otevřeného kontejneru [27].

3.2 Offline algoritmy

Oproti OnA, offline algoritmy (OfA) pracují s předem známými vstupními daty. Od OfA je požadováno zpracování veškerých dat a vykázání požadovaného výsledku v co nejmenším čase [19].

Next Fit sestupně

Algoritmus setřídí předměty dle určitého parametru v sestupném pořadí. Je zvolen parametr (v případě rozměrů jedna dimenze), dle kterého se metoda řídí, například šířka předmětu w_i a předměty jsou setříděny sestupně dle šířky a uloženy v seznamu. Po setřídění předmětů je setřídění seznam podroben metodě Next Fit [27].

First Fit sestupně

Algoritmus setřídí předměty v sestupném pořadí dle určitého parametru a použije metodu First Fit [27].

Worst Fit sestupně

Algoritmus setřídí předměty v sestupném pořadí dle určitého parametru a použije metodu Worst Fit [27].

3.3 Heuristika

V každém heuristickém přístupu k řešení BPP existuje mechanismus pro rozhodování o tom, jak umístit předmět (bednu) do úložného prostoru (kontejner, paleta atd.), ať už jde o generování počátečního řešení, nebo o integrální část přístupu (vstup do již existujícího stavu řešení). Pokud jsou bedny slabě heterogenní, pak nejběžnějším přístupem je metoda „Stavění zdi“ („Wall Building“), nebo „Vrstvení“ („Layer Building“). Ve velmi heterogenním případě je upřednostňováno umisťovat bedny po jedné. Pro všechny proveditelné pozice umístění je vytvořen seznam volných pozic. Jakmile se do pozice dá umístit zeď, nebo vrstva beden ze zmíněných dvou přístupů, vygenerují se nové pozice k umístění [50].

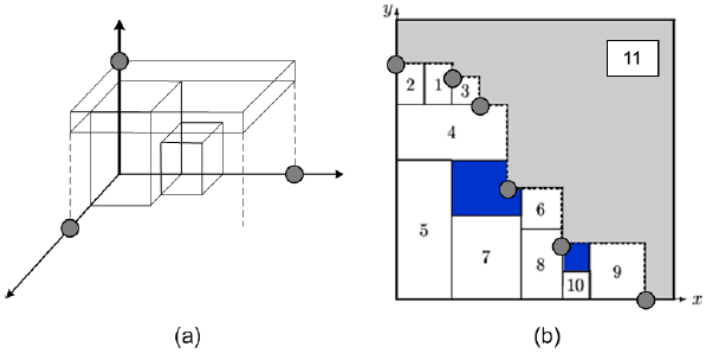
3.3.1 Umístění předmětů

Pro efektivní umístění většího množství předmětů do kontejnerů pomocí heuristického přístupu je velice důležité klást důraz na způsobu volby umístění. Předmět do prostoru k uložení může být vkládán dle jedné z metod, viz kapitoly 3.3.1.1 a 3.3.1.2. V kapitolách jsou uvedeny metody, které se zakládají na umístění předmětu dle orientačních bodů.

3.3.1.1 Rohové body

V [48] jsou rohové body („Corner Points“ – CP) popsány, jako rohy strany nakládaných předmětů, dle kterých je ověřováno okolí v prostoru pro uložení předmětů a zda je možné provést uložení na kontrolované místo (okolí) či nikoliv. V rámci 2D CP jsou předměty umisťovány svým levým dolním rohem do levého dolního rohu kontejneru, nebo do levého dolního rohu vzniklého vložením předchozího předmětu. V případě 2D se body vytváří v souřadnicích, kde se mění rozměry předmětu z šířky na výšku a opačně, viz *Obr. 3-2 (b)*. V případě 3D lze CP nalézt pomocí 2D algoritmu na základě výšky kontejneru dle spodní a horní hrany každého z předmětů, viz Obr. 3-2 *(a)*.

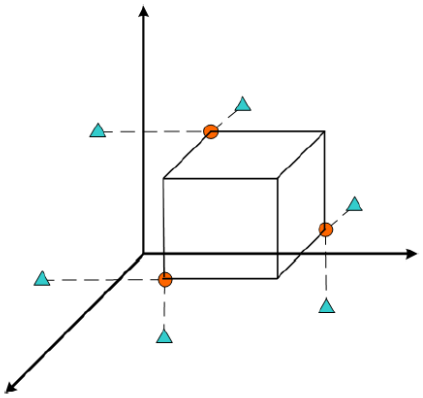
Na základě principu CP [48] vytvořili metodiku umisťování nazvanou „Branch & Bound“ (B&B), která ověřuje, zda je možné veškeré předměty naložit do daného kontejneru. V B&B je díky CP zmenšen počet řešení o méně výhodná řešení, což usnadňuje nalezení optimálního řešení.



Obr. 3-2 Ukázka pravidla rohových bodů [13]

3.3.1.2 Extrémní body

Pravidlo extrémních bodů („Extreme Points“ – EP) je aplikovatelné jak pro 2D, tak i pro 3D BPP. EP zohledňují časovou stránku výpočtu a zároveň se snaží o úsporu místa v kontejneru. Pravidlo je založeno na principu CP, viz kapitola 3.3.1.1. Pravidlo EP funguje na principu ukládání předmětů do kontejneru dle rozměrů a tvarů již vložených. Princip samotného EP spočívá v uložení předmětu *n* s rozměry w_i, h_i a d_i do kontejneru levým dolním rohem na pozici (x_i, y_i, z_i). Následuje vygenerování série nových potencionálních EP, dle kterých může být umístěn další předmět ze seznamu. Nový EP je vygenerován pomocí projekce bodů o pozicích ($x_i + w_i$, y_i, z_i), (x_i, $y_i + d_i$, z_i) a (x_i, y_i, $z_i + h_i$) na ortogonální ose kontejneru, viz *Obr. 2-2 [13]*.



Obr. 3-3 EP určeny pomocí předmětu (trojúhelníky) [13]

V [26] autor využil a modifikoval metodu extrémních bodů, kde předměty byly uloženy do kontejnerů v určitém pořadí na základě extrémních bodů. Metodika je založena na umístění předmětu o rozměrech w_i, h_i a d_i do prázdného kontejneru na pozici (0, 0, 0). Dále jsou vygenerovány 3 extrémní body pro šířku, výšku a hloubku vkládaného předmětu.

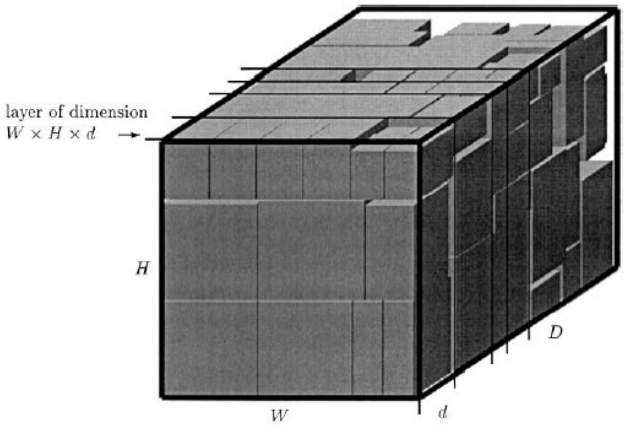
3.3.2 Vrstvení

Vrstvení („Layer Building“) se běžně využívá při řešení umístění v prostoru v rámci heuristiky. Layer Building spadá pod kategorii odhadových algoritmů („Approximation Algorithm“). Algoritmus tvoří řady z předmětů k uložení. Každá řada může být orientována dle šířky, nebo délky kontejneru. Předměty v řadě mají stejnou jednu souřadnici (x, y nebo z). Systematika postupuje vkládáním předmětů od dna kontejneru, kde vytvoří základní vrstvu („Level“). Vrstva je tvořena z řad předmětů orientovaných příčně či podélně vůči kontejneru. Výška jedné vrstvy je dána výškou nejvyššího předmětu ve vrstvě. Při naplnění vrstvy je vytvořena další vrstva nad předchozí, a to pokračuje až do výšky kontejneru [50]. Vrstvy mohou být zformovány do „Stacku“ – dotýkající se vrstvy předmětů naložené podél výšky kontejneru. Požadovaná celková dimenze Stacku je rovna sumě dané dimenze všech vrstev obsažené ve Stacku. Dimenze Stacku je limitována dimenzí kontejneru. Stack musí být postaven od dna kontejneru a postupovat podél dané dimenze kontejneru (například dle výšky H). Pro „Vrstvení“ je dáno omezení v rámci velkosti jednotlivých Stacků, kde kromě poslední vrstvy v kontejneru, musí všechny vrstvy splňovat podmínku rozměrů pro možnost připojení další vrstvy či rovnou Stacku. Algoritmus se zabývá, jakým způsobem vygenerovat seznam vrstev s předměty a jakým způsobem vybrat volné místo a vložit zvolenou vrstvu [37].

V rámci „Vrstvení“ řešitel čelí dvěma hlavním problémům, které ve výsledku prokážou výslednou efektivitu systému nakládání. Prvním problémem je účinné plnění dimenze (v případě vertikální výšky), kde je snaha skládat do jedné vrstvy předměty o podobné výšce. Druhým problémem je vyřešení horizontálního plnění vrstev (dimenze šířky a hloubky). V tomto případě je k dispozici přístup „Nejdřív výška, poté plocha“ („Height first Area second“). Princip metody se zakládá na určení spodní plochy předmětu, jako plochy první vrstvy a výšky nejvyššího předmětu přítomného v dané vrstvě, jako počátek vrstvy druhé [25].

3.3.3 Stavění zdi

Princip stavění zdi („Wall Building“ -WB) se zakládá na naplnění prostoru k uložení předmětů několika vrstvami podél hloubky kontejneru D. V publikaci [34] se uvádí, že jsou brány v potaz pouze vrstvy s určitou hloubkou *d,* která se rovná některému z rozměru předmětů (w_i, d_i, h_i). I přes splnění podmínky se samotná hloubka vrstev musí dobře zvolit, jelikož se na tomto rozměru zakládá výkonnost celého výpočtu. Když je vytvořena nová vrstva, je podrobena bodovacímu systému, dle kterého se stanovuje hloubka prostoru zbývajících předmětů k uložení. Upřednostňuje se taková bedna, která má největší rozměr hrany (W, H) u nejmenší dimenze (X, Y, Z) ze všech jejích dimenzí. Výběr je odůvodněn tím, že by se předmět s takovými rozměry hůře nakládal v pozdější fázi procesu uložení. V tomto případu musí být povolena rotace, kde je s předmětem otáčeno za účelem dosažení maximální hloubky předmětu d_i a celková hloubka vrstvy tak je $d = d_i$.



Obr. 3-4 Vizualizace rozměru hloubky vrstvy u WB [34]

Pokud byla určena hloubka vrstvy, pak stěna je naplněna dle přístupu „Greedy Way“ – je volena pouze lokálně optimální varianta uložení v podobě horizontálních pruhů, viz *Obr. 3-4*. Do každého pruhu jsou ukládány předměty s nejvyšším obodováním (čím větší rozměr, tím vyšší bodové hodnocení). V případě shod v bodovém systému se rozhodne na základě typu předmětu a velikosti největšího rozměru zvolené dimenze předmětů se stejným obodováním [34].

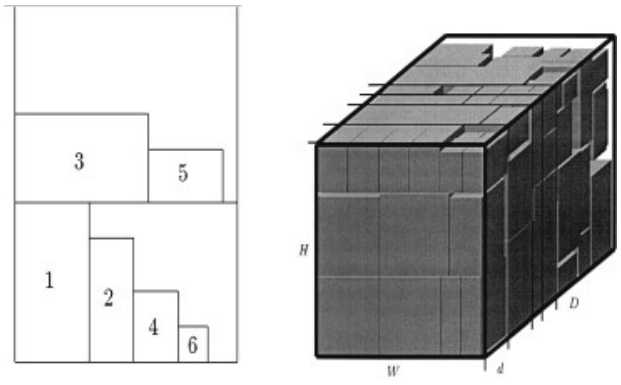
3.3.4 Policové algoritmy

Stejně tak jako „Layer Building“, je policový algoritmus („Shelf Algorithm“) využíván k umisťování předmětů do kontejnerů [23]. Algoritmus pro 3D BPP vychází z verze algoritmu pro 2D BPP. V tomto případě jsou vytvořeny police pro ukládání předmětů v kontejneru. Police je obdélníkového tvaru o určité výšce a šířce, kde se tyto rozměry stanovují dle šířky předmětů ukládaných na polici. Problematika je tímto způsobem zjednodušena z 2D na 1D problém. Pro 3D BPP se vytvoří 2D verze police a dále pomocí zvoleného 1D algoritmu jsou umístěny do trojrozměrného kontejneru, kde jedna z dimenzí předmětu je stejná s výškou police v kontejneru H. Pokud byl použit 2D algoritmus pro rozmístění polic, jednalo by se o přístup „Wall Building“. Nevýhodou přístupu je nevyužití možných variant řešení (možná existence vhodnějšího řešení) v případě využití gilotinového algoritmu pro výpočet nově vzniklého

volného místa uvnitř kontejneru po vložení aktuálního předmětu jež se běžně využívá. Zde mohou nastat situace, kdy není plně využit prostor na polici ať už v dimenzi hloubky, nebo výšky kontejneru [15].

Police jsou očíslovány ode dna kontejneru. Následně jsou předměty v určeném směru umisťovány na police, dle některých z algoritmů uvedených v *kapitole 2.1* a *2.2.* V případě povolení rotace předmětů je cíleno na maximální využití volného místa mezi bednami a dnem police o patro výše (v případě poslední police stropu kontejneru). Pro 3D BPP platí, že se dle orientace nakládání předmětů hledá minimum mezi rozměry předmětů (bedny s rozměry w_i, h_i a d_i). Samozřejmě zde imponují subjektivní případy, jako například způsob vkládání předmětů do poslední police k dosažení úspory místa. Mezi uloženými předměty na nejvyšší polici a stropem kontejneru vznikají nevyužité prostory, které lze hůře využívat z důvodu nemožnosti nastavení výšky posledního patra

V rámci užití policových algoritmů, lze upravit algoritmy uvedené v kapitole 3.1 a následné použít v kombinaci s algoritmem policovým. V rámci First Fit přístupu se lze zaměřit na šířku předmětu, kde je předmět uložen na polici s výsledným nejmenším zbývajícím volným místem na polici. Dále se můžeme zaměřit na výškový rozměr, kde je snaha o zmenšení plýtvání místem mezi předmětem a policí nad ním (nebo stropem kontejneru). To samé platí i pro hloubku police. Při umisťování předmětů v 3D prostoru se většinou volí varianta rohů, kde se předměty začínají skládat od levého dolního rohu podél stěny, a poté je řešeno ukládání předmětů do hloubky police. V případě, že hloubka police je stejně orientována, jako hloubka kontejneru. V případě využití Worst Fit přístupu je naopak snahou dosáhnout co nejvíce volného místa pro uložení dalších předmětů. Pokud je cílem zmenšení nevyužitých míst mezi vrchem předmětů a vrchní policí, je k dispozici přístup „Floor-Ceilling“. Existují také kombinace metod, jakou je například „Waste Map“, kde je kombinován přístup policových algoritmů s algoritmy gilotinovými. Principiálně jsou předměty umístěny do polic až k úplnému naplnění policovým algoritmem. Zbylé volné místo je podrobeno gilotinovému algoritmu. V případě vkládání dalších předmětů je kontrolováno volné místo gilotinovým algoritmem, který by předmět umístil do volného prostoru zbylém po výpočtu policovým algoritmem.



Obr. 3-5 Uložení předmětů v kontejneru dle policového algoritmu [25]

3.3.5 Generování sloupů

Metoda generování sloupů („Column Generation“ – CG) je účinnou metodou optimalizace. CG je schopna vyřešit úlohu v modelu lineárního programování s vysokým počtem proměnných, které jsou přítomny ve výpočtu. Metoda nejdříve rozdělí model

do několika částí dle stanovených proměnných na počátku algoritmu. Metoda nejdříve počítá s menší částí matematického modelu problému. Pro tuto část je cílem najít lokální řešení. Následuje nalezení optimálního řešení z nalezených lokálních. Po nalezení optimálního řešení metoda pokračuje k další části modelu. Při řešení části modelu je možné během procesu přidávat nové proměnné do výpočtu a je zde i pravděpodobnost, že samotná metodika sama nalezne nové, nedefinované proměnné v řešené části. Proces je opakován do té doby, kdy je dosaženo přijatelného řešení pro celkový model [26].

V [26] se zabývalo řešením CG pro 3D BPP. Problematika byla rozdělena do 3 částí. Hlavním problémem byl stanoven model s menšími omezeními, týkajících se proměnné *x* (počet beden) v dané části modelu. Další částí byl hlavní problém s omezeními v rámci zjednodušeného lineárního programování pro proměnné v druhé řešené části modelu. Poslední část tvořil sub-problém, který měl zá úkol najít nové proměnné v modelu. V rámci [26] proměnnými byly samotné předměty. Pro 3D BPP měl hlavní význam sub-problém, kde se hledala přijatelná řešení v rámci systematiky ukládání předmětů do jednoho kontejneru. Přístup se podobal problematice batohu, viz kapitola *1.2.3.* Dále bylo využito systematiky umisťování pomocí EP, viz kapitola *2.3.1.2*.

3.3.6 Vyhledávání dle větvení – model stromu

Model stromu slouží k zobrazení dat v hierarchii. Algoritmus lze přirovnat ke skutečnému stromu. Algoritmus („Tree Search - TS“) disponuje větvením (uzly), větvemi a listy. Běžně využívaný je binární strom, kde každý uzel odkazuje na další dva uzly (bez zacyklení). Jsou zde také hrany (spojnice mezi uzly), které reprezentují vztahy mezi jednotlivými uzly [12].

V práci [15] byl použit přístup tvorby bloků z předmětů (beden) o stejných rozměrech. Cílem blokového přístupu je naložit co největší množství předmětů s co nejmenším plýtváním v rámci volného místa v kontejneru. Přístup uspořádání předmětů do bloků je více vhodný pro homogenní předměty a nelze přístupy za daného stavu využít. V případě heterogenních předmětů se přístup podstatně komplikuje. Je možné vytvořit pouze malý či nulový počet bloků. Pro efektivní implementaci přístupu byla dle [15] nutnost generalizace modelu.

V [15] je vyhledávání rozděleno do dvou stádií, pro které byla vygenerována skupina předmětů v blocích. První stádium se zabývá pouze homogenními předměty. Druhé stádium se zabývá heterogenními předměty. Optimální přístup nakládání je vygenerován pomocí obou stádií. Vyhledávání dle větvení bylo využito v části algoritmu, kde je snahou nalézt nejlepší blok předmětů k umístění do volného místa v kontejneru. Umístění se řídí dle parametru obtížnosti, který je dán pro každý blok. Parametr se zvyšuje s každým řešením a díky tomu je možné nalezení lepších řešení. Díky nízkému parametru obtížnosti po zahájení výpočtu, jsou počáteční řešení nalezena sice za krátký čas, ale pravděpodobně daleko od optima. Výstupem metody jsou optimální řešení pro naložení předmětů z hlediska objemnosti za středně dlouhý výpočetní čas.

Hlavní větev stromu

Metoda je založena modelu stromu. Jsou zde přiřazeny předměty do rozdílných kontejnerů bez specifikování jejich přesné pozice. Předměty před naložením jsou v sestupném pořadí setříděny dle jejich objemu a do kontejneru jsou předměty nakládány pomocí strategie

„Hloubka První“. Z názvu přístupu plyne, že je zaměřen na rozměr hloubky předmětu d_i. Metoda obsahuje rozhodovací uzly, kde každý další předmět je přiřazen do kontejneru, do kterého lze předmět umístit. Pokud žádný takový neexistuje, je otevřen kontejner nový. Přijatelnost řešení nakládání heterogenních předmětů do kontejneru se dle [29] ověří pomocí dolní limity pro 3D BPP. Ta je stanovena v [29] jako suma objemů pro všechny předměty v_i děleno objemem kontejneru *V*, viz rovnice [29]:

$$L = \frac{\sum_{i=1}^{n} v_i}{V} \tag{8}$$

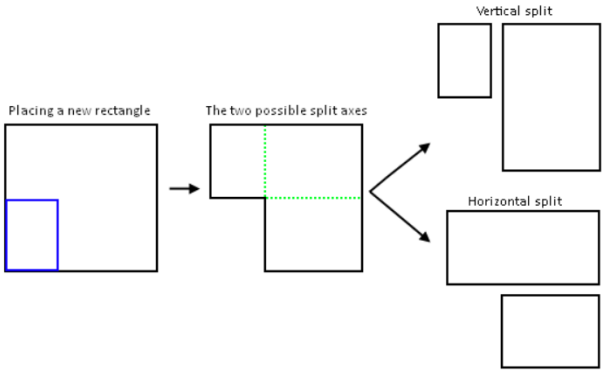
Pokud je limita překročena pro předmět *k*, je uzel, ve kterém byla hranice překročena, je předmět vyřazen a bedna je uzavřena. Pokud hranice překročena není, je otestováno pravidlo dominance. Pokud aproximační algoritmy naleznou příležitost pro uplatnění metodiky „Jedné Bedny“ („ONEBIN“), prezentováno v [29], je dáno, že neexistuje žádné jiné lepší řešení proumístění daných předmětů a kontejner, do kterého byly bedny vloženy, je uzavřena. Jakmile je kontejner uzavřen, je přepočítána spodní limita pro zbývající nenaložené předměty. ONEBIN je tedy použit pouze na specifické rozhodovací uzly, kde je testováno, zda předměty je možno umístit do kontejneru. Využívá se k tomu také přístupů sestupného First Fit a modifikovaného ONEBIN. Podobný přístup byl využit i v práci [29].

3.3.7 Lokální vyhledávání

Lokální vyhledávání („Local Search“ – LS) se snaží iterativně nalézt lepší řešení pro nakládání předmětů do kontejneru. Nová řešení jsou dle metodiky generována pomocí funkce souseda *N* („Neighbourhood Function“) v rozmezích možných řešení *X* pro naložení předmětů. Možnými řešeními se mají na mysli všechny možné přístupy k naložení předmětu do kontejneru. Funkce *N* přiřadí každému řešení *x* náležící *X* soubor funkcí *N(x)*, který je podmnožinou *X*. Tyto *x* (řešení) jsou v určitém „dosahu“ dané funkce, například řešení pomocí přemístění předmětu do jiné lokace v kontejneru. Pokud je splněna podmínka objektivní funkce pro možná řešení *X*, LS je podroben lokální optimalizaci, která je zastavena až nejlepším řešením (*x*) v lokálním minimu. Metodika je většinou aplikována spolu s meta-heuristikou [16].

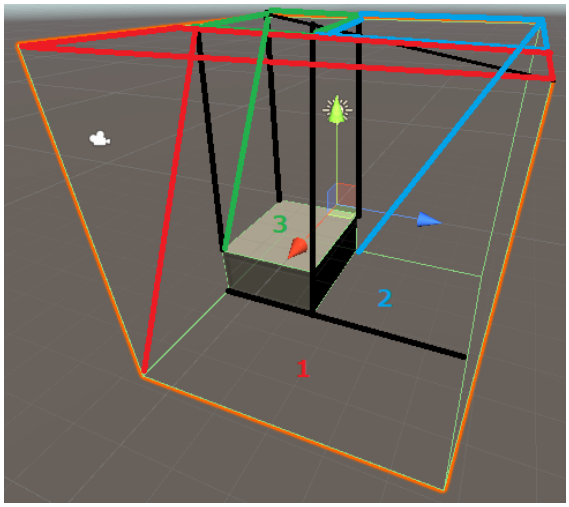
3.3.8 Gilotinové algoritmy

Gilotinové algoritmy se zakládají na rozdělování velkých objektů, v tomto případě prostoru uvnitř kontejneru, na objekty menší. Řez je veden paralelně podél strany kontejneru vedený napříč celým kontejnerem. Všechny provedené řezy, musí být řezy gilotinovými. Řezy mohou být rozděleny do stádií. Kromě prvního stádia, jsou řezy prováděny na objektech (volném prostoru), které vznikají po předchozím stádiu. Všechny řezy ve stádiu musí být ortogonální vůči řezům ve stádium předchozím. V rámci řezů je možné se setkat s pojmem „Cutting Patterns“, neboli systematikou řezů. V rámci 3D BPP je předpokládáno, že jak předměty, tak kontejner mají své dimenze pro šířku, výšku a hloubku. Pozice (0, 0, 0) se dle [35] stanovuje v levém dolní rohu kontejneru. V rámci 2D BPP jsou předměty vkládány do rohu (kontejneru nebo rohu vzniklého vložením předmětu) a vznikají tak dva nové obdélníkové volné prostory, které se dalšími řezy (vložením následujícího předmětu) dělí na menší.



Obr. 3-6 Vizualizace funkčnosti gilotinových algoritmů ve 2D [19]

V rámci 3D se jedná o vzniklé prostory o určité šířce, výšce a hloubce – kvádr nebo krychle prázdného prostoru. Navíc vzniknou tyto prostory tři namísto dvou, jak tomu je u 2D verze. Vzniklé útvary se nesmí překrývat, nebo sebou pronikat. Výhodou této metody je přehlednost volných míst v kontejneru [19]. V [35] je jedním z přístupů k řešení 3D gilotinových řezů „3D unbounded Knapsack Problem“, kde bylo pomocí tzv. „Raster Points“ (RP), určeno, kudy mohou být jednotlivé řezy provedeny. Byly určeny diskretizační body dle výšky, délky nebo hloubky, kde RP jsou podmnožinou těchto bodů. Je tak docíleno zmenšení časové náročnosti algoritmu pomocí ohraničení možností řešení pro optimální uložení.



Obr. 3-7 Vizualizace jedné z možností provedení řezů ve 3D

Možností přístupu v rámci gilotinových algoritmů, dle [19] existuje hned několik. Varianta zabývající se plochou obdélníků pracuje s nejmenší volnou plochou, do které se aktuálně vkládaný předmět rozměrově dá uložit. V rámci 3D by se jednalo o objem namísto plochy.

Dále je varianta se zaměřením na kratší hrany předmětu, kde je cílem minimalizace rozdílu délek uvažovaných stran předmětu a stran kontejneru. Existuje i opačná varianta pro delší hrany.

Jako u policových algoritmů i zde lze využít přístup Worst Fit. Přístupem jsou předměty ukládány s cílem dosažení co největšího volného místa uvnitř kontejneru.

Pod oblasti gilotinových algoritmů spadají též algoritmy maximálních obdélníků. V případě 3D je však nutné se zabývat kvádry. Proto se problematika musí převést do 3. dimenze, jak je vysvětleno v [32]. Funkcionalita se zakládá na daném počtu bodů *b* náhodně umístěných v prostou kontejneru, kde je cílem najít všechny kvádry volného místa, které byly vytvořeny pomocí šesti párů bodů. Algoritmus si ukládá seznam těchto volných kvádrů, do kterých vkládá další předměty k naložení. Maximální kvádry v kontextu metody mají funkci pro nalezení nových volných míst. Ta jsou vytvořena tak, aby jejich strany byly nejdelší možné v každém směru, tudíž se jedná o maximalizaci volného místa uvnitř kontejneru. Z toho vyplývá, že strany volného kvádru se dotýkají některého z vložených předmětů, nebo stěny kontejneru. Definice 2D gilotinového algoritmu dle [32] zaručuje vložení předmětu za předpokladu existence dostatečně velkého kvádru volného místa díky vlastnosti množiny maximálních volných kvádrů. V případě gilotinových algoritmů existují též variace dle nejlepší shody, či přístupu se zaměřením na rozměry stran volných obdélníků (ve 3D – kvádrů) [32].

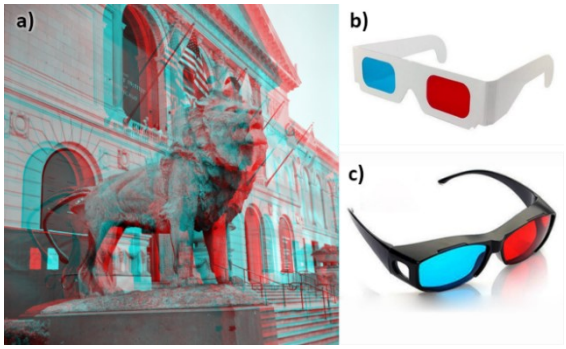
4 Virtuální realita

Pro možnost vizualizace ukládání beden, rozmístěných v cílové lokaci (police stojanu) dle výsledků algoritmů, byl využit model virtuální reality. Virtuální model dokáže propojit pracoviště a například i řídit procesy příkazy přijímaných od jiných pracovišť/strojů [18]. V případě této práce by bylo zajímavé propojit všechny procesy skladování. V rámci této aplikace je zaměřeno pouze na proces uložení (přijmutí beden k uložení a následné uložení do stojanu). V případě vytvoření modelu pracoviště i v reálném světě, začalo by se v případě virtuálního modelu jednat o „Digitální dvojče“ pracoviště („Digital twin“) [18]. Následně by v rámci virtuálního modelu bylo možné provádět různé simulace blížící se reálnému stavu (záleží na propracovanosti definice proměnných reálného světa). Pro možnost představy způsobu tvorby a využití virtuálního modelu je stručně popsána definice VR, její princip funkčnosti a vybavení (Hardware/Software) potřebné k možnosti využití VR.

Jak se v [18] uvádí, je vhodné si pro začátek položit otázku co to vlastně je virtuální realita (VR). Pohledy se různí pozicí, ze které se na VR nahlíží. Ze zábavního průmyslu je považována skvělým prostředkem pro vytvoření imerzní hry. Naopak u nebezpečných povolání, jako jsou hasič, pilot nebo voják, VR slouží k nácviku situací bez rizik fyzického zranění. Jedna z vědeckých definic VR reality zní: „*Umělé prostředí, které je vnímáno pomocí senzorických stimulů (jako například obrazových nebo zvukových) zprostředkovaných počítačem. Chování prostředí je ovlivněno akcemi uživatele.“ [18]*. V [18] je odkázáno na definici VR: *„Imerzní počítačově simulovaná realita vytvářející prostředí, které ve skutečnosti neexistuje“.* V rámci imerze do virtuální reality se rozlišuje míra, v jaké je uživatel do VR vtažen. Ze strany vjemů se imerze rozděluje na vizuální, haptickou, aurální (sluchovou) a pomocí čichu nebo chuti. Velkou roli ve VR hraje stereoskopie, prostřednictvím které lze vnímat 3. rozměr v obrazu.

4.1 Stereoskopie

V rámci stereoskopie lze vnímat 3D obraz pomocí anaglyfu, pasivní/aktivní projekcí, aktivně-pasivní projekcí a auto-stereoskopickými monitory. Anaglyfem lze vnímat 3D obraz pomocí vytištění dvou zdrojových obrazů na objekt (obraz), které jsou horizontálně posunuté o rozměr vnímané hloubky pro vyjevení dle požadavku. Charakteristikou pro metodu je odstínění obrazů dvěma barvami (červená a modrá).



Obr. 4-1 Příklad vizualizace 3D obrazu anaglyfem [18]

Pasivní projekce se zakládá na rozložení obrazu do dvou rovin. Pomocí brýlí s polarizačními filtry se přenese obraz do 3D pomocí propouštění určitého světla do jednoho oka přes jedno ze sklíček brýlí.

Aktivní projekce se zakládá na promítání o vysoké frekvenci (110 ÷ 144 Hz), kde z dataprojektoru jsou střídavě vysílány signály pro pravé a levé oko a obraz je zpracován pomocí 3D brýlí, které se synchronizují s projektorem a střídavě zatmívají sklíčka brýlí. Tím je rozdělen snímek na sudé a liché obrazy a nastává obdobný případ anaglyfu.

4.2 Implementace v průmyslu

V průmyslu lze pomocí VR simulovat procesy, které by v případě testování v realitě mohli zapříčinit závažnou škodu podniku, či ohrozit životy zaměstnanců podniku. Virtuální realita má limitace v rámci převedení reality do modelu virtuálního, což je velmi složité. Je velmi obtížné stanovit proměnné reálného světa a omezit realitu, tak aby se co nejblíže dala popsat ve VR. V případě přijatelného napodobení skutečnosti, lze s VR zaznamenávat posuzovaný proces, replikovat jej a dále upravovat bez větších nákladů na tyto úpravy, než by tomu bylo ve skutečnosti (záleží na případu implementace).

Největší nevýhodou imerzních technologií při implementaci jsou počáteční náklady. Vytvoření 3D modelu je časově náročné a finančně nákladné. Dále se zde vyskytují stále některé nedostatky v rámci hardwaru (HW), které znemožňují plynulé zavedení technologie. Jedná se především o umístění a zapojení senzorů, zapojení samotného VR zařízení a jejich kalibrace. Dle Gartnerovy křivky „Hype Cycle“ pro vhodnost („zralost“) technologie k implementaci, lze prohlásit, že VR řešení, není nadále vyvíjející se technologií a je již běžně v praxi využívána [33].

4.3 Head Mounted Display

Jak bylo v *kapitole 4.1* popsáno, existuje několik druhů projekcí. Každá využívá jiné zařízení či systémy k dosažení požadovaného vtáhnutí uživatele do virtuálního světa. V dnešní době jsou oblíbené VR brýle z kartonu či plastu pro chytré telefony díky jejich nízké ceně. V této práci je však použito zařízení Head Mounted Display (HMD).

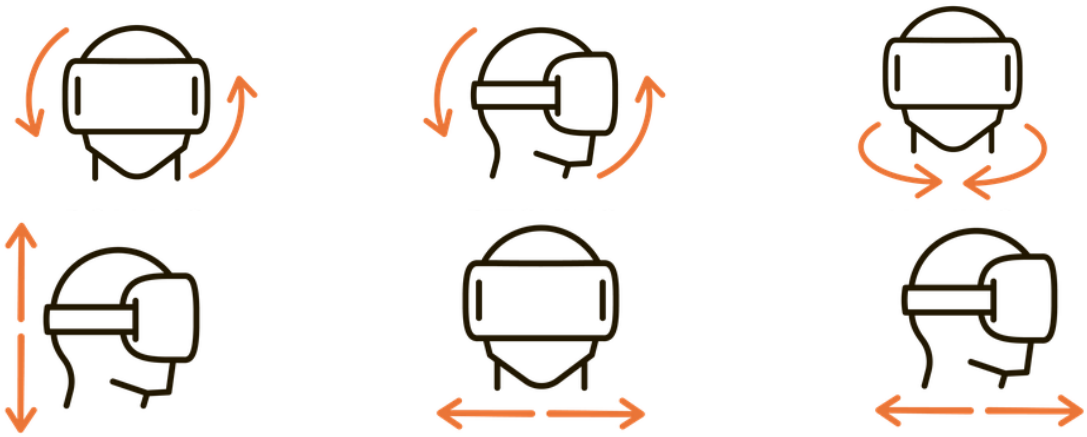
Zařízení HMD je určeno k využívání pouze pro jednoho uživatele v okamžiku používání. Nevýhodu využívání HMD jedním uživatelem lze částečně pokrýt promítáním obrazu z VR externě (na monitor), aby mohli další lidé vidět, co se ve VR odehrává. V rámci průmyslu záleží, pro jaký účel je VR využíváno (design nebo vizualizace). Sestava VR je následně postavena v místě, kde je nutné vyčlenit dostatečný prostor pro možné pohyby uživatele (pokud pohyby jsou zapotřebí). V rámci HMD dělíme HW na desktopové (v rámci počítačové aplikace) a mobilní zařízení (chytrý telefon) [21]. VR se dělí také dle prostoru, který působnost zařízení pokrývá (pomocí senzorů). Jedná se o prostory místnosti, kde se uživatel pohybuje a senzory snímají jeho pohyby, které jsou přenášeny do VR. Druhou možností je promítání VR kolem uživatele a ten se bez vlastního fyzického pohybu přesouvá pouze ve virtuální realitě.

4.4 Implementace hardwaru

K umožnění zobrazení VR a interakce s tímto prostředím bylo nutné nastavit vhodným způsobem hardware. Využitím takových prostředků k zobrazení VR dopřeje uživateli lepší imerzi do modelované situace. Pro tuto práci byl využit Oculus Quest zapůjčený od katedry Průmyslového inženýrství a managementu – FST – ZČU, viz *Obr. 4-2*.

Obr. 4-2 HMD Oculus Quest [7]

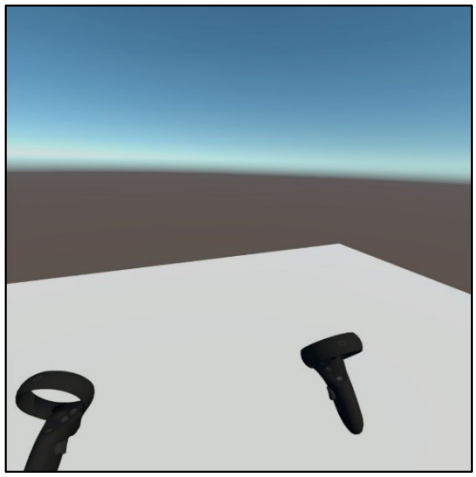
Tento typ brýlí pro virtuální realitu je „all-in-one“ varianta (bez kabelu, bez vnějších senzorů), která nepotřebuje připojení k počítači pro možnost spuštění aplikací. Vše je zabudované již v samotných brýlích. HW set se skládá z „Head Mounted Displeje“, tj. brýlí pro zobrazení VR a ovladačů - kontroléry pro snímání pohybu rukou uživatele. Díky těmto HW ovladačům je možné manipulovat s bednami a měnit jejich polohu. Oculus Quest využívá šesti stupňů volnosti, kde v rámci pohybu headset zaznamenává naklánění hlavy do stran, předklon a záklon hlavy a její otáčení. Dále disponuje detekcí změny výšky vůči nastavené hladině podlahy, pohybu do stran, vpřed a vzad (viz *Obr. 4-3*) v rámci vyznačené bezpečné plochy, která je nastavena uživatelem při počátečním spuštění brýlí.



Obr. 4-3 Vizualizace šesti stupňů volnosti HMD Oculus Quest [4]

Dle Gartnerova Hype cyklu je sice VR již zralá inovace k běžné implementaci v podnicích [50], avšak podpora běžného uživatele v rámci implementace HW do práce není stále ve fázi, kdy by bylo možné říct, že se jedná o uživatelsky vstřícnou technologii. Pro potřeby této práce bylo nutné stáhnout a nainstalovat několik aplikací a ovladačů, které umožňují propojení HW s počítačem. Pro využití Oculus Quest HMD bylo zapotřebí nainstalovat aplikace Oculus pro registraci zařízení na využívaném počítači, které předcházela instalace ovladačů pro využívaný typ HMD. Další potřebná aplikace, k možnosti importu verze vyvíjené aplikace, byla aplikace SideQuest. Tyto aplikace byly vybrány z důvodu synchronizace s využívaným vývojovým softwarem Unity3D. Pro možnost ovládání byly

implementovány assety SteamVR a VRTK, které implementují logiku ovládání ve vyvíjené aplikaci. Tyto assety obsahují možnosti zaměřování na objekt, uchopitelnost nebo samotnou vizualizaci prostředí pomocí integrovaného kódu pro virtuální kameru snímající danou scénu („CameraRig“). Po ukončení implementace všech nutných částí pro využívání HMD byla vytvořena testovací aplikace, kde byla ověřena funkcionalita všech částí hardwaru a implementovaného softwaru pro HMD uvnitř aplikace Unity3D, viz *Obr. 4-4*.

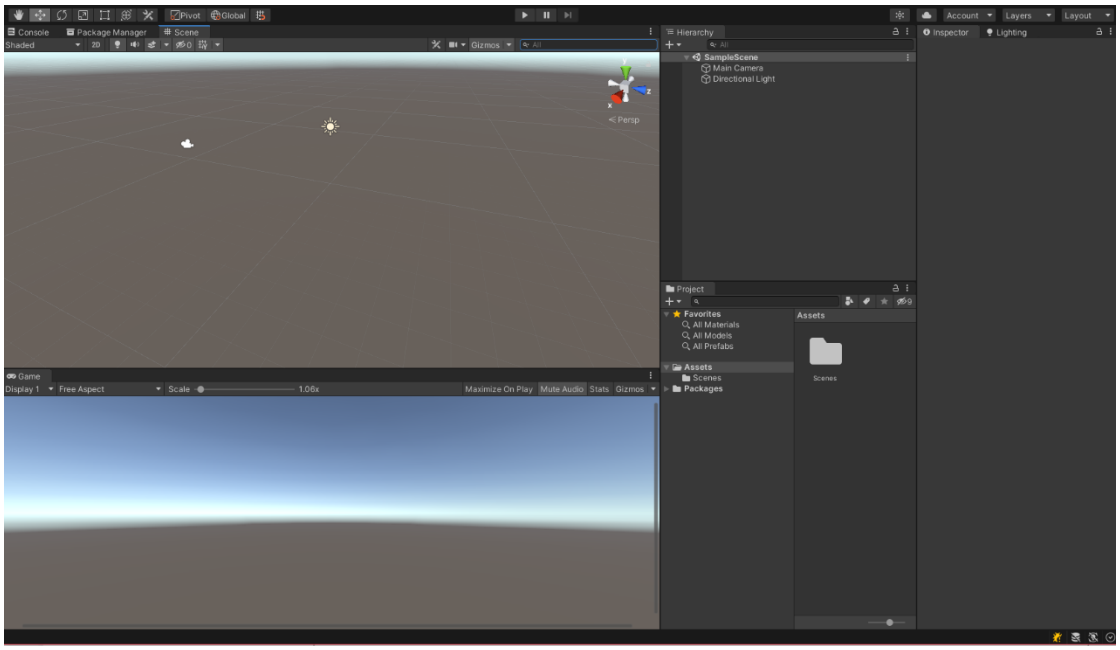


Obr. 4-4 Test ovládání ve VR

4.5 Unity3D

Důležitou stránkou je též uživatelské prostředí („User Interface“ – UI), ve kterém se odehrávají veškeré akce VR. Typicky se v případě VR jedná buď o prostorové interface nebo o diegetický interface. Pro tuto práci se více hodí spíše prostorový interface, kde uživatel uvidí UI přímo uvnitř scény [18].

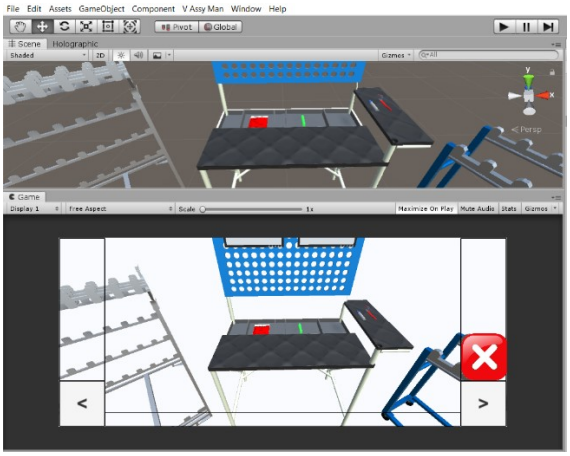
Pro realizaci řešení bylo využito vývojové prostředí Unity3D. Tento nástroj je multiplatformní herní „engine“, který je především využíván pro vývoj v rámci zábavního průmyslu. Unity3D podporuje jak tvorbu v 2D, tak i ve 3D. V našem případu se jedná o model kontejneru s uloženými předměty ve 3D prostoru. Hlavní předností Unity3D je možnost tvorby grafického prostředí pro uživatelem požadovanou skutečnost. Jednou z hlavních předností tohoto enginu je velká komunita vývojářů, díky které lze snadno s prostředím pracovat a dohledat spoustu informací ohledně funkčnosti a kódů pro naprogramování logiky celého VR. Jedním z těchto prostředků je Asset Store, přímo v prostředí Unity3D, kde lze stáhnout spoustu modelů k urychlení návrhu aplikace. Ukázka vývojového prostředí aplikace Unity3D je možné vidět v *Obr. 4-5*.



Obr. 4-5 Ukázka vývojového prostředí Unity3D

4.5.1 Způsob tvorby 3D prostředí v Unity3D.

Na začátku tohoto procesu je zapotřebí vytvoření 3D modelu. Po vymodelování objektu je zapotřebí udělat ještě jeden důležitý krok. Tím je export modelu do vhodného formátu. Většina dnešních enginů používá formát Autodesk [48]. Postup vývoje prostředí aplikace, která vizualizuje 3D BPP, je popsán v kapitole 6.1.



Obr. 4-6 Příklad modelu pracoviště v Unity3D [18]

4.5.2 Skripty jazyka C#

Unity3D pro vytvoření mechanismů a logiky funkčnosti celého VR používá skriptování. V Unity3D je možné psát zdrojový kód (skript) v některém ze tří programovacích jazyků. Těmito jazyky jsou UnityScript, C# a Boo. Je dokonce možné v rámci jednoho projektu, používat kódy napsané různými jazyky, to se ovšem není doporučeno díky své různorodosti. Mezi skriptovacími jazyky není prakticky žádný výkonový rozdíl. Volba jazyka je závislá spíše na preferenci vývojáře. Samotné skriptovací jazyky jsou velmi dobře zdokumentované.

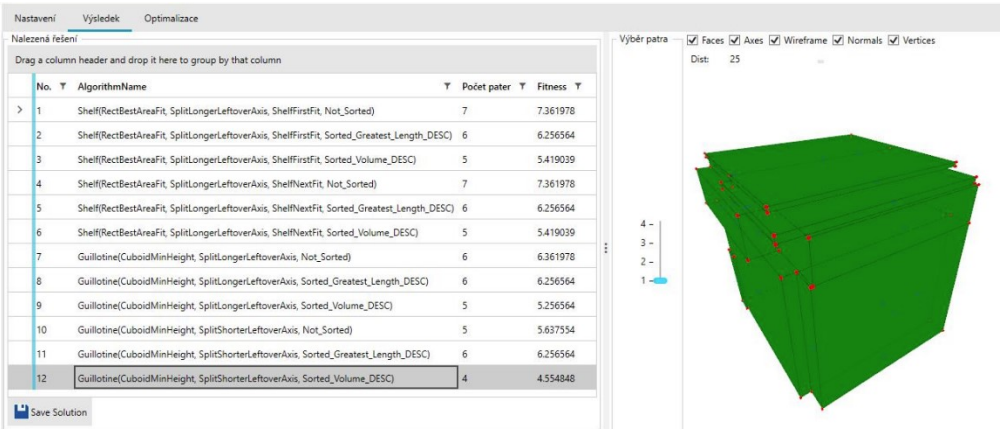
Dokumentace je dostupná jak online, tak i lokálně na počítači přímo v aplikaci Unity3D [48]. Vytvořený zdrojový kód pro funkcionalitu vizualizace 3DBPP v rámci vytvořené aplikace je detailně popsán v kapitole 6.5 a v přílohách této diplomové práce.

5 Návrh zobrazení ve 3D

Cílem této práce je vyobrazení dat obdržených z výpočtu rozmístění předmětů v 3D prostoru. Pro tento účel je cílem vývoj aplikace pro uskladňovaní beden uvnitř interaktivní aplikace, kde by uživatel byl schopný dle určitého systému umisťovat předměty (bedny) na stanovené pozice dle vypočtených hodnot algoritmem. V rámci způsobu provedení procesu uložení beden uživatelem, je požadavek implementace podpůrného systému, který by uživatele naváděl, na jakou pozici se daná bedna má uložit. Aplikace je cílena na školící účely v rámci průmyslu či školství. V průmyslovém podniku by se pomocí aplikace mohli zaškolovat noví zaměstnanci například skladového úseku. V rámci využití k výuce, například na katedře Průmyslového inženýrství a managementu na Západočeské univerzitě, lze studentům vyložit látku týkajících se různých problematik (ergonomie, časové náměry, Kaizen, programování, atd.) pomocí virtuálního modelu aplikace.

5.1 Vstupní data

Vstupní data byla obdržena z aplikace, která byla vytvořena na katedře Průmyslového inženýrství a managementu Fakulty strojní Západočeské univerzity v Plzni. V aplikaci je BPP řešen pomocí metaheuristických algoritmů (využívající základních metod viz kapitola 3.1), dle kterých jsou bedny umisťovány do kontejnerů.



Obr. 5-1 Vizualizace prostředí aplikace provádějící výpočet umístění pomocí 3DBPP algoritmu

Data jsou uložena v souboru formátu „*.json“, kde jsou uvedeny rozměry beden s jejich označením ID, umístění v kontejneru, váhou, označením a zda byla bedna uložena. Seznam v podobě generického listu („List“) s názvem „Cuboids“ (kvádry) obsahuje všechny bedny k uložení v počáteční nulté pozici (0, 0, 0). Důležitou částí pro výstup této práce jsou data v generickém listu „Results“. V této části vstupních dat byly bednám již pomocí algoritmu přiřazeny polohy. Poloha je dána vůči patru, ke kterému je bedna přiřazena, viz **Chyba! N enalezen zdroj odkazů.**. Ve vstupních datech jsou využívána 4 patra, do kterých jsou bedny ukládány. Dále byly poskytnuty zdrojové kódy jednotlivých tříd. Každá zaujímá určitou část celku algoritmu pro setřídění beden. Některé části tříd a kódu jsou využity k vizualizaci setříděných beden ve virtuální realitě. Celý soubor *.json je k dispozici v příloze 1 na stránce 80.

```
"Results": [
  {
    "Patro": 1,
    "PatroBedny": [
      {
        "ID": 18,
        "Poradi": 0,
        "Width": 89.0,
        "Height": 74.0,
        "Depth": 93.0,
        "X": 0.0,
        "Y": 0.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": true
      },
      {
        "ID": 12,
        "Poradi": 0,
        "Width": 100.0,
        "Height": 19.0,
        "Depth": 86.0,
        "X": 0.0,
        "Y": 74.0,
        "Z": 0.0,
        "Weight": 0.0,
```


Obr. 5-2 Vstupní data popisující atributy beden (velikost, umístění atd.)

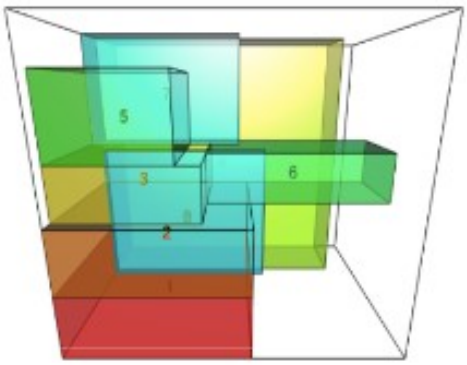
5.2 Návrh vizualizace vstupních dat ve 3D

Pro účel této práce byl vybrán programovací jazyk C# z důvodu jednoduchosti vývoje aplikace.

V rámci Unity3D se též jedná o nejčastěji využívaný jazyk k vývoji, a proto je snadné dohledat v případě potřeby informace a pro dosažení požadovaných funkcionalit v rámci jak kódu, tak vytvářeného virtuálního modelu [11].

Každá aplikace vytvářená v Unity3D má své „assety“ (zdroje). Jedná se o veškeré soubory, které aplikace obsahuje, jako jsou 3D modely („prefaby“), skripty, textury a další. Aplikace se většinou skládá z několika scén (mapa/prostředí), avšak v rámci této aplikace je použita pouze jedna scéna. Ve scéně jsou umístěny objekty aplikace („GameObjects“) vytvářející vizuální stránku aplikace. Tyto objekty jsou nositeli funkčních komponent, které určují, jak objekt má vypadat, jak se po spuštění objekt má chovat v rámci prostředí, nebo kde má být objekt umístěn. Prefaby jsou složitější objekty, se kterými se dobře pracuje i po spuštění aplikace. Význam těchto prefabů je především v jejich zhmotnění v aktivním stavu aplikace. Například pomocí funkce objektů „Instantiate“ (instance) je vytvořena instance objektu v rámci objektivně orientovaného programovacího jazyka (C#). Po zavolání funkce je instacionalizovanému objektu přidělen název a je uložen v paměti [5].

Cílem této diplomové práce je vizualizace beden uvnitř prostoru, ať už ohraničeného, nebo jen ve vyhrazeném pro skládku/nakládku. Podstata myšlenky je znázorněna na *Obr. 5-3*. Jednotlivé fáze stavby modelu ve virtuální realitě jsou popsány v následujících kapitolách.



Obr. 5-3 Vizualizace 3D BPP výstupu v VR [14]

Jednotlivé pozice k uložení beden (navigační kvádry) jsou barevně zvýrazněny pro jednoduché rozlišení. Uživatel má možnost se na předměty podívat z různých úhlů a lze jednoznačně určit uložení jednotlivých předmětů. Aplikace naviguje uživatele při ukládání beden s jistou mírou pomoci. Uživatel by si měl během procesu ukládání beden všimnout, jakým způsobem se bedny ukládají do polic stojanu. S pomocí navigačního systému lze uživatele též připravit na nepřehledné situace, které mohou nastat při skládání vícero beden do jednoho patra stojanu (např. vložení menšího předmětu do vzniklých prostor mezi předměty již uloženými).

5.3 Výstupní data

Výstupní data budou strukturně podobná datům vstupujících do aplikace za účelem možnosti jejich nahrání do prostředí aplikace. Výstupní data budou zastupovat objekty beden uvnitř aplikace. Důležité budou atributy dimenzí a pozic těchto objektů. Výstupní data budou reprezentovat uložená data získána v průběhu spuštění aplikace a následně s nimi dál pracovat.

6 Implementace „3D Bin Packing Problem“ ve VR

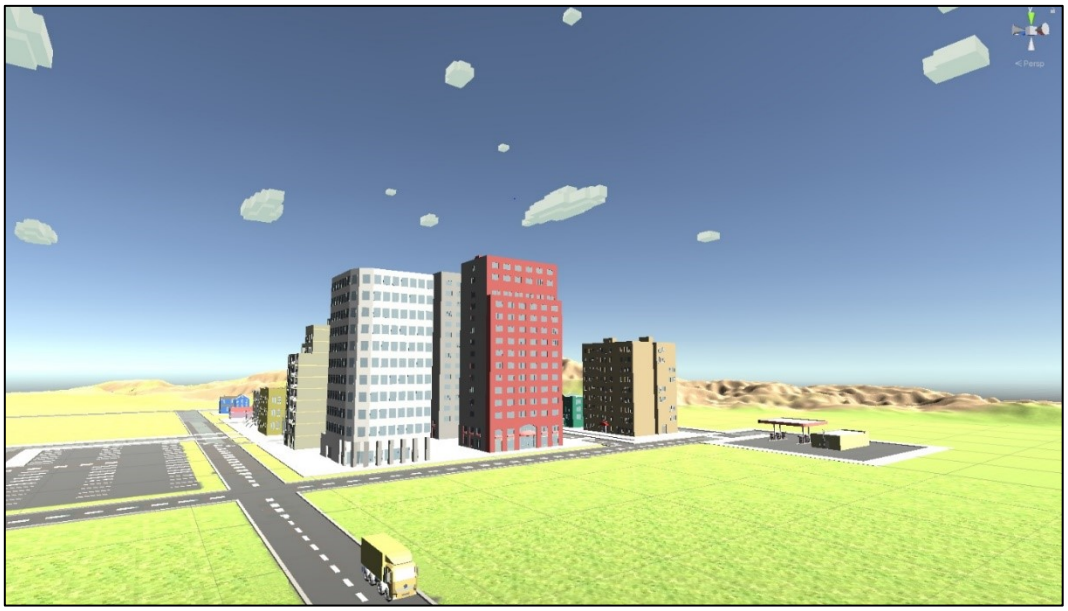
Praktická část byla zahájena vytvořením vhodného virtuálního prostředí pro následnou implementaci logiky samotného BPP. Tvorba samotného prostředí pouze vlastním modelováním by byla velmi časově náročná, a proto byla zvolena varianta importu již modelovaných objektů pro užití ve VR a 3D prostoru. Volba vhodného prostředí je důležitá pro lepší vnoření uživatele do virtuální reality. Pro tvorbu bylo upřednostněno pozadí s urbanistickými prvky, kam lze snadno vložit a realizovat logistické prvky a operace v rámci BPP.

V kapitole 6.5 je sepsán programovací kód v jazyce C# a v rámci logiky jsou implementovány některé z algoritmů uvedených v *kapitole 3* této práce. Jednotlivé předměty jsou barevně zvýrazněny pro jednoduché rozlišení. Uživatel má možnost se na předměty podívat z různých úhlů a lze jednoznačně určit uložení jednotlivých předmětů. Další logika a funkce by byly doplněny během vytváření praktické části práce.

Před započetím četby kapitoly 6 a dále je doporučeno shlédnutí videa s ukázkou celého procesu uložení beden, které je obsaženo v přiloženém CD k této diplomové práci.

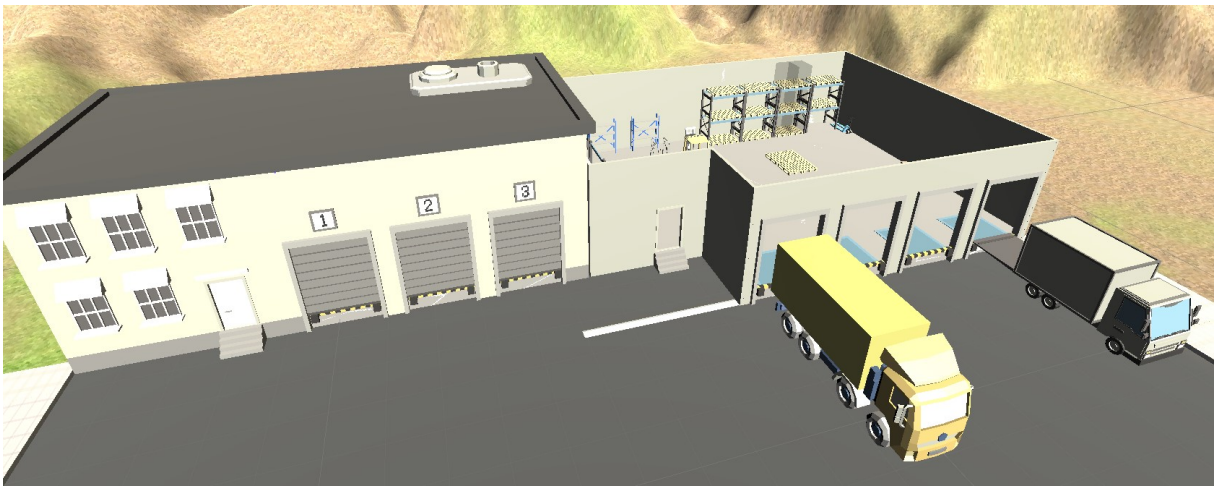
6.1 Vytvoření prostředí ve VR

Pro pozadí modelu byla po diskusi s konzultantem diplomové práce vybrána scéna města, viz *Obr. 6-1*, kde se hlavní pozornost upírá k objektu skladu. Pozadí bylo vybráno pro lepší imerzi uživatele do virtuálního světa. V případě dalšího vývoje aplikace je možné prostory skladu rozšířit, zvýšit interaktivnost a docílit lepšího zážitku a porozumění ze strany běžného uživatele. Záměrně bylo též zvoleno nízko-polygonální („Low-Poly“) grafické prostředí, které není tolik náročné na výkon zařízení v případu all-in-one provedení HMD, pomocí kterého se VR ovládá. Low-Poly aplikace má také výhodu v rámci úspory potřebného celkového místa k uložení. Modely obsahují méně polygonů, tak je jednoduché vytvořit aplikaci, která svou velikostí může mít jen několik desítek megabytů. Předpřipravené modely a textury pro tvorbu prostředí byly staženy z internetových stránek pro vývoj v programu Unity3D, „Unity Asset Store“ [1].



Obr. 6-1 Vizualizace modelu města jako pozadí modelu

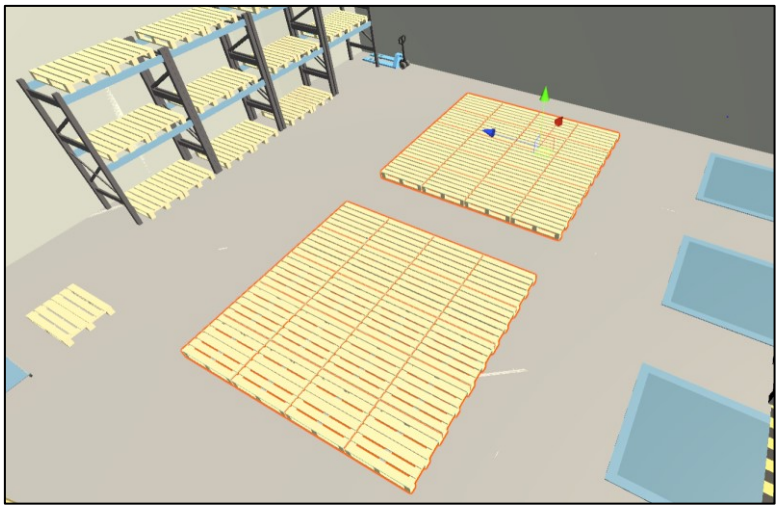
Model budovy skladu byl vytvořen na základě logistických modelů dostupných v balíčku logistického skladu v [2]. Do budovy byly implementovány logistické prvky, kde na některé z nich byla implementována vizualizační logika. Úmyslně byla vytvořena budova s otevřenou střechou pro lepší orientaci v případě pohledu na logistický sklad z ptačí perspektivy, *Obr. 6-2*.



Obr. 6-2 Vizualizace modelu logistického skladu

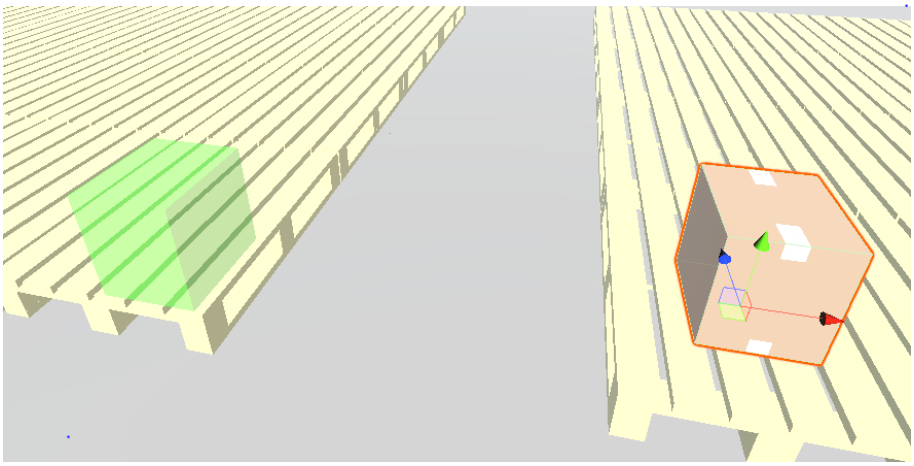
6.1.1 Vytvoření pracoviště uvnitř logistické budovy skladu

Pro možnost prvotního otestování vizualizační logiky byl vytvořen vymezený prostor pomocí palet uvnitř logistické budovy skladu, viz *Obr. 6-3*.



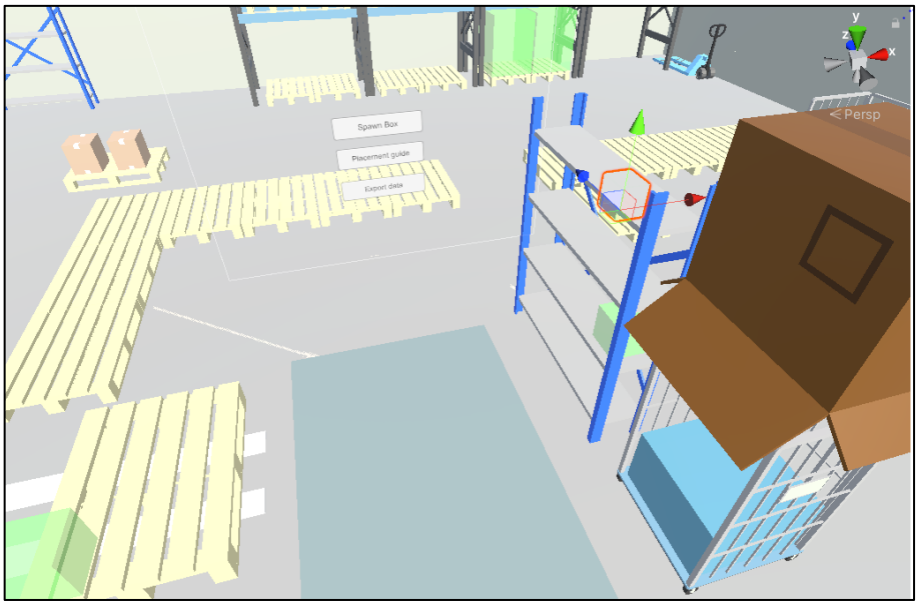
Obr. 6-3 Vyhrazené místo pro manipulaci s bednami

Na počátku tvorby pracoviště byla vybrána jednoduchá varianta pro ukládání beden na palety, aby mohla být otestována základní funkčnost modelu a jeho prvků. Byly vytvořeny dvě větší plochy z jednotlivých palet, kde na pravém seskupení palet (blíže u zdi – viz *Obr. 6-3*) by se umísťovaly nesetříděné bedny a na levé by byly generovány setříděné vzory beden, které by vyznačovaly pozici k umístění. V rámci této práce jsou nazývány „navigačními kvádry“.



Obr. 6-4 Prvotní návrh vizualizace navigačních prvků pro ukládání beden

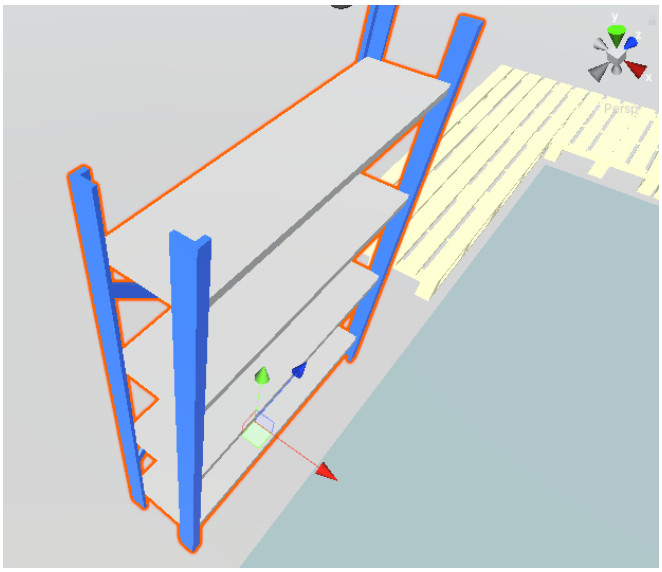
V prostoru bližších palet by se naopak objevila nápomocná navigace pro umístění beden do regálů, jejichž rozmístění vypočetl algoritmus. Způsob generování a přemístění beden je detailně popsán v kapitole *6.2*.



Obr. 6-5 Počáteční rozmístění upraveného pracoviště

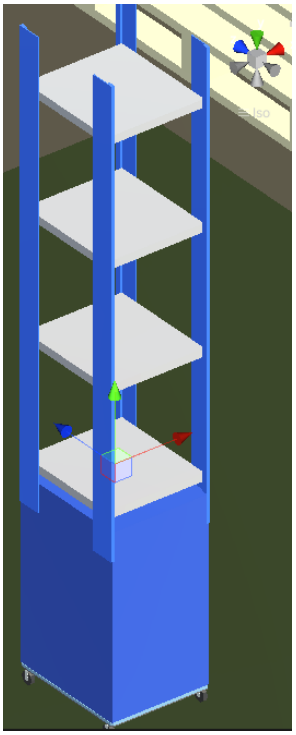
Při přemisťování beden v počátečním layoutu bylo obtížné přenášet vygenerované bedny z počátečního do cílového místa. Pracoviště bylo následně upraveno, aby uživatel nemusel několikrát přemisťovat svoji virtuální postavu („Avatar“) pomocí teleportu, viz Obr. 6-5. Pro usnadnění umístění uživatele do pracovní polohy byla na zem pracoviště přidána podložka, která je umístěna před rollkontejenerem a stojanem – viz Obr. 6-5. Došlo též ke zmenšení plochy pro generování beden. V prvotním rozložení bylo pro uživatele obtížné se orientovat a pohybovat. V novém rozložení je pro uživatele mnohem snazší odebírat bedny z místa generování. Místo pro generování beden bylo přesunuto z původního místa (palety) do „levitující“ krabice nad rollkontejnerem (viz Obr. 6-9). Zde jsou bedny postupně generovány v prostor nad rollkontejnerem a padají na vyvýšené dno rollkontejneru. V rámci vizualizace pro ukládání vygenerovaných beden byl přidán vysokozdvižný vozík, kde si může uživatel vyzkoušet uložení bedny ve vyznačeném místě na paletě pomocí průhledného kvádru zelené barvy. Funkcionalita a možnosti této navigace jsou detailně pospány v kapitole *6.4*.

Pro uložení vygenerovaných beden byl vedle rollkontejneru umístěn stojan s poličkami, na které jsou bedny ukládány. Stojan je vhodný i z hlediska policového algoritmu, použitého pro setřídění beden do pater. Stojan má 4 patra, který odpovídá počtu pater v datech získaných po proběhnutí optimalizace umístění beden pomocí algoritmu. Princip je stejný jako na paletě, kde se zobrazují průhledné navigační kvádry. Příslušný zdrojový kód je vysvětlen v kapitole *6.5*. Pro úlohu vizualizace BPP v 3D je toto rozložení přijatelné. Pro skutečný stav by musel být layout více upraven, aby bylo dosaženo vhodného rozmístění v rámci reálného skladu.



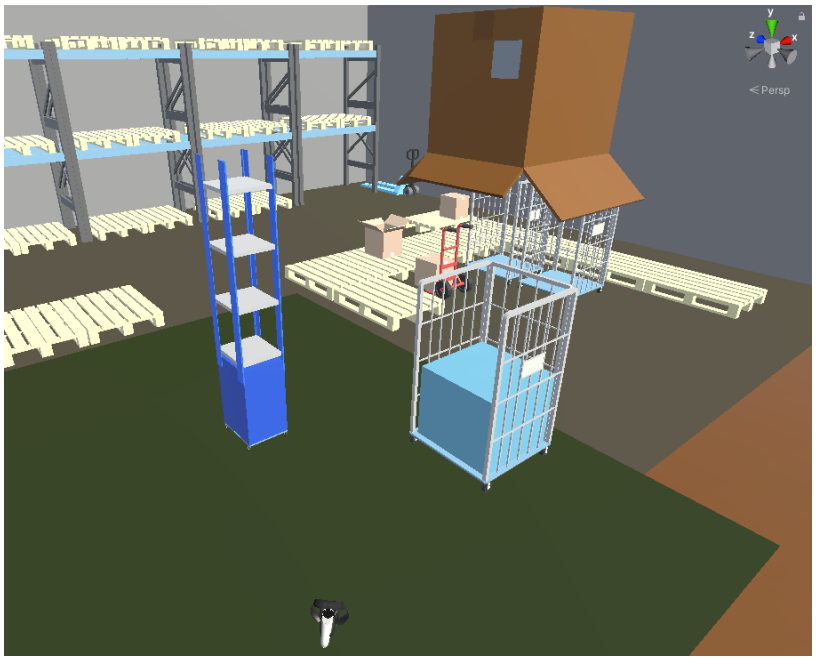
Obr. 6-6 Stojan pro ukládání vygenerovaných beden v původní podobě

Po implementaci kódu pro vizualizaci navigačních kvádrů uvnitř stojanu bylo nutné pracoviště upravit pro bezproblémový průběh procesu umisťování beden. Prvním problémem byl rozměr stojanu, kde patra stojanu byla v ose „z“ (modrá šipka v *Obr. 6-6*) byla příliš úzká a bedny by se na police nevešly. Rozšíření polic bylo též provedeno v ose „y“ (zelená šipka v *Obr. 6-6)*. Při ukládání bedny na již uloženou bednu se mohla vyskytnou situace, kde bedna nemohla být vložena z důvodu nízké výšky patra. Ve stejném porovnání lze vypozorovat další rozdíl mezi původním (viz *Obr. 6-6*) a upraveným modelem stojanu (viz *Obr. 6-8*) v části konstrukce. Pomocí programu na úpravu modelů s příponou „*.fbx“, v tomto případě byl využit program „Blenderer“, pomocí kterého byla odebrána podpůrná část konstrukce ve tvaru „X“ ze zad stojanu a změněny celkové rozměry stojanu, viz *Obr. 6-7*.



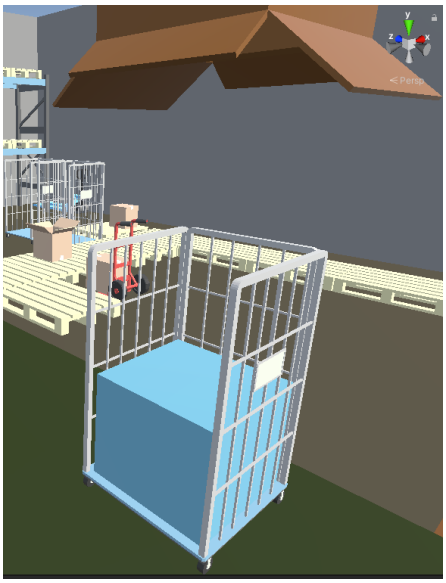
Obr. 6-7 Upravený model stojanu

Bylo tak učiněno z důvodu možnosti ukládání beden z vícero stran. Z důvodu zvětšení rozměrů konstrukce stojanu by bylo obtížné pro uživatele bedny ukládat pouze z čela stojanu. V *Obr. 6-9* lze již zřetelně vidět „levitující“ krabici umístěnou nad rollkontejnerem, která zastupuje vizualizaci generátoru beden, viz *Obr. 6-10*.



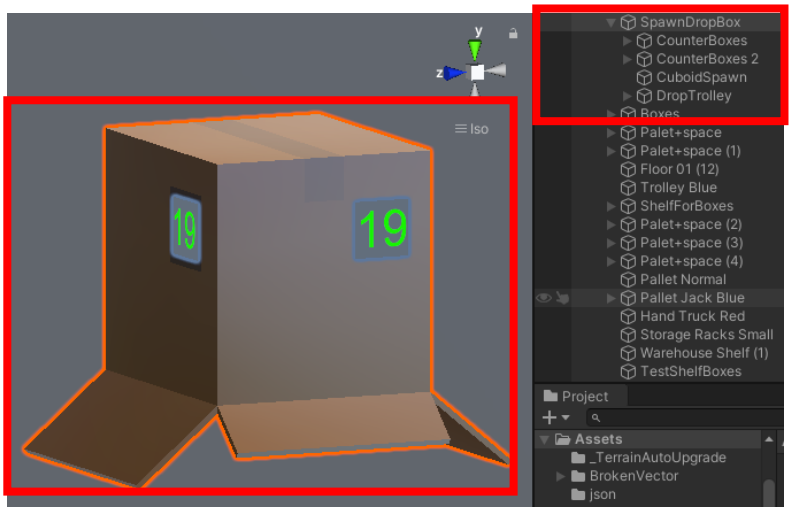
Obr. 6-8 Vizualizace konečného rozmístění pracoviště po úpravách

Po testování nové verze pracoviště s rollkontejnerem ve virtuální realitě, bylo na rollkontejner aplikována ergonomická racionalizace v rámci výšky dna kontejneru. Při provádění úkonu přemisťování bedny ze dna kontejneru, bylo nutné, aby se uživatel ohýbal téměř k podlaze v reálném světě. Navíc rozměry některých vygenerovaných beden jsou o velikosti dopisové obálky, a proto by bylo obtížné na tyto bedny z pohledu uživatele dosáhnout. Dno bylo zvednuto o 0,35 m, což umožnilo snazší průběh celé operace přemístění bedny – viz *Obr. 6-9*. Ve stejném obrázku je možné vidět přidanou bednu „vznášející“ se ve vzduchu. Tato bedna byla přidána pro vizualizaci místa generování nových beden určených k následnému uložení, se kterým uživatel nemůže nikterak manipulovat (zamezení kvůli možnému zkomplikování systému postupného generování).



Obr. 6-9 Rollkontejner se zvýšeným dnem

Pro lepší vizualizaci počtu zbývajících beden k uložení a snadnější vizuální kontroly beden k umístění nacházející se v seznamu (ve skriptu jako generický List), byla na generátor beden umístěna dvě počítadla (v případě pohybu kolem stojanu pro ukládání beden, nebyl displej počítadla čitelný). Počítadlo se řídí počtem beden uvnitř seznamu „CuboidList“ s názvem „Cuboid“ ve třídě BoxesFromJson. Třídy a jejich kód jsou detailně popsány v kaptiole *6.5.2*.



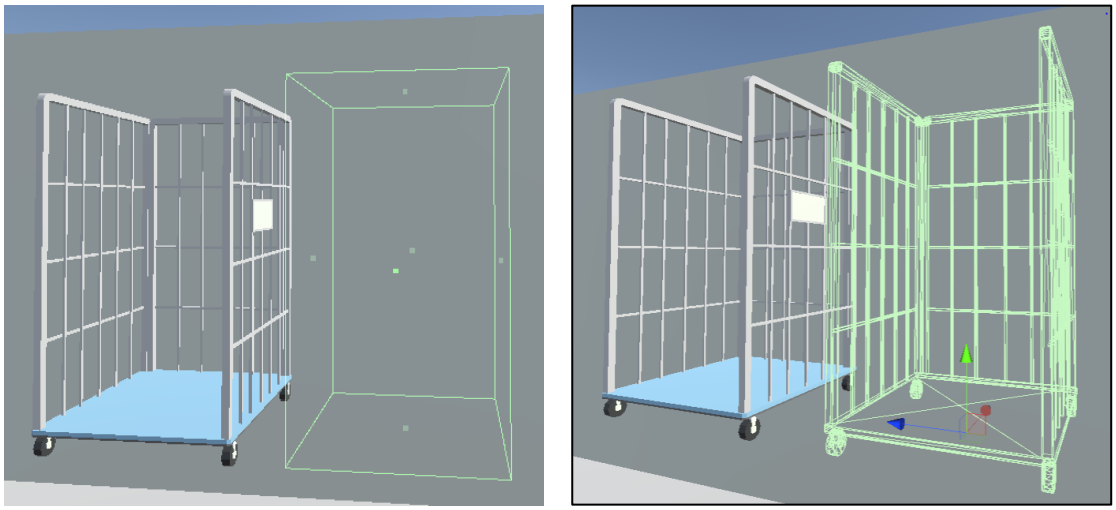
Obr. 6-10 Krabice obsahující generátor beden určených k umístění

6.1.2 Detekce dotyku na požadované objekty

Jelikož jsou bedny generovány jako předměty, které podléhají fyzikálním zákonům nasimulovaných uvnitř aplikace, je zapotřebí přizpůsobit pracoviště takovým způsobem, aby reakce objektu (bedny) například se zemí, nebo policí ve stojanu se přibližovala reakci v reálném světě. K umožnění působení nasimulovaných fyzikálních zákonů na objekt, které spravuje „Unity physics engine“ (správce fyzikálních dějů), je nutné objektu přidělit komponentu „Rigidbody“. Komponentu lze využít bez implementace jakéhokoliv zdrojového kódu. Díky této komponentě je možné přiřadit působení gravitace, nebo reakci na dotyk s jiným

objektem. V případě nutnosti lze komponentu upravit pomocí zdrojového kódu. V tomto případě bylo využito výchozího nastavení [41].

V případě využití „Rigidbody“ lze využít komponentu pro detekci kolize, která je v prostředí Unity3D popsána jako „Collider“. Pokud se dva objekty s touto komponentou střetnou, dochází k události, kde se z výchozího nastavení objekty řídí fyzikálními zákony, nebo je možné událost dotyku pozměnit pomocí kódu komponenty. Stejně jako v případu „Rigidbody“ bylo u kolize využito výchozího nastavení v Unity3D.



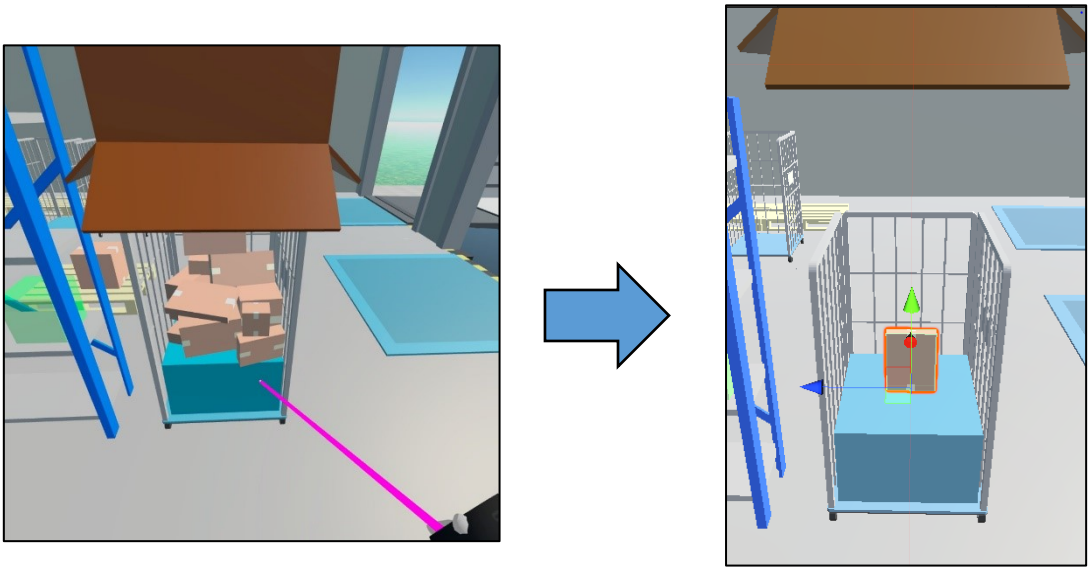
Obr. 6-11 "Box Collideru", vlevo ukázka "Mesh Collideru"

V Unity3D je k dispozici několik typů těchto detektorů kolize. Pro počáteční návrh rollkontejneru, byl využit detektor „Box Collider“, který vytvoří neviditelný kvádr o určitých rozměrech a poloze (viz Obr. 6-11), kde je pivot přiřazen k vytvořenému bodu v prostoru. Tímto detektorem byly prvotně vytvořeny čtyři mantinely, ohraničující stěny a dno rollkontejneru. „BoxCollider“ neumožnil umístění bedny do rollkontejneru z důvodu kvádrového ohraničení rollkontejneru. Po detailnějším prostudování nabídky detektorů kolize, byl vybrán detektor „Mesh Collider“, který nevytváří neviditelné kvádrové útvary, ale kopíruje tvar objektu (jeho „kostru“), ke kterému je přiřazen, viz Obr. 6-11 (levý obrázek). V případě této práce pomocí tohoto detektoru lze docílit přesnější interakce předmětů. Detektor byl aplikován na objekty přilehlé k pozici pracoviště, aby objekty v případě vzájemného kontaktu spolu interagovaly a bylo dosaženo důvěryhodnější simulace v reálném prostředí. V kapitole 6.5.5 je popsána událost v momentě kontaktu podrobněji.

6.2 Vizualizace nesetříděných a setříděných beden

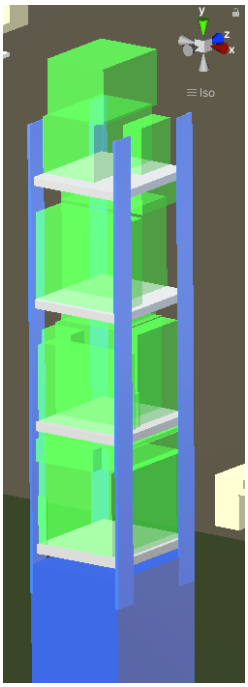
Za účelem snadnější operace s bednami a intuitivnější orientace byly v aplikaci vytvořeny dva generátory objektů. Generátorem je myšlen prázdný objekt, umístěný v prostoru skladu, specificky v prostoru nad rollkontejnerem, viz *Obr. 6-10*. Tomuto objektu je přidán skript, který provádí vytvoření nových objektů na základě vzoru objektu již existujícího – „Prefab“. První generátor má za úkol generování beden k umístění. Tento generátor je umístěn uvnitř krabice, která se „vznáší“ nad rollkontejnerem, viz *Obr. 6-10*. Z počátku bylo zamýšleno, že všechny nesetříděné bedny budou vygenerovány ve stejném okamžiku v prostoru palet (*Obr. 6-3* – vzdálenější seskupení palet). Avšak tento způsob by byl pro uživatele méně přehledný,

a proto byla nakonec zvolena varianta postupného zobrazení (generování) jednotlivých beden, viz *Obr. 6-15*. Bedny byly setříděny na základě jednoznačného identifikátoru bedny – identifikačního čísla bedny (ID). Uživatel tak má zhmotněnu pouze aktuální bednu ze seznamu beden a tu následně ukládá do vyznačeného prostoru určeného k umístění bedny v regálu. Tímto prostorem se myslí navigační průhledný kvádr zelené barvy. Funkcionalita navigačního systému je vysvětlena podrobněji v kapitole *6.4*. Uživatel tímto způsobem postupně přeskládá všechny bedny do libovolného prostoru v přistaveném regálu.



Obr. 6-12 Vygenerovaná bedna – hromadné vs. postupné generování beden

Druhý generátor, který generuje navigační kvádry, je umístěn ve stojanu, do kterého mají být bedny ukládány. Tento generátor má za úkol vytvářet kvádr tvarově totožný s právě vygenerovanou bednou z prvního generátoru, kde se vytváří bedny určené k umístění. Kvádr je vygenerován na pozici, která je určena ve vstupních datech viz kapitola **Chyba! Nenalezen z droj odkazů.**. Tyto kvádry imitující rozměry bedny k uložení, jsou dle vstupních dat setříděny pomocí použitého policového a gilotinového algoritmu. Navigačnímu kvádru („Kuboidu“) byl přiřazen průhledný materiál zářivě zelené barvy. Vlastnost průhlednosti byla přidána pro možnost pozorování prostoru uložení, před samotným vložením bedny. Barevné zvýraznění bylo využito pro lepší atraktivitu pozornosti na místo uložení bedny, se kterou v daný okamžik uživatel právě manipuluje. Generování začíná na spodním patře stojanu, kde do levého dolního rohu police je umístěn prázdný objekt (bod), na kterém se vygeneruje první navigační kvádr („Kuboid“). Ukázku celkového setřídění navigačních kvádrů je možné vidět v *Obr. 6-13*. Tento obrázek je využit pouze pro vizualizaci principu umístění navigačních kvádrů. Za běžného užívání je vygenerován pouze jeden navigační kvádr pro zachování přehlednosti. Systém podpůrné navigace je detailněji popsán v kapitole *6.4*.



Obr. 6-13 Vygenerované setříděné navigační kvádry v prostoru stojanu

6.3 Vytvoření uživatelského prostředí k provádění akcí

Pro ovládání aplikace uživatelem bylo vytvořeno uživatelské prostředí, které napomáhá uživateli provádět požadované akce jako je zvednutí bedny, přemístění, nebo generování nové bedny.

Aplikace je navržena s ohledem na jednoduchost ovládání a celkově na uživatelskou přívětivost. V rámci uživatelského prostředí (pohledu uživatele ve VR) bylo prvotně vytvořeno jednoduché menu, kde měl uživatel možnost pomocí stisknutí jednotlivých tlačítek (umístěných pouze v pohledu uživatele) ovládat aplikaci. Pro snadné ovládání byla implementována pouze tři tlačítka. V původním návrhu mělo první tlačítko („Spawn Box“) generovat novou skupinu beden na stejné ploše skupiny palet. Po implementaci změny je však vygenerována pouze jedna bedna z daného pořadí ve frontě (seznamu beden a jejich umístění v prostoru). Druhé tlačítko („Placement guide“) mělo zobrazovat cílovou polohu k uložení bedny na seskupení palet pomocí průhledného, zeleně zbarveného navigačního kvádru. Navigační systém pomáhá uživateli jak s orientací v prostoru, tak i s umístěním beden. Detailně je tato část popsána v kapitole *6.4*. Třetí tlačítko mělo ukládat provedené změny rozmístění beden a následného exportu dat do soboru typu *.json.

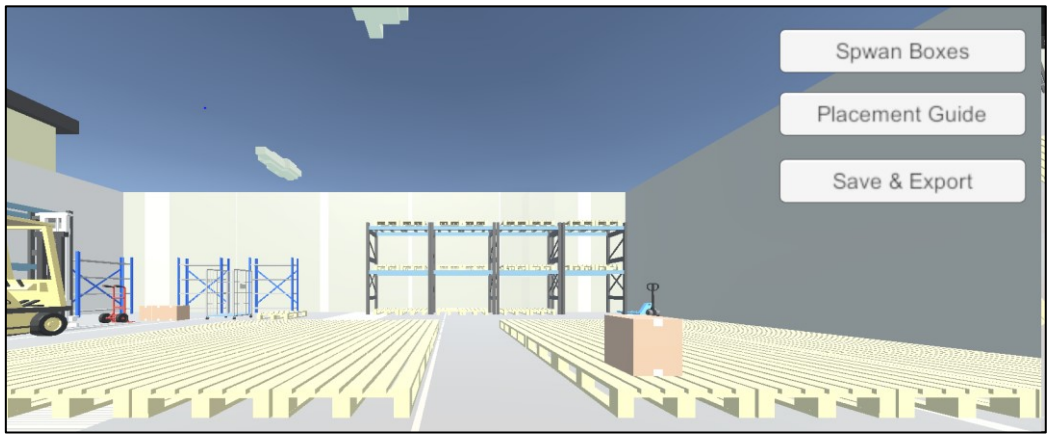
6.3.1 Menu k nahrání vstupních dat do prostředí aplikace

Pro možnost ovládání akce nahrávání vstupních dat bylo vytvořeno počáteční menu (viz *Obr. 6-14*), kde si uživatel může zvolit, zda chce pomocí tlačítka „Start“ nahrát uložení beden v souboru *.json, který je výstupem aplikace s algoritmy řešící umístění jednotlivých beden. Při stisku tlačítka „Load“ se nahrají uložená data z předchozího uloženého spuštění aplikace VR, kde již uživatel mohl bedny sám přesouvat na jiné pozice podle svého uvážení (toto nové umístění beden je ukládáno taktéž v souboru *.json). Kód pro funkcionalitu menu je možné vidět v příloze 2 na stránce 88.

*Obr. 6-14 Počáteční menu pro nahrávání dat ze souboru *.json*

6.3.2 Menu k ovládání generátorů

Z počátku bylo vytvořeno menu s tlačítky, které za normálních okolností bylo skryto a až po stisknutí jednoho z tlačítek na ovladači HMD (kontroloru), by se menu zviditelnilo. Následně by sledovalo pohyb HMD (směr pohledu) po dobu viditelnosti. Na *Obr. 6-15* je vizualizovaný prvotní koncept menu.



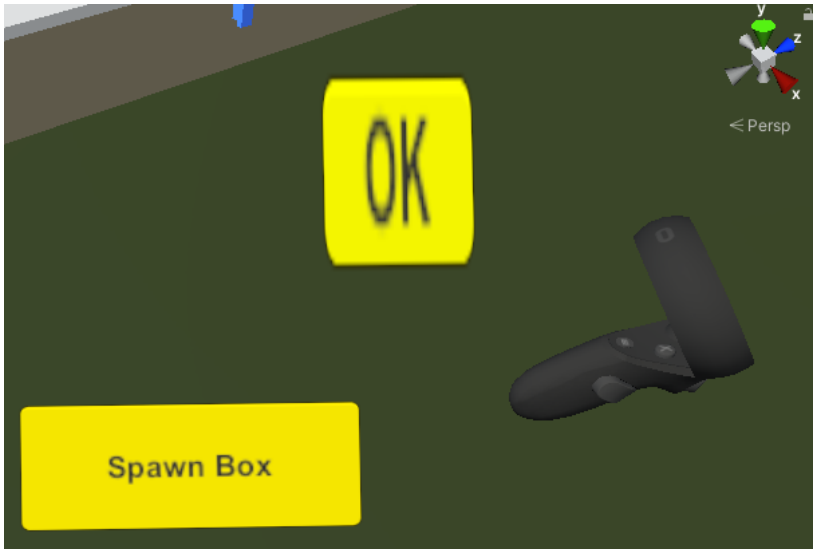
Obr. 6-15 Vizualizace pohledu uživatele na scénu s aktivovaným původním menu obsahující tlačítka

Po otestování této varianty se skrytým menu bylo ale usouzeno, že pro efektivnější práci s touto aplikací bude zvoleno dietetické viditelné menu [49], které je rychleji dostupné bez nutnosti stisku tlačítka pro zviditelnění. Tato menu také nezakrývá pohled uživatele, nebo nepřekrývá předměty v prostoru pracoviště. Menu bylo připnuto k objektu znázorňující levý kontrolér. Tím bylo docílena neustálá viditelnost díky trakci pohybu ovladače, a tudíž i samotného menu. Navigační systém „Placement Guide“ byl přemístěn do tlačítka dlaně – „Hand Trigger“ kontroléru Oculus Quest, viz Obr. 6-18. Uživatel může jednoduše aplikaci ovládat během operace skládání a zároveň jsou tlačítka uvnitř menu dobře

viditelná – viz Obr. 6-16. Z obrázku se může zdát, že tlačítka jsou nečitelná, avšak ve virtuální realitě jsou tlačítka zřetelně čitelná, a to i z pozice, ve které se nachází uživatel v momentě pořízení obrázku. Přemístěním tlačítek bylo dosaženo snazší orientace, kde v menu zůstalo pouze tlačítko „Spawn Box“ pro vygenerování nové bedny, navigačního kvádru spárovaným s bednou přiřadí přetočené dimenze (Width, Height, Depth) bedny, dle vstupních dat a nakonec přiřadí objektu „guideM“ ve třídě „TriggerDestroyGuide“ objekt z prostředí Unity3D, Assets. Popis funkcionality je detailněji popsán v kapitole *6.5.2* nebo v příloze 4 na stránce 94. Tlačítko je znepřístupněno ve chvíli, kdy je s aktuální bednou manipulováno. Jakmile je bedna umístěna do cílové pozice navigačního kvádru a pozice je zkontrolována dle validačního menu, viz kapitola 6.3.3, je tlačítko znovu zpřístupněno ke kliknutí a je možné vygenerovat další bednu v pořadí.

6.3.3 Menu pro validaci umístění bedny

Tlačítko „OK“ je již součástí jiného menu, a to validačního menu s názvem „IsPlacedMenu“. Tlačítko a samotné menu je implementováno za účelem validace ukládané bedny. Zda se jedná o správné propojení mezi právě ukládanou bednou a vygenerovaným navigačním kvádrem, který slouží k vytyčení místa k uložení bedny. Tlačítko se zviditelní v momentě, kdy je bedna, s pomocí „SnapDropZone“ umístěna do cílové pozice. Po potvrzení pozice se tlačítko zneviditelní, zničí se objekt „SnapDropZone“ a bedna je označena, jako uložená („IsPlaced“). Navíc jsou přiřazeny hodnoty atributům objektu reprezentující bednu. Funkcionalita tlačítka je provedena metodou „DestroySnapZone“ ve třídě „GlowManager“, viz příloha 5 na stránce 101 nebo kapitola *6.5.4*.



Obr. 6-16 Ukázka menu připojeného k levému ovladači Oculus Quest

6.3.4 Menu k ukončení menu

Pro ukončení aplikace bylo vytvořeno menu, kde lze pomocí tlačítek ovládat chod aplikace, nebo umístění všech beden do cílový pozic uvnitř polic stojanu, viz *Obr. 6-17*. Tlačítkem „Restart“ je možné znovu načíst aplikaci. U této varianty načtení se však určité početní hodnoty proměnných nenulují a je tak nutné počítat s rostoucím celkovým počtem

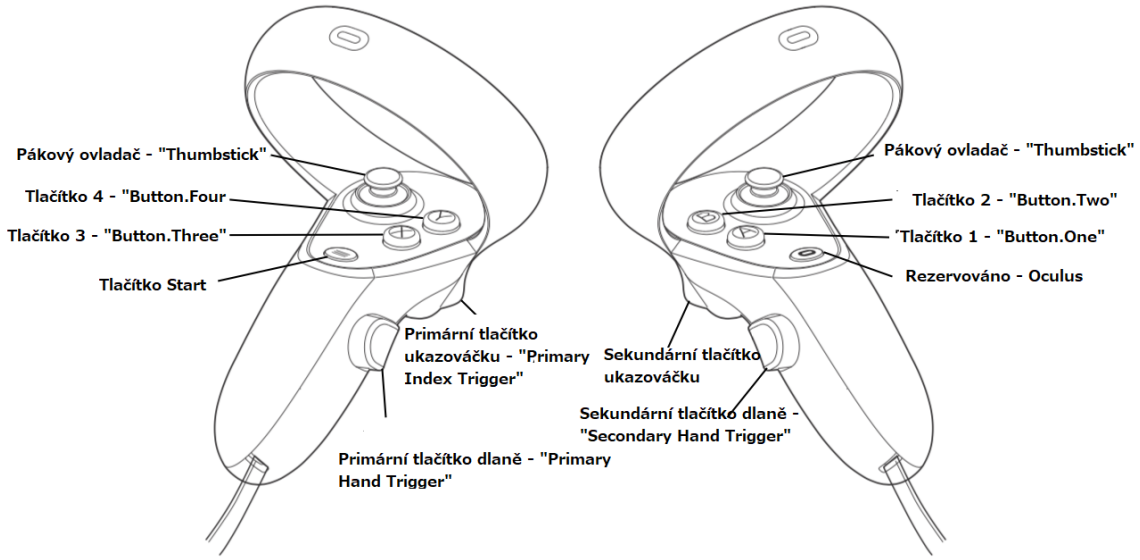
některých z počítadel. Tlačítko „Save & Exit“ má funkci uložení a ukončení aplikace. Kód, který tlačítko ovládá, ukládá vygenerované objekty beden s hodnotami přiřazenými k jednotlivým atributům v rámci metody „AddBoxToList“ ve třídě „BoxesFromJson“. Tlačítko „Exit“ ukončí aplikaci bez uložení dat.

Obr. 6-17 Vizualizace menu k ukončení aplikace

6.3.5 Ovládání aplikace pomocí Oculus Quest ovladačů

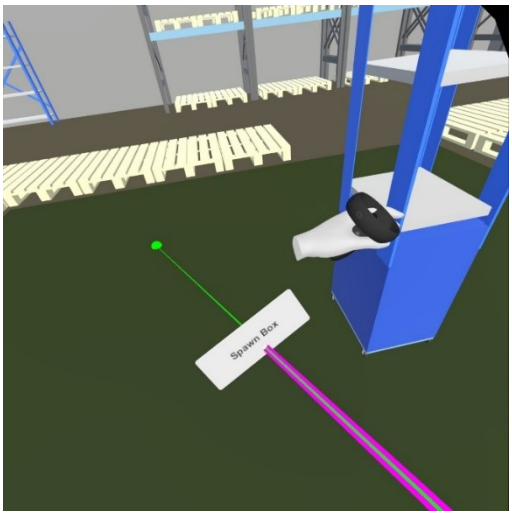
HMD zařízení pro vizualizaci virtuální reality je možné využít spolu s dvojicí ovladačů, viz *Obr. 6-18*. Pomocí ovladačů mohou být snímány pohyby končetin uživatele a zobrazeny ve virtuální realitě. Dále jsou na obrázku popsána tlačítka, pomocí kterých lze v aplikaci vykonávat určitou akci, která byla danému tlačítku na ovladači přidělena.

V rámci vytvořené aplikace je snahou klást důraz na snadné ovládání. K tomuto faktu se váže i nadefinování využití ovladačů HMD (Oculus Quest HMD a jeho ovladače) a funkce přiřazené tlačítkům. Pro lehčí orientaci v prostoru a zaměření se na určité místo, nebo předmět, byl přidán viditelný paprsek na pravém ovladači (fialově zbarvený) - viz *Obr. 6-19*. Tímto paprskem je možné zaměřit jak zhmotněné předměty, tak i objekty uživatelského prostředí, jako jsou tlačítka uvnitř řídícího menu. Díky dobré viditelnosti tohoto paprsku je snazší interakce s menu, než kdyby byl využit paprsek, který je nutné zviditelnit. K interakci s uživatelským prostředí není tak nutné držet ve stisku „Thumbstick“ a následně stisknout další z tlačítek, určených k interakci s menu (Tlačítko 1 – „Button-One“).



Obr. 6-18 Ovladače pro snímání pohybu ve VR a provedení akcí uživatelem s popisem jednotlivých tlačítek [42]

Levý paprsek je viditelný pouze tehdy, pokud se stiskne pákový ovladač „Thumbstick“, který je vizualizován na ovladačích – viz *Obr. 6-18*. Tento paprsek slouží k přemístění uživatele ve virtuálním světě. V případě této aplikace je však prostor vymezen pouze na plochu budovy skladu, jelikož se všechny děje odehrávají uvnitř skladových prostor. V případě rozšíření plochy určené k možnosti pohybu uživatele, stačí pouze upravit štítek („Tag“) jednotlivých objektů (chodník, silnice, budova, …), které by se chtěli zpřístupnit uživateli pro pohyb. Štítek lze nalézt v záložce „Inspector“ jednotlivých objektů aplikace v Unity3D. Přemisťovací paprsek lze vyvolat stisknutím pákového ovladače jak na pravém, tak i levém ovladači Oculus Quest. Na následujícím obrázku (viz *Obr. 6-19*) je demonstrován přemisťovací paprsek pomocí levého ovladače a je možné si všimnout, že paprsek je zbarven do červena. To je zapříčiněno tím, že cílová plocha, na kterou paprsek ukazuje, není zpřístupněna k přemístění uživatele. Pokud by se barva paprsku změnila na zelenou, znamená to, že se uživatel na dané místo může přemístit. Jak bylo zmíněno již v předchozích kapitolách týkajících se uživatelských menu, tlačítkům dlaně („Hand Trigger“) byla přiřazena funkce „Placement Guide“ při stisku. Po uvolnění tlačítka se funkce zvýraznění lokality pro umístění bedny sama uvede do neaktivního stavu. Dále tlačítku ukazováčku („Index Trigger“) byla přiřazena funkce pro uchopení předmětů (beden).



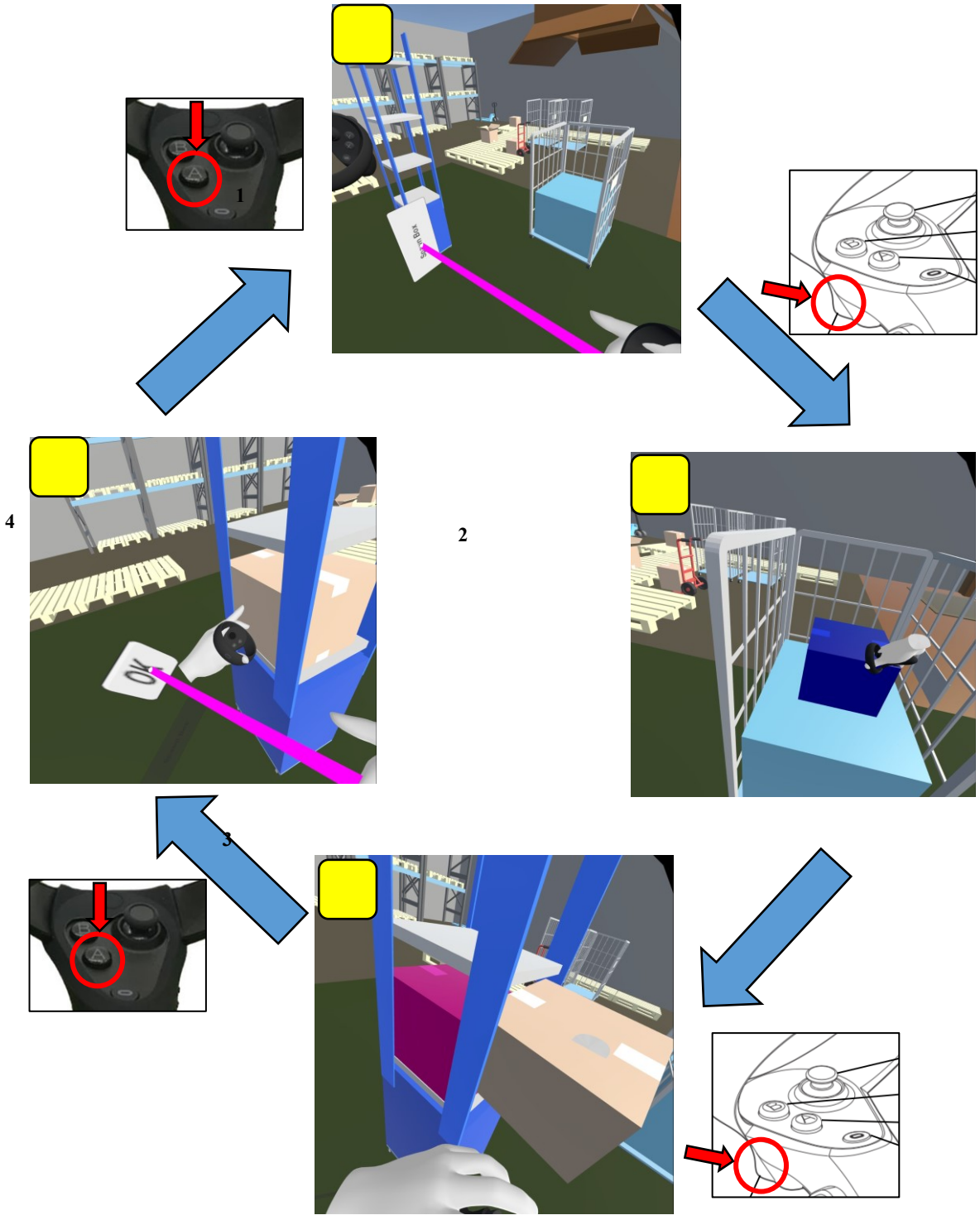
Obr. 6-19 Vizualizace zaměřovacích paprsků při interakci s tlačítky menu("Pointer")

6.3.6 Postup pro přemístění bedny

Sekvence úkonu pro přesun bedny v rámci rozmístění pracoviště (viz *Obr. 6-20*), byla definována do čtyř jednoduchých kroků. Pro zahájení postupu je nutný přesun do libovolné ideální pozice na pracovišti. Doporučenou pozicí pro začátek sekvence přesunu bedny je prostor před rollkontejnerem na pracovní podložce ve vzdálenosti dosahu natažených paží na bednu, viz *Obr. 6-20* – obrázek 1 (horní obrázek s popiskem 1 ve žlutém čtverci). Uživatel je připraven k zahájení úkonu a při předklonu či natažení paží je schopný na bednu dosáhnout. Prvním krokem je zmáčknutí tlačítka pro vygenerování bedny „Spawn Box“ zaměřením se ukazovátkem na tlačítko a stisknutím tlačítka kontroléru „A“ (Tlačítko 1 – „Button One“). Stisknutí tlačítka zároveň vygeneruje spárovaný navigační kvádr, který slouží jako vizualizace cílového umístění. Přiřadí se velikost bedny v jednotlivých dimenzích, dle upraveného stavu metodou „AssignScale( )“. Proměnným ve třídě „TriggerDestroyGuide“ jsou přiřazeny hodnoty. Tato třída je přiřazena k prefabu bedny. Zároveň se v rámci třídy (statické proměnné "guideM") přiřadí objekt "GuideManager" pro propojení navigačního systému s aktuálním objektem bedny.

Při kontaktu uživatele s bednou (pomocí kontroléru), se barva bedny změní z výchozí barvy na tmavě modrou, viz *Obr. 6-20* – obrázek 2. Na obrázku je názorně vidět kontakt ruky uživatele a bedny. Při stisknutí tlačítka ovladače (kontroléru) „Index Trigger“ (které je na obou ovladačích znázorněné v *Obr. 6-18*), lze bednu díky komponentě určené k přemístění „VRTK_InteractableObject“ objektu, tj. přidružené k bedně, lze bednu ovladačem uchopit a držet (virtuální reference na ruce uživatele). Při opětovném stisknutí tlačítka se úchop uvolní a bedna díky nasimulované gravitaci spadne na zem, nebo je umístěna na objekt, např. přilehlý stojan, který má přidělenou komponentu pro detekci kolize, viz *Obr. 6-20* – obrázek 3.

Bedna se následně umístí do pozice vygenerovaného spárovaného navigačního kvádru. Ten se při dotyku s bednou zničí (destruktor) a zůstává pouze objekt „SnapDropZone“, který má za úkol lehčí umístění předmětu do vyhrazeného prostoru (prostor o stejných dimenzích jako má bedna i navigační kvádr). Následuje potvrzení pozice bedny pomocí validačního tlačítka „OK“. Následuje vygenerování další bedny v pořadí pomocí stisku tlačítka „Spawn Box“.



Obr. 6-20 Vizualizace postupu při skládání beden z počátečního místa generování na cílové místo na polici ve stojanu

6.4 Navigační systém pro umístění beden

Za účelem lepší orientace při provádění procesu uskladnění bedny v prostředí pracoviště, byla vytvořena podpůrná navigace pro ukládání beden do stojanu. V případě této práce je aplikace nápomocného systému demonstrována pouze na jednom stojanu. Pokud by se brala v potaz reálná situace skladu, kde by bylo těchto stojanů v prostorech skladu několik, mohla by nastat situace, kdy by dohledání cílového místa uložení pouze pomocí navigačních kvádrů bylo obtížné. Z tohoto důvodu byly vytvořeny další dvě složky navigačního systému, které jsou popsány v této kapitole. Kód s detailnějším vysvětlením funkcionality všech navigačních prvků je vysvětlen v kapitole 6.5 této práce.

6.4.1 Navigační kvádr

Navigace obsahuje čtyři druhy podpůrných prvků pro umístění bedny v prostoru police stojanu. Prvním ukazatelem cílové pozice pro uložení je průhledný zelený navigační kvádr o stejných rozměrech, jakými disponuje bedna („Kuboid“), se kterou je právě manipulováno. Tento prvek slouží převážně k finálnímu uložení bedny na polici stojanu, kdy uživatel je v těsné blízkosti místa uložení. Tento prvek má též přiřazenou funkci destrukce při kontaktu s manipulovanou bednou pomocí komponenty „Collider“ z důvodu zachování přehlednosti. Tento prvek je jako jediný viditelný od okamžiku generování nové bedny. Ostatní prvky navigace je nutné aktivovat pomocí zmáčknutí tlačítka „Placement guide“ (pomocníka pro umístění). Pro vizualizaci zeleného navádějícího kvádru je uveden původní návrh paletového rozložení pracoviště – viz *Obr. 6-4.*

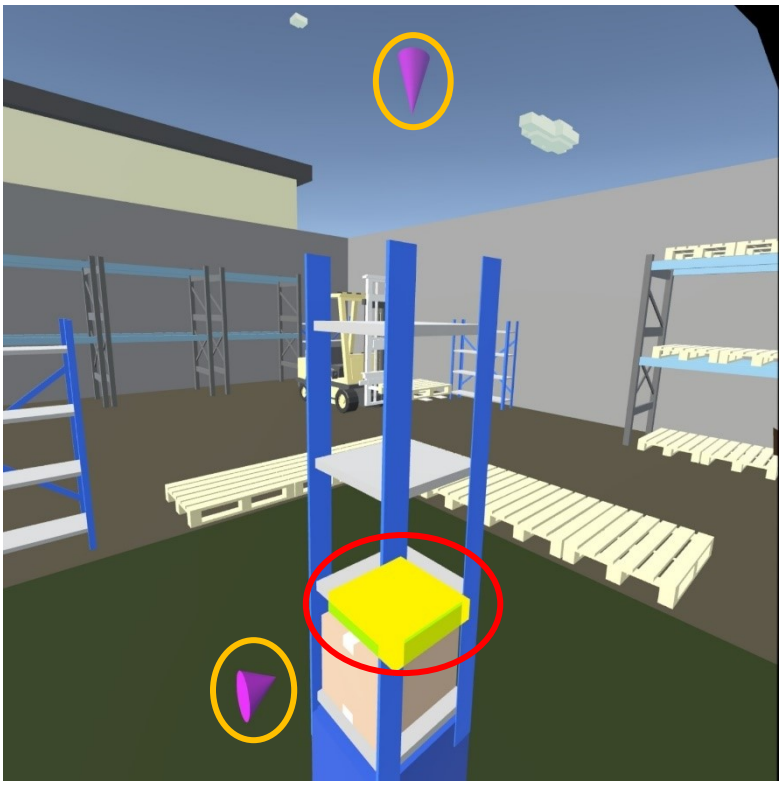
6.4.2 Navigační ukazatele – kužele

Druhým navigačním prvkem jsou dva kužele, které se vznášejí před (osa z) a nad (osa y) stojanem, kde jsou kužele vyznačeny oranžovými kruhy – viz *Obr. 6-21.* Kužele jsou neviditelné do té doby, než je zavolán navigační pomocník (stisknutím tlačítka „Placement guide“). Kužele si vůči stojanu udržují stálou polohu, ale při generování nového navigačního kvádru se kužele přesunou na střed nově vygenerovaného pomocného kvádru. Zároveň při destrukci (zrušení objektu) aktuálního navigačního kvádru se v okamžiku reakce „Collideru“ (detekce kolize) s bednou k uložení navigační kužele zneviditelní. Opětovně se zviditelní se stisknutím tlačítka pro navigaci pro jinou bednu určenou k uložení. Tento prvek je viditelný z větší vzdálenosti a v případě existence vícero stojanů, lze pomocí něj snadněji lokalizovat místo pro uskladnění bedny. Aktivace kuželů je spuštěna stiskem a podržením tlačítka „Hand Trigger“ na hardwarovém kontroléru, viz kapitola 6.3.5.

6.4.3 Zvýraznění navigačního kvádru

Třetím prvkem je přidělení navigačnímu kvádru zvýrazňující efekt v podobě vyzařujícího povrchu pomocí změny materiálu přiděleného k objektu. Navigační kvádr (viz *Obr. 6-21*) vyznačený v červeném kruhu, je díky volbě tohoto materiálu lépe zpozorovatelný (základní objekt obklopuje zářivý „obal“). Navíc je možné objekt, který má přiřazen tento materiál, vidět i skrze ostatní zhmotněné předměty, jako je například stojan. Tento navigační prvek, stejně jako navigační kužely, lze zobrazit (zaktivovat) pomocí stisknutí tlačítka navigace umístění. K docílení lepší intuitivnosti byla brána v potaz komponenta „Halo“, která vytvoří zář kolem

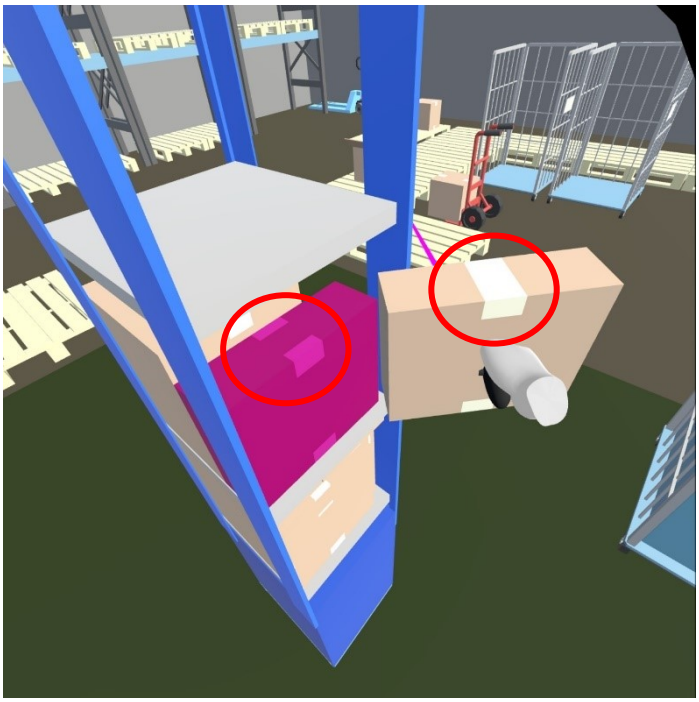
objektu, ale z určitých pohledů tato varianta nebyla dobře viditelná. Stávalo se, že záře procházela i skrze objekty umístěné v těsné blízkosti ohniska, což zapříčinilo nejednoznačné určení pozice k uložení, a proto komponenta nebyla v navigačním systému použita. Funkce zvýraznění je spuštěna spolu s aktivací kuželů stiskem a podržením tlačítka „Hand Trigger“.



Obr. 6-21 Vizualizace druhého a třetího navigačního prvku

6.4.4 Zóna pro přesné umístění bedny

Posledním navigačním prvkem pro uložení bedny do cílové pozice (pozice navigačního kvádru), je pozice objektu „SnapDropZone“ („Zóna“) - viz *Obr. 6-22*. Zóna ukládá vybraný interaktivní objekt (bednu) přesně na specifikovanou cílenou pozici. Zóna disponuje stejnými dimenzemi, jako objekt, který je do ní umisťován (pokud není naprogramována odlišná funkcionalita) [8]. Navíc je i možné v *Obr. 6-22* pozorovat natočení bedny v jednotlivých dimenzích a fialově zvýrazněného objektu Zóny, kde objekt Zóny udává správné natočení (rotaci) bedny při ukládání. Atributy zóny se přiřazuji z větší části v kódu třídy „BoxesFromJsonForSorted“, kde jsou přiřazovány atributy objektu bedny aktuálně určené k uložení. Samotnému objektu je dále přiřazeno ID pro možnost spárování s navigačním kvádrem a aktuální bednou k uložení pomocí třídy „SnapMatcher“. Funkcionalita kódu je vysvětlena v kapitole 6.5.3 a v příloze 9 na straně 113.



Obr. 6-22 Vizualizace objektu zóny pro přesné umístění "SnapDropZone"

6.5 Programová část modelu VR

Pro správnou funkcionalitu modelu VR a možnosti provádět požadované úkony procesu uložení beden na pracovišti musí být chování jednotlivých prvků naprogramováno. Pro psaní a úpravu zdrojového kódu bylo využito vývojářské prostředí Visual Studio.

Z počátku vývoje aplikace, kdy bylo tvořeno pouze prostředí a objekty (viz kapitola 6.1) a to především kvůli logickému uspořádání objektů do skupin, dle účelu a role pro výkon procesu. Dále toto uspořádání mělo důležitou roli při psaní kódu pro generátor beden a navigačních kvádrů, kde se pozice objektů v prostředí aplikace odvíjí v závislosti na nadřazeném objektu – v případě generátoru beden se jedná o objekt „SpawnDropBox“ s dalšími objekty v hierarchii (Obr. 6-10), nebo „ShelfForBoxes“ (Obr. 6-7) pro uskladnění vygenerovaných beden.

V podkapitolách jsou rozepsány a stručně vysvětleny jednotlivé stěžejní programové části a proto je vhodné vysvětlit několik užívaných pojmů:

- „prefab“ – vzor pro vytvoření, konfiguraci a uložení herního objektu („GameObject“) se všemi komponentami a hodnotami připojenými k prefabu (lze vícenásobně použít pro různé účely – „Asset“), který lze instancionalizovat uvnitř scény aplikace [39]
- Bedna – kódové vyjádření bedny (v „BoxesClass“ třída „Cuboid“ – čtení „Cuboids“ ze vstupních dat)
- Navigační kvádr – kódové vyjádření navigačního kvádru (v „BoxesClass“ třída „Cuboid“ – čtení Results - „PatroBedny“ ze vstupních dat)
- „SnapDropZone“ – prefab pro zónu („Snap“) provádějící přesné umístění bedny
- Objekt bedny – vygenerovaný objekt uvnitř aplikace zastupující aktuální bednu k uložení (instancionalizovaný prefab bedny)

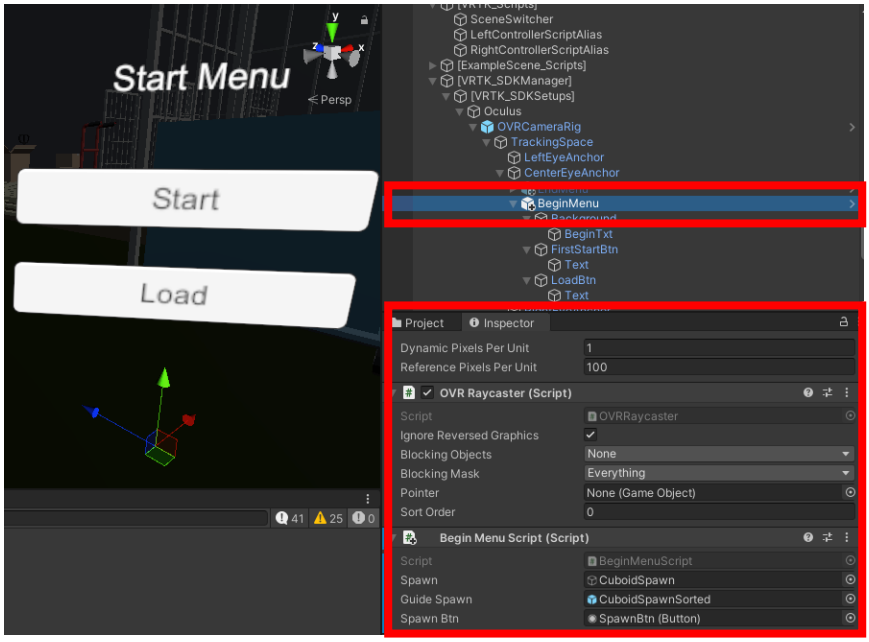
6.5.1 Import dat do prostředí VR

Prvním krokem k dosažení požadované funkčnosti aplikace, je import dat z poskytnutého souboru formátu „*.json“ s názvem „JsonTextBedny“, který byl popsán v kapitole 5.1 a celá využitá část kódu pro účel této aplikace je dostupná v příloze 1 na stránce 80.

6.5.1.1 Způsob načtení dat (třída BeginMenuScript)

Pro zajištění požadované funkcionality jsou pomocí kódu vytvářeny objekty generátoru beden („CuboidSpawn“) a generátoru navigačních kvádrů (CuboidSpawnSorted). Třída obsahuje dvě metody pro spuštění následujících funkcí: první funkce „StartApp( )“ zkontroluje, jaká data se mají načíst dle výběru uživatele (tlačítka „Start“ a „Load“). V případě stisknutí „Start“ se načtou vstupní data vygenerovaná z aplikace pro řešení BPP. V druhém případě („LoadApp( )“) se načtou uložená data z předchozího uloženého stavu této aplikace. Pro případ chyby je implementován kód pro zpracování výjimek [45]. Dále je ve třídě obsažena metoda „ReadFromFile(string filename)“, která umožňuje načtení souboru z definované složky „filename“ v počítači uživatele (metoda pro obdržení cesty – „GetFilePath(string filename)“.

V průběhu testování aplikace byla testována data k načtení. V případě neexistence složky s daty k načtení je tlačítko „LoadBtn“ deaktivováno.



Obr. 6-23 Objekt "Start Menu" uvnitř aplikace

6.5.1.2 Čtení vstupních dat (třída „BoxesClass“)

Pro správnou vizualizaci rozměrů a pozic beden ve VR bylo nutné převést data z textové podoby do podoby kódu programovacího jazyka C#. Tato část byla vyřešena pomocí vytvoření třídy BoxesClass, viz příloha 3 na stránce 91, která se zakládá na poskytnuté třídě využívané v aplikaci pro řešení BPP. Převod dat z textové podoby byl proveden pomocí deserializace dat, která zajišťuje převedení proudu bytů zpět do podoby kopie původního kódu [43]. Serializace a následná deserializace je používána pro přesun objektu mezi aplikacemi a pro případ této práce byla využita pro vytvoření kopie původního objektu jiné aplikace nevyužívající Unity3D

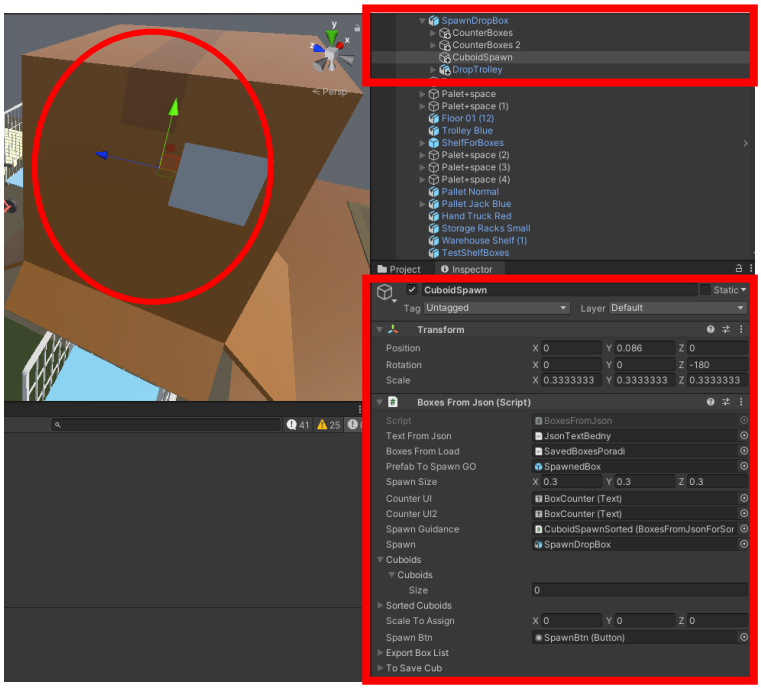
[3]. V poskytnutých datech jsou uvedeny rozměry beden – šířka (atribut „Width“), výška („Height“) a hloubka beden („Depth“) - vizkapitola 5.1. Jednotlivé rozměry bedny v různých dimenzích načtené ze vstupních dat jsou přiřazeny atributům třídy „BoxesClass“.

V rámci určitých částí kódu bylo zapotřebí použít převodu na metry. Pokud by se rozměry nepřevedly, Unity3D by rozměry specifikované ve vstupních dat bralo jako metry, které jsou výchozí jednotkou aplikace. V případě rozměrů krabic by přímé čtení rozměru mohlo zapříčinit nadměrnou velikost beden, což je nežádoucí. Pro převod byla vytvořena metoda („calculate ( )“) typu „Vector3“, která pracuje s trojrozměrným vektorem a umožňuje tak pracovat se všemi rozměry bedny v jeden okamžik. Převedení jednotek bylo provedeno též pro pozice beden ve vstupních datech („X“, „Y“ a „Z“). Další metoda slouží pro úpravu textového formátu v případě nutnosti převádět obsah třídy do textové podoby.

Do třídy „BoxesClass“ byla přidána vnořená třída, která byla vytvořena na základě poskytnutých dat „ResultsPatro“. Pomocí třídy jsou generovány navigační objekty beden na základě načtených dat. Popis je uveden v kapitole *6.5.3*. Datové složky třídy popisují číslo patra, ve kterém se umístěná bedna nachází a na seznam beden umístěných v daném patře. V případě této vnořené třídy, se jedná o výsledky optimalizace uložení beden provedené pomocí algoritmu aplikace pro řešení BPP. Zdrojový kód třídy „BoxesClass“ je uveden v příloze – viz příloha 3 na stránce 91.

6.5.2 Manažer pro generování beden k uložení (třída „BoxesFromJson“)

V dalším kroku vývoje aplikace byl vytvořen generátor beden – viz Obr. 6-12. Generátor,spadající pod objekt s názvem „CuboidSpawn“ v aplikaci zastupuje třídu s kódem pro generování beden s názvem „**BoxesFromJson**“, viz příloha 4 na stránce 94. Objekt je nazván manažerem z důvodu zastupování třídy uvnitř aplikace. Manažer byl přiřazen na požadovanou pozici a následně byl psán kód pro základní funkčnost – viz *Obr. 6-24*. Zbytek této podkapitoly popisuje funkcionalitu třídy „BoxesFromJson“.



Obr. 6-24 Vizualizace manažeru pro generování beden jako objektu uvnitř aplikace

Na počátku bylo nutné deklarovat atributy třídy, pomocí kterých jsou pomocí kódu vytvořeny objekty aplikace, na základě vstupních data určujících rozměry beden („Cuboids“) ze souboru „JsonTextBedny.json“. Pro tento účel byla vytvořena proměnná „textFromJson“ datového typu „TextAsset“. Proměnná „boxesFromLoad“ stejného typu je použita v rámci třídy pro načtení vstupních dat z uloženého stavu aplikace. Uložená data „SavedBoxes.json“ musí být přiřazena proměnné manuálně. Model bedny z „Assets“ v Unity3D. Proměnná „spawnSize“ slouží k určení místa generování beden. Proměnné „counterUI“ a „counterUI2“ jsou počítadla, ukazující zbývající počet beden k umístění. Proměnná „spawnGuidance“ slouží k propojení s generátorem navigačních kvádrů. Poslední deklarovanou proměnnou je „spawn“, jež zastupuje objekt generátoru beden. Proměnná „prefabToSpawnGO“ slouží jako „nosič“ modelu bedny, dle kterého se bedny budou generovat v prostředí aplikace.

Metoda „LoadBoxes( )“ byla implementována pro využití v rámci „Start Menu“ pro načtení vstupních dat. Metoda „SaveBoxes( ) naopak data ukládá pod textovou proměnnou „toJsonEnd“ pomocí serializace dat. Metoda „SortBoxes( )“ byla vytvořena z důvodu rotace bedny v jednotlivých dimenzích (pro seznam beden „Cuboids“ před podrobením výpočtu algoritmu). V případě načtení uložených dat z předchozího spuštění aplikace pro VR jsou data načtena ze souboru „SavedBoxes.json“ namísto původního souboru „JsonTextBedny.json“.

V rámci třídy „BoxesFromJson“ byla vytvořena vnořená třída „CuboidList“ pro deserializaci dat do datového typu „CuboidList<Cuboid>“ s názvem „Cuboids“ . Tento seznam je v kódu využit k vytvoření seznamu beden k uložení ze vstupních dat a umožňuje s ním pracovat v prostředí třídy.

Dále jsou ve zdrojovém kódu využity proměnné jejichž význam bude vysvětlen v bodech:

- „newOrLoed“ – kontrola způsobu načtení počátečních dat
- „cub“ – zastoupení objektu bedny, kterému jsou měněny hodnoty atributů i v jiných třídách
- „counterBoxes“ – počítadlo uložených beden
- „prefabToSpawn“ – objekt zastupující prefab bedny
- „scaleToAssign“ – trojrozměrný vektor pro změnu rozměru beden v jednotlivých dimenzích
- „obj“ – zastoupení objektu bedny pro zpřístupnění objektu ostatním třídám
- „spawnBtn“ – tlačítko pro generování nové bedny
- „exportBoxList“ – třída „CuboidList“ zastupující generický list beden určených k exportu

Metoda „SpawnBoxes( )“ generuje bedny dle proměnné „prefabToSpawnGO“ a mění hodnoty jejich atributů dle dat ze souboru „JsonTextBedny“. Objekt bedny je vytvořen pomocí funkce „Instantiate( )“. Dále jsou na objekt bedny aplikovány modifikace dle vstupních dat. Proces úpravy objektu bedny („prefabToSpawn“) uvnitř třídy „BoxesFromJson“ začíná selekcí první bedny ze seznamu, následuje přiřazení rozměrů v jednotlivých dimenzí, pozice a rozpoznávacích znaků (“Tag” a “IsPlaced”). Štítek „Tag“ identifikuje objekt bedny, který vyvolává událost kolize s navigačním kvádrem. Detailněji je párování popsáno v kapitole *6.5.5*. Metoda též přiřadí ID bedny komponentě připojené k modelu bedny „IDReceiver“, dle kterého se provádí párování objektů. Na konci metody „SpawnBoxes( )“ je vybraná bedna (první ze seznamu) odstraněna ze seznamu a nahradí ji bedna následující v pořadí.

Proměnná s názvem “cub” je deklarována s modifikátorem “static“ z důvodu zpřístupnění proměnné druhému generátoru za účelem synchronizace postupné inicializace s generováním

navigačních kvádrů. Vzhledem k tomu, že generátor navigačních kvádrů generuje kvádry v závislosti na aktuální bedně k umístění, bylo nutné zajistit prvotní inicializaci generátoru beden a následně generátoru navigačních kvádrů. Na stejném principu provázanosti je založeno párování „SnapDropZone“ (kapitola 6.5.6). Posloupnost inicializací byla docílena pomocí vytvoření metody „SpawnNextBox( )“, která má za úkol dodržet správné pořadí spouštění metod. V případě, že vstupní data byla nahrána z uložených dat aplikace pro VR, je přistoupeno k druhé části podmínky pro kontrolu formy vstupních dat, podle které jsou bedny vygenerovány na uložených pozicích, o daných rozměrech v jednotilvých dimenzích, rotacích, s přiřazeným štítkem objektu „PBox“ a pořadím, ve kterém byly bedny umístěny. Následně jsou hodnoty atributů beden přiřazeny k atributům objektů beden.

Metoda „Rotate( )“ slouží ke změně hodnot dimenzí objektu bedny dle dimenzí dané bedny, která byla v rámci vstupních dat podrobena výpočtu algoritmu (v rámci „JsonTextBedny.json“ se jedná o seznam „Results“). Metoda „AssignScale( ) přiřadí dimenze podrobené rotaci z metody „Rotate( )“ objektu bedny a uloží je též každému z objektů do třídy „IDReceiver“ připojené k prefabu bedny.

Poslední metodou této třídy je „AddBoxToList( )“, která ukládá bedny s jejich atributy do generického listu s datovým typem „Cuboid“ s názvem „exportBoxList“, který je pomocí serializace ukládán do souboru „SavedBoxes.json“.

Po testování aplikace byly dodatečně přidány části kódu řídící situace výskytu výjimek v částech kódu náchylných k poruchovosti. Dále byla přidán kód pro změny hodnoty proměnné „allowNavi“, která ve třídě „VRTK_ControllerEvents“ určuje, zda je možné aktivovat navigační systém. V rámci testování ukončení aplikace byla přidána podmínka metodě „Rotate ( )“, pokud by v případě načtení uložených dat nebyl dostupný objekt zastupující bednu „obj“ (hondoty „null“). V rámci testování ukončení aplikace bylo deaktivováno tlačítko „Spawn Boxes“ po validaci polohy poslední bedny ze seznamu k uložení.

6.5.3 Manažer pro generování navigačních kvádrů (třída „BoxesFromJsonForSorted“)

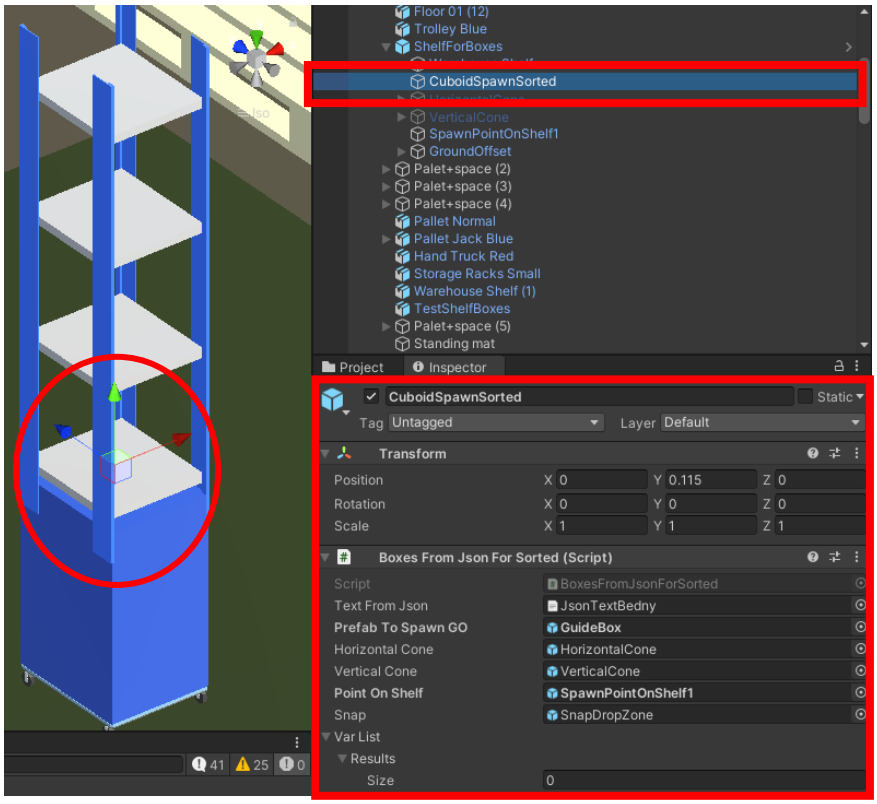
Vytvoření a naprogramování generátoru pro tvorbu navigačních kvádrů s názvem „BoxesFromJsonSorted“, viz *Obr. 6-13*, proběhlo obdobným stylem, jako tomu bylo u tvorby generátoru beden v předchozí kapitole. Manažer byl umístěn na dno první police stojanu, viz *Obr. 6-25*. Na počátku třídy jsou deklarovány proměnné napodobující styl deklarace proměnných v prvním generátoru. U této třídy byly navíc přidány proměnné pro udání výšky patra, „shelfHeight“, objekty zastupující navigační kužele, proměnná vymezující mezery mezi vygenerovanými navigačními kvádry pro zamezení kolize a nakonec objekt „snap“, kterému v kódu je přiřazen prefab „SnapDropZone“ pro tvorbu zón k uchycení bedny na pozici. Rozdílů oproti předchozímu generátoru je několik. Prvním větším je odlišnost v načtených datech. V tomto generátoru se načítají data pro bedny, které byly podrobeny výpočtovému algoritmu pro BPP a mají již přiřazené pozice a byly podrobeny případné rotaci v dimenzích. V rámci tohoto generátoru jsou nahrávány data z části vstupních dat spadající pod kolonku „Results“. Pro import počátečního bodu do třídy byl vytvořen objekt zastupující tento bod uvnitř modelu aplikace. V rámci stojanu byl bod umístěn do rohu první police (pravidlo levého dolního rohu, viz kapitola 3.3.1). Tato pozice nadále slouží jako výchozí počáteční pozice pro generování navigačních kvádrů na policích stojanu.

Obdobně s prvním generátorem byla vytvořena vnořená třída pod názvem „JsonClass“, která má na starosti deserializaci dat ze souboru „JsonTextBedny“. Třída ukládá do seznamu datového typu „BoxesClass“ data, dle vnořené třídy „ResultPatro“ (bedny uložené v patrech). Následně je použit konstruktor pro vytvoření nového seznamu pod názvem „varList“, který je využíván k selekci požadovaného kvádru ze seznamu v závislosti na vygenerované bedně k umístění. Načítání kvádrů z textového formátu „JsonTextBedny“ je provedeno stejným způsobem, jak je tomu u manažera pro generování beden k uložení.

Dále jsou deklarovány proměnné využívané v metodě „SpawnBoxesSorted( )“:

- cuboidFromBuffer – aktuálně vygenerovaná bedna z „CuboidSpawn“ - za účelem párování
- obj – deklarace statické proměnné pro zpřístupnění proměnné objektu navigačního kvádru ostatním třídám
- snapDes – objekt reprezentující objekt „snapGuide“ určený k odstranění (destrukci) v případě splnění podmínky umístění
- cubToDes – objekt reprezentující navigační kvádr určený k odstranění

V metodě „SpawnBoxesSorted( )“ jsou přiřazovány hodnoty objektu navigačního kvádru dle vstupních dat ze souboru *.json, viz kapitola *6.4*. Metoda začíná přiřazením hodnoty proměnné „cuboidFromBuffer“, který se odkazuje na statickou proměnnou „cub“ uvnitř třídy „BoxesFromJson“ připnuté k objektu generátoru beden. Je vytvořen generický list „cuboids“, který obsahuje všechny navigační kvádry (v rámci vstupních dat – bedny s přiřazenou pozicí a rotací). Nad tímto listem se následně v aplikaci provádí postupné procházení pomocí vyřazení prvního objektu listu, který je nahrazen objektem následným (postupné zmenšování celkového počtu objektů uvnitř listu). Před přiřazováním hodnot atributům objektu navigačního kvádru je kontrolováno podmínkou párování navigačního kvádru „cub“ s aktuální bednou k umístění „cuboidFromBuffer“ pomocí porovnání „ID“ atributy. Když kód splní předchozí podmínku, je následně kontrolována pozice navigačního kvádru (v jakém patře se nachází). V případě, že patro (police) obsahuje daný navigační kvádr, začne metoda přiřazovat hodnoty atributům objektu navigačního kvádru „gO“. Objektu „gO“ má pro párování přidělenou třídu „IDMatcher“, která ukládá přiřazené ID do proměnné „IDGuideBox“ (je přiřazováno ID aktuální bedny k uložení). Proměnné „cubToDes“ je přiřazen navigační kvádr, který bude v případě kolize s objektem bedny odstraněn (destrukce objektu navigačního kvádru). Statické proměnné třídy „TriggerDestroyGuide“ (třída připnuta na prefab bedny) je přiděleno ID navigačního kvádru pro spárování kvádrů a beden.



Obr. 6-25 Vizualizace manažeru pro generaci navigačních kvádrů

Pro možnost odkazování se na právě vygenerovaný navigační kvádr, byla přidána proměnná s názvem „obj“, která dovoluje ostatním třídám se na tuto proměnnou odkazovat díky „static“ modifikátoru datového přístupu.

V případě navigačních kvádrů bylo nutné přidat koeficienty na přepočet skutečné velikosti modelu kvádru (bedny). Koeficienty jsou datového typu „float“ s názvy “koefX” a “koefYZ”, jelikož rozměry v jednotlivých dimenzích jsou ve skutečnosti nižší, než je měřítko „Scale“ na modelu bedny/kvádru. Měřítko modelu bedny bylo importováno společně s celým balíčkem pro tvorbu prostředí skladu [2]. Přepočet byl nutný z důvodu správného umístění navigačních kvádrů do police stojanu.

V další části kódu metody „SpawnBoxesSorted( )“ jsou přiřazovány hodnoty atributům objektu Zóny pro přesné umístění bedny („SnapDropZone“) dle hodnot objektu navigačního kvádru. Objektu Zóny je též přiděleno ID v rámci připnuté třídy „SnapMatcher“ (proměnná „snapID“ pro možnost párování s navigačním kvádrem.

V poslední části byl přidělen štítek („Tag“) pro rozeznání objektu kvádru v momentě detekce kontaktu s bednou k uložení. Následuje kód pro nastavení pozic navigačních kuželů v závislosti na pozici stojanu a vygenerovaného navigačního kvádru, kde kužele směřují svým hrotem na prostředek kvádru. Nakonec je odebrán kvádr ze seznamu „PatroBedny“ pro docílení logiky procházení listu pomocí načtení první položky a smazání pro uvolnění místa následující položce ze seznamu.

Během testování v metodě „SpawnBoxesSorted( )“ bylo přidáno přiřazení hodnoty proměnné „wasDestroyed“, na kterou je odkazováno do třídy „GlowMaager“ (statická

proměnná) za účelem označení objektu navigačního kvádru, jako stále existujícího objektu (nedestruovaného dle třídy „TriggerDestroyGuide“).

6.5.4 Manažer pro ovládání navigace ukládání beden (třída „GlowManager“)

Pro ovládání navigačního systému, který napomáhá určení pozice na polici stojanu pro uložení aktuálně vygenerované bedny, byl vytvořen „manažer“, který byl umístěn na objekt „Menu“ pod názvem „GuideManager“. Ten obsahuje skript s kódem pod názvem „GlowManager“, který je uveden v příloze 6 na stránce 105. Samotný manažer je v hierarchické struktuře „potomkem“ levého kontroléru Oculus Quest („OculusTouchForQuestAndRiftSLeftModel“). Pro zajištění správné funkčnosti byl upraven kód importovaného assetu pro ovládací prvky uživatele „VRTK“, přesněji kód třídy pro kontrolu událostí „VRTK_ControllerEvents“, kde byla doplněna funkcionalita tlačítek dlaně pro spuštění navigačního systému.

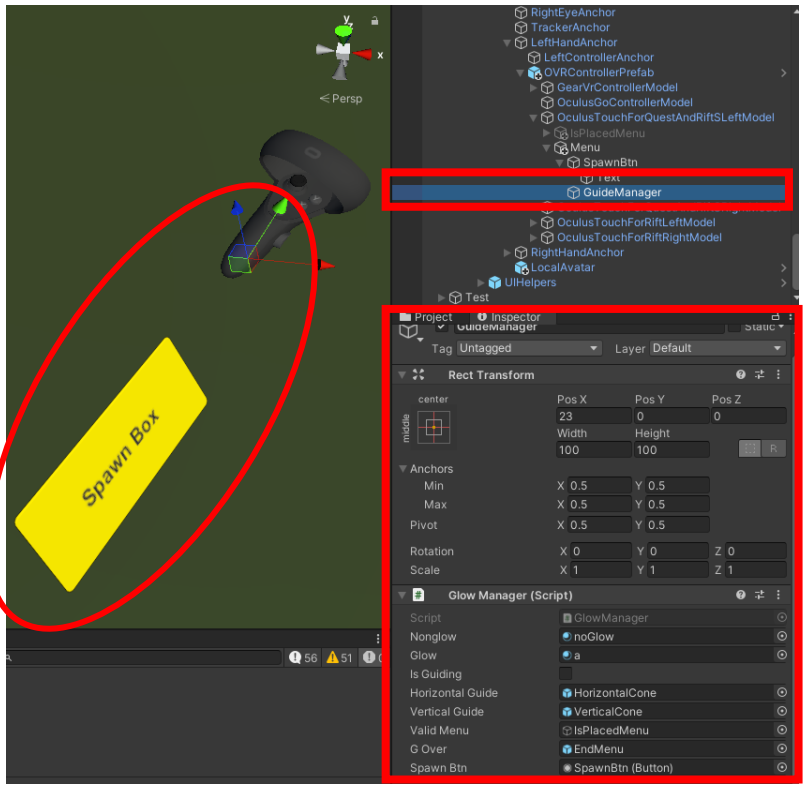
Na počátku třídy jsou deklarovány vstupní proměnné, kde je:

- „nonglow“ – výchozí materiál navigačního kvádru
- „glow“ – materiál pro zvýraznění navigačního kvádru při spuštění navigačního systému
- „isGuiding“ – kontrola stavu navigačního systému (zapnut/vypnut)
- „horizontalGuide“ a „verticalGuide“ – navigační kužele ukazující přibližnou polohu navigačního kvádru
- „validMenu“ – validační menu pro potvrzení umístění bedny do cílové pozice (pozice navigačního kvádru a Zóny)
- „gOver“ – menu pro ukončení aplikace

Metoda „ToggleGlow( )“, v případě načtení poskytnutých vstupních dat, zviditelní navigační kužele a změní materiál navigačního kvádru (*Obr. 6-21.*) na „glow“ pro lepší viditelnost cílového místa („rentgenový pohled“). Kontrola stavu navigace „IsGuiding“ je v aktivním stavu („true“). Naopak metoda „TurnOffGlow( )“ deaktivuje navigační kužele a mění materiál navigačního kvádru na původní „noglow“ materiál (*Obr. 6-4*). Kontrola stavu navigace „IsGuiding“ je v neaktivním stavu („false“).

Dále jsou deklarovány proměnné:

- „counterToFin“ – počítadlo zbývajícího počtu beden k umístění
- „spawnBtn“ – tlačítko pro generování další bedny, navigačního kvádru a Zóny
- „checkOver“ – kontrola, zda byla aplikace restartována (během aplikace po restartu)

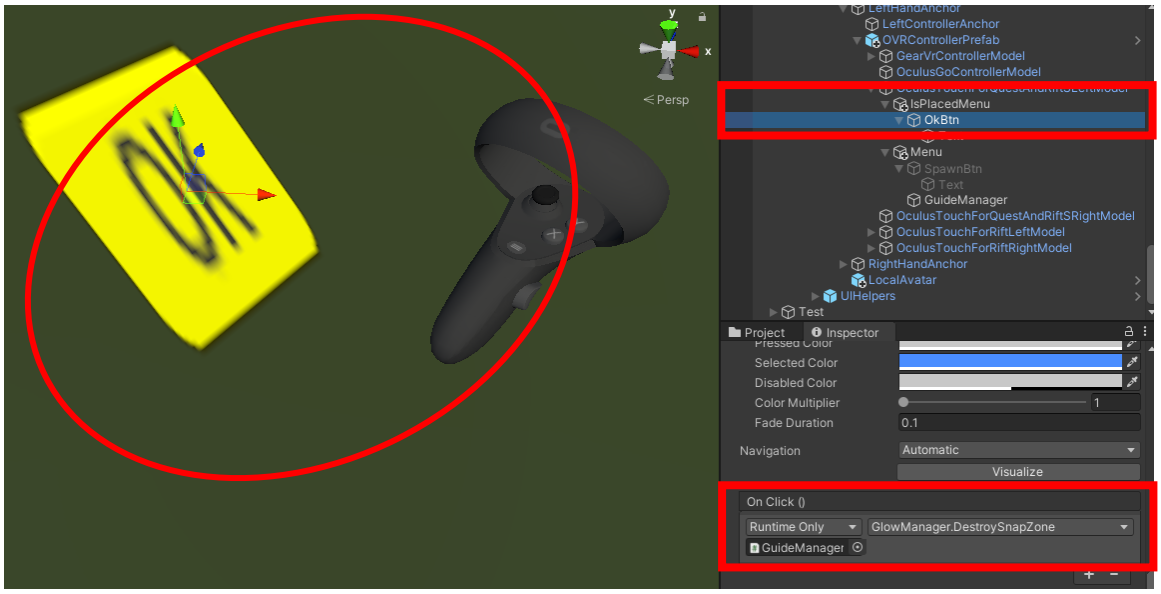


Obr. 6-26 Vizualizace manažeru pro ovládání navigace ukládání beden jako objektu uvnitř aplikace

Metoda „CanvasSetActive( )“ aktivuje menu pod názvem „IsPlacedMenu“ (*Obr. 6-27*), kde je následně dostupné tlačítko pro validaci pozice právě ukládaného objektu bedny. Metoda „BtnOn( )“ umožňuje interakci s tlačítkem „Spawn Box“ (s tlačítkem do doby, než je validována poloha objektu bedny, není možná interakce). Metoda „DestroySnapZone( )“ pomocí počítadla „counterToFin“ definuje počet uložených beden. Dále označí bednu („BoxesFromJson.cub“) a objekt bedny („BoxesFromJson.obj“), jako uložený pomocí boolean proměnné „IsPlaced“ („true“). Objektu bedny „obj“ (třídě „IDReceiver“ přpnuté na prefabu bedny) jsou přiděleny hodnoty atributů (dimenze, pozice, štítek a pořadí) a následně je objekt uložen do generického list „goToExport“, kterým budou data beden uložena do souboru „SavedBoxes.json“. Následuje odstranění příslušné Zóny („snapDes“) a validační menu je deaktivováno. V případě shodné hodnoty počítadla „counterToFin“ a „numbToPlace“ je zavolána metoda „CheckStored( )“.

Metoda „CheckStored( )“ kontroluje počet uložených beden „counterBoxes“ a pokud se rovná celkovému počtu beden k umístění (proměnná „numbToPlace“ přístupná ze třídy „BoxesFromJson), metoda aktivuje menu k ukončení aplikace „EndMenu“.

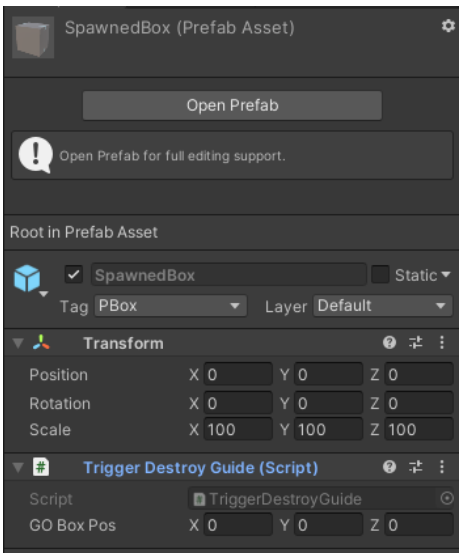
Během testování aplikace (kapitola 7) byla do třídy dodatečně přidána proměnná datového boolean „wasDestroyed“ určující, zda lze prvek navigačního systému (změna materiálu – „rentgenový“ pohled) aktivovat či nikoliv. Při testování ukončení aplikace bylo tlačítko „Spawn Box“ stále aktivní a mohlo by dojít k výskytu chyby. V metodě „CheckSorted( )“ byl doplněn kód pro deaktivaci tlačítka.



Obr. 6-27 Vizualizace validačního menu "IsPlacedMenu"

6.5.5 Třída pro detekci doteku beden s navigačním kvádrem („TriggerDestroyGuide“)

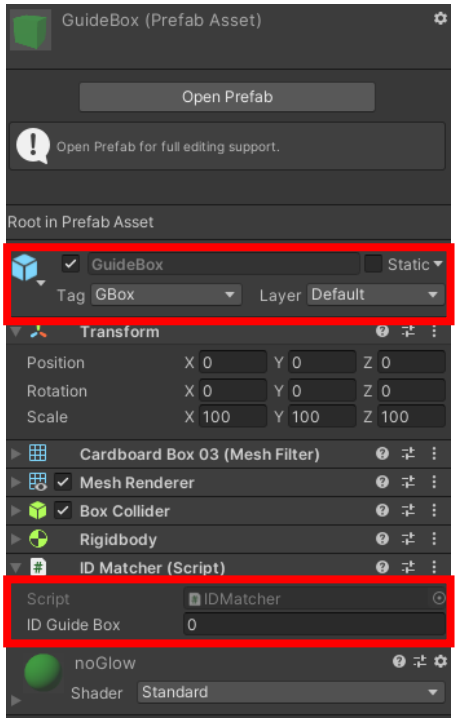
Pro udržení přehlednosti navigačních kvádrů na policích byla vytvořena třída s názvem „TriggerDestroyGuide“, která v případě kontaktu („Collider“) odstraní navigační kvádr pomocí metody „OnTriggerEnter(Collider collider).



Obr. č. 6-28 Vizualizace umístění třídy "TriggerDestroyGuide" přiřazené k prefabu bedny

V případě zanechání objektů navigačních kvádrů na policích by docházelo k nepřehlednosti prostoru uložení. Navíc by se uložená bedna s navigačním kvádrem překrývala a vytvářel by se nežádaný efekt prolínaní textur. Třída pro odstranění navigačního kvádru („TriggerDestroyGuide“) je přidělena jako komponenta prefabu bedny. K odstranění navigačního kvádru dochází pouze v případě, kdy se detektor kolize „BoxCollider“ objektu bedny dostává do kontaktu s detektorem kolize objektu navigačního kvádru, kterému je přiřazen štítek („Tag“) s názvem „GBox“ a objekty se shodují svými ID. Zmíněný štítek je přiřazen pouze objektu navigačního kvádru, viz *Obr. 6-29*, který lze nalézt v horní části

záložky „Inspector“ v programovacím prostředí aplikace Unity3D po kliknutí na požadovaný objekt. Bedna k umístění je označena štítkem „PBox“. Pro možnost reakce navigačního systému na destrukci navigačního kvádru byla třídě přidělena vstupní proměnná „guideM“, která nabývá hodnoty až po spuštění aplikace díky modifikátoru “static”, díky kterému byla proměnná zpřístupněna i ostatním třídám.



Obr. 6-29 Ukázka prefabu navigačního kvádru

Z důvodu změny materiálu navigačních kvádrů při aktivaci navigačního systému, bylo nutné pro správnou funkcionalitu detekce kolizí změnit vykreslování materiálu objektu. K docílení požadovaného efektu je změněn atribut materiálu „shader“ ve třídě “TriggerDestroyGuide”. V obecné definici lze „shader“ popsat, jako skript, který obsahuje matematické výpočty a algoritmy pro vykreslení barev každého pixelu určeného k renderování (vykreslení) na základě konfigurace materiálu [40]. Celý kód třídy je uveden v příloze 7 na stránce 109.

V rámci destrukce navigačního kvádru bylo vhodné též odstranit objekt Zóny („SnapDropZone“) k udržení „pořádku“ na policích stojanu pro snazší provedení uskladnění všech beden do cílových pozic. Za tímto účelem byla třída rozšířena o část kódu, kde se kontroluje kontakt objektu bedny s objektem Zóny (štítek „SnapZone“) za splnění podmínky kontaktu objektu bedny s objektem označeným korektním štítkem a podmínky shody hodnoty ID obou objektů. Pro předejití chyb je dodatečně uvnitř podmínky vytvořena další podmínka pro kontrolu pozic obou objektů. V případě shody se přiřadí aktuální pozice objektu bedny k proměnné „gOBoxPOs“. Navíc pro snazší proces ukládání je objektu bedny zamezen jakákoliv pohyb („RigidbodyConstraints“ – pozice i rotace) v momentě uložení bedny (vtažení bedny Zónou). Při procesu ukládání se často stávalo, že při přemisťování uživatele pomocí teleportu byla bedna neúmyslně posunuta v momentě manipulace s novou bednou k umístění. K možnosti pozorování uložení beden s působností gravitace je nutné uložit a nahrát uložená data z aplikace. V nahrané verzi jsou restrikce na bedny deaktivovány.

Během fáze testování aplikace bylo přidáno přiřazení hodnoty statické proměnné „wasDestroyed“ třídy „GlowManager“, dle které je označen objekt navigačního kvádru za ostraněný.

6.5.6 Třídy pro spárování objektů pomocí ID

Třídy pro spárování objektů jsou rozděleny následovně:

- objekt bedny – třída „IDReceiver“
- objekt navigačního kvádru – třída „IDMatcher“
- objekt Zóny („SnapDropZone“) – třída „SnapMatcher“

Třída „IDReceiver“ je přiřazena objektu bedny za účelem přiřazení atributů objektu bedny za účelem uložení a exportu dat (viz příloha 8 na stránce 111). Třída obsahuje proměnné, jež jsou téměř totožné s proměnnými třídy „BoxClass“. Pomocí této třídy je objekt bedny schopen přijímat hodnoty atributů bedny. Pomocí metody „AssignPosPrfb( )“ (datového typu „Vector3“) je přiřazena pozice objektu bedny proměnným třídy X, Y a Z. Metoda „AssignSclPrfb( ) přiřazuje dimenze objektu bedny proměnným třídy „Width“, „Height“ a „Depth“. Metoda „AssignTag( )“ přiřadí proměnné třídy štítek objektu. Poslední metoda „AssignPoradi( )“ přiřadí pořadí, ve kterém byly jednotlivé objekty beden umístěny.

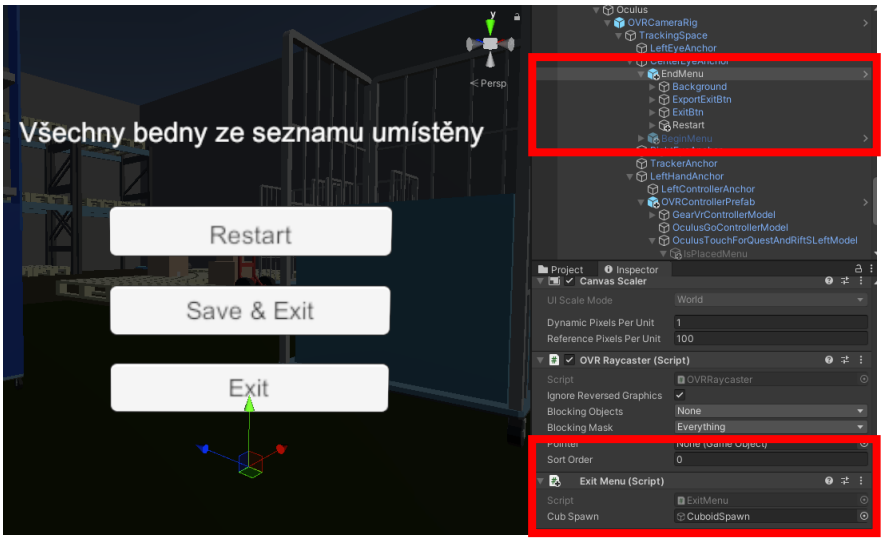
Třída „IDMatcher“ a „SnapMatcher“ obsahují pouze proměnnou pro uchování hodnoty ID daného objektu. Třída „IDMatcher“ je připnuta k prefabu navigačního kvádru a třída „SnapMatcher“ je připnuta k prefabu Zóny (prefab „SnapDropZone“), viz příloha 9 na stránce 113.

6.5.7 Menu k ukončení aplikace (třída „ExitMenu“)

Menu bylo vytvořeno pro možnost volby akce při dokončení procesu uložení všech beden ze seznamu. V menu lze zvolit restart aplikace („Restart“), vypnutí („Exit“) a uložení dat uložených beden s následným vypnutím („Save & Exit“), viz Obr. 6-30. Do třídy „ExitMenu“ přiřazené k objektu „EndMenu“ má pouze jednu proměnnou, která se odkazuje na generátor beden „CuboidSpawn“ pro možnost přístupu k metodě „AddBoxToList( )“.

Metoda „ExitApp( ) vypne celou aplikaci bez uložení jakýchkoliv dat. Metoda je provedena po stisknutí tlačítka „Exit“. Metoda „Reset( )“ znovu nahraje scénu „My BPP 5“. Zde musel být vyřešen problém v rámci zachování určitých dat (např. hodnota počítadel) pro zajištění správné funkčnosti aplikace. Metoda „SaveExit( )“ ukládá data beden určených k exportu (uložení) do generického listu „exportBoxList“ třídy „BoxesFromJson“ pomocí volání metody „AddBoxToList( )“ (obsažena ve třídě „BoxesFromJson“) pro přidání bedny do listu. Následuje vepsání textu do složky pomocí metody „WriteToFile( )“, využívající „FileStream“ pro zápis textu do souboru *.json („SavedBoxes.json“). Nakonec je zavolána metoda „ExitApp( )“ pro ukončení aplikace. Metoda „WriteToString( )“ ukládá data v podobě textu do složky v počítači dle cesty „path“ (cesta z metody „GetFilePath( )“). Využití cesty do složky uživatele „PersistantDataPath“ – složka ani data nemizí s vypnutím aplikace.

V případě ukončení aplikace s uložením dat aplikace je vytvořen nový soubor s názvem „SavedBoxes.json“ (příloha 11 na stránce 117), který je uložen ve složce: „C:\Users\uživatel\AppData\LocalLow\DefaultCompany\název složky projektu“.



Obr. 6-30 Vizualizace menu k ukončení aplikace

7 Testování aplikace

Pro účely správného využívání aplikace uživatelem bylo provedeno testování aplikace. Byla otestována vstupní data pro možnost pozorování chování aplikace při změně ve vstupních datech. Dále byl testován samotný proces ukládání beden. Zda uživatel je schopen ukládat bez problémů bedny do cílových míst a také byla testována celková interakce uživatele s prostředím. Závěrem kapitoly je uveden test uložení a exportu dat.

7.1 Testování vstupních dat

Pro test vstupních dat byl upraven soubor „JsonTextBedny.json“, kde byla pozměněna data v rámci pojmenování, hodnot či pořadí v jakém se data nacházejí. Pro účel testu tak byla vytvořena kopie vstupních dat s názvem „JsonTextBednyTest.json“. Kopie následně byla přiřazena oběma generátorům („CuboidSpawn“ a „CuboidSpawnSorted“). Problémy v rámci této aplikace byly detekovány pomocí kódu pro zpracování výjimek „try-catch“, dle kterého lze lépe určit, jaká část kódu je nefunkční. Stejný přístup zjišťování závad je použit i v testování dalších částí.

Prvotně je otestována náchylnost textu v případě textové změny ve vstupních datech (názvy oddílů). V části souboru „JsonTextBednyTest“ byla přidána mezera za názvy skupin dat „Results“ a „Cuboids“. Při spuštění aplikace s upraveným souborem *.json a stisknutí tlačítka „Spawn Box“ pro vygenerování objektů bedny a navigačního kvádru, nebyl vygenerován ani jeden z objektů. Z toho vyplývá, že vstupní data v rámci názvů skupin (nutné ke správné deserializaci) jsou textově náchylná. Řešení by se v textové náchylnosti muselo provést propojením obou aplikací (výpočetní aplikace s algoritmy s aplikací vyvinuté v této práci). V danou chvíli by bylo možné naprogramovat automatickou korekci názvů. V případě vymazání dat důležitých pro správné zobrazení bedny pomocí kódu, například „ID“ či vymazání jedné hodnoty z rozměrů v jednotlivých dimenzích, nastává situace, kdy není možné do aplikace načíst jakákoliv data z testového souboru „JsonTextBednyTest.json“. Znovu by bylo nutné propojení k obdržení správných hodnot dat. V případě načtení nulové hodnoty jedné z dimenzí, je sice objekt bedny vygenerován, avšak z důvodu struktury kódu ve třídě „BoxesFromJson“ není pozorovatelný (dimenze „Scale“ - (0,0,0)).

7.2 Testování procesu ukládání beden

První menší problém se objevil hned při výběru způsobu načítání vstupních dat. V momentě, kdy je startovací menu aktivní, by uživatel neměl mít vliv na jakoukoliv funkcionalitu co se týče procesu umisťování beden. Bylo ale zjištěno, že navigační systém je aktivní i při přítomnosti startovacího menu, která způsobí nahlášení chyby prázdných proměnných třídy „GlowManager“ z důvodu nenačtení dat. Problém byl napraven přidáním podmínky pro aktivace navigačního systému až po deaktivaci startovacího menu. Do třídy s ovládáním kontroléru Oculus Quest („VRTK_ControllerEvents“) byla dopsána podmínka pro umožnění spuštění navigace až po deaktivaci (načtení dat aplikace) počátečního menu. Ve stejný moment byla přidána další podmínka k aktivaci systému navigace až po vygenerování první bedny „allowNavi“ do třídy „BoxesFromJson“ (bylo možné spuštění navigace bez jakéhokoliv objektu k navigování). Navigační systém (pozice navigačních kuželů a změna materiálu byl vázán přímo na objekt a jeho atributy). V případě uložení bedny, kdy dochází k odstranění objektu navigačního kvádru, nastala situace, kdy se systém odkazoval na odstraněný, již neexistující objekt. Z tohoto důvodu byly přidány podmínky do třídy „GlowManager“ (metody „ToggleGlow( ) a „TurnOffGlow( )“). V rámci stejné třídy byla deklarována proměnná datového typu boolean „wasDestroyed“, která svou hodnotou určuje, zda se mají jednotlivé složky navigačního systému aktivovat či nikoliv. Ve třídě „BoxesFromJsonForSorted“ byla přiřazena hodnota proměnné „wasDestroyed“, která signalizuje existenci objektu navigačního kvádru. Poloha navigačních kuželů byla umístěna pod samostatnou proměnnou pro zachování hodnot v případě zničení objektu navigačního kvádru. Ve třídě „TriggerDestroyGuide“ byla přiřazena hodnota proměnné „wasDestroyed“ pro potvrzení odstranění objektu navigačního kvádru.

7.3 Testování uložení a exportu dat

Při testování ukončování aplikace spolu s uložením a exportem dat beden bylo nalezeno několik chyb. V případě načtení uložených beden, byl problém s navigačním systémem v rámci párování s vygenerovanou bednou. V aplikaci se stisknutím tlačítka „Load“ a následně tlačítka „Spawn Box“ umístí všechny bedny v jednom okamžiku na pozice, ve kterých byly uloženy. Z tohoto důvodu nebylo možné uskutečnit párování a byla by třeba návaznosti dat pro další umisťování nahraných objektů beden. Problém byl ošetřen prázdnou podmínkou (navigační systém nebude spuštěn). Stejný problém nastal v rámci třídy „BoxesFromJson“, která obsahuje metodu „Rotate( ) pro uskutečnění rotace bedny dle vstupních dat ze souboru *.json. Proměnným v rámci metody „xSor, ySor, zSor“ nelze přiřadit hodnoty objektu bedny „obj“ z důvodu načítání již umístěných beden s rotací (pokud byla zapotřebí). Kód je ošetřen podmínkou vypisující problém v rámci „Console“ aplikace Unity3D. Při zobrazení menu k ukončení aplikace bylo stále aktivní tlačítko „Spawn Box“ jež bylo nežádoucí. Ve třídě „GlowManager“ byl přidán kód s deaktivací v rámci metody „CheckSorted( )“, která aktivuje ukončovací menu „EndMenu“. Vypnutí tlačítka nastává ve stejný moment. Ve třídě „BoxesFromJson“ bylo tlačítko „LoadBtn“ opatřeno podmínkou pro umožnění/znemožnění interakce na základě existence složky, ze kterých se nahrávají uložená data. Název složky je textově náchylný a musí přesně odpovídat textem názvu „SavedBoxes.json“.

8 Závěr

Hlavním cílem této diplomové práce byla vizualizace „Bin Packing“ problému ve 3D. Provedení vizualizace bylo uskutečněno ve VR při tvorbě modelu aplikace v Unity3D a programování kódu v aplikaci Visual Studio. Vytvořená aplikace je cílena pro zaškolovací či vyučovací účely v rámci podniku či školy.

V úvodní části práce jsou nastíněny základní principy uplatňované při BPP. Tyto principy jsou využity v rámci vytvořené aplikace pro řešení umístění jednotlivých beden do pater, kde je snahou co nejvíce zaplnit prostor a redukovat prázdná místa mezi bednami. Seznam jednotlivých beden (setříděných i nesetříděných) jen uložen výstupního souboru, který je základem pro vizualizaci ve vytvořené aplikaci ve vývojovém prostředí Unity3D. Tato aplikace je určena pro vizualizaci beden a jejich rozložení ve 3D prostředí pomocí VR. Pro testování během vývoje a pro následné využití aplikace, byl použit VR headset Oculus Quest (HMD). Headset byl poskytnut katedrou Průmyslové inženýrství a managementu Fakulty strojní na ZČU pro možnost vývoje aplikace, za což bych chtěl vyjádřit poděkování.

Při tvorbě samotného virtuálního modelu byla brána zřetel na jednoduchost a intuici uživatele. Uživatel by tedy neměl problém s akceptováním postupu procesu ukládání beden na daném pracovišti. Bylo použito jednoduché, přívětivé grafické prostředí, které by uživatele nemělo svým vzhledem nikterak mást. Aplikace cílí na vizualizaci BPP v podobě setříděných navigačních kvádrů, které fungují jako vzor pro ukládání vygenerovaných beden k uložení. Byl též vytvořen vizuální pomocník ve formě implementace navigačního systému, který uživateli pomáhá při procesu uložení bedny.

V této práci byl brán v potaz pouze jeden stojan k ukládání beden s definovaným počtem beden určených k uložení. V případě reálné aplikace by bylo nutné naprogramovat další situace, které by se s rozšířením modelu aplikace objevily. V případě možných rozšíření by bylo zajímavé vytvořit kód, kde by uživatel byl schopný po naplnění stojanu ten samý stojan přesunout do nákladového prostoru auta a následně pokračovat ukládáním beden do stojanu dalšího. Proces by pokračoval až do naplnění celého nákladního prostoru auta. Pro plnění nákladového prostoru auta by bylo možné využít některý z algoritmů, popsaných v kapitole 3. Dále by byla užitečná implementace načtení více souborů se vstupními daty, kde by každý soubor obsahoval data obdržená od různých algoritmů řešící problematiku 3D BPP. Následovalo by vizuální porovnání jednotlivých variant, na jehož základě by si uživatel zvolil vhodnou variantu (ve VR lze implementovat i základní fyzikální vlastnosti např. gravitace).

9 Bibliografie

[1] 255 PIXEL STUDIOS, 2020. *CITY package | 3D Urban | Unity Asset Store* [online] [vid. 2021-04-14]. Dostupné z: https://assetstore.unity.com/packages/3d/environments/urban/city-package-107224

[2] A3D, 2020. *Low Poly Logistics | 3D Environments | Unity Asset Store* [online] [vid. 2021-04-14]. Dostupné z: https://assetstore.unity.com/packages/3d/environments/low-poly-logistics-137583

[3] ALBAHARI, Joseph, Ben ALBAHARI a Peter DRAYTON, 2012. *C# 5.0 in a nutshell*. 5th ed. Beijing ; Sebastopol: O'Reilly. In a nutshell. ISBN 978-1-4493-2010-2.

[4] Anon., 2018. A quick guide to Degrees of Freedom in Virtual Reality. *Kei Studios* [online] [vid. 2021-04-22]. Dostupné z: https://kei-studios.com/quick-guide-degrees-of-freedom-virtual-reality-vr/

[5] Anon., nedatováno. Definition: Instantiate. *Educative: Interactive Courses for Software Developers* [online] [vid. 2021a-04-22]. Dostupné z: https://www.educative.io/edpresso/definition-instantiate

[6] Anon., nedatováno. *Kapitola 5.2.1.1. Mezioborové srovnání přepravních výkonů nákladní dopravy | Ročenka dopravy 2019* [online] [vid. 2021b-05-02]. Dostupné z: https://www.sydos.cz/cs/rocenka-2019/rocenka/htm_cz/cz19_520110.html

[7] Anon., nedatováno. *Oculus Quest od 9 990 Kč* [online] [vid. 2021c-05-26]. Dostupné z: https://www.zbozi.cz/vyrobek/oculus-quest/

[8] Anon., nedatováno. Snap Drop Zone · VRTK - Virtual Reality Toolkit. *VRTK - Virtual Reality Toolkit* [online] [vid. 2021c-05-23]. Dostupné z: https://vrtoolkit.readme.io/v3.1.0/docs/snap-drop-zone

[9] BÖHM, Martin a Jiří SETNIČKA, nedatováno. *Těžké problémy – Recepty z programátorské kuchařky* [online] [vid. 2021-05-02]. Dostupné z: https://ksp.mff.cuni.cz/kucharky/tezke-problemy/

[10] BORODIN, Allan a Denis PANKRATOV, 2019. *Online Algorithms* [online]. 14. březen 2019. Dostupné z: http://www.cs.toronto.edu/~bor/2420s19/papers/draft-ch1-8.pdf

[11] BRABEC, Tomáš, nedatováno. *Lekce 1 - Úvod do Unity 3D* [online] [vid. 2021-04-22]. Dostupné z: https://www.itnetwork.cz/uvod-do-unity-3d

[12] BRILLIANT.ORG, nedatováno. Tree search | Brilliant Math & Science Wiki. *Brilliant* [online] [vid. 2021-05-02]. Dostupné z: https://brilliant.org/wiki/tree-search/

[13] CRAINIC, Teodor Gabriel, Guido PERBOLI a Roberto TADEI, 2008. Extreme Point-Based Heuristics for Three-Dimensional Bin Packing. *INFORMS Journal on Computing* [online]. **20**(3), 368–384. ISSN 1091-9856, 1526-5528. Dostupné z: doi:10.1287/ijoc.1070.0250

[14] DUAN, Lu, Haoyuan HU, Yu QIAN, Yu GONG, Xiaodong ZHANG, Yinghui XU a Jiangwen WEI, 2019. A Multi-task Selected Learning Approach for Solving 3D Flexible Bin Packing Problem. *arXiv:1804.06896 [cs, stat]* [online]. [vid. 2021-04-14]. Dostupné z: http://arxiv.org/abs/1804.06896

[15] FANSLAU, Tobias a Andreas BORTFELDT, 2010. A Tree Search Algorithm for Solving the Container Loading Problem. *INFORMS Journal on Computing* [online]. **22**, 222–235. Dostupné z: doi:10.1287/ijoc.1090.0338

[16] FAROE, Oluf, David PISINGER a Martin ZACHARIASEN, 2000. Guided Local Search for the Three-Dimensional Bin Packing Problem. *INFORMS Journal on Computing* [online]. **15**. Dostupné z: doi:10.1287/ijoc.15.3.267.16080

[17] GOLDIN, Dina, Scott A. SMOLKA a Peter WEGNER, ed., 2006. *Interactive Computation: The New Paradigm* [online]. Berlin Heidelberg: Springer-Verlag [vid. 2021-05-02]. ISBN 978-3-540-34666-1. Dostupné z: doi:10.1007/3-540-34874-3

[18] HOŘEJŠÍ, Petr, nedatováno. *Kurz: KPV/DPVR Digitální podnik a virtuální realita (ZS)* [online] [vid. 2021-05-02]. Dostupné z: https://phix.zcu.cz/moodle/course/view.php?id=4740

[19] JYLÄNKI, Jukka, 2010. *A thousand ways to pack the bin – a practical approach to two-dimensional rectangle bin packing*.

[20] KAMALI, Shahin a Alejandro LÓPEZ-ORTIZ, 2015. All-Around Near-Optimal Solutions for the Online Bin Packing Problem. In: Khaled ELBASSIONI a Kazuhisa MAKINO, ed. *Algorithms and Computation* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, Lecture Notes in Computer Science, s. 727–739 [vid. 2021-05-03]. ISBN 978-3-662-48970-3. Dostupné z: doi:10.1007/978-3-662-48971-0_61

[21] KANNA, Rajesh, ed., nedatováno. *GENETIC ALGORITHM FOR BIN PACKING PROBLEM*. ISBN ASIN : B01GVA7NX4.

[22] KRISTIANSSON, Bengt, nedatováno. *Loading and unloading goods: How to get it right in your warehouse* [online] [vid. 2021-05-02]. Dostupné z: https://blog.unicarrierseurope.com/material-handling-blog/loading-unloading-lorry-warehouse-pallet-trucks-or-counterbalance-forklifts-advice-analysis

[23] KYSELA, Marek, 2014. *Plošná optimalizace - Bin Packing Problem*. Plzeň. Západočeská univerzita v Plzni.

[24] LINOWES, Jonathan, 2015. *Unity Virtual Reality Projects by Jonathan Linowes | eBook* [online] [vid. 2021-04-15]. ISBN 978-1-78528-680-3. Dostupné z: https://www.scribd.com/book/277715067/Unity-Virtual-Reality-Projects

[25] LODI, Andrea, Silvano MARTELLO a Daniele VIGO, 2002. Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research* [online]. **141**, 410–420. Dostupné z: doi:10.1016/S0377-2217(02)00134-0

[26] MAHVASH, Batoul a Anjali AWASTHI, 2017. A column generation-based heuristic for the three-dimensional bin packing problem with rotation. *Journal of the Operational Research Society* [online]. Dostupné z: doi:10.1057/s41274-017-0186-7

[27] MALKEVITCH, Joseph, nedatováno. *Joseph Malkevitch: Tidbit: Bin Packing* [online] [vid. 2021-05-25]. Dostupné z: https://www.york.cuny.edu/~malk/tidbits/tidbit-bin-packing.html

[28] MARÍA TERESA ALONSO MARTÍNEZ, nedatováno. *Models and algorithms for solving packing problems in logistics* [online] [vid. 2021-05-02]. Dostupné z: https://1library.co/document/zk33191y-models-algorithms-solving-packing-problems-logistics.html

[29] MARTELLO, Silvano, David PISINGER a Daniele VIGO, 1998. The Three-Dimensional Bin Packing Problem. *Operations Research* [online]. **48**. Dostupné z: doi:10.1287/opre.48.2.256.12386

[30] MARTELLO, Silvano a Paolo TOTH, 1990. *Knapsack problems: algorithms and computer implementations*. Chichester ; New York: J. Wiley & Sons. Wiley-Interscience series in discrete mathematics and optimization. ISBN 978-0-471-92420-3.

[31] MEALY, Paul, 2018. *Virtual & Augmented Reality For Dummies*. 1st edition. Indianapolis, IN: For Dummies. ISBN 978-1-119-48134-8.

[32] NANDY, S. C. a B. B. BHATTACHARYA, 1998. Maximal empty cuboids among points and blocks. *Computers & Mathematics with Applications* [online]. **36**(3), 11–20. ISSN 0898-1221. Dostupné z: doi:10.1016/S0898-1221(98)00125-4

[33] O'LEARY, Daniel E., 2008. Gartner's hype cycle and information system research issues. *International Journal of Accounting Information Systems* [online]. **9**(4), 240–252. ISSN 14670895. Dostupné z: doi:10.1016/j.accinf.2008.09.001

[34] PISINGER, David, 2002. Heuristics for the container loading problem. *European Journal of Operational Research* [online]. **141**(2), 382–392. ISSN 0377-2217. Dostupné z: doi:10.1016/S0377-2217(02)00132-7

[35] QUEIROZ, Thiago A. De, Flávio K. MIYAZAWA, Yoshiko WAKABAYASHI a Eduardo C. XAVIER, 2012. Algorithms for 3D guillotine cutting problems: Unbounded knapsack, cutting stock and strip packing. *Computers and Operations Research* [online]. **39**(2), 200–212. ISSN 03050548. Dostupné z: doi:10.1016/j.cor.2011.03.011

[36] SALTON, Jessie, 2017. *Summary of bin-packing algorithms* [online]. 2017. Dostupné z: http://www.math.unl.edu/~s-sjessie1/203Handouts/Bin%20Packing.pdf

[37] SARAIVA, Rommel, Napoleao NEPOMUCENO a Plácido PINHEIRO, 2015. A layer-building algorithm for the three- dimensional multiple bin packing problem: a case study in an

automotive company. *IFAC-PapersOnLine* [online]. **48**, 490–495. Dostupné z: doi:10.1016/j.ifacol.2015.06.129

[38] SASKA, Colton, 2020. Exploring the Bin Packing Problem. *Medium* [online] [vid. 2021-05-02]. Dostupné z: https://medium.com/swlh/exploring-the-bin-packing-problem-f54a93ebdbe5

[39] TECHNOLOGIES, Unity, nedatováno. *Unity - Manual: Prefabs* [online] [vid. 2021-05-24]. Dostupné z: https://docs.unity3d.com/Manual/Prefabs.html

[40] UNITY TECHNOLOGIES, 2017. Unity - Manual: Materials, Shaders & Textures. *Unity Documentation* [online] [vid. 2021-04-14]. Dostupné z: https://docs.unity3d.com/560/Documentation/Manual/Shaders.html

[41] UNITY TECHNOLOGIES, 2021. Unity - Scripting API: Rigidbody. *Unity Documentation* [online] [vid. 2021-04-14]. Dostupné z: https://docs.unity3d.com/ScriptReference/Rigidbody.html

[42] UNITY TECHNOLOGIES, nedatováno. *Map Controllers | Oculus Developers* [online] [vid. 2021-04-14]. Dostupné z: https://developer.oculus.com/documentation/unity/unity-ovrinput/

[43] VARGOVSKÝ, Jan, nedatováno. Lekce 10 - Serializace a deserializace v C# .NET. *ITnetwork.cz* [online] [vid. 2021-04-14]. Dostupné z: https://www.itnetwork.cz/tutorial-csharp-serializace-a-deserializace

[44] VLEUGELS, Kenney, 2018. An Intro to Low-Poly and Flat Design : Appstore Blogs. *Amazon appstore* [online] [vid. 2021-04-14]. Dostupné z: https://developer.amazon.com/blogs/appstore/post/aaedd3b8-5e3f-4b4b-a567-c3257139cbcf/an-intro-to-low-poly-and-flat-design

[45] WAGNER, Bill, nedatováno. *Zpracování výjimek – Průvodce programováním v C#* [online] [vid. 2021-05-23]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/csharp/fundamentals/exceptions/exception-handling

[46] WÄSCHER, Gerhard, Heike HAUSSNER a Holger SCHUMANN, 2007. An improved typology of cutting and packing problems. *European Journal of Operational Research* [online]. **183**(3), 1109–1130. ISSN 0377-2217. Dostupné z: doi:10.1016/j.ejor.2005.12.047

[47] WIKIPEDIA, nedatováno. *Oculus Rift - Wikipedia* [online] [vid. 2021-05-02]. Dostupné z: https://en.wikipedia.org/wiki/Oculus_Rift?oldid=761323847

[48] WLOSOK, Jan, 2014. *Unity 3D* [online]. B.m. Technická univerzita Ostrava. Dostupné z: : https://dspace.vsb.cz/bitstream/handle/10084/104220/WLO0008_FEI_B2647_2612R025_2014.pdf?sequence=1&isAllowed=y

[49] ZEALOUSYS, nedatováno. Unity Virtual Reality: Types Of UI Placements. *zealous* [online] [vid. 2021-04-23]. Dostupné z: https://www.zealousys.com/blog/unity-virtual-reality-types-ui-placements/

[50] ZHAO, Xiaozhou, Julia A. BENNELL, Tolga BEKTAŞ a Kath DOWSLAND, 2016. A comparative review of 3D container loading algorithms: A comparative review of 3D container loading algorithms. *International Transactions in Operational Research* [online]. **23**(1–2), 287–320. ISSN 09696016. Dostupné z: doi:10.1111/itor.12094

PŘÍLOHA č. 1

Ukázka souboru se vstupními daty JsonTextBedny.json

```
"Results": [
  {
    "Patro": 1,
    "PatroBedny": [
      {
        "ID": 18,
        "Poradi": 0,
        "Width": 89.0,
        "Height": 74.0,
        "Depth": 93.0,
        "X": 0.0,
        "Y": 0.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": true
      },
      {
        "ID": 12,
        "Poradi": 0,
        "Width": 100.0,
        "Height": 19.0,
        "Depth": 86.0,
        "X": 0.0,
        "Y": 74.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": true
      },
      {
        "ID": 2,
        "Poradi": 0,
        "Width": 11.0,
        "Height": 74.0,
        "Depth": 87.0,
        "X": 89.0,
        "Y": 0.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": true
      },
      {
        "ID": 5,
        "Poradi": 0,
        "Width": 40.0,
        "Height": 5.0,
        "Depth": 92.0,
        "X": 0.0,
        "Y": 93.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": true
      },
      {
        "ID": 21,
        "Poradi": 0,
        "Width": 71.0,
        "Height": 19.0,
        "Depth": 13.0,
        "X": 0.0,
        "Y": 74.0,
        "Z": 86.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": true
      },
      {
        "ID": 15,
        "Poradi": 0,
        "Width": 58.0,
        "Height": 3.0,
        "Depth": 96.0,
        "X": 40.0,
        "Y": 93.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": true
      },
      {
        "ID": 8,
        "Poradi": 0,
        "Width": 86.0,
        "Height": 2.0,
        "Depth": 93.0,
        "X": 0.0,
        "Y": 98.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": true
      },
      {
        "ID": 10,
        "Poradi": 0,
        "Width": 10.0,
        "Height": 42.0,
        "Depth": 1.0,
        "X": 0.0,
        "Y": 0.0,
        "Z": 93.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": true
      }
    ]
  },
  {
    "Patro": 2,
    "PatroBedny": [
      {
        "ID": 3,
        "Poradi": 0,
        "Width": 72.0,
        "Height": 81.0,
        "Depth": 68.0,
        "X": 0.0,
        "Y": 0.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
```

```
      "IsPlaced": true
    },
    {
      "ID": 17,
      "Poradi": 0,
      "Width": 97.0,
      "Height": 67.0,
      "Depth": 25.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 68.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": true
    },
    {
      "ID": 11,
      "Poradi": 0,
      "Width": 20.0,
      "Height": 80.0,
      "Depth": 58.0,
      "X": 72.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": true
    },
    {
      "ID": 4,
      "Poradi": 0,
      "Width": 65.0,
      "Height": 18.0,
      "Depth": 60.0,
      "X": 0.0,
      "Y": 81.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": true
    },
    {
      "ID": 9,
      "Poradi": 0,
      "Width": 35.0,
      "Height": 12.0,
      "Depth": 19.0,
      "X": 0.0,
      "Y": 67.0,
      "Z": 68.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": true
    },
    {
      "ID": 6,
      "Poradi": 0,
      "Width": 23.0,
      "Height": 30.0,
      "Depth": 9.0,
      "X": 72.0,
      "Y": 0.0,
      "Z": 58.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": true
    }
  ]
},
{
  "Patro": 3,
  "PatroBedny": [
    {
      "ID": 20,
      "Poradi": 0,
      "Width": 79.0,
      "Height": 79.0,
      "Depth": 58.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": true
    },
    {
      "ID": 19,
      "Poradi": 0,
      "Width": 98.0,
      "Height": 76.0,
      "Depth": 37.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 58.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": true
    },
    {
      "ID": 13,
      "Poradi": 0,
      "Width": 99.0,
      "Height": 21.0,
      "Depth": 59.0,
      "X": 0.0,
      "Y": 79.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": true
    }
  ]
},
{
  "Patro": 4,
  "PatroBedny": [
    {
      "ID": 7,
      "Poradi": 0,
      "Width": 62.0,
      "Height": 50.0,
      "Depth": 84.0,
      "X": 0.0,
      "Y": 0.0,
```

```
          "Z": 0.0,
          "Weight": 0.0,
          "Tag": null,
          "IsPlaced": true
        },
        {
          "ID": 14,
          "Poradi": 0,
          "Width": 68.0,
          "Height": 49.0,
          "Depth": 77.0,
          "X": 0.0,
          "Y": 50.0,
          "Z": 0.0,
          "Weight": 0.0,
          "Tag": null,
          "IsPlaced": true
        },
        {
          "ID": 16,
          "Poradi": 0,
          "Width": 22.0,
          "Height": 41.0,
          "Depth": 42.0,
          "X": 62.0,
          "Y": 0.0,
          "Z": 0.0,
          "Weight": 0.0,
          "Tag": null,
          "IsPlaced": true
        }
      ]

"Cuboids": [
    {
      "ID": 18,
      "Poradi": 0,
      "Width": 89.0,
      "Height": 74.0,
      "Depth": 93.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": false
    },
    {
      "ID": 3,
      "Poradi": 0,
      "Width": 72.0,
      "Height": 81.0,
      "Depth": 68.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": false
    },
    {
      "ID": 20,
      "Poradi": 0,
      "Width": 79.0,
      "Height": 79.0,
      "Depth": 58.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": false
    },
    {
      "ID": 19,
      "Poradi": 0,
      "Width": 37.0,
      "Height": 98.0,
      "Depth": 76.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": false
    },
    {
      "ID": 7,
      "Poradi": 0,
      "Width": 62.0,
      "Height": 50.0,
      "Depth": 84.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": false
    },
    {
      "ID": 14,
      "Poradi": 0,
      "Width": 77.0,
      "Height": 68.0,
      "Depth": 49.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": false
    },
    {
      "ID": 12,
      "Poradi": 0,
      "Width": 100.0,
      "Height": 19.0,
      "Depth": 86.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": false
```

```
  },
  {
    "ID": 17,
    "Poradi": 0,
    "Width": 25.0,
    "Height": 97.0,
    "Depth": 67.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 13,
    "Poradi": 0,
    "Width": 21.0,
    "Height": 99.0,
    "Depth": 59.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 11,
    "Poradi": 0,
    "Width": 20.0,
    "Height": 58.0,
    "Depth": 80.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 2,
    "Poradi": 0,
    "Width": 11.0,
    "Height": 87.0,
    "Depth": 74.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 4,
    "Poradi": 0,
    "Width": 65.0,
    "Height": 18.0,
    "Depth": 60.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 16,
    "Poradi": 0,
    "Width": 41.0,
    "Height": 22.0,
    "Depth": 42.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 5,
    "Poradi": 0,
    "Width": 40.0,
    "Height": 5.0,
    "Depth": 92.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 21,
    "Poradi": 0,
    "Width": 19.0,
    "Height": 13.0,
    "Depth": 71.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 15,
    "Poradi": 0,
    "Width": 58.0,
    "Height": 3.0,
    "Depth": 96.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 8,
    "Poradi": 0,
    "Width": 93.0,
    "Height": 86.0,
    "Depth": 2.0,
```

```
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": false
    },
    {
      "ID": 9,
      "Poradi": 0,
      "Width": 12.0,
      "Height": 19.0,
      "Depth": 35.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": false
    },
    {
      "ID": 6,
      "Poradi": 0,
      "Width": 9.0,
      "Height": 30.0,
      "Depth": 23.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": false
    },
    {
      "ID": 10,
      "Poradi": 0,
      "Width": 10.0,
      "Height": 1.0,
      "Depth": 42.0,
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0,
      "Weight": 0.0,
      "Tag": null,
      "IsPlaced": false
    }
  ],
  "Parameter": {
    "BinID": 1,
    "BinWidth": 100.0,
    "BinHeight": 100.0,
    "BinDepth": 100.0,
    "BinWeight": 0.0,
    "AllowRotateVertically": true,
    "Cuboids": [
      {
        "ID": 2,
        "Poradi": 0,
        "Width": 11.0,
        "Height": 87.0,
        "Depth": 74.0,
        "X": 0.0,
        "Y": 0.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": false
      },
      {
        "ID": 3,
        "Poradi": 0,
        "Width": 72.0,
        "Height": 81.0,
        "Depth": 68.0,
        "X": 0.0,
        "Y": 0.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": false
      },
      {
        "ID": 4,
        "Poradi": 0,
        "Width": 65.0,
        "Height": 18.0,
        "Depth": 60.0,
        "X": 0.0,
        "Y": 0.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": false
      },
      {
        "ID": 5,
        "Poradi": 0,
        "Width": 40.0,
        "Height": 5.0,
        "Depth": 92.0,
        "X": 0.0,
        "Y": 0.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": false
      },
      {
        "ID": 6,
        "Poradi": 0,
        "Width": 9.0,
        "Height": 30.0,
        "Depth": 23.0,
        "X": 0.0,
        "Y": 0.0,
        "Z": 0.0,
        "Weight": 0.0,
        "Tag": null,
        "IsPlaced": false
      },
      {
        "ID": 7,
        "Poradi": 0,
        "Width": 62.0,
```

```
    "Height": 50.0,
    "Depth": 84.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 8,
    "Poradi": 0,
    "Width": 93.0,
    "Height": 86.0,
    "Depth": 2.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 9,
    "Poradi": 0,
    "Width": 12.0,
    "Height": 19.0,
    "Depth": 35.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 10,
    "Poradi": 0,
    "Width": 10.0,
    "Height": 1.0,
    "Depth": 42.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 11,
    "Poradi": 0,
    "Width": 20.0,
    "Height": 58.0,
    "Depth": 80.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 12,
    "Poradi": 0,
    "Width": 100.0,
    "Height": 19.0,
    "Depth": 86.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 13,
    "Poradi": 0,
    "Width": 21.0,
    "Height": 99.0,
    "Depth": 59.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 14,
    "Poradi": 0,
    "Width": 77.0,
    "Height": 68.0,
    "Depth": 49.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 15,
    "Poradi": 0,
    "Width": 58.0,
    "Height": 3.0,
    "Depth": 96.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 16,
    "Poradi": 0,
    "Width": 41.0,
    "Height": 22.0,
    "Depth": 42.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
```

```
    "IsPlaced": false
  },
  {
    "ID": 17,
    "Poradi": 0,
    "Width": 25.0,
    "Height": 97.0,
    "Depth": 67.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 18,
    "Poradi": 0,
    "Width": 89.0,
    "Height": 74.0,
    "Depth": 93.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 19,
    "Poradi": 0,
    "Width": 37.0,
    "Height": 98.0,
    "Depth": 76.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 20,
    "Poradi": 0,
    "Width": 79.0,
    "Height": 79.0,
    "Depth": 58.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  },
  {
    "ID": 21,
    "Poradi": 0,
    "Width": 19.0,
    "Height": 13.0,
    "Depth": 71.0,
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "Weight": 0.0,
    "Tag": null,
    "IsPlaced": false
  }
]
```

PŘÍLOHA č. 2

Zdrojový kód třídy „BeginMenuScript“ pro výběr způsobu načtení vstupních dat

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

public class BeginMenuScript : MonoBehaviour
{
    // deklarace vstupních proměnných
    // objekt zastupující generátor beden
    public GameObject spawn;
    // objekt zastupující generátor navigačních kvádrů
    public GameObject guideSpawn;
    // tlačítko pro generování nové bedny
    public Button spawnBtn;

    // metoda pro spouštěcí menu aplikace
    public void StartApp()
    {
        try
        {
            BoxesFromJson.newOrLoad = false;
            spawn.transform.GetComponent<BoxesFromJson>().LoadBoxes();
            guideSpawn.transform.GetComponent<BoxesFromJsonForSorted>().loadBoxes();
            spawn.transform.GetComponent<BoxesFromJson>().SortBoxes();

            gameObject.SetActive(false);
            spawnBtn.gameObject.SetActive(true);
        }
        catch (Exception e)
        {
            Debug.LogError(e.Message);
        }

    }

    // metoda pro načítání vstupních dat
    public void LoadApp()
    {
        try
        {
            BoxesFromJson.newOrLoad = true;
            spawn.transform.GetComponent<BoxesFromJson>().LoadBoxes();
            gameObject.SetActive(false);
            spawnBtn.gameObject.SetActive(true);
        }
        catch(Exception e)
        {
            Debug.LogError(e.Message);
        }

    }

    // metoda pro čtení dat ze souboru uložených beden (*.json)
    public string ReadFromFile(string filename)
    {
        string path = GetFilePath(filename);
        if(File.Exists(path))
        {
            using(StreamReader reader = new StreamReader(path))
            {
```

```
                string readJson = reader.ReadToEnd();
                return readJson;
            }
        }
        else
        {
            Debug.LogWarning("Soubor nenalezen");
            return "";
        }

    }

    // metoda pro nalezení cesty k cílovému uložení souboru s uloženými daty
    private string GetFilePath(string filename)
    {
            return Application.persistentDataPath + "/" + filename;
    }
}
```

PŘÍLOHA č. 3

Zdrojový kód třídy „BoxesClass“ pro import a načtení vstupních dat

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

public class BoxesClass : MonoBehaviour
{
    // čtení objektu z poskytnutého souboru JsonTextBedny.json pro výměnu dat .json -
JsonTextBedny jako bedny (kuboidu) s jejími atributy
    [System.Serializable]
    public class Cuboid
    {
        // deklarace atributů bedny
        public int ID;
        public int Poradi;
        public double Width;
        public double Height;
        public double Depth;
        public double X;
        public double Y;
        public double Z;
        public double Weight;
        public object Tag;
        public bool IsPlaced;

        // převedení jednotek z cm na m pro správné vyobrazení velikosti bedny v
Unity3D
        public double WidthInMeters => Width/100f;
        public double HeightInMeters => Height/100f;
        public double DepthInMeters => Depth/100f;

        // převedení jednotek z cm na m pro správné vyobrazení pozice bedny v Unity3D
        public double BoxX => X / 100f;
        public double BoxY => Y / 100f;
        public double BoxZ => Z / 100f;

        // konstruktor pro třídu Cuboid
        public Cuboid() { }
        public Cuboid(int id, double width, double height, double depth) :
            this(id, width, height, depth, 0, 0, 0, 0, null)
        { }
        public Cuboid(int id, double width, double height, double depth, double
weight, object tag) :
            this(id, width, height, depth, 0, 0, 0, weight, tag)
        { }
        public Cuboid(int id, double width, double height, double depth, double x,
double y, double z) :
            this(id, width, height, depth, x, y, z, 0, null)
        { }
        public Cuboid(int id, double width, double height, double depth, double x,
double y, double z, double weight, object tag)
        {
            ID = id;
            Width = width;
            Height = height;
            Depth = depth;
            X = x;
            Y = y;
            Z = z;
            Weight = weight;
            Tag = tag;
        }
```

```
        // metoda pro přiřazení rozměrů bedny ze souboru formát *.json - využíván v
generátorech
        public Vector3 calculate()
        {
            return new Vector3((float)Width, (float)Height, (float)Depth);
        }

        // úprava převedení do textového formátu
        public override string ToString()
        {
            return $"Cuboid(X: {X}, Y: {Y}, Z:{Z}, Width: {Width}, Height:{Height},
Depth:{Depth}, Weight: {Weight}, Tag: {Tag})";
        }
    }

    // čtení atributů beden - výsledek optimalizace algoritmů BPP - data v souboru
formátu *.json - Results
    [System.Serializable]
    public class ResultPatro
    {
        //vstupní atributy pro třídu ResultPatro pro možnost čtení rozřazení beden dle
polic
        public int Patro;
        public List<Cuboid> PatroBedny;

        //konstruktor třídy ResultPatro
        public ResultPatro(int patro, List<Cuboid> patroBedny)
        {
            Patro = patro;
            PatroBedny = patroBedny;
        }
    }
}
```

PŘÍLOHA č. 4

Zdrojový kód třídy „BoxesFromJson“ umožňující generování beden určených k umístění

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.UI;
using static BoxesClass;
using System.IO;
using System;

public class BoxesFromJson : MonoBehaviour
{
    // Deklarace vstupních proměnných

    //  import dat ze souboru *.json - vstupní data
    public TextAsset textFromJson;
    public TextAsset boxesFromLoad;
    // objekt k instancionalizaci - model bedny
    public GameObject prefabToSpawnGO;
    // proměnná slouží k usnadnění vizualizace plochy
    public Vector3 spawnSize;
    // proměná pro počítání zbylých beden k umístění – na ose z
    public Text counterUI;
    // proměnná pro počítání zbylých beden k umístění – na ose x
    public Text counterUI2;
    // propojení generátoru beden s generátorem navigačních kvádrů
    public BoxesFromJsonForSorted spawnGuidance;
    // objekt generátoru beden
    public GameObject spawn;

    // Třída pro serializaci (čtení) dat ze souboru *.json
    [System.Serializable]
    public class CuboidList
    {
        public List<Cuboid> Cuboids;

    }

    void Start()
    {
        CheckNull();
    }
    // vytvoření nového kuboid (bedny) listu dle třídy pro serializaci dat
    public CuboidList Cuboids = new CuboidList();

    // boolean pro způsob načítání vstupních dat
    public static bool newOrLoad = false;
    // objekt startovacího menu aplikace
    public GameObject startMenu;

    // načítání dat do vytvořeného listu s kuboidy (bednami)
    public void LoadBoxes()
    {
        try
        {
            // výběr požadovaných vstupních dat dle výběru uživatele ve vstupním menu
aplikace
            if (newOrLoad == false)
            {
                Cuboids = JsonUtility.FromJson<CuboidList>(textFromJson.text);
            }
            else if (newOrLoad == true)
            {
```

```
                Cuboids =
JsonUtility.FromJson<CuboidList>(startMenu.transform.GetComponent<BeginMenuScript>().R
eadFromFile("SavedBoxes.json"));
            }
        }
        catch(Exception e)
        {
            Debug.LogError(e.Message);
        }
    }

    // metoda k ukládání dat beden do souboru *.json
    public string SaveBoxes()
    {
        try
        {
            string toJsonEnd = JsonUtility.ToJson(exportBoxList, true);
            return toJsonEnd;
        }
        catch(Exception e)
        {
            Debug.LogError(e.Message);
            return "";
        }
    }

    public Button loadBtn;
    public void CheckNull()
    {
        if( System.IO.File.Exists(Application.persistentDataPath + "/" +
"SavedBoxes.json"))
        {
            loadBtn.interactable = true;
        }
        else
        {
            loadBtn.interactable = false;
        }
    }

    // generický list pro získání beden v určitém pořadí dle použitého algoritmu pro
výpočet
    public List<Cuboid> sortedCuboids;
    // počítadlo beden k uložení
    public static int numbToPlace;
    // metoda pro setřídění beden
    public void SortBoxes()
    {
        Cuboid sort;
        numbToPlace = 0;
        for (int patro = 0; patro < spawnGuidance.varList.Results.Count; patro++)
        {
            foreach (Cuboid cuboid in spawnGuidance.varList.Results[patro].PatroBedny)
            {
                try
                {
                    sort = Cuboids.Cuboids.Find(d => d.ID == cuboid.ID);
                    sort.Poradi = numbToPlace;
                    sortedCuboids.Add(sort);
                    numbToPlace = numbToPlace + 1;
                }
                catch(Exception e)
```

```
                {
                    Debug.LogError(e.Message);
                }
            }
        }
    }

    // vytvoření jednoho kuboidu (bedny) pro možnost přidělení atributů kuboidu
(bendy) k inicializovanému objektu (prefabu)
    public static Cuboid cub;
    // počítadlo uložených beden
    public static int counterBoxes;
    // objekt zastupující prefab bedny
    GameObject prefabToSpawn;
    // trojrozměrný vektro pro příjem dimenzí beden
    public Vector3 scaleToAssign;
    // zastoupení objektu bedny využívané pro zpřístupnění objektu ostatním třídám
    public static GameObject obj;
    // objekty tlačítka pro generování nové bedny
    public Button spawnBtn;
    // třída CuboidList zastupující generický list beden určených k exportu
    public CuboidList exportBoxList = new CuboidList();

    // metoda pro postupné generování beden k umístění
    public void SpawnBoxes()
    {
        // kontrola způsobu načtení vstupních dat)
        if(newOrLoad == false)
        {
            // vytvoření instancionalizovaného objektu
            // pro zachování původního objektu (prefabu) a změnu atributů
instancionalizovaého objektu
            prefabToSpawn = Instantiate(prefabToSpawnGO);

            // vezme první bednu z listu beden k umístění
            cub = sortedCuboids[0];
            // přiřazení ID objektu bedny
            prefabToSpawn.transform.GetComponent<IDReceiver>().iDOfBox = cub.ID;
            // přiřazení hodnot dimenzí bedny proměnné -  využito k porovnání dimenzí
bedny po provedení rotace
            scaleToAssign = cub.calculate();

            // přidělení pozice inicionalizovanému prefabu
            prefabToSpawn.transform.localPosition = spawn.transform.position + new
Vector3(0, -spawnSize.y / 2, 0);
            prefabToSpawn.transform.rotation = Quaternion.identity;
            // přidělení štítku ("Tag") k rozeznání objektů uvnitř aplikace
            prefabToSpawn.transform.tag = "PBox";
            // bedna není umístěna ve stojanu
            cub.IsPlaced = false;

            // přiřazení hodnoty statickému objektu "obj" k umožnění přístupu k
objektu bedny ostatním třídám
            obj = prefabToSpawn;
            // objekt bedny je vygenerován a není uložen do cílové pozice
            obj.transform.GetComponent<IDReceiver>().IsPlaced = false;
            // navýšení počítadla vygenerovaných beden
            counterBoxes = counterBoxes + 1;

            // odstranění první bedny z listu kuboidů (beden) k umístění
            sortedCuboids.RemoveAt(0);
```

```
            // vypsání zbývajícího počtu beden k umístění na počítadle
            counterUI.text = sortedCuboids.Count.ToString();
            counterUI2.text = sortedCuboids.Count.ToString();
            // znepřístupnění tlačítko pro generování další bedny
            spawnBtn.interactable = false;
            // kontrola vygenerování bedny
            VRTK.VRTK_ControllerEvents.allowNavi = true;
        }
        else
        {
            try
            {
                // pokud jsou vstupní data nahráta po předchozím uložení - přiřazení
vstupních atributů jednotlivým objektům beden
                foreach (Cuboid cubFromLoad in Cuboids.Cuboids)
                {
                    // vytvoření variabilní instancionalizovaného objektu
                    // pro zachování původního objektu (prefabu) a upravování atributů
instancionalizovaého objektu
                    prefabToSpawn = Instantiate(prefabToSpawnGO);
                    // přiřazení hodnot třídě IDReceiver připojené k prefabu bedny
                    prefabToSpawn.transform.GetComponent<IDReceiver>().iDOfBox =
cubFromLoad.ID;
                    prefabToSpawn.transform.GetComponent<IDReceiver>().X =
cubFromLoad.X;
                    prefabToSpawn.transform.GetComponent<IDReceiver>().Y =
cubFromLoad.Y;
                    prefabToSpawn.transform.GetComponent<IDReceiver>().Z =
cubFromLoad.Z;
                    prefabToSpawn.transform.GetComponent<IDReceiver>().Width =
cubFromLoad.Width;
                    prefabToSpawn.transform.GetComponent<IDReceiver>().Height =
cubFromLoad.Height;
                    prefabToSpawn.transform.GetComponent<IDReceiver>().Depth =
cubFromLoad.Depth;
                    prefabToSpawn.transform.GetComponent<IDReceiver>().Weight =
cubFromLoad.Weight;
                    prefabToSpawn.transform.GetComponent<IDReceiver>().IsPlaced =
cubFromLoad.IsPlaced;
                    prefabToSpawn.transform.GetComponent<IDReceiver>().tag = "PBox";
                    prefabToSpawn.transform.GetComponent<IDReceiver>().Poradi =
cubFromLoad.Poradi;
                    // přiřazení atributů objektu bedny
                    prefabToSpawn.transform.localScale = new
Vector3((float)prefabToSpawn.transform.GetComponent<IDReceiver>().Width,
(float)prefabToSpawn.transform.GetComponent<IDReceiver>().Height,
(float)prefabToSpawn.transform.GetComponent<IDReceiver>().Depth);
                    prefabToSpawn.transform.position = new
Vector3((float)prefabToSpawn.transform.GetComponent<IDReceiver>().X,
(float)prefabToSpawn.transform.GetComponent<IDReceiver>().Y,
(float)prefabToSpawn.transform.GetComponent<IDReceiver>().Z);
                    prefabToSpawn.transform.rotation = Quaternion.identity;
                    prefabToSpawn.transform.tag =
prefabToSpawn.transform.GetComponent<IDReceiver>().tag;
                    prefabToSpawn.transform.GetComponent<Rigidbody>().constraints =
RigidbodyConstraints.None;
                }

                spawnBtn.interactable = false;
            }
            catch (Exception e)
            {
```

```
                Debug.LogError(e.Message);
            }
        }
    }

// metoda pro rotaci bedny dle vstupních dat
    public Vector3 Rotate()
    {
        // výstupní atribut v podobě trojrozměrného vektoru zastupující dimenze bedny
        Vector3 rotatedDim = new Vector3();
        // generický list pro bedny s přetočení
        List<double> sorted = new List<double>();

        if (newOrLoad == false)
        {
            double xSor = BoxesFromJsonForSorted.obj.transform.localScale.x;
            double ySor = BoxesFromJsonForSorted.obj.transform.localScale.y;
            double zSor = BoxesFromJsonForSorted.obj.transform.localScale.z;

            sorted.Add(xSor);
            sorted.Add(ySor);
            sorted.Add(zSor);

            // generický list pro bedny bez přetočení
            List<double> untouched = new List<double>();

            double xCub = cub.Width;
            double yCub = cub.Height;
            double zCub = cub.Depth;

            untouched.Add(xCub);
            untouched.Add(yCub);
            untouched.Add(zCub);

            int c = 0;

            foreach (double d in untouched)
            {
                try
                {
                    if (sorted.Contains(d))
                    {
                        c = c + 1;
                    }
                }
                catch (Exception e)
                {
                    Debug.LogError(e.Message);
                }
            }

            if (c == 3)
            {
                rotatedDim.x = (float)xSor;
                rotatedDim.y = (float)ySor;
                rotatedDim.z = (float)zSor;
            }
        }
        else
        {
            Debug.LogError("Data pro načtení hodnot beden nejsou správná či k
dispozici.");
```

```
        }
        return rotatedDim;
    }

    // metoda pro přiřazení přetočených hodnot dimenzí bedny
    public void AssignScale()
    {
        Rotate();
        // přidělení atributů vybraného kuboidu (bedny) inicionalizovanému objektu
(prefabu)
        // upravení rozměru prefabu dle rozměrů vybraného kuboidu
        if (BoxesFromJsonForSorted.obj.transform.localScale == scaleToAssign)
        {
            prefabToSpawn.transform.localScale = new Vector3((float)cub.Width,
(float)cub.Height, (float)cub.Depth);
        }
        else
        {
            prefabToSpawn.transform.localScale = Rotate();
        }

        obj.transform.GetComponent<IDReceiver>().sizeOfBox =
prefabToSpawn.transform.localScale;
    }

    // statický generický list pro typ objektů GameObject - ukládání objektů beden k
exportu
    public static List<GameObject> gOToExport = new List<GameObject>();
    // objekty třídy Cuboid pro uložení dat bedny
    public Cuboid toSaveCub;
    // metoda pro ukládání dat beden do souboru *.json k exportu
    public void AddBoxToList()
    {
        try
        {
            foreach (GameObject gOb in gOToExport)
            {
                toSaveCub = new Cuboid();
                toSaveCub.ID = gOb.GetComponent<IDReceiver>().iDOfBox;
                toSaveCub.X = gOb.GetComponent<IDReceiver>().X;
                toSaveCub.Y = gOb.GetComponent<IDReceiver>().Y;
                toSaveCub.Z = gOb.GetComponent<IDReceiver>().Z;
                toSaveCub.Width = gOb.GetComponent<IDReceiver>().Width;
                toSaveCub.Height = gOb.GetComponent<IDReceiver>().Height;
                toSaveCub.Depth = gOb.GetComponent<IDReceiver>().Depth;
                toSaveCub.Weight = gOb.GetComponent<IDReceiver>().Weight;
                toSaveCub.IsPlaced = gOb.GetComponent<IDReceiver>().IsPlaced;
                toSaveCub.Tag = gOb.GetComponent<IDReceiver>().tag;
                toSaveCub.Poradi = gOb.GetComponent<IDReceiver>().Poradi;

                string saveString = toSaveCub.ToString();
                exportBoxList.Cuboids.Add(toSaveCub);
            }
        }
        catch(Exception e)
        {
            Debug.LogError(e.Message);
        }
    }

    // metoda pro vygenerování bedny a navigačního kvádru ve správném pořadí a
přidělení objektu třídě pro kontrolu doteku
```

```
    public void SpawnNextBox()
    {
        SpawnBoxes();
        spawnGuidance.SpawnBoxesSorted();
        AssignScale();
        TriggerDestroyGuide.guideM = GameObject.Find("GuideManager");
    }
}
```

PŘÍLOHA č. 5

Zdrojový kód třídy „BoxesFromJsonSorted“ umožňující generování navigačních kvádrů

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;
using static BoxesClass;
using System;

public class BoxesFromJsonForSorted : MonoBehaviour
{
    // Deklarace vstupních proměnných
    // import dat ze souboru *.json - JsonTextBedny - vstupní data
    public TextAsset textFromJson;
    // objekt k instancionalizaci navigačního kvádru - model krabice/boxu
    public GameObject prefabToSpawnGO;
    // proměnná udávající výšku patra stojanu
    Vector3 shelfHeight = new Vector3(0, 0.2455f, 0);
    // proměnné pro import objektů zastupující navigační kužele
    public GameObject horizontalCone;
    public GameObject verticalCone;
    // proměnná zastupující objekt počátečního bodu pro generování navigačních kvádrů
    public GameObject pointOnShelf;
    // trojrozměrný vektor pro vytvoření mezery mezi jednotlivými navigačními kvádry -
zabránění kolize
    Vector3 spaceBetween = new Vector3(0.0001f, 0.0001f, -0.0001f);
    // objekt zastupující SnapDropZone pro snažší umístění bedny do cílového prostoru
    public GameObject snap;

    // pomocná vnořená třída pro načtení dat z JsonTextBedny
    [Serializable]
    public class JsonClass
    {
        // způsob rozlišení a uložení dat do seznamu Results
        public List<BoxesClass.ResultPatro> Results;
    }

    // konstruktor pro pomocnou vnořenou třídu JsonClass
    public JsonClass varList = new JsonClass();

    // metoda pro načtení kvádrů z JsonTextBedny
    public void loadBoxes()
    {
        // uložení načtených dat do seznamu varList
        varList = JsonUtility.FromJson<JsonClass>(textFromJson.text);
    }

    // vytvoření pomocné proměnné pro kontrolu vygenerovaných beden pomocí třídy
BoxesFromJson
    Cuboid cuboidFromBuffer = new Cuboid();
    // deklarace statické proměnné pro zpřístupnění proměnné objektu navigačního
kvádru ostatním třídám
    public static GameObject obj;

    // objekt reprezentující SnapDropZone objekt určený ke zničení
    public static GameObject snapDes;
    // objekt reprezentující navigační kvádr určený ke zničení
    public static Cuboid cubToDes;

    // metoda pro generování navigačních kvádrů
    public void SpawnBoxesSorted()
    {
        // proměnné je přiřazena proměnná ze třídy BoxesFromJson
        // slouží jako propojení generátorů
```

```
        cuboidFromBuffer = BoxesFromJson.cub;

        // vytvoření seznamu navigačních kvádrů, které jsem vyselektovány ze seznamu
varList
        List<Cuboid> cuboids = varList.Results.SelectMany(x =>
x.PatroBedny).ToList();// list se všemi cuboidy ze všech pater

        if (BoxesFromJson.newOrLoad == false)
        {
            // cyklus procházející jednotlivá patra
            for (int patro = 0; patro < varList.Results.Count; patro++)
            {
                // cyklus procházející všechny objekty datového typu "Cuboid" v
seznamu "cuboids"
                foreach (Cuboid cub in cuboids)
                {
                    // podmínka pro sjednocení výběru navigačního kvádru s
vygenerovanou bednou k uložení
                    if (cub.ID == cuboidFromBuffer.ID)
                    {
                        // podmínka za účelem určení patra, ve kterém se navigační
kvádr vygeneruje
                        if (varList.Results[patro].PatroBedny.Contains(cub))
                        {
                            // vytvoření variabilní instancionalizovaného objektu
                            // pro zachování původního objektu (prefabu) a upravování
atributů instancionalizovaného objektu
                            var gO = Instantiate(prefabToSpawnGO);
                            // instancionalizování objektu SnapDropZone pro usnadnění
umístění bedny
                            var snapGuide = Instantiate(snap);
                            // přiřazení ID objektu navigačního kvádru v rámci
"připnuté" třídy IDMatcher
                            gO.GetComponent<IDMatcher>().iDGuideBox = cub.ID;
                            // přiřazení bedny proměnné zastupující bednu ke zničení
                            cubToDes = cub;
                            // přiřazení ID k proměnné sortedPlaceID ve třídě
TriggerDestroyGuide přiopojené k objektu bedny
                            TriggerDestroyGuide.sortedPlaceID = cub.ID;

                            // pomocné koeficienty k obdržení skutečných rozměrů beden
                            float koefX = 0.1948263f;
                            float koefYZ = 0.2270044f;


                            // přidělení atributů vybraného kuboidu (bedny)
inicializovanému objektu (prefabu)
                            // upravení rozměru prefabu dle rozměrů vybrané bedny
(cuboid)
                            gO.transform.localScale = cub.calculate();

                            // deklarace proměnných zastupující skutečnou pozici
navigačního kvádru uvnitř aplikace
                            float colliderX = (float)cub.BoxX * koefX +
(float)((gO.GetComponent<Renderer>().bounds.size.x) / 2);
                            float colliderY = (float)cub.BoxY * koefYZ + shelfHeight.y
* patro + gO.GetComponent<Renderer>().bounds.size.y / 2;
                            float colliderZ = (float)cub.BoxZ * koefYZ +
(float)((gO.GetComponent<Renderer>().bounds.size.z) / 2);
                            gO.transform.position = pointOnShelf.transform.position +
new Vector3(colliderX, colliderY, -colliderZ) + spaceBetween;
                            gO.transform.rotation = Quaternion.identity;
```

```
                            // deklarace proměnných pro definici rozměru, pozice a
rotace objektu SnapDropZone
                            snapGuide.transform.localScale = gO.transform.localScale /
100;
                            snapGuide.transform.position = gO.transform.position;
                            snapGuide.transform.rotation = Quaternion.identity;
                            // přiřazení ID objektu SnapDropZone
                            snapGuide.transform.GetComponent<SnapMatcher>().snapID =
cub.ID;
                            // přiřazení SnapDropZone proměnné zastupující
SnapDropeZone ke zničení
                            snapDes = snapGuide;

                            // přidělení štítku "Tag" pro rozeznání objektu v rámci
detekce kolize s vygenerovanou bednou
                            gO.transform.tag = "GBox";
                            // přiřazení hodnoty proměnné za účelem propojení kódu
tříd
                            obj = gO;

                            GlowManager.wasDestroyed = false;

                            // umístění navigačních kuželů dle pozice bedny
                            Vector3 posHor = new Vector3(gO.transform.position.x,
gO.transform.position.y, horizontalCone.transform.position.z);
                            Vector3 posVert = new Vector3(gO.transform.position.x,
verticalCone.transform.position.y, gO.transform.position.z);
                            horizontalCone.transform.position = posHor;
                            verticalCone.transform.position = posVert;

                            // odebrání první bedny ze seznamu - procházení seznamu
                            varList.Results[patro].PatroBedny.Remove(cub);
                        }
                    }
                }
            }
        }
        else
        {
            // v případě nahrání uložených vstupních dat (z předchozího procesu
ukládání beden) vypsána zpráva v "Console" záložce Unity3D
            Debug.Log("Načtena uložená data");
            Debug.Log("Zadejte data pro nové rozmístění navigačních kvádrů");
        }
    }
}
```

PŘÍLOHA č. 6

Zdrojový kód třídy spravující podpůrný navigační systém „GlowManager“

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

public class GlowManager : MonoBehaviour
{
    // Deklarace datových složek třídy

    // materiál pro navigační kvádr ve stavu bez navigace
    public Material nonglow;
    // materiál pro navigační kvádr ve stavu s navigací
    public  Material glow;
    // kontrolor režimu navigačního systému
    public  bool isGuiding = false;
    // objekty zastupující navigační kužele
    public  GameObject horizontalGuide;
    public  GameObject verticalGuide;
    // objekt zastupující menu pro validaci umístěné bedny
    public GameObject validMenu;
    // objekt zastupující menu pro ukončení aplikace
    public GameObject gOver;

    // aktivní režim navigačního systému
    public void ToggleGlow()
    {
        // kontrola způsobu načtení počátečních dat
        if (BoxesFromJson.newOrLoad == false)
        {
            // zviditelnění kuželů v prostředí pracoviště
            horizontalGuide.SetActive(true);
            verticalGuide.SetActive(true);

            // změna materiálu navigačního kvádru
            if(wasDestroyed == false)
            {

BoxesFromJsonForSorted.obj.transform.GetComponent<MeshRenderer>().material = glow;
            }

            // změna režimu navigačního systému
            isGuiding = true;
        }
    }

    // neaktivní režim navigačního systému
    public void  TurnOffGlow()
    {
        if(BoxesFromJson.newOrLoad == false)
        {
            horizontalGuide.SetActive(false);
            verticalGuide.SetActive(false);

            if(wasDestroyed == false)
            {

BoxesFromJsonForSorted.obj.transform.GetComponent<MeshRenderer>().material = nonglow;
            }

            isGuiding = false;
        }
```

```
    }

    // počítadlo zbývajících beden k umístění
    private int counterToFin = 0;
    // metoda pro aktivaci validačního menu
    public void CanvasSetActvie()
    {
        try
        {
            validMenu.SetActive(true);
        }
        catch(Exception e)
        {
            Debug.LogError(e.Message);
        }
    }
    // tlačítko pro generování bedny
    public Button spawnBtn;
    // metoda pro zpřístupnění tlačítka pro generování bedny
    public void BtnOn()
    {
        try
        {
            spawnBtn.interactable = true;
        }
        catch (Exception e)
        {
            Debug.LogError(e.Message);
        }
    }

    public static bool wasDestroyed = false;

    // metoda pro zničení SnapDropZone s ID shodným s ID aktuální bedny k uložení
    public void DestroySnapZone()
    {
        if (BoxesFromJson.obj.transform.GetComponent<TriggerDestroyGuide>().gOBoxPos
== BoxesFromJsonForSorted.snapDes.transform.position)
        {
            counterToFin = counterToFin + 1;

            // bedna je umístěna na požadovaném cílovém místě
            BoxesFromJson.cub.IsPlaced = true;
            // přidělení hodnot objektu bedny v rámci připojené třídy k objektu
IDReceiver
            BoxesFromJson.obj.transform.GetComponent<IDReceiver>().IsPlaced = true;
            BoxesFromJson.obj.transform.GetComponent<IDReceiver>().AssignPosPrfb();
            BoxesFromJson.obj.transform.GetComponent<IDReceiver>().AssignSclPrfb();

BoxesFromJson.obj.transform.GetComponent<IDReceiver>().AssignTag(BoxesFromJson.obj);

BoxesFromJson.obj.transform.GetComponent<IDReceiver>().AssignPoradi(counterToFin);
            BoxesFromJson.gOToExport.Add(BoxesFromJson.obj);

            // zničení objektu SnapDropZone
            Destroy(BoxesFromJsonForSorted.snapDes);
            validMenu.SetActive(false);

            // kontrola stavu aplikace pro případné ukončení
            if (counterToFin == BoxesFromJson.numbToPlace)
            {
                CheckStored();
```

```
            }

            // zviditelnění tlačítka pro generování bedny
            BtnOn();
        }
    }

    //  počítadlo pro kontrolu stavu aplikace v příapdě uzavření či restartu
    public static int checkOver = 0;
    //metoda pro zobrazení menu ukončení aplikace
    public void CheckStored()
    {
        checkOver += 1;
        if (BoxesFromJson.counterBoxes/checkOver == BoxesFromJson.numbToPlace)
        {
            gOver.SetActive(true);
            spawnBtn.interactable = false;
        }
    }
}
```

PŘÍLOHA č. 7

Zdrojový kód třídy detekce kontaktu „TriggerDestroyGuide“

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

public class TriggerDestroyGuide : MonoBehaviour
{
    // vstupní proměnná, která je přiřazena až po spuštění aplikace
    // díky "static" modifikátoru lze se odkazovat na proměnou i z jiných tříd
    public static GameObject guideM;
    // deklarace proměnné, které se bude přiřazovat pozice objektu bedny
    public Vector3 gOBoxPos;
    // přiřazení ID k proměnné sortedPlaceID ze třídy BoxesFromJsonForSorted
    public static int sortedPlaceID;

    // metoda pro zaznamenání dotyku bedny s navigačním kvádrem
    private void OnTriggerEnter(Collider collider)
    {
        //  pokud objekt, kterého se bedna dotýká je navigační kvádr, odstraň
navigační kvádr
        if (collider.gameObject.CompareTag("GBox"))
        {
            if(gameObject.transform.GetComponent<IDReceiver>().iDOfBox ==
collider.gameObject.transform.GetComponent<IDMatcher>().iDGuideBox)
            {
                // kód přidán z důvod nedetekování dotyku v případě, kdy je spuštěna
navigace k uložení
                // změna "Shaderu" navigačního kvádru v momentě dotyku
                collider.gameObject.GetComponent<Renderer>().material.shader =
Shader.Find("Standard");
                // přiřazení komponenty objektu pro možnost ovládání navigačního
systému
                guideM.GetComponent<GlowManager>().TurnOffGlow();
                // odstanění objektu (destruktor)
                Destroy(collider.gameObject);

                GlowManager.wasDestroyed = true;

                // aktivace validačního menu
                guideM.GetComponent<GlowManager>().CanvasSetActvie();
            }
        }

        // kontrola zda kolidující objekt má přiřazen štítek "SnapZone" a zda ID
objektu bedny se shoduje s ID kolidujícího objektu SnapDropZone
        if (collider.gameObject.CompareTag("SnapZone") &&
gameObject.transform.GetComponent<IDReceiver>().iDOfBox ==
collider.gameObject.transform.GetComponent<SnapMatcher>().snapID)
        {
            // pokud se pozice objektů shodují, je uložena aktuální pozice objektu
bedny a zároveň je zamezen je zamezen jakýkoliv pohyb s objektem bedny
            if (gameObject.transform.position ==
collider.gameObject.transform.position)
            {
                gOBoxPos = gameObject.transform.position;
                gameObject.transform.GetComponent<Rigidbody>().constraints =
RigidbodyConstraints.FreezeAll;
            }
        }
    }
}
```

PŘÍLOHA č. 8

Zdrojový kód třídy „IDReceiver“ pro přiřazení hodnot proměnných objektu bedny

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

public class IDReceiver : MonoBehaviour
{
    // deklarace vstupních proměnných třídy - přiřazení atributů objektu bedny
    public int iDOfBox;
    public Vector3 pos;
    public double Width;
    public double Height;
    public double Depth;
    public double X;
    public double Y;
    public double Z;
    public bool IsPlaced;
    public Vector3 sizeOfBox;
    public double Weight = 0;
    public int Poradi;

    // metoda pro přiřazení pozice v prostoru aplikace objekut bedny
    public Vector3 AssignPosPrfb()
    {
        pos = gameObject.transform.position;
        X = pos.x;
        Y = pos.y;
        Z = pos.z;
        return pos;
    }

    // metoda pro přiřazení dimnzí objektu bedny
    public Vector3 AssignSclPrfb()
    {
        sizeOfBox = gameObject.transform.localScale;
        Width = sizeOfBox.x;
        Height = sizeOfBox.y;
        Depth = sizeOfBox.z;
        return sizeOfBox;
    }

    // metoda pro přiřazení štítku objektu bedny
    public void AssignTag(GameObject go)
    {
        gameObject.transform.tag = go.transform.tag;
    }

    // metoda pro přiřazení pořadí objektu bedny, v jakém jsou generovány
    public int AssignPoradi(int poradi)
    {
        Poradi = poradi;
        return Poradi;
    }
}
```

PŘÍLOHA č. 9

Zdrojový kód tříd pro přiřazení ID k přiřazeným objektům - „IDMatcher“ a „SnapMatcher“

IDMatcher

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

public class SnapMatcher : MonoBehaviour
{
    // proměnná pro přiřazení ID k možnosti propojení navigačního kvádru s aktuální
bednou k uložení
    public int iDGuideBox;
}
```

SnapMatcher

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

public class SnapMatcher : MonoBehaviour
{
    // proměnná pro přiřazení ID k možnosti propojení navigačního kvádru s aktuální
bednou k uložení
    public int snapID;
}
```

PŘÍLOHA č. 10

Zdrojový kód třídy „ExitMenu“ pro možnosti ukončení aplikace pro VR

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

public class ExitMenu : MonoBehaviour
{
    // proměnná reprezentující generátor beden
    public GameObject CubSpawn;

    // metoda k ukončení aplikace bez uložení
    public void ExitApp()
    {
        Application.Quit();
    }

    // metoda pro znovunačtení počáteční scény aplikace
    public void Reset()
    {
        SceneManager.LoadScene(0);
    }

    // metoda pro uložení dat a následné ukončení aplikace
    public void SaveExit()
    {
        CubSpawn.transform.GetComponent<BoxesFromJson>().AddBoxToList();

        WriteToFile(file,
CubSpawn.transform.GetComponent<BoxesFromJson>().SaveBoxes());

        Application.Quit();
    }

    // formát pro uložení dat beden po ukončení procesu umístění beden
    private string file = "SavedBoxes.json";

    // metoda pro způsob zapisování dat k uložení do vytvořeného souboru *.json
    private void WriteToFile(string fileName, string json)
    {
        string path = GetFilePath(fileName);
        FileStream fileStream = new FileStream(path, FileMode.Create);

        using (StreamWriter writer = new StreamWriter(fileStream))
        {
            writer.Write(json);
        }
    }

    // cesta k místu uložení vytvořeného souboru s uloženými daty
    private string GetFilePath(string fileName)
    {
        return Application.persistentDataPath + "/" + fileName;
    }
}
```

PŘÍLOHA č. 11

Zdrojový kód s uloženými daty z aplikace pro VR „SavedBoxes.json“

```
{
    "Cuboids": [
        {
            "ID": 18,
            "Poradi": 1,
            "Width": 89.0000228881836,
            "Height": 74.00000762939453,
            "Depth": 93.00001525878906,
            "X": 92.10908508300781,
            "Y": 0.9431042075157166,
            "Z": 176.85626220703126,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 12,
            "Poradi": 2,
            "Width": 100.0000228881836,
            "Height": 19.0,
            "Depth": 86.00001525878906,
            "X": 92.11979675292969,
            "Y": 1.048661470413208,
            "Z": 176.8642120361328,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 2,
            "Poradi": 3,
            "Width": 11.000003814697266,
            "Height": 74.00003051757813,
            "Depth": 86.99999237060547,
            "X": 92.20649719238281,
            "Y": 0.9431042075157166,
            "Z": 176.8630828857422,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 5,
            "Poradi": 4,
            "Width": 40.000003814697269,
            "Height": 4.999999523162842,
            "Depth": 92.00000762939453,
            "X": 92.06134796142578,
            "Y": 1.075901985168457,
            "Z": 176.85740661621095,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 21,
            "Poradi": 5,
            "Width": 71.00006103515625,
            "Height":
19.000003814697267,
            "Depth": 13.000009536743164,
            "X": 92.09154510498047,
            "Y": 1.048661470413208,
            "Z": 176.75184631347657,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 15,
            "Poradi": 6,
            "Width": 58.00001525878906,
            "Height": 3.0,
            "Depth": 96.0,
            "X": 92.15681457519531,
            "Y": 1.073632001876831,
            "Z": 176.8528594970703,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 8,
            "Poradi": 7,
            "Width": 86.00000762939453,
            "Height": 2.000000238418579,
            "Depth": 93.0,
            "X": 92.10616302490235,
            "Y": 1.083847165107727,
            "Z": 176.85626220703126,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 10,
            "Poradi": 8,
            "Width": 10.000003814697266,
            "Height": 42.0000114440918,
            "Depth": 0.9999998807907105,
            "X": 92.0321273803711,
            "Y": 0.9067836403846741,
            "Z": 176.74957275390626,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 3,
            "Poradi": 9,
            "Width": 72.00001525878906,
            "Height": 81.00000762939453,
            "Depth": 68.0,
            "X": 92.09252166748047,
            "Y": 1.1965492963790894,
            "Z": 176.8846435546875,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 17,
            "Poradi": 10,
            "Width": 97.00005340576172,
            "Height": 67.00000762939453,
            "Depth": 25.000015258789064,
            "X": 92.11687469482422,
            "Y": 1.1806590557098389,
            "Z": 176.77908325195313,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
```

```
            "ID": 11,
            "Poradi": 11,
            "Width": 20.00000762939453,
            "Height": 80.00001525878906,
            "Depth": 58.0000114440918,
            "X": 92.18214416503906,
            "Y": 1.1954143047332764,
            "Z": 176.89599609375,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 4,
            "Poradi": 12,
            "Width": 65.00000762939453,
            "Height":
18.000001907348634,
            "Depth": 59.99999237060547,
            "X": 92.08570098876953,
            "Y": 1.308916687965393,
            "Z": 176.8937225341797,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 9,
            "Poradi": 13,
            "Width": 35.000003814697269,
            "Height":
12.000001907348633,
            "Depth": 19.0000057220459,
            "X": 92.05648040771485,
            "Y": 1.270326018333435,
            "Z": 176.785888671875,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 6,
            "Poradi": 14,
            "Width": 23.000011444091798,
            "Height": 30.00000762939453,
            "Depth": 9.000003814697266,
            "X": 92.18506622314453,
            "Y": 1.138663411140442,
            "Z": 176.8199462890625,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 20,
            "Poradi": 15,
            "Width": 79.00001525878906,
            "Height": 79.00000762939453,
            "Depth": 58.00000762939453,
            "X": 92.0993423461914,
            "Y": 1.439779281616211,
            "Z": 176.89599609375,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 19,
            "Poradi": 16,
            "Width": 97.99999237060547,
            "Height": 76.00000762939453,
            "Depth": 37.0,
            "X": 92.11785125732422,
            "Y": 1.4363741874694825,
            "Z": 176.7881622314453,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 13,
            "Poradi": 17,
            "Width": 99.0000228881836,
            "Height": 21.00000762939453,
            "Depth": 59.000003814697269,
            "X": 92.11882019042969,
            "Y": 1.5532816648483277,
            "Z": 176.8948516845703,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 7,
            "Poradi": 18,
            "Width": 62.00001525878906,
            "Height":
50.000003814697269,
            "Depth": 84.0,
            "X": 92.08277893066406,
            "Y": 1.6523637771606446,
            "Z": 176.86648559570313,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 14,
            "Poradi": 19,
            "Width": 67.99999237060547,
            "Height": 49.00000762939453,
            "Depth": 77.0,
            "X": 92.088623046875,
            "Y": 1.7647309303283692,
            "Z": 176.87442016601563,
            "Weight": 0.0,
            "IsPlaced": true
        },
        {
            "ID": 16,
            "Poradi": 20,
            "Width": 22.000003814697267,
            "Height": 41.0000114440918,
            "Depth": 42.000003814697269,
            "X": 92.16460418701172,
            "Y": 1.6421486139297486,
            "Z": 176.91415405273438,
            "Weight": 0.0,
            "IsPlaced": true
        }
    ]
}
```