FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Diplomová práce

Demonstrační aplikace pro laboratoř techniky

Dominik Poch

PLZEŇ

2023

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Diplomová práce

Demonstrační aplikace pro laboratoř techniky

Bc. Dominik Poch

Vedoucí práce
Doc. Ing. Libor Váša, Ph.D.

PLZEŇ **2023**

Citace v seznamu literatury:
POCH, Dominik. *Demonstrační aplikace pro laboratoř techniky*. Plzeň, 2023. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Doc. Ing. Libor Váša, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Dominik POCH**
Osobní číslo:	**A21N0026P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Počítačová grafika**
Téma práce:	**Demonstrační aplikace pro laboratoř techniky**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s hardwarovým vybavením demonstrační učebny techniky na Gymnáziu Sokolov (virtuální realita, 3d senzory, vícekamerový systém, interaktivní stůl, haptické zařízení, robot založený na platformě arduino). Prostudujte jejich parametry a způsob ovládání (API).
2. Navrhněte způsob, jak demonstrovat vlastnosti a schopnosti dostupných zařízení v jedné komplexní aplikaci (její kompletní realizace není součástí práce).
3. Implementujte centrální součásti komplexní aplikace (tj. serverovou část) nutné pro vzájemnou komunikaci jednotlivých zařízení. Dbejte na obecnost a stabilitu.
4. Zvolte dvě konkrétní zařízení (popř. kombinace zařízení) a vytvořte pro každé z nich zadání pro sadu úloh směřující k realizaci komplexní aplikace. Sada se bude skládat ze základní úlohy, realizovatelné bez dohledu podle návodu, která návštěvníkovi laboratoře bude demonstrovat smysl zařízení, a dále navazující úlohy, při jejímž řešení návštěvník získá kontrolu nad konkrétním zařízením, typicky formou skriptování.
5. Vytvořte referenční řešení navržených úloh.
6. Vytvořené metodické materiály a referenční řešení důkladně otestujte v demonstrační učebně techniky a popište vlastnosti vytvořených řešení a možnosti pro další rozvoj.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Doc. Ing. Libor Váša, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **9. září 2022**
Termín odevzdání diplomové práce: **18. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2022

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

Plzeň dne 22. června 2023

. . . . . .
Dominik Poch

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Diplomová práce pojedná o tvorbě demonstračních aplikací a metodických materiálů pro laboratoř techniky na Gymnáziu Sokolov. Pro demonstraci celé laboratoře je navržený centrální systém, který propojuje jednotlivá zařízení do jedné komplexní aplikace. Bližší rozbor a implementace se provádí jen u centrální serverové části systému. Z popisu dostupného hardwarového vybavení jsou vybrána dvě zařízení - stolní počítač a haptické zařízení, pro které jsou vytvořeny úlohy s návody a podpůrným softwarem. Pro budoucí programy studentů gymnázia a pro vývoj podpůrného softwaru jsou připraveny knihovny, které usnadní vývoj komunikace mezi klienty a centrálním serverem.

Abstract

The master thesis discusses a creation of demonstrative applications and methodological materials for a technology laboratory of Gymnasium Sokolov. A central system is designed to demonstrate the laboratory. It interconnects all devices into one complex application. Deeper analysis and implementation are available just for the central server of the system. From a description of all available hardware in the laboratory, two devices are selected - a desktop computer and a haptic device, that receive assignments, instructions and supporting software throughout the thesis. Students of the gymnasium may create their own software in the future. For that purpose and to implement supporting software for the assignments, two libraries are created to help with communication between clients and the central server.

Klíčová slova

laboratoř techniky • centrální systém • metodické materiály • haptické zařízení • Unity • REST API

Poděkování

Chtěl bych poděkovat mému vedoucímu Doc. Ing. Liboru Vášovi, Ph.D. za poskytnuté rady, vstřícnost a trpělivost v průběhu vypracování diplomové práce. Mé díky patří také Ing. Alexi Königovi za pomoc při testování.

Obsah

Úvod 1

Moderní technologie se za poslední desítky let staly nedílnou součástí našich životů. Jen těžko si dnes dokážeme představit život bez mobilních telefonů a osobních počítačů. Není proto divu, že informační technologie prostupují do všech oborů lidské činnosti. Jedná se o rychle se vyvíjející odvětví a nastupující generace by měly být připravené na to, že v budoucnu bude závislost na informačních technologiích ještě větší než nyní, což bude jistě umocněno nástupem strojového učení, jehož výrazný posun je sledovatelný v posledních měsících a letech.

Naneštěstí školské systémy, které by měly přípravu mladých lidí zajistit, typicky reagují na změny v tomto oboru pomaleji a přizpůsobení trvá déle. Při pohledu na změny způsobené zavedením moderních technologií do zdravotnictví, dopravy nebo komunikací se zdá, že školství tahá za kratší konec pomyslného technologického provazu. I přesto však ve školství informační technologie v menší míře uplatnění nalezly. Ve většině škol jsou k dispozici počítačové učebny či interaktivní tabule. Je však otázkou, zda by v dnešní době nemohla být integrace do výuky výrazně vyšší.

Situace se začala rychle vyvíjet v roce 2020 s nástupem pandemie onemocnění COVID-19, kdy většina států světa přistoupila k omezení sociálních kontaktů a uzavření škol. Místo vyučování v lavicích se přešlo k distanční výuce. Množství škol narazilo na problémy s tímto typem výuky a byly nuceny vylepšit svoji IT infrastrukturu a vyučovat přes komunikační platformy typu Zoom, Google Classroom a jiné. I přes porodní obtíže se tento způsob výuky využíval dva roky. Objektivně lze říci, že přispěl ke zvýšení počítačové gramotnosti učitelů, díky čemuž dnes vidíme reálnou možnost hlubší integrace moderních technologií do každodenního provozu. Například Západočeská univerzita v Plzni má v plánu do roku 2025 zvýšit využívání distančních metod vzdělávání v prezenčních studijních programech a přidat počet předmětů s flexibilní formou vzdělávání z 858 (k 10.10.2020) na 4200 [Duc21].

Jelikož se většinou jedná o finančně náročná zařízení na pořízení, lze vzhledem k aktuální ekonomické situaci předpokládat, že postup začlenění informačních technologií bude postupný. Problém však spočívá především v pořízení složitějších zařízení a budování odborných laboratoří, kde například nelze očekávat, že by školy

přistoupily k pořízení brýlí pro virtuální realitu pro všechny jejich studenty. Pravděpodobněji budeme vídat několik volně dostupných zařízení v rámci celé školy.

K takovému řešení přistoupilo i Gymnázium Sokolov. Pro studenty vybavilo laboratoř se zaměřením na počítačové vidění a počítačovou grafiku. Laboratoř vybavili v rámci projektu Implementace Krajského akčního plánu 2 v Karlovarském kraji (IKAP) různými zařízeními od 3D skenerů až po brýle pro virtuální realitu.

V rámci této práce je cílem demonstrovat daná zařízení studentům a návštěvníkům technologické laboratoře a vytvořit metodické materiály pro práci se zařízeními. V dalších kapitolách tohoto textu jsou popsané připravené úlohy s různými úrovněmi obtížnosti. Studenti by měli být schopni je vlastními silami vyřešit. Žáci dostanou k zadáním také zrealizovaná ukázková řešení daných úloh. Předpokládá se, že je využijí jako pomocný materiál při vlastní tvorbě. Různá zařízení budou využita v jednom komplexním řešení. Jedná se o systém vzájemně komunikujících a spolupracujících aplikací. Úlohy pro studenty jsou pouze vhodně zvolenými částmi těchto komplexních celků.

Laboratoř techniky na Gymnáziu Sokolov — 2

2.1 Dostupná zařízení

Znalost zařízení a jejich výkonu popřípadě nároků je důležitá pro správné nastavení úloh. V rámci projektu IKAP bylo dodáno do laboratoře kromě popsaných zařízení, potenciálně uvažovaných pro návrh úloh v kapitole 3, více různorodého vybavení - například výkonné stolní počítače, interaktivní stůl, roboti založení na platformě Arduino a další.

2.1.1 Virtuální realita

Pro virtuální realitu jsou v laboratoři dostupná dvě zařízení HTC Vive Pro. Každé má vytyčené své vlastní území pomocí základových stanic, jež obsahují senzory pro snímání pohybu na daném prostoru. Dva uživatelé díky tomu mohou používat oboje brýle najednou a vzájemně si nepřekážet v pohybu, což by mohlo být při nasazených brýlích nebezpečné. Brýle zakrývají výhled a díky integrovaným reproduktorům dokáží přehlušit i zvuky z okolí.

Základními parametry HTC Vive Pro [18b] jsou

- Obrazovka: Dual AMOLED 3.5",
- Rozlišení: 1440 x 1600 pixelů pro každé oko,
- Obnovovací frekvence: 90 Hz,
- Zorné pole: 110°.

HTC Vive Pro byl vydaný v roce 2018. Z dnešního pohledu se stále jedná o velmi kvalitní zařízení. U nově vycházejících brýlí pro virtuální realitu je reálně patrný pouze malý pokrok. V roce 2021 byla uvedena na trh aktualizovaná verze HTC Vive Pro 2 s vyšším rozlišením, obnovovací frekvencí i zorným polem [23j]. Na začátku roku 2023 vydal Playstation vlastní VR headset s rozlišením 2000 x 2040, obnovovací

frekvencí 90/120 Hz, a zorným polem 110° [23i]. Jedná se o horší parametry než má HTC Vive Pro 2, ale i za poloviční cenu předčí zařízení dostupné v laboratoři.

Vývoj aplikací pro HTC Vive vyžaduje speciální software. Vive Wave SDK [23k] je poskytováno výrobcem zařízení zdarma. Obsahuje nativní SDK pro Android a pluginy pro herní enginy Unity i Unreal Engine. Další možností je využít otevřeného standardu pro přístup k VR a AR zařízením OpenXR, který je také podporován v Unity a Unreal Engine. Jeho výhodou je multiplatformita, jelikož podporuje i zařízení bez operačního systému Android od ostatních výrobců.

2.1.2 3D skenery

Domácích nebo hobby 3D skenerů není na trhu velké množství. Většinou se jedná o profesionální zařízení, jejichž cena začíná na několika desítkách tisíc korun. Dnes již nevyráběný Microsoft Kinect byl jedním z mála přístrojů v této kategorii a zahájil revoluci v počítačovém vidění, jelikož přinesl schopnosti 3D snímání široké veřejnosti. Microsoft ukončil jeho výrobu v roce 2017 [Wei18] a dnes využívá cloudové řešení Azure Kinect [23c]. Pomyslně na původní Kinect navázal Intel se svojí řadou RealSense, jehož skenery D400, byly vydány v roce 2018 [GSW18]. Po konzultaci se Západočeskou univerzitou v Plzni pořídilo Gymnázium Sokolov tři skenery Intel RealSense D415.

D415 je druhý z řady 3D skenerů a jeho hloubkový senzor má následující vlastnosti [23d]:

- Zorné pole: 65° x 40°,
- Minimální vzdálenost: 45 cm,
- Rozlišení: 1280 x 720 pixelů,
- Snímková frekvence: 90 fps .

Skener obsahuje také barevnou kameru s parametry:

- Zorné pole: 69° x 42°,
- Rozlišení: 1920 x 1080 pixelů,
- Snímková frekvence: 30 fps .

Intel RealSense poskytuje SDK pro širokou škálu programovacích jazyků a nástrojů. Z webových stránek zařízení [21b] lze zjistit, že podporují jazyky C/C++, C#/.NET, Matlab, Node.js a Python společně s řadou vývojových prostředí od rozsáhlých herních enginů Unity, Unreal Engine přes knihovny pro 3D a počítačové vidění Open3D, PCL, OpenCV, OpenVINO až po robotí operační systémy ROS /

ROS 2. SDK od Intelu funguje na Windows 10, Windows 7, Linux, macOS, Android, Jetson, Raspberry Pi 3 a Firefly.

2.1.3 Kamery

Mezi dostupnými kamerami je dnes velký výběr. Volba padla na stálici mezi přenosnými kamery - GoPro, kde lze očekávat dobrou podporu a dostupnost pomocných materiálů pro práci s kamerou. Bohužel zvolená HERO 7 White je jedna z nejslabších dostupných GoPro verzí a používá jiný firmware [18a] než ostatní verze Black, které běží na speciálně vytvořené platformě pro GoPro - GP1 a později GP2 [17], z čehož plyne řada omezení. Kromě horší kvality videa nebo pořízených fotek

- Rozlišení videa: 1440p 60 fps,
- Rozlišení fotografie: 10 Mpx,

také neschopnost měnit parametry kamery [Leb23]. Obtíže způsobené horší kvalitou záznamu by bylo možné bez problémů ignorovat, bohužel větší problém nastává s kompatibilitou.

Pro GoPro kamery existuje množství volně dostupných aplikací a knihoven pro různé programovací jazyky [Itu23b; Itu22; Itu23a; 16a], které rozšiřují jejich funkčnost nad rámec výrobcem dodávaného softwaru. Manipulace s kamerami je umožněna díky rozsáhlému, výrobcem dodávanému API [23h]. Kamery poskytují připojení přes Wifi, Bluetooth, HDMI nebo USB. Kombinace vybraných standardů závisí na konkrétním modelu. GoPro HERO 7 White poskytuje pouze připojení přes Wifi a Bluetooth. Všechny GoPro však mají stejné IP adresy a i v případě kompatibility se softwarem je možné připojit pouze jednu kameru k jedné bezdrátové síti [23g]. Přes Bluetooth nebo USB lze připojit více kamer najednou.

2.1.4 Haptické zařízení

Haptické zařízení 3DSystems Touch [16c] je motorizovaný přístroj, jenž umožní uživateli cítit zpětnou vazbu při manipulaci s virtuálními objekty. Uplatnění najde v simulačních úlohách, 3D modelování, rehabilitaci a dalších. Zařízení poskytuje šest stupňů volnosti a dokáže tak plně reagovat na uživatelův pohyb zápěstím.

K zařízení je dodáván OpenHaptics toolkit [16b] umožňující práci se zařízením. Mimo tento toolkit existuje bezplatný plugin Haptics Direct for Unity V1 [23f], který přidává podporu haptických zařízení do Unity a HapticsDirect for Unreal [21a] pro vývoj v Unreal Enginu.

2.2 Využití laboratoře

Laboratoř byla navržena jako víceúčelové zázemí pro výuku, studentské zájmové kroužky i veřejnost se zájmem o různé oblasti informačních technologií. Z dostupných zařízení je patrné, že tento cíl realizace laboratoře splnila. Problémem je dostupný software a neexistence výukových (metodických) materiálů pro zařízení uvnitř laboratoře. Jak bylo ukázáno výše, lze vyhledat řadu existujících aplikací a knihoven. Na nich by bylo možné schopnosti zařízení demonstrovat, ale návody a úlohy, které by uživatele aplikacemi provedly, stimulovaly by zájem o zařízení a prohlubovaly vědomosti o zařízeních, se vyskytují jen velmi sporadicky, pokud vůbec existují. Toto omezení výrazně snižuje atraktivitu zařízení, jelikož klesá použitelnost laboratoře v rámci výuky.

2.2.1 Aktéři

Veřejnost Návštěvníci z řad veřejnosti musí být schopni zařízení zprovoznit, vyzkoušet a seznámit se s jejich základní funkčností. To vše samostatně, bez dohledu odborného pracovníka. K tomuto účelu by měly sloužit připravené návody pro různá zařízení a na míru vytvořené aplikace. Návod návštěvníky provede ovládáním a dostupnými funkcemi, které vhodně prezentují schopnosti příslušného zařízení.

Žáci při výuce Pro žáky budou v rámci výuky k dispozici stejné manuály a aplikace jako pro veřejnost. Nad rámec těchto dokumentů budou mít k dispozici i metodické materiály se zadáním úloh, jejichž cílem je provést uživatele úpravou vlastností zařízení či běžící aplikace. Úpravy by měly probíhat uvnitř aplikace, typicky formou změny vlastností připojených zařízení nebo psaním jednoduchých skriptů. Nemělo by však být nutné opětovně sestavovat aplikaci. Nejedná se tedy o editaci aplikace jako takové, nýbrž pouze o lehkou úpravu výsledku existujícího řešení.

Žáci při zájmové činnosti Žáci se zájmem nad rámec běžné výuky budou mít k dispozici také navazující úlohy včetně všech výše zmíněných podkladů z předchozích odstavců. Cílem těchto úloh je úprava kódu existujících aplikací, popřípadě vytvoření vlastního řešení s pomocí dostupných knihoven.

Návrh úloh pro laboratoř

3

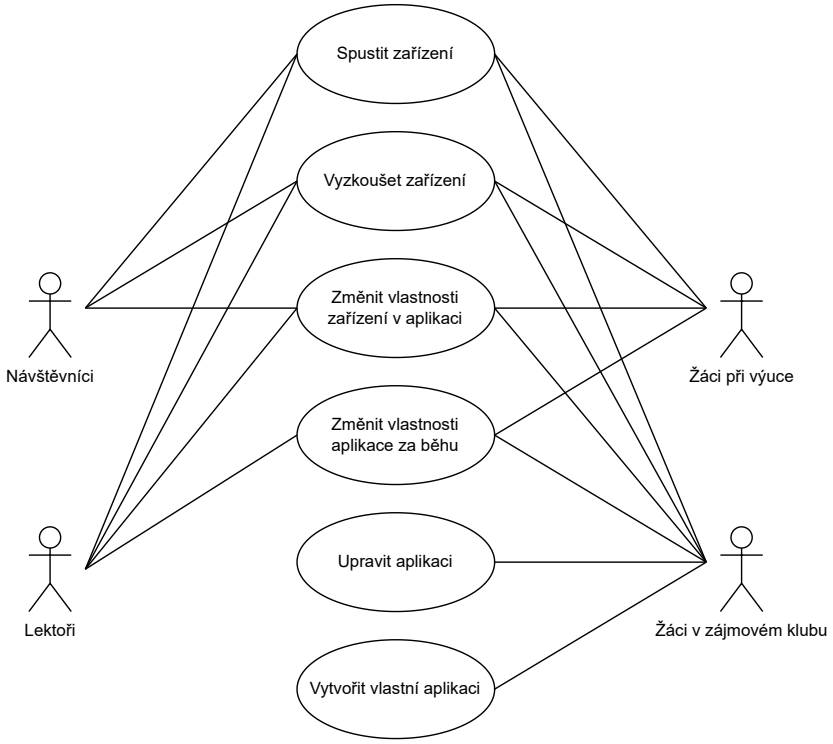
V kapitole 2.2 byli představeni aktéři systému a potřeby stakeholderů na gymnáziu. S tímto vědomím musí práce uspokojovat potřeby rozličných skupin uživatelů na obr. 3.1 a odpovídat požadavkům gymnázia. Primárními artefakty jsou

- návody pro zvolená zařízení,
- úlohy pro zvolená zařízení,
- vzorová řešení úloh,
- podpůrné aplikace pro úlohy.

3.1 Centrální systém

Navržené úlohy budou pomyslnou součástí většího řešení, zastřešujícího všechny vyvíjené programy a využívaná zařízení. Cílem systému je propojit více zařízení najednou, ačkoliv není typicky běžné nebo možné je spolu využívat. Tato přidaná hodnota otevírá nové možnosti, jak uživatelům demonstrovat vlastnosti a schopnosti zařízení. Může být možné například vzdáleně zobrazovat výstup 3D skeneru ve VR brýlích nebo manipulovat se stejným obsahem ve dvou VR brýlích. Nároky kladené na stolní počítače totiž bývají při připojení brýlí pro virtuální realitu příliš vysoké na to, aby jich bylo možné použít více najednou.

Celý systém bude navržen jako internetová aplikace. Dokáže propojit programy ovládající různá zařízení pomocí definovaného API. Díky němu lze začlenit do systému i nové programy, které mohou být v budoucnu vyvíjeny samotnými žáky gymnázia. Celkové řešení tak umožní bezproblémovou interakci mezi různými zařízeními. Systém bude schopný udržovat svůj globální stav. Jelikož zařízení typu VR brýlí a 3D skeneru pracují v trojrozměrném prostoru a výstupy ostatních zařízeních do něj lze jednoduše převést, je globálním stavem myšlena virtuální scéna trojrozměrných objektů. Pomocí zmíněného API bude možné scénu měnit a manipulovat s objekty uvnitř ní. Objekty bude možné



Obrázek 3.1: Případy užití pro úlohy

- přidávat,
- odebírat,
- stahovat.

Aby bylo možné objekty ve virtuální scéně umístit, musejí mít definovanou transformaci - pozici, rotaci a měřítko. Objekty mohou vyžadovat pro svůj popis i další vlastnosti. Například obrázek by mohl chtít sdílet svojí výšku, šířku a barevné hodnoty pixelů. Je nutné zajistit přístup k těmto vlastnostem. Systém by proto měl být schopen

- upravit vlastnosti objektu,
- vrátit vlastnosti objektu.

Aby bylo možné systém rozšířit a přidávat do něj nové programy, musí být zajištěna dostatečná obecnost rozhraní. Cílem je vytvořit dostatečně generickou strukturu dat pro komunikaci mezi odlišnými částmi systému. Základním kamenem návrhu by měla být možnost rozesílat jakékoliv typy objektů. Do budoucna je nutné počítat s tím, že studenti mohou chtít rozesílat i jiné typy objektů kromě

těch definovaných v této práci. Již nyní vznikají pro centrální systém i další klienti v rámci projektu IKAP, kteří nejsou součástí této práce, na nichž bude také testován a bude s nimi muset komunikovat.

Důraz by měl být kladen také na jednoduchost navrženého systému a jeho rozhraní, aby studenti gymnázia mohli bez větších obtíží upravovat existující řešení nebo vytvářet své vlastní. Z toho důvodu bude dbáno na použití obecně rozšířených technologií.

Systém bude fungovat pouze v rámci laboratoře a nepovoluje přístup klientů zvenčí. Z důvodu velikosti technologické laboratoře se nepředpokládá velké množství konkurentně připojených klientů. Škálovatelnost a výkonnost systému proto nejsou primárními cíli a mělo by na ně být myšleno pouze okrajově.

3.2 Správa systému

První sada úloh bude směřována na správu celého systému, aby studenti získali přehled o tom, jak systém funguje a co s ním mohou potenciálně tvořit. Úlohy budou zpracovávány na stolních počítačích nebo velkém interaktivním stole s aplikací, přes kterou budou moci systém ovládat.

3.2.1 Základní úloha

Cílem základní úlohy je pomocí dodané aplikace naučit uživatele pracovat se sdílenými objekty. Úloha provede uživatele

- připojením k systému,
- nahráním objektu,
- základním ovládáním,
- manipulací s kamerou a objekty,
- výběrem objektu,
- smazáním objektu,
- získáváním změn ze systému.

3.2.2 Navazující úloha

Navazující úloha předvede uživateli, jak manipulovat s objekty na serveru pomocí skriptů. Skripty budou psány a spouštěny za běhu dodané aplikace, aby měly přístup ke sdíleným objektům. V návodu bude popsáno, jak pracovat se skriptovacím

frameworkem, co očekávat za objekty a jak se výsledek metody dále zpracovává. Součástí zadání bude také úkol, který studenti musejí ve skriptovacím jazyce vypracovat.

3.3 Haptika

Druhá sada úloh se bude zabývat haptickým zařízením. Úlohy poskytnou uživateli průpravu pro ovládání zařízení a naučí ho, jak pracovat s haptickým zařízením ve zdrojovém kódu.

3.3.1 Základní úloha

Cílem základní úlohy pro haptické zařízení je vyzkoušet si fyzikální vlastnosti, které dokáže simulovat a naučit uživatele, jak používat haptické pero. Typickými jevy, které lze přes haptické zařízení pozorovat jsou

- tvrdost,
- viskozita
- tření,
- hmotnost,
- síla.

Náročnost používání haptického pera tkví především v tom, osvojit si způsob kolizí s ostatními objekty. Proto by uživatel měl být schopen vyzkoušet si chování na objektech různých tvarů.

3.3.2 Navazující úloha

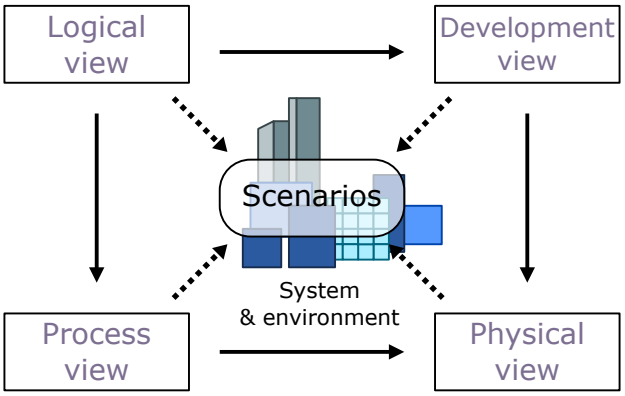
Základním kamenem navazující úlohy je aplikace, která umožňuje pomocí haptického zařízení kreslit na načtené objekty. Z existujícího řešení bude odebrán kód, který převádí data z uživatelského rozhraní do ovládacích tříd haptického zařízení. Úkolem bude doplnit odebraný kód a zprovoznit aplikaci tak, aby odpovídala zadání. Na rozdíl od navazující úlohy správy systému se nebude pracovat se skriptováním za běhu aplikace, ale bude se přistupovat přímo ke zdrojovému kódu. Podmínkou této úlohy je dostupnost vývojového prostředí.

Architektura systému 4

Před návrhem architektury se hlavní úsilí vynakládá na pochopení řešeného problému a nalezení teoreticky uskutečnitelného řešení. Určení architektury přidává do nalezeného řešení definovanou, obecně srozumitelnou strukturu přístupnou pro další zainteresované osoby. Existují předepsané návody, jak systematicky navrhnout architekturu systému. V práci je využit 4+1 model [Kru95].

První fáze tvorby architektury se soustředí pouze na rozeznání, oddělení a tvorbu abstraktní logické struktury hlavních úkonů z daného problému. K jejich rozpoznání využívá funkčních požadavků na vytvářený produkt. Tento postup se v 4+1 modelu nazývá logická architektura. Zde se části problému logicky rozčlení na abstraktní objekty (třídy), logicky souvisejí funkční celky, a vytvoří se mezi nimi komunikační rozhraní. Rozdělení problému na menší celky pomáhá identifikovat společné části z různých koutů systému. Toto rozdělení musí fungovat nezávisle na programovacím jazyce nebo jiných technických a implementačních podrobnostech. K určení vhodné struktury pomáhají známé architektonické vzory, například monolitický systém, vrstvená architektura (MVC, MVVM, MVP, ...), peer-to-peer, událostmi řízená architektura a další.

Když je logická struktura vytvořená, realizuje se řešení v konkrétních technic-



Obrázek 4.1: 4+1 architektonický model [Dek16]

kých artefaktech. Návrh těchto artefaktů je v 4+1 modelu řešen ve fázi vývojové architektury. Cílem je převést logické třídy do implementovatelných jednotek a eventuálně je spojit do menších subsystémů (balíků nebo knihoven). Aby bylo možné mezi subsystémy komunikovat, musí se definovat vazby mezi nimi. Návrh již musí být možné realizovat v konkretním programovacím jazyce. Z tohoto tvrzení vyplývá, že ve vývojovém pohledu na architekturu jsou řešeny technické záležitosti spojené s implementací a zvoleným programovacím jazykem. Kromě volby programovacího jazyka jsou očekávatelnými kroky rovněž výběr frameworků nebo knihoven a vytvoření konvencí a politik pro implementaci.

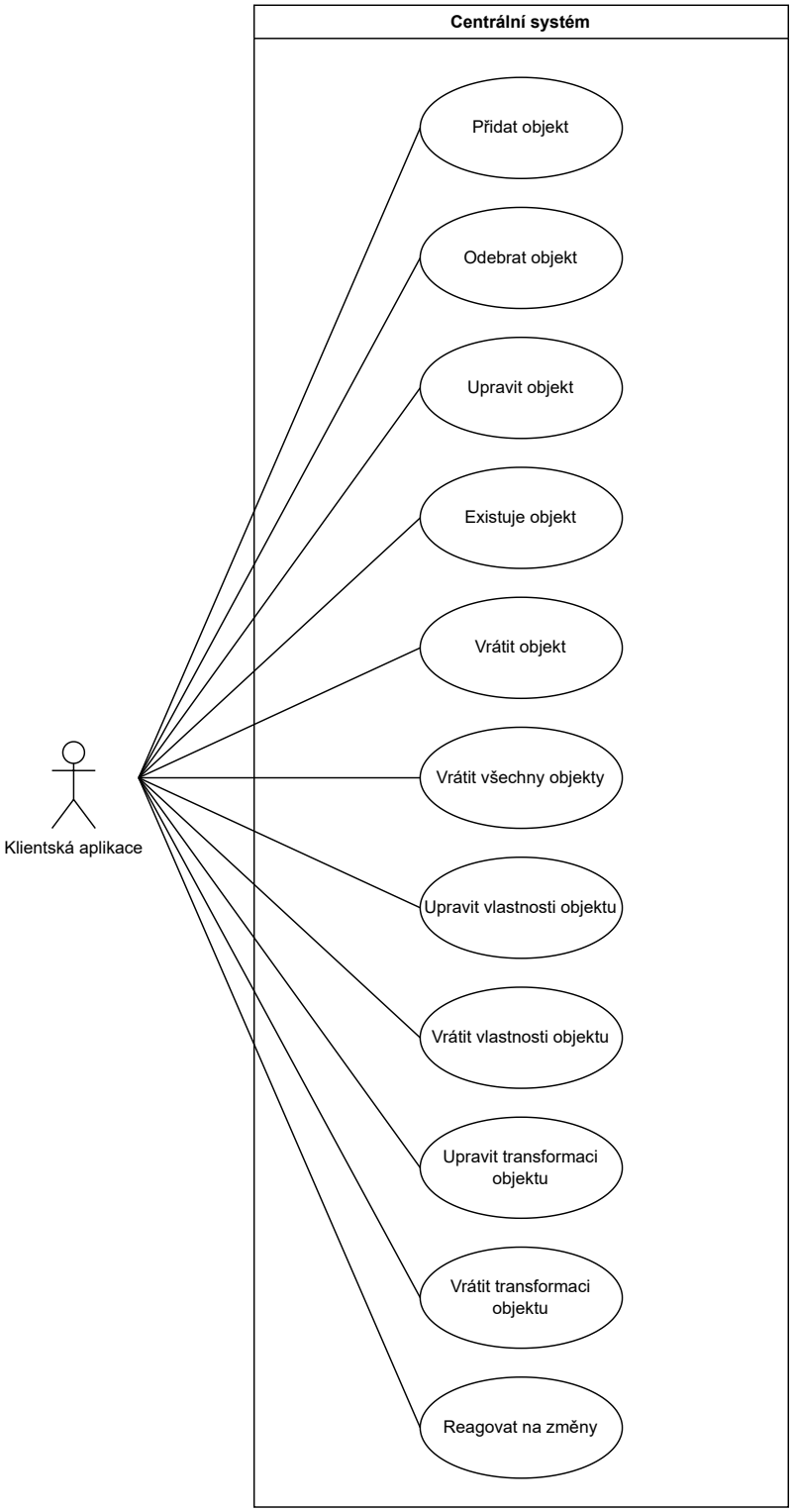
Zformované subsystémy je nutné spojit do větších celků, které jsou vhodné pro nasazení a následnou údržbu. Toto sjednocení se provádí v návrhu fyzické architektury. Nezbytné je při spojení dodržet závislosti mezi balíky, aby bylo možné knihovny a aplikace sestavit. Když jsou celky vytvořeny, určí se způsob jejich sestavení a nasazení. Identifikuje se na jaké platformě a jakých zařízeních mají fungovat. Ohled je brán především na požadavky spolehlivosti, výkonnosti a škálovatelnosti.

Předchozí kroky pokrývají většinu fází implementace produktu. Do této doby však bylo ignorováno, jakým způsobem aplikace poběží na cílové platformě. Proto je nutné navrhnout strukturu procesů a vláken. Součástí je taktéž definice komunikace a synchronizace mezi nimi. Procesní model se vytváří uvnitř procesní architektury 4+1 modelu.

4.1 Logická architektura

Z požadavků na centrální systém v kapitole 3.1 a z popisu úloh lze vyvodit všechny nezbytné případy užití, jež se v systému vyskytují. Jejich kompletní souhrn lze vidět v UML diagramech na obr. 4.2, 4.3, 4.4. Z pohledu klientské aplikace, jakožto aktéra centrálního systému, musí být možné

- nahrát objekt do systému, pokud ho aktér chce sdílet s ostatními aktéry,
- odstranit existující objekt ze systému,
- upravit celý objekt včetně transformace a vlastností,
- zeptat se na existenci objektu, aby se předešlo zbytečným dotazům,
- vrátit existující objekt ze systému,
- vrátit všechny existující objekty v systému,
- upravit vlastnosti existujícího objektu,
- vrátit vlastnosti existujícího objektu,



Obrázek 4.2: Případy užití klientských aplikací

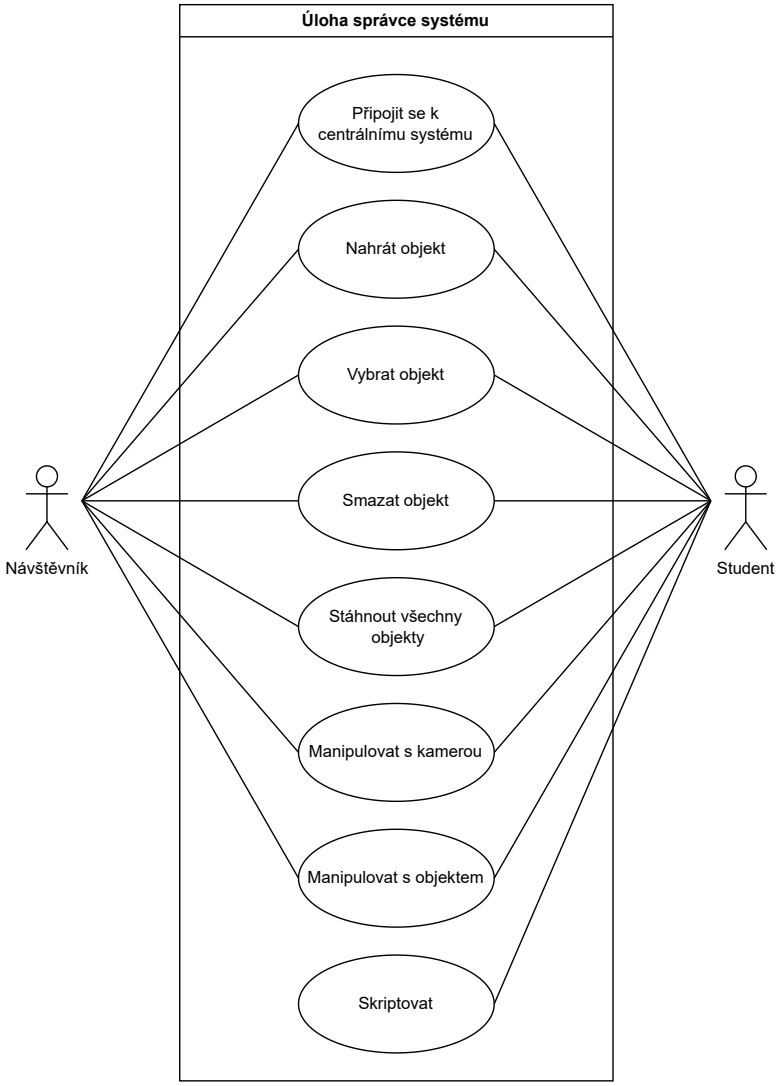
- upravit transformaci existujícího objektu,
- vrátit transformaci existujícího objektu,
- reagovat na změny v systému.

Tím by mělo být možné pokrýt požadavky na centrální systém i do budoucna. Případy užití úloh na správu systému jsou následující

- připojit se k systému,
- nahrát objekt,
- vybrat objekt,
- smazat objekt,
- stáhnout všechny objekty ze systému,
- manipulovat s kamerou,
- manipulovat s vybraným objektem,
- skriptovat.

Vypozorované případy užití pro haptickou úlohu jsou

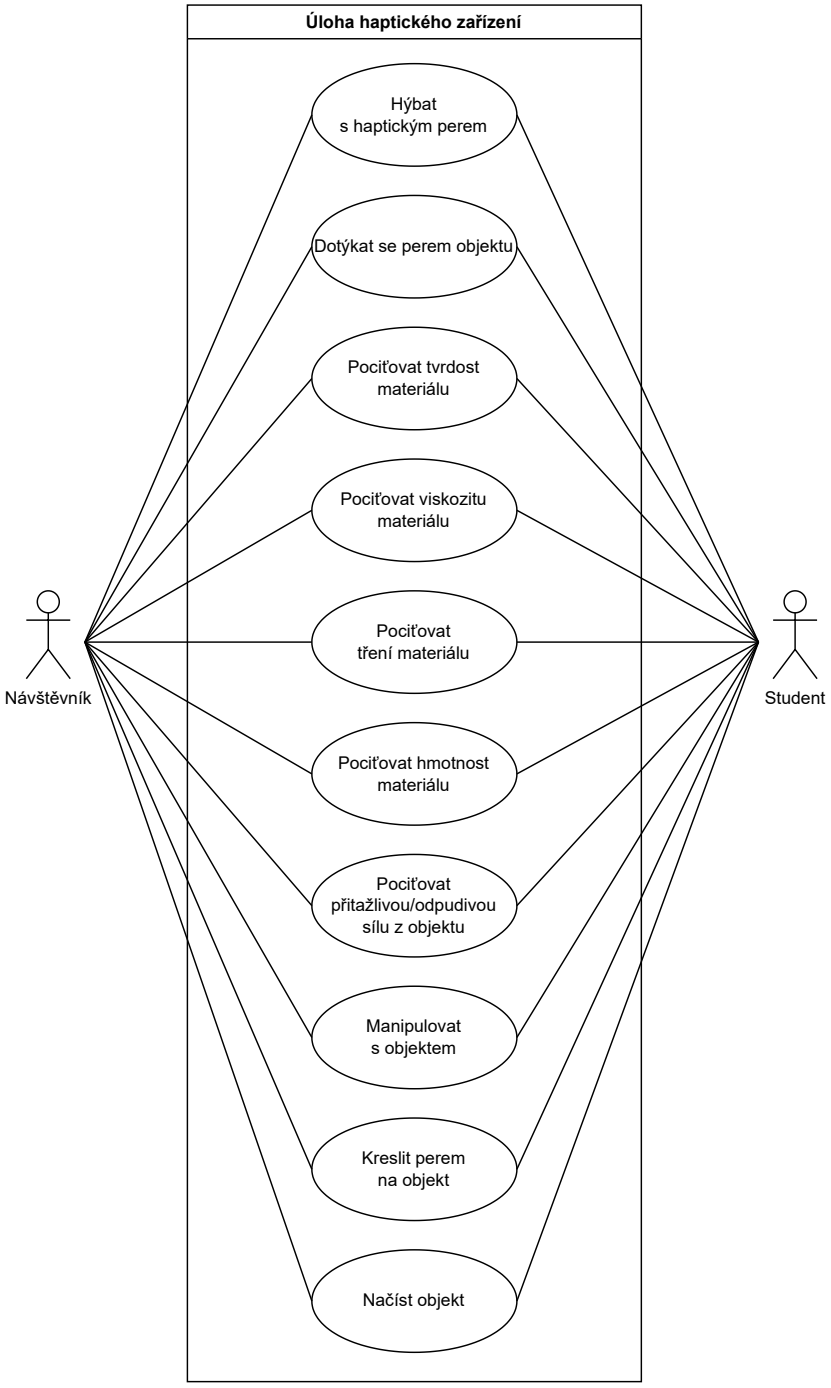
- hýbat s haptickým perem,
- dotýkat se perem objektů,
- pociťovat tvrdost materiálu,
- pociťovat viskozitu materiálu,
- pociťovat tření materiálu,
- pociťovat hmotnost materiálu,
- pociťovat přitažlivou/odpudivou sílu z objektu,
- manipulovat s objektem pomocí pera,
- kreslit perem na objekty,
- načíst objekt.



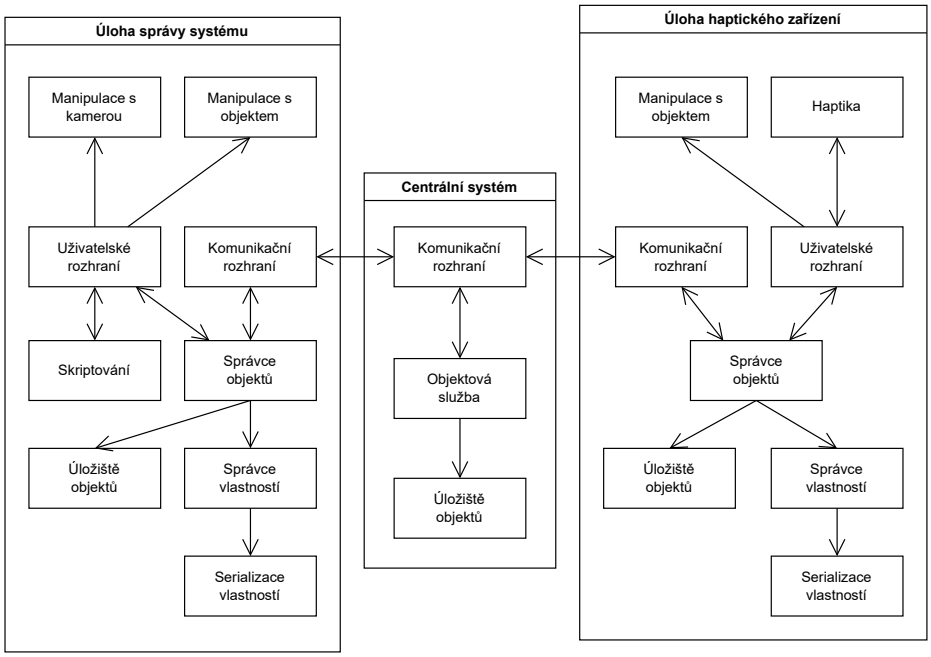
Obrázek 4.3: Případy užití úlohy správce systému

Z případů užití lze identifikovat společné jmenovatele. Ty pomohou při rozdělení systému na logické třídy. Centrální systém a klientské aplikace spolu potřebují komunikovat, aby bylo vůbec možné mezi nimi předávat zprávy. Jedny z tříd tedy budou komunikační rozhraní. Aby centrální systém mohl objekty zpracovávat, bude vyžadovat existenci nějaké služby, která se o objekty stará. Pouze s touto službou by však systém nedokázal udržovat svůj stav, proto bude obsahovat také úložiště objektů.

Aby klientská aplikace mohla pracovat s objekty, bude pravděpodobně potřebovat správce objektů. Od něj se očekává, že bude synchronizovat lokální stav objektů v uživatelském rozhraní s centrálním systémem. Znalost lokálních objektů by měla být osamostatněna od funkce správce, protože se jedná o jiné chování. V architektuře



Obrázek 4.4: Případy užití úlohy haptického zařízení



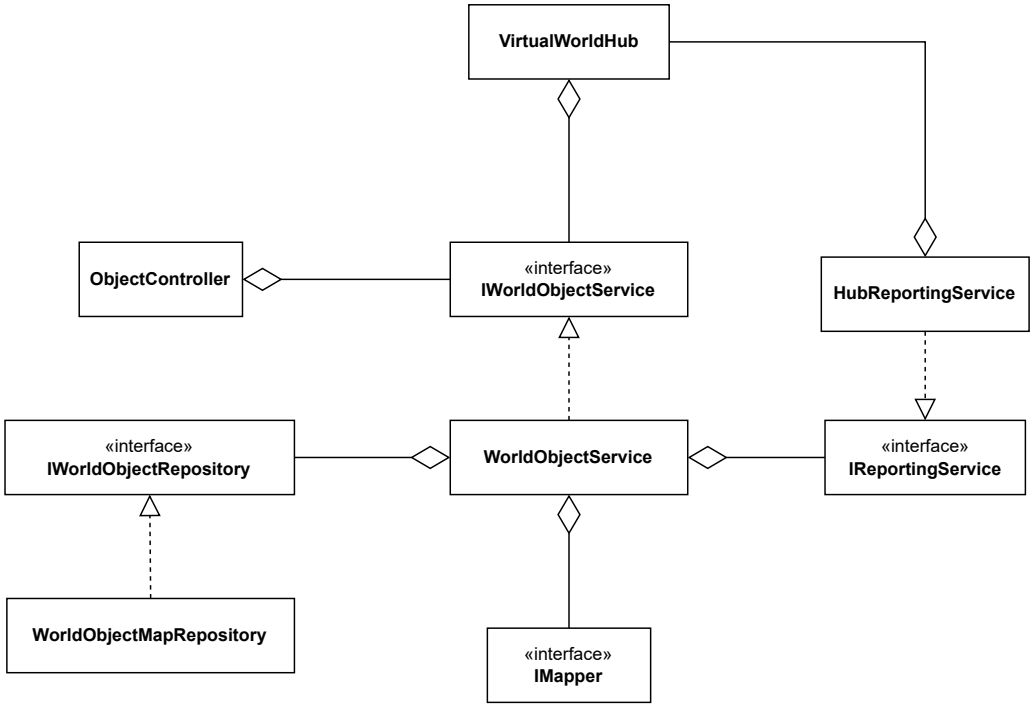
Obrázek 4.5: Doménový model

přibude třída pro lokální úložiště objektů.

Objekty obsahují specifické vlastnosti, jak bylo popsáno dříve, například pixely u obrázku. Těchto vlastností může být libovolné množství podle typů přenášených objektů. Podle případů užití bude s vlastnostmi objektu také možné pracovat, proto další třídou bude správce vlastností. V návrhu centrálního systému v kapitole 3.1 je zmíněno o rozšiřitelnosti a obecnosti struktury posílaných dat, kvůli tomu lze očekávat, že vlastnosti objektů bude nutné nějakým způsobem před odesláním upravit. K tomu bude sloužit subsystém serializace vlastností.

V zadání úloh jsou zmíněny kroky, které prozatím nejsou zakomponovány do žádné ze tříd. V úloze správy systému musí být možné manipulovat s kamerou a objektem. To jsou úkoly, které jdou přímo převést do tříd, jelikož budou vyžadovat nějaký systém, který se o danou funkčnost stará. Stejně tak skriptování, které musí být možné provádět v rámci uživatelského rozhraní. Úloha haptického zařízení umožňuje také manipulaci s objektem, ale pravděpodobně se bude jednat o jiný způsob manipulace než v první úloze. Jelikož se jedná o úlohu pro haptické zařízení bude nutné propojit haptiku s uživatelským rozhraním. K tomu může sloužit samostatná logická třída.

Na obr. 4.5 lze vidět popsanou strukturu v jednoduchém diagramu. Z digramu je patrné, že se jedná o server-klient architekturu.



Obrázek 4.6: Architektura serveru

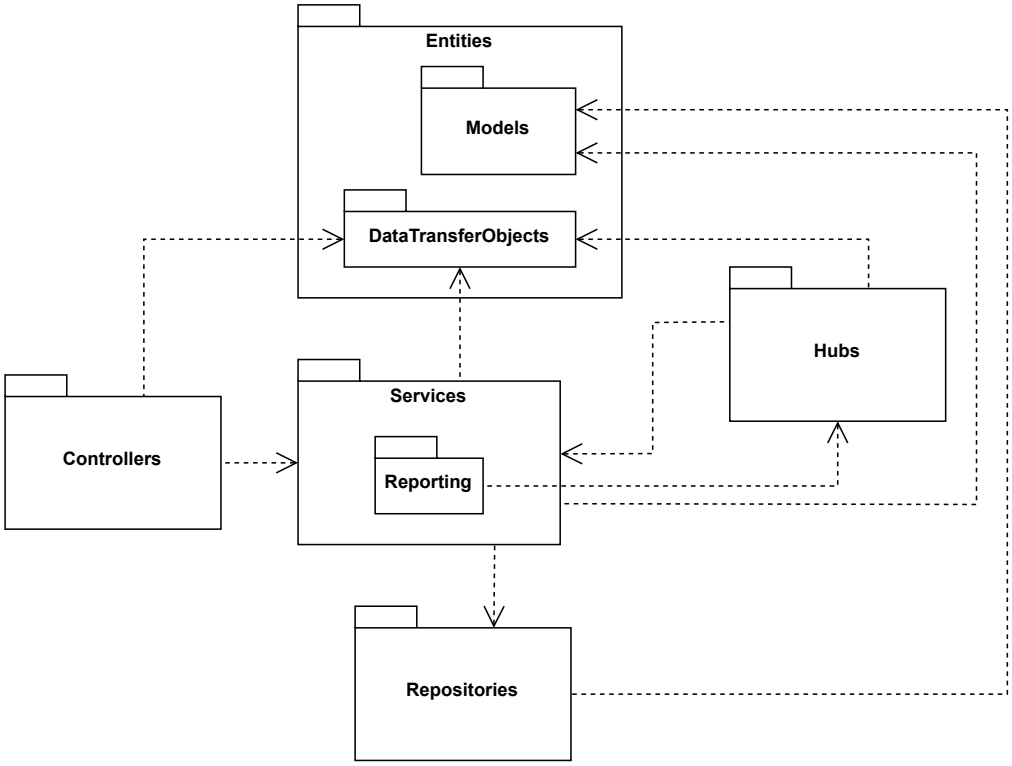
4.2 Vývojová a procesní architektura

4.2.1 Centrální systém

Centrální systém bude vyvíjen jako serverová aplikace na platformě .NET 6 v programovacím jazyce C#. Při implementaci budou dodržovány .NET konvence pro pojmenování [Wag+23]. K usnadnění vývoje webové aplikace se použije volně dostupný framework ASP.NET od Microsoftu pro zajištění základní webové funkcionality.

Na obr. 4.1 je systém navržen velmi abstraktně. Architektura musí být zkonkretizována, aby podle ní šlo vytvořit implementaci. Komunikační rozhraní se rozdělí na `ObjectController` obsluhující dotazy na server, `VirtualWorldHub` a `IReportingService` pro informování připojených klientů. Objektová služba zůstane téměř totožná, pouze se oddělí mapování mezi posílanými objekty a lokálními modely. Úložiště objektů bude reprezentováno rozhraním `IWorldObjectRepository`. Vytvořenou strukturu lze vidět na obr. 4.6.

Implementovatelné třídy se ve vývojové architektuře sjednocují do balíků, projektů a knihoven. Struktura serveru patří mezi jednodušší, proto balíky vesměs odpovídají jednotlivým třídám. V diagramu 4.7 přibyly balíky s entitami. Posílané objekty jsou oddělené od těch používaných uvnitř aplikace. Z diagramu 4.6 byly pro



Obrázek 4.7: Balíky centrálního systému

přehlednost vynechány.

Server bude reprezentován třemi projekty. Hlavní vývoj bude probíhat v projektu ZCU.TechnologyLab.VirtualWorldServer, zatímco zbytek bude sloužit pro testování. Server je centrální částí celého systému, měl by proto být stabilní. K tomu slouží kvalitní a rozsáhlé testování. Vzhledem k rozličným způsobům psaní testů došlo k rozdělení testovacích projektů. Tím dojde k zpřehlednění testů. Jednotkové testy půjde nalézt v projektu ZCU.TechnologyLab.VirtualWorldServer.Test. Dalším stupněm jsou integrační a funkční testy. Oběma bude příslušet projekt ZCU.TechnologyLab.VirtualWorldServer.IntegrationTest. Oba projekty budou záviset na knihovně NUnit, která bude sloužit jako framework pro psaní testů.

Bližší popis API pro webovou komunikaci je dostupný v kapitole 4.2.2. S využitím Swaggeru/OpenAPI [23a] lze vygenerovat jeho dokumentaci a vystavit ji veřejně přes url. Podporu tohoto frameworku do ASP.NET přidává knihovna Swashbuckle [Mor23]. Stejná cesta je využita i v této práci. Swagger má více funkcí. Umožňuje vygenerovat API podle definovaného předpisu a podporuje také testování existujícího API. Primární účel v této práci je však pouze dokumentační.

Komunikace v reálném čase a rozesílání zpráv o změnách na serveru jsou postaveny na opensource knihovně SignalR [Gas+23]. SignalR vytváří relaci mezi serve-

rem a klientem, v níž umožňuje provozovat vzdálené volání procedur (RPC). Komunikace probíhá buďto přes WebSockety, serverem odesílané události (Server-Sent Events) nebo long polling. Způsob záleží na schopnostech komunikujících stran. Rozhraní pro vytváření RPC se nazývá hub. Definuje dostupné metody, které mohou být mezi serverem a klientem volány. Podporované je silné typování parametrů. Formát zasílaných dat je textový - JSON nebo binární pomocí protokolu MessagePack [13].

Přijímané nebo odesílané objekty přes veřejné API serveru by měly být odlišné od modelů používaných v backendu aplikace. Typicky se pro každý dotaz vytváří nový objekt, který reprezentuje nějaký model nebo jeho poměrnou část. Pokud jsou modely uloženy například v databázi, jsou pozitiva tohoto řešení ještě větší, protože lze bez problémů měnit veřejné rozhraní bez nutnosti změn v databázi. Pro převod z posílaných objektů na interní modely se použije knihovna AutoMapper [23b].

V případě nutnosti provádět asynchronní operace, bude vždy cíleno na využití tasků. Vykonávání všech dotazů na serveru bude probíhat asynchronně. SignalR knihovna také využívá pro zpracování procedur tasky. Tasky fungují jako rozhraní vytvořené nad vlákny z thread poolu a jsou plánovány .NET runtimem.

4.2.2 Komunikační API

Server podporuje komunikaci přes dvě rozhraní - REST API a SignalR. REST API se používá pro přístup ke zdrojům, které jsou uloženy na serveru. Vzhledem k tomu, že uložené zdroje mohou obsahovat velké množství dat, je REST vhodným kandidátem pro tuto činnost. SignalR slouží pouze pro předávání zpráv, které vyžadují komunikaci v reálném čase. Jeho primárním účelem je informovat o změnách ve zdrojích, které byly provedeny přes REST.

Poskytnutá API se v práci vzájemně doplňují, proto je doporučeno využívat pro plnohodnotné využití serveru obě zároveň.

4.2.2.1 REST API

REST API je běžně využívaný standard pro komunikaci mezi serverem a klientem. Jeho rozsáhlé použití, jednoduchost a obecná znalost mezi programátory byly jedny z rozhodujících faktorů, které vedly k jeho zapojení do komunikace mezi serverem a klienty.

GET api/objects/{name} Vrací konstrukci, která je k vidění v ukázce 4.1. Konstrukce se dále v práci nazývá world objekt.

Zdrojový kód 4.1: JSON vracející objekt

```
1 {
```

```
  "Name": "string",
  "Position": {
    "X": 0,
    "Y": 0,
    "Z": 0
  },
  "Rotation": {
    "X": 0,
    "Y": 0,
    "Z": 0
  },
  "Scale": {
    "X": 0,
    "Y": 0,
    "Z": 0
  },
  "Type": "string",
  "Properties": {
    "additionalProp1": "string",
    "additionalProp2": "string",
    "additionalProp3": "string"
  }
}
```

GET api/objects/ Vrací seznam všech world objektů. Jeho JSON struktura je k vidění v ukázce 4.2.

Zdrojový kód 4.2: JSON vracející všechny objekty

```
{
  [
    {
      "Name": "string",
      "Position": {
        "X": 0,
        "Y": 0,
        "Z": 0
      },
      "Rotation": {
        "X": 0,
        "Y": 0,
        "Z": 0
      },
      "Scale": {
        "X": 0,
        "Y": 0,
        "Z": 0
      },
```

```
      "Type": "string",
      "Properties": {
        "additionalProp1": "string",
        "additionalProp2": "string",
        "additionalProp3": "string"
      }
    },
    {
    ...
    },
    ...
  ]
}
```

POST api/objects/ Přidává world objekt 4.1.

DELETE api/objects/{name} Odebere world objekt podle jména. Smazaný world objekt se nevrací.

PUT api/objects/{name} Upravuje world objekt. Objekt je určen jeho jménem. Dotaz využívá speciální objekt pro komunikaci, který je v ukázce 4.3.

Zdrojový kód 4.3: JSON pro úpravu objektu

```
{
  "Position": {
    "X": 0,
    "Y": 0,
    "Z": 0
  },
  "Rotation": {
    "X": 0,
    "Y": 0,
    "Z": 0
  },
  "Scale": {
    "X": 0,
    "Y": 0,
    "Z": 0
  },
  "Type": "string",
  "Properties": {
    "additionalProp1": "string",
    "additionalProp2": "string",
    "additionalProp3": "string"
  }
}
```

GET api/objects/{name}/properties Vrací vlastnosti world objektu. Vlastnosti world objektu mají vlastní objekt v JSONu 4.4.

Zdrojový kód 4.4: JSON vracející vlastnosti objektu

```
{
  "Properties": {
    "additionalProp1": "string",
    "additionalProp2": "string",
    "additionalProp3": "string"
  }
}
```

PUT api/objects/{name}/properties Upravuje vlastnosti world objektu.
GET api/objects/contains/{name} Zjišťuje existenci world objektu.

4.2.2.2 SignalR API

Metody na serveru

- `TransformWorldObject(WorldObjectTransformDto?)` - mění transformace u world objektu

Metody na klientovi

- `AddWorldObject(string)` - informuje klienta o přidání objektu
- `UpdateWorldObject(string)` - informuje klienta o úpravě objektu
- `RemoveWorldObject(string)` - informuje klienta o odstranění objektu
- `UpdateWorldObjectProperties(string)` - informuje klienta o úpravě vlastností
- `TransformWorldObject(WorldObjectTransformDto)` - informuje klienta o změně transformace

4.2.2.3 Typy posílaných world objektů

Momentálně jsou v práci definované dva typy posílaných world objektů z JSONu 4.1, které se odlišují hodnotou property Type a obsahem Properties. Vlastnosti (obsah Properties) ve world objektu 4.1 jsou závislé na hodnotě Type. Některé property mohou být volitelné a ve world objektu se vyskytovat nemusí.

Obrázek

- Type: „Bitmap“
- Properties:
  - „Width“: int
  - „Height“: int
  - „Format“: string(„RGB“, „RGBA“, ...)
  - „Pixels“: byte[]

Prvním posílaným typem je obrázek. Odesílané vlastnosti jsou výška, šířka, formát a pole pixelů. Všechny vlastnosti obrázku jsou povinné. Kdyby jakákoliv hodnota ve slovníku Properties chyběla, obrázek by nebylo možné zrekonstruovat. Width (šířka) a Height (výška) jsou nutné pro znalost rozlišení obrázku. Bez nich by nebylo možné určit rozložení pixelů do řad a sloupců. Formát pixelů může mít několik předdefinovaných hodnot. RGB říká, že obrazová data v poli budou obsahovat tři složky pro jeden pixel - červenou, zelenou a modrou. RGBA pouze přidává alfa kanál s informací o průhlednosti. Formát pixelů může být libovolný, avšak všichni klienti, kteří se k API připojují, by měli být schopni s daným formátem pracovat.

Mesh

- Type: „Mesh“
- Properties:
  - „Vertices“: float[]
  - „Indices“: int[]
  - „Primitive“: string(„Triangle“, „Quad“, ...)
  - „UV“: float[]
  - „DiffuseTextureWidth“: int
  - „DiffuseTextureHeight“: int
  - „DiffuseTextureFormat“: string(„RGB“, „RGBA“, ...)
  - „DiffuseTexturePixels“: byte[]

  nebo

  - „Ply“: byte[]

  nebo

– „TextPly“: string

Mesh může obsahovat několik dalších podtypů v závislosti na obsahu slovníku Properties. Mesh, která obsahuje první sadu vlastností se v rámci práce nazývá raw mesh. Říká se jí raw, protože obsahuje pouze přímý výčet jednotlivých nekomprimovaných vlastností, které se dají obecně u meshe očekávat. Vertices, Indices a Primitive jsou povinné. Zbytek vlastností je volitelný a v meshi být nemusí. Pokud však v meshi budou vlastnosti DiffuseTexture, musejí tam být všechny. Vlastnost Primitive může, podobně jako Format u obrázku, obsahovat jakýkoliv řetězec, ale je vhodné domluvit možné hodnoty, které se mohou ve vlastnosti nacházet. V práci je podpořeno pouze primitivum Triangle. Jedná se o jediné, s kterým mohou pracovat všichni klienti. Určuje počet prvků pole Indices, které tvoří jedno primitivum meshe. Difuzní textura obsahuje stejné vlastnosti jako obrázek.

Druhá verze world objektu typu Mesh obsahuje textový nebo binární souborový formát PLY. Dále bude v práci pojmenována jako ply mesh. World objekt obsahuje v tomto případě pouze jednu ze dvou možných vlastností. Vlastnosti se vzájemně vylučují, nemohou být v meshi obě. Pokud obsahuje vlastnost Ply jedná se o binární verzi formátu, pokud se vlastnost jmenuje TextPly, pak je formát textový. Property obsahují přímo obsah souborů.

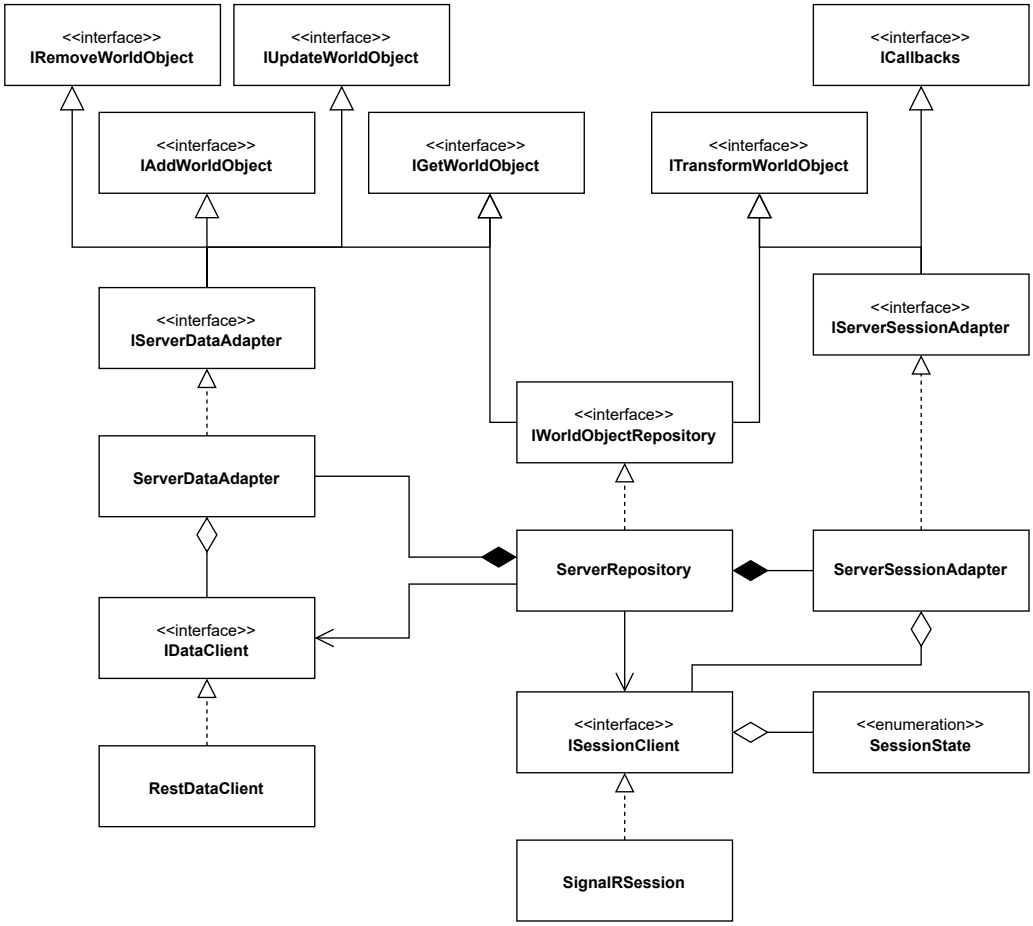
4.2.3 .NET knihovna

Na obr. 4.5 mají viditelně obě úlohy stejné části. Aby se předešlo duplicitám, budou vyvíjeny odděleně od vlastních aplikací jako jedna knihovna. Knihovna bude vytvořena na platformě .NET v programovacím jazyce C#. Stejně jako v případě serveru budou dodržovány .NET konvence pro pojmenování. Vývoj bude cílit na architektury .NET 6 a .NET Standard 2.1.

Pro připojení k serveru slouží rozhraní `IDataClient` a `ISessionClient`. Obalují funkcionalitu HTTP klienta respektive SignalR připojení a jsou implementovány ve třídách `RestDataClient` a `SignalRSession`. SignalR je dostupný přes nuget balíček Microsoft.AspNetCore.SignalR.Client.

Pro přístup k serverovým funkcím se používá abstraktní model zvaný `IWorldObjectRepository`. Pokud by bylo v budoucnu nutné zasílat data místo serveru jinam, například je ukládat do souboru nebo lokální databáze, bude možné použít stejné rozhraní. Operace uvnitř `IWorldObjectRepository` se dále dělí na

- `IAddWorldObject` - přidá objekt do úložiště,
- `IGetWorldObject` - vrátí objekty nebo jejich vlastnosti z úložiště,
- `IRemoveWorldObject` - odstraní objekt z úložiště,

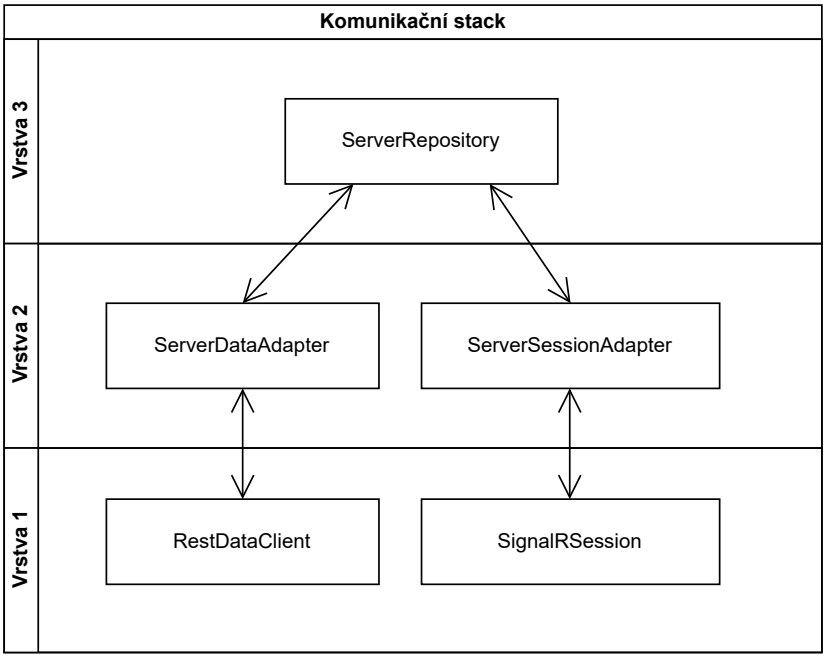


Obrázek 4.8: Architektura připojení v .NET knihovně

- `ITransformWorldObject` - změní transformaci objektu v úložišti,
- `IUpdateWorldObject` - upraví objekt nebo jeho vlastnosti v úložišti,
- `ICallbacks` - reportuje změny z úložiště.

Serverové úložiště reprezentuje třída `ServerRepository`, která implementuje rozhraní `IWorldObjectRepository`. Jedná se však pouze o použití návrhového vzoru proxy nad `ServerDataAdapter` a `ServerSessionAdapter`. Každý z nich implementuje pouze některá z výše vyjmenovaných rozhraní. Umožňují rozdělit dotazy na server podle způsobu komunikace. Celý komunikační stack je k vidění na obr. 4.9.

Serializaci vlastností posílaných world objektů předepisuje rozhraní `ISerializable` a abstraktní třída `PropertySerializer`, která implementuje společné metody všech serializovatelných datových typů. Každý serializovatelný typ posílaný přes komunikační rozhraní musí implementovat vlastní serializátor. V knihovně



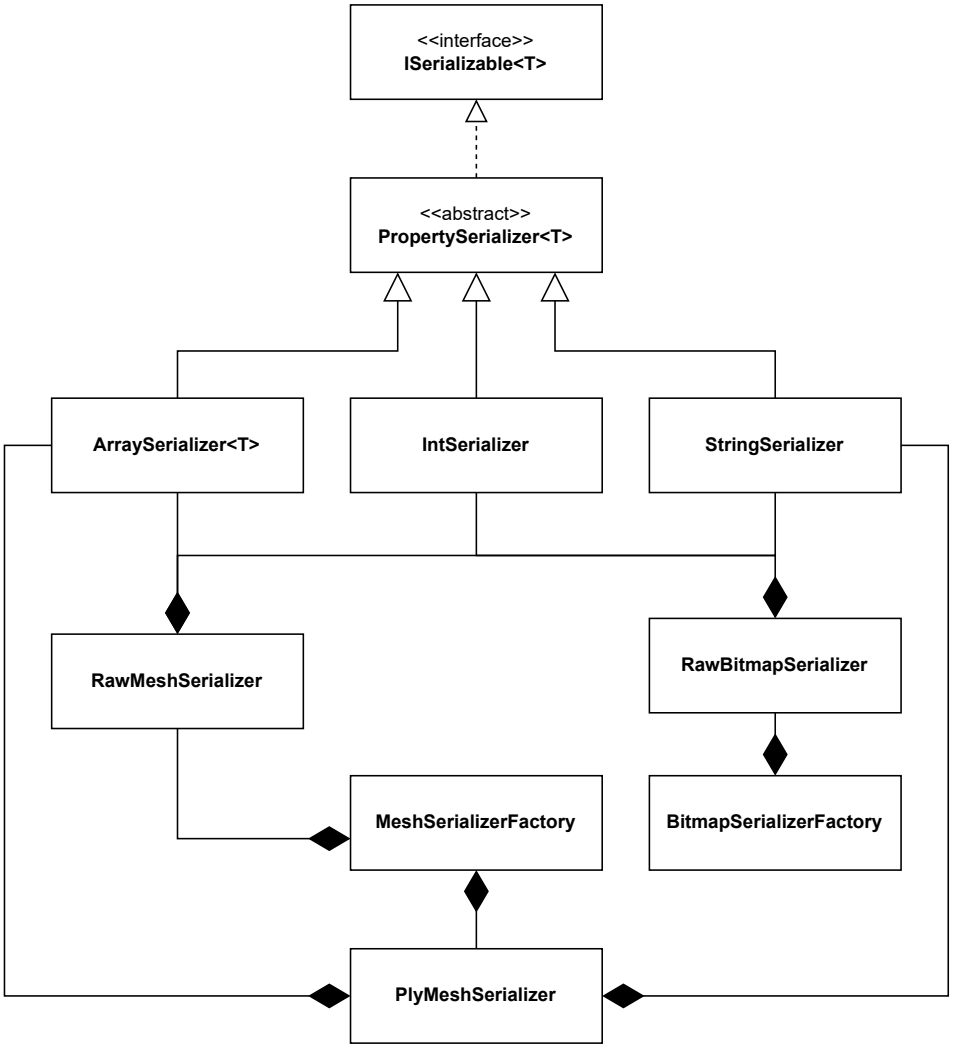
Obrázek 4.9: Komunikační stack v .NET knihovně

je připravena trojice `ArraySerializer`, `IntSerializer` a `StringSerializer`. Z definice představených posílaných world objektů by nemělo být nutné přidávat jiné datové typy než `int`, `Array` a `string`. Pro serializaci world objektů stačí pouze složit potřebné serializátory jejich vlastností.

Serializační mechanismus by měl být schopný rozeznat o jaký typ dat se jedná a podle něj vydat správnou serializační třídu. K tomu účelu existují továrny. Konkrétně se jedná o `BitmapSerializerFactory` a `MeshSerializerFactory`. Podle typu world objektu je možné získat `RawMeshSerializer`, `PlyMeshSerializer` a `RawBitmapSerializer`.

Rozdělení do balíků je trochu pracnější než u serveru. Celá architektura na obr. 4.8 tvoří balík Connection. Takový počet v jednom balíku by mohl být nepřehledný. Řešením je vytvořit ještě sadu podbalíků, které členění tříd více zjemní. Vrchní řada rozhraní z obr. 4.8 včetně `IWorldObjectRepository` lze spojit do jednoho společného balíku Repository. Další balík nazvaný Server obsahuje vše od `IServerDataAdapter` a `IServerSessionAdapter` až po `ServerRepository`. Nakonec balík pro připojení se jménem Client dále rozděluje `IDataClient`, `ISessionClient` a jejich implementace mezi balíky Data respektive Session.

Serializace rozděluje třídy do balíků velmi přímočaře. Všechny serializátory, které se zabývají vlastnostmi jsou umístěny v balíků Serialization/Properties. Továrny a serializátory world objektů existují v balících Serialization/Bitmap a Serialization/Mesh. Výsledné schéma je k vidění na obr. 4.11.



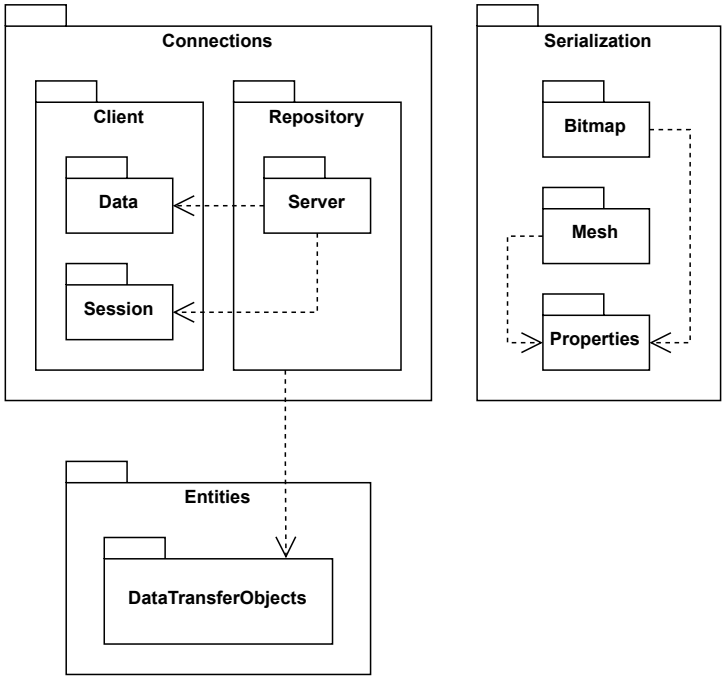
Obrázek 4.10: Architektura serializace v .NET knihovně

Knihovna se skládá ze dvou projektů. Výkonný kód je obsažen v projektu ZCU.TechnologyLab.Common a jednotkové testy v ZCU.TechnologyLab.Common.Test. Knihovna na rozdíl od serveru neobsahuje žádný projekt pro integrační testování.

4.2.4 Unity knihovna

Úlohy budou vypracovány v herním enginu Unity. Aby bylo možné využít .NET knihovnu uvnitř Unity, je potřeba vytvořit adaptéry, které umožní přistupovat ke knihovně z Unity scény. K tomuto účelu se vytvoří další knihovna. Součástí knihovny nebudou pouze adaptéry, ale také společné třídy, které se uplatní v obou úlohách. Programování bude probíhat v jazyce C#.

Připojení odpovídá .NET knihovně z předcházející kapitoly. Použitím návr-

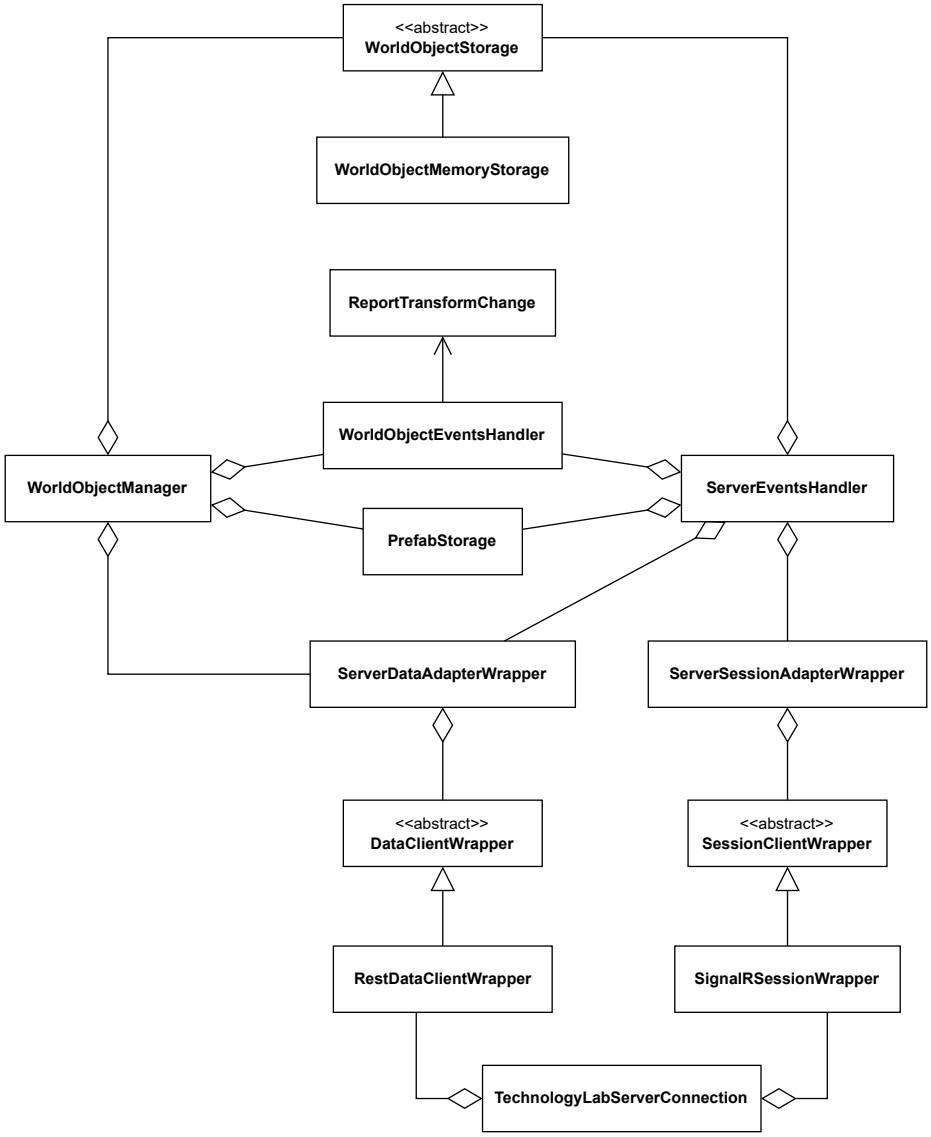


Obrázek 4.11: Balíky .NET knihovny

hového vzoru adaptér se nad třídami z .NET knihovny vytvoří obalovací objekty `RestDataClientWrapper` a `SignalRSessionWrapper`. Jejich rodiče `DataClientWrapper` a `SessionClientWrapper` plní roli rozhraní pro Unity scénu. Také `ServerDataAdapter` a `ServerSessionAdapter` mají své adaptéry.

Správce objektů je reprezentovaný třídami `WorldObjectManager` a `ServerEventsHandler`. Obě se starají o synchronizaci se serverem. Rozdíl se nachází ve způsobu synchronizace. `WorldObjectManager` se soustředí na lokální, uživatelem způsobené akce, které odesílá na server. Komunikace směrem ze serveru je hlídána uvnitř `ServerEventsHandler`. Lokální úložiště je také rozdělené na dvě třídy. `WorldObjectStorage` udržuje informaci o objektech, které přišly ze serveru nebo na něj byly nahrány. `PrefabStorage` je pomocná třída, která obsahuje připravené vzory pro vytváření nových Unity objektů ve scéně. Architektura připojení společně se vztahy mezi třídami je k vidění na obr. 4.12.

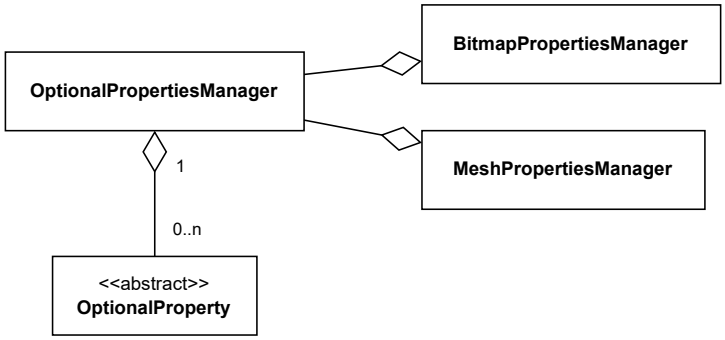
Vlastnosti odlišných typů objektů se zpracovávají různě. Každý typ poslaný ze serveru do Unity musí být schopný převést své vlastnosti do objektů ve scéně. K tomu slouží `BitmapPropertiesManager` a `MeshPropertiesManager`. Kromě pevně daných vlastností, které byly popsány dříve, je možné k serializaci přidat i volitelné vlastnosti. Pokud by uživatel chtěl například k meshi přibalit animace a také je poslat na server, může k `MeshPropertiesManager` přidat `OptionalPropertyManager` s konkrétní implementací `OptionalProperty`, která animace převede do odesílatelného stavu. Vazby mezi správci vlastností jsou uvedeny na obr. 4.13.



Obrázek 4.12: Architektura připojení a správy objektů v Unity knihovně

Z logické architektury je patrné, že bude nutné načítat data ze souborů, aby měly úlohy s čím pracovat. Hned druhý případ užití v 4.3 hovoří o nahrání objektu. Zpřístupnění načítání souborů ve scéně bude základním kamenem práce se soubory. Třída, která se přesně o to stará se nazývá `FileLoader`. Jedná se o obecný objekt, který umožní načítání libovolného množství souborových formátů. Každý typ formátů má však pro čtení svoji vlastní třídu. Knihovna podporuje načítání meshí a obrázků přes třídy `BitmapFileReader` a `MeshFileReader`.

Knihovna obsahuje také společné části uživatelského rozhraní, které se sdílí napříč úlohami. Budou rozděleny na třídy reprezentující ovládací prvky uživatelského rozhraní a jejich kontrolery.



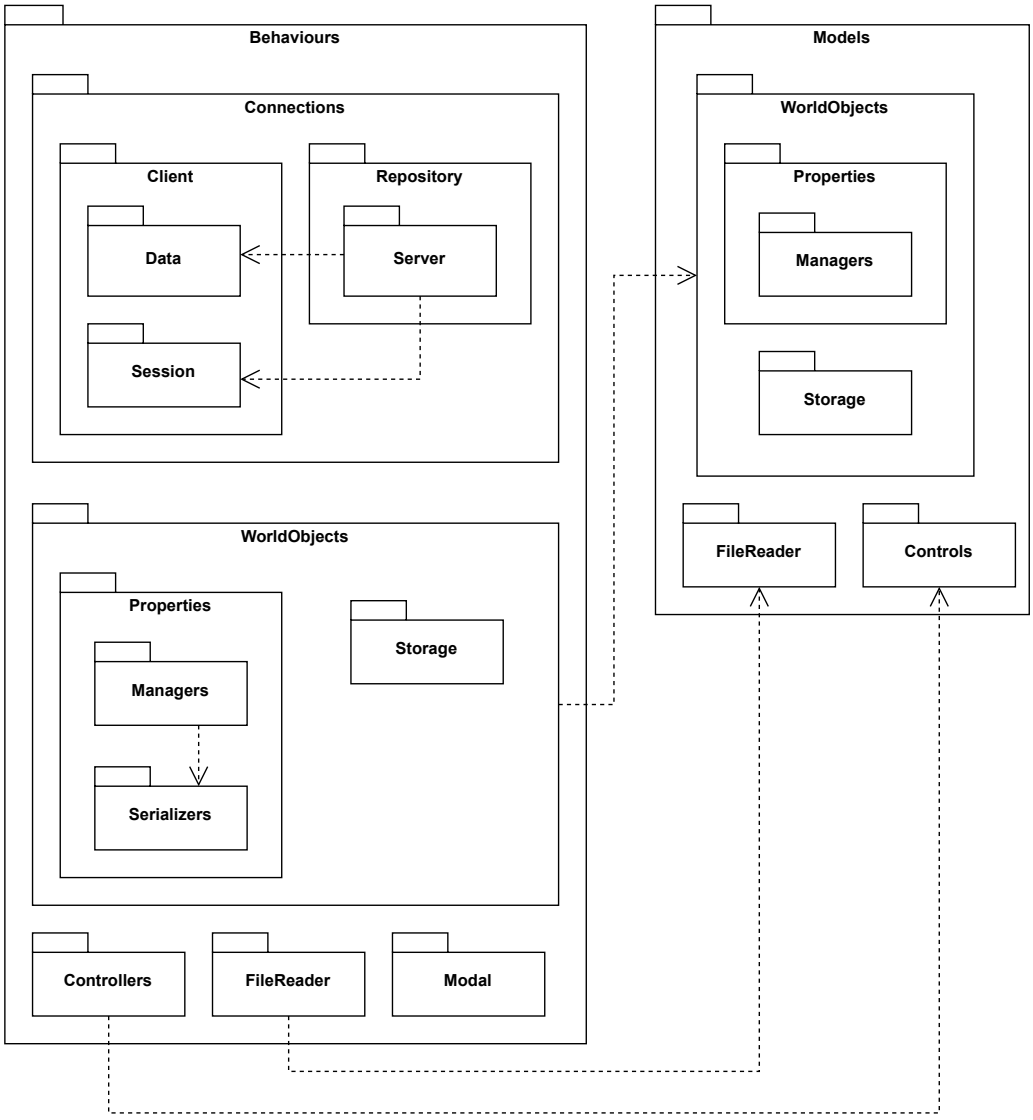
Obrázek 4.13: Architektura manažerů vlastností v Unity knihovně

Tvorba návrhu je v Unity složitější než v klasické .NET aplikaci. Důvodem je velké propojení s Unity frameworkem, které klade zvýšené nároky na architekturu. Ukázkou budiž neschopnost referencovat rozhraní mezi objekty uvnitř Unity scény. Jedním z častých řešení, jak tomuto problému předejít, je co nejvíce oddělit objekty od Unity frameworku. I ve vztahu k jednotkovému testování je tento způsob v Unity hojně používaný. Cílem je navrhnout architekturu nezávisle na Unity a následně pro všechny objekty, s nimiž musí Unity pracovat, vytvořit mezivrstvu pomocí návrhového vzoru adaptér nebo proxy.

Knihovna se tedy rozdělí na dva hlavní balíky, v nichž v jednom budou klasické objekty (Models) a v druhém vrstva přístupná pro Unity (Behaviours). Vnitřní struktura těchto dvou balíků bude z velké části podobná. Správa objektů, úložiště a správa vlastností se umístí do balíku WorldObjects. Připojení bude mít stejnou strukturu jako v .NET knihovně. Další balíky spojují uživatelské rozhraní a načítání souborů. Celá struktura knihovny je dostupná na obr. 4.14.

Unity knihovna je rozdělena do sedmi projektů

- ZCU.TechnologyLab.Common.Unity - obsahuje obecný zdrojový kód (balíky Connections a WorldObjects)
- ZCU.TechnologyLab.Common.Unity.Editor - uživatelské rozhraní pro komponenty dostupné z inspektoru v Unity Editoru.
- ZCU.TechnologyLab.Common.Unity.UI - obsahuje návrh uživatelského rozhraní a obslužný zdrojový kód (balíky Controllers, FileReader, Model, Controls)
- ZCU.TechnologyLab.Common.Unity.Tests - jednotkové testy pro zdrojový kód z ZCU.TechnologyLab.Common.Unity
- ZCU.TechnologyLab.Common.Unity.Editor.Tests - jednotkové testy pro zdrojový kód z ZCU.TechnologyLab.Common.Unity.Editor



Obrázek 4.14: Balíky v Unity knihovně

- ZCU.TechnologyLab.Common.Unity.Tests.Library - knihovna pro pomocné třídy k testování
- ZCU.TechnologyLab.Common.Unity.IntegrationTests - integrační testy

V ZCU.TechnologyLab.Common.Unity.UI se využívají knihovny pro načítání souborů z adresáře. Knihovna pro otevření dialogu na výběr souboru z adresáře Runtime File Browser [20] a knihovna na načítání meshí Runtime OBJ Importer [15].

Unity není thread-safe. Přístup ke všem Unity objektům musí být prováděn na hlavním vláknu. Unity knihovna však staví svojí funkcionalitu na .NET knihovně, jejíž implementace připojení může využívat jiná vlákna. Pokud dojde ke střetu vláken, bude nutné odeslat problematické volání na hlavní vlákno a provést ho až tam.

4.2.5 Klienti

Klienti jsou implementováni v herním enginu Unity ve verzi 2022.3. Programování bude probíhat v jazyce C#. Klienti jsou si architektonicky velmi blízcí, proto budou všichni popsáni společně. Klienti jsou postaveni na architektuře MVP (Model-View-Presenter). Od toho se odvíjí návrh všech klientských aplikací.

Logická třída uživatelského rozhraní se rozdělí na prvky uživatelského rozhraní a jejich kontrolery. Klienti budou vytvářet vlastní třídy pro reprezentaci panelů a jiných prvků uživatelského rozhraní. Z kontrolerů pak bude probíhat veškeré ovládání zbytku aplikace.

V logické architektuře je zmíněno o nějakém způsobu manipulace s world objekty, kterou obě úlohy vyžadují. V případě správy serveru se provádí manipulace s vybraným, zvýrazněným objektem. Tento úkon se postupně rozdělí na jednotlivé části. Zvýraznění se provádí třídou `HighlightBox`. Výběr objektu umožňuje třída zvaná `ObjectMouseSelection`. Změnu pozice, rotace a měřítka zajišťuje `ObjectMouseManipulation`. Jak názvy napovídají, manipulace se provádí myší. V haptické úloze musí být většina manipulace svázána s haptickým zařízením. Bylo by nepraktické pokaždé pouštět haptické pero a manipulovat s objektem pomocí myši nebo klávesnice. Mediátor `ObjectSelection` rozesílá zprávu o vybrání world objektu mezi výše popsané třídy. Správa serveru podporuje kromě manipulace s objektem také pohyb s kamerou. Převedení tohoto logického modelu do vývojové architektury je jednoduché. Je plně definován třídou `CameraMovement`.

Jelikož se jedná o Unity aplikace, je zachována podobná struktura balíků, která rozděluje třídy na Models a Behaviours. Stejně jako v případě Unity knihovny. Je snaha o to, aby balíky zhruba kopírovaly architekturu. Balík UI odpovídá vrstvám View a Presenter. V podbalíku Controls jsou obsaženy speciálně vytvořené panely a jiné prvky uživatelského rozhraní. Třídy popsané v předchozím odstavci

zhruba kopírují výsledné balíky. Manipulation je spojením tříd `HighlightBox`, `ObjectMouseManipulation` a `ObjectMouseSelection`. Pohyb s kamerou žádný svůj speciální balík nemá.

Úlohy se rozdělí do tří projektů

- ZCU.TechnologyLab.ServerManager - základní i navazující úloha na správu projektů,
- ZCU.TechnologyLab.ScriptableHaptic - základní i navazující úloha na haptické zařízení.

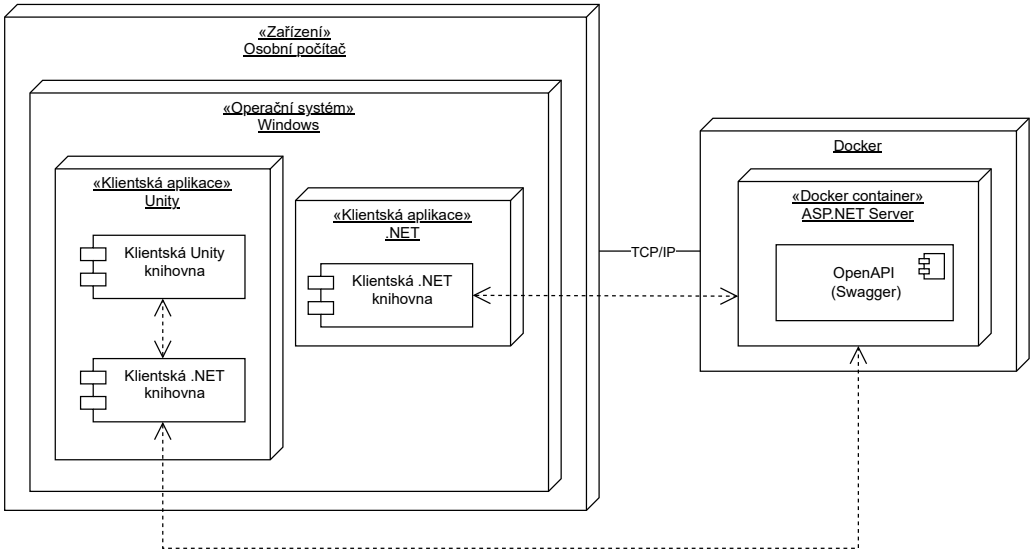
Obě aplikace staví na ZCU.TechnologyLab.Common.Unity knihovně. Sdílí s ní stejné knihovny třetí party. Haptické aplikace používá pro ovládání haptických zařízení, simulaci fyzikálních vlastností a jevů, manipulaci objektů haptickým perem knihovnu Haptics Direct for Unity V1. Skriptování za běhu zajišťuje knihovna ZCU.PythonExecutionLibrary [Kön23]. Může se hodit také knihovna na rozšíření funkcionality uživatelského rozhraní UI Toolkit pojmenovaná UIElements [You21].

4.3 Fyzická architektura

Při nasazení je myšleno primárně na jednoduchost použití. Všechny závislosti by měly být vyřešené automaticky při instalaci bez zásahů správce. Z toho důvodu je server vyvíjen jako Docker aplikace.

Docker je vývojová platforma založená na kontejnerizaci. Kontejner [21c] si lze představit jako proces izolovaný od zbytku systému. Oddělení aplikace od konkrétního prostředí zajišťuje multiplatformitu a usnadňuje kompatibilitu. Toto je zásadní výhoda, kterou kontejnery do vývoje přináší. Obrazy aplikací (docker images) pro kontejnery obsahují všechno potřebné pro běh aplikace včetně knihoven, systémových nastavení a dalších. Nejedná se však o virtualizaci. Využití kontejnerů nevyžaduje takové množství zdrojů, jelikož kontejnery nespouští vlastní operační systém, ale fungují na stejném kernelu, na jehož prostředí běží celý systém.

Sestavení obrazů aplikací se definuje pomocí takzvaných dockerfilů. Obsahují příkazy, které popisují způsob sestavení a popis vlastností prostředí, ve kterém má být aplikace spuštěna. Nad dockerfilem existuje ještě funkce zvaná docker compose [23e]. Primárně slouží pro případy, kdy se spouští aplikace s větším množstvím kontejnerů a je nezbytné je umístit do jednoho společného prostředí. Typickým příkladem by mohl být server běžící v jednom kontejneru a databáze ve druhém, které potřebují komunikovat po stejné síti. Konfigurace společného prostředí je důležitá z hlediska spolupráce a kompatibility kontejnerů. Lze například nastavit čísla portů pro komunikaci a jiné užitečné vlastnosti.



Obrázek 4.15: Nasazení

Docker compose byl zvolen jako způsob, kterým serverovou aplikaci do Dockeru nainstalovat. Hlavním důvodem je výše zmíněné nastavení portů pro internetovou komunikaci mimo kontejnery. Přiřazení pevného portu umožní snazší propojení s klientskými aplikacemi.

Obě klientské aplikace jsou vytvořené v Unity, byť multiplatformní vývojové prostředí, tak jsou klientské aplikace sestaveny pouze na platformu Windows. Cílová architektura musí být 64 bitová. Distribuce budou zajištěny pomocí ZIP archivů se spustitelným EXE souborem. Diagram nasazení všech podpořených aplikací je k dispozici na obr. 4.15.

Implementace serveru

5

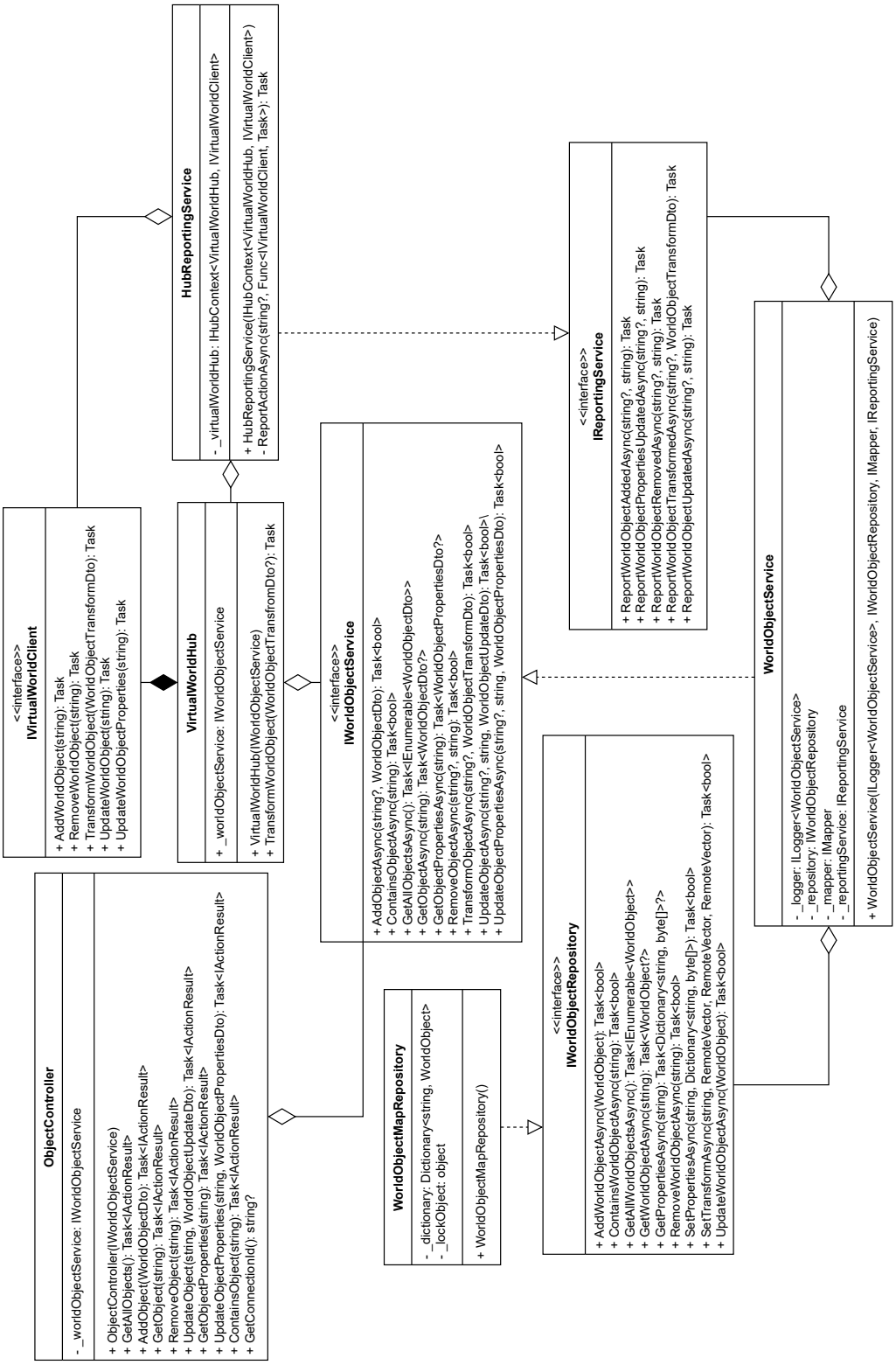
V kapitole 4.2.1 byl představen návrh architektury serveru a jeho konkrétní podoba je k vidění na obr. 4.6. Finální implementace vychází z této architektury. Na obr. 5.1 jsou k vidění implementované třídy s jejich metodami a atributy. Jedná se o zjednodušený diagram. V parametrech metod se vyskytují objekty, které v diagramu neexistují. Bylo tak učiněno z důvodu přehlednosti. Chybějící objekty jsou uvedeny na samostatném diagramu na obr. 5.2. V této kapitole bude implementace jednotlivých částí diagramů jedna po druhé podrobněji popsána.

5.1 Entity

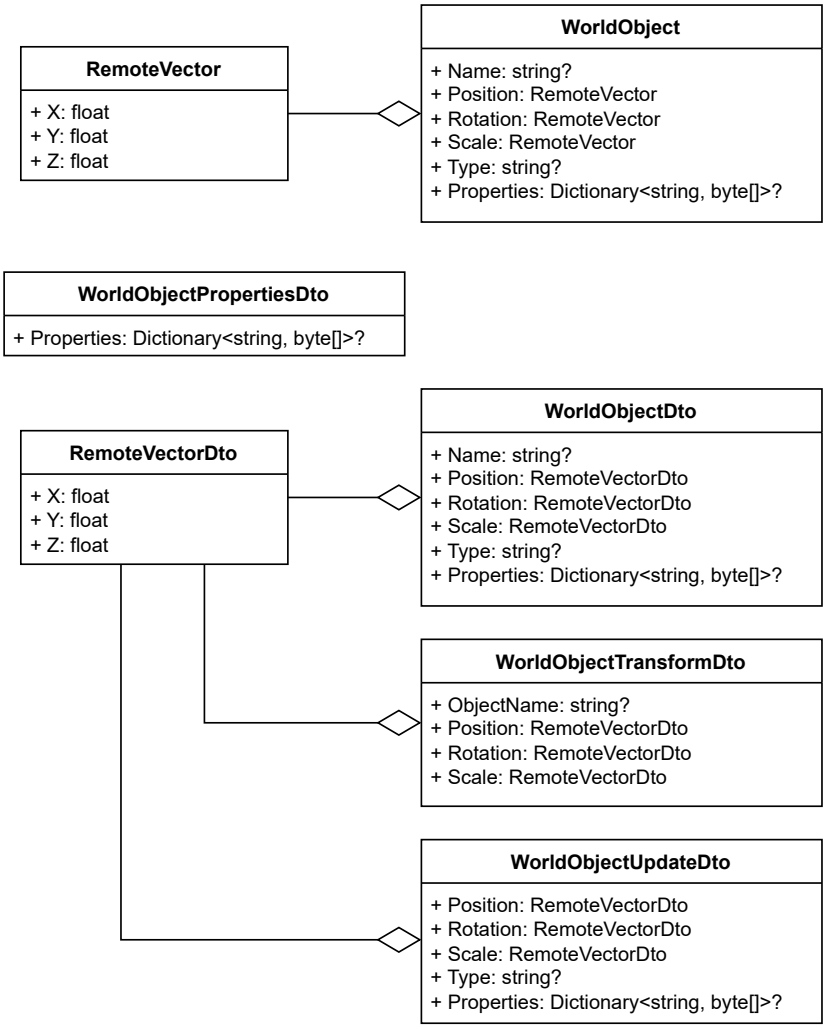
Entity pro reprezentaci objektů na serveru se rozdělují do dvou kategorií. Modely představují objekty uvnitř serverové aplikace, zatímco objekty pro přenos dat (data transfer objects), zkráceně DTO, se uplatňují v komunikaci se serverem. Oddělení modelů a DTO je běžnou praxí při tvorbě datové vrstvy. Modely a DTO by na sobě měly být nezávislé. Modely bývají stabilnější částí serveru. Především v případě napojení na externí databázi. Ke změnám modelů dochází jen zřídka, aby nebylo nutné přetvářet tabulky v databázi. Na druhou stranu komunikační rozhraní se může měnit relativně často. Nebývá výjimkou, že se postupem času rozšiřují schopnosti serveru nebo se mění existující dotazy. Pokud by modely i DTO byly tvořeny totožnými třídami, pak by každá takováto změna vyústila v úpravu databáze a možnou ztrátu dat. Aktuálně sice server databázi pro ukládání dat nepoužívá, ale je pro jistotu myšleno na zadní vrátka.

5.1.1 Modely

V serveru se nepoužívá Entity Framework, takže vlastnosti modelů nejsou nijak omezovány. Z čehož však plyne, že nejsou ani kontrolovány na správné hodnoty. Validace musí být provedena na úrovni DTO.



Obrázek 5.1: Implementace serveru



Obrázek 5.2: Implementace entit na serveru

5.1.1.1 WorldObject

Třída `WorldObject` udržuje všechny informace o world objektu.

Zdrojový kód 5.1: World objekt

```
1  /// <summary>
2  /// Model of a world object.
3  /// </summary>
4  public class WorldObject
5  {
6      /// <summary>
7      /// Name of a world object.
8      /// </summary>
9      /// <remarks>
10     /// Name is needed to communicate with a server.
```

```
11      /// It is used to reference a world object in multiple
        calls.
12      /// </remarks>
13      public string? Name { get; set; }
14
15      /// <summary>
16      /// World space position.
17      /// </summary>
18      public RemoteVector Position { get; set; }
19
20      /// <summary>
21      /// World space rotation as Euler angles in degrees.
22      /// </summary>
23      /// <remarks>
24      /// Euler angles represent a three dimensional rotation
        by performing three separate rotations around individual
        axes.
25      /// </remarks>
26      public RemoteVector Rotation { get; set; }
27
28      /// <summary>
29      /// World space scale.
30      /// </summary>
31      public RemoteVector Scale { get; set; }
32
33      /// <summary>
34      /// Type of a world object.
35      /// </summary>
36      /// <remarks>
37      /// Types depend on clients that connect to a server.
38      /// A client can create objects with a new type and
39      /// it depends on others if they will recognize that type
        .
40      /// Common types are:
41      /// - "Bitmap"
42      /// - "Mesh"
43      /// </remarks>
44      public string? Type { get; set; }
45
46      /// <summary>
47      /// Properties of a world object.
48      /// </summary>
49      /// <remarks>
50      /// Properties depend on the <see cref="Type"/> of the
        world object.
51      /// </remarks>
52      public Dictionary<string, byte[]>? Properties { get; set;
        }
```

```
53 }
```

5.1.1.2 RemoteVector

`WorldObject` obsahuje data o pozici, rotaci a měřítku. Všechny se skládají ze tří hodnot v osách x, y, z. Spojením hodnot dohromady vznikne vektor, který je reprezentovaný třídou `RemoteVector` v ukázce 5.2. Díky tomu se třída `WorldObject` stává přehlednější.

Zdrojový kód 5.2: Vektor

```
1  /// <summary>
2  /// Model of a vector.
3  /// </summary>
4  public struct RemoteVector
5  {
6      /// <summary>
7      /// X coordinate.
8      /// </summary>
9      public float X { get; init; }
10
11     /// <summary>
12     /// Y coordinate.
13     /// </summary>
14     public float Y { get; init; }
15
16     /// <summary>
17     /// Z coordinate.
18     /// </summary>
19     public float Z { get; init; }
20 }
```

5.1.2 Objekty pro přenos dat

Základní pravidlo pro tvorbu DTO zní, že pro každý dotaz i odpověď by měl existovat speciální objekt, který je reprezentuje. Sdílení objektů napříč dotazy může přinést problémy při rozšiřování nebo úpravách existujících zpráv. Změna tvaru zprávy by neměla ovlivňovat ostatní. Důvodem pro toto pravidlo je nutnost reprezentovat objekty zároveň na klientovi i serveru. Každá změna ovlivní nejen rozhraní serveru, ale vyžádá si i úpravy klienta. V případě, že klientských aplikací existuje mnoho, může počet potřebných změn výrazně narůst.

5.1.2.1 WorldObjectDto

Základní třída pro posílání celých world objektů. Všechny property v objektu jsou povinné.

Zdrojový kód 5.3: DTO world objektu

```
1  /// <summary>
2  /// A crate that holds data of an object, which can be placed
       in 3D world, during a transfer.
3  /// </summary>
4  public record WorldObjectDto
5  {
6      /// <summary>
7      /// Name of a world object.
8      /// </summary>
9      /// <remarks>
10     /// Name is needed to communicate with a server.
11     /// It is used to reference a world object in multiple
       calls.
12     /// </remarks>
13     [Required(ErrorMessage = $"{nameof(Name)} cannot be null
       or empty")]
14     public string? Name { get; init; }
15
16     /// <summary>
17     /// World space position.
18     /// </summary>
19     [Required(ErrorMessage = $"{nameof(Position)} cannot be
       null")]
20     public RemoteVectorDto? Position { get; init; }
21
22     /// <summary>
23     /// World space rotation as Euler angles in degrees.
24     /// </summary>
25     /// <remarks>
26     /// Euler angles represent a three dimensional rotation
       by performing three separate rotations around individual
       axes.
27     /// </remarks>
28     [Required(ErrorMessage = $"{nameof(Rotation)} cannot be
       null")]
29     public RemoteVectorDto? Rotation { get; init; }
30
31     /// <summary>
32     /// World space scale.
33     /// </summary>
34     [Required(ErrorMessage = $"{nameof(Scale)} cannot be null
       ")]
```

```
35      public RemoteVectorDto? Scale { get; init; }
36
37      /// <summary>
38      /// Type of a world object.
39      /// </summary>
40      /// <remarks>
41      /// Types depend on clients that connect to a server.
42      /// A client can create objects with a new type and
43      /// it depends on others if they will recognize that type
        .
44      /// Common types are:
45      /// - "Bitmap"
46      /// - "Mesh"
47      /// </remarks>
48      [Required(ErrorMessage = $"{nameof(Type)} cannot be null
        or empty")]
49      public string? Type { get; init; }
50
51      /// <summary>
52      /// Properties of a world object.
53      /// </summary>
54      /// <remarks>
55      /// Properties depend on the <see cref="Type"/> of the
        world object.
56      /// </remarks>
57      [Required(ErrorMessage = $"{nameof(Properties)} cannot be
        null")]
58      public Dictionary<string, byte[]>? Properties { get; init
        ; }
59  }
```

5.1.2.2 WorldObjectUpdateDto

Úprava objektu vyžaduje stejné property jako `WorldObject`. Jedinou výjimkou je `Name`, která je součástí cesty.

5.1.2.3 WorldObjectPropertiesDto

DTO pro odesílání vlastností world objektů obsahuje pouze propertu Properties z world objektu ve zdrojovém kódu 5.3.

5.1.2.4 WorldObjectTransformDto

`WorldObjectTransformDto` je podmnožina `WorldObjectDto` zabývající se transformací world objektu. Kromě pozice, rotace a měřítka obsahuje také jméno objektu. U předešlých tříd nebylo uvedeno jméno objektu, jelikož se posílají přes

kontrolery a dle návrhu v kapitole 4.2.2 bylo jméno uvedeno v cestě REST API. `WorldObjectTransformDto` se odesílá přes SignalR, které takovou možnost nenabízí.

5.2 Kontrolery

Kontrolery se v ASP.NET používají jako přístupové body webového HTTP API. Jejich názvy by měly vždy končit sufixem Controller. Každý kontroler má definovanou společnou část cesty pro všechny podporované dotazy. Zpřesnění cesty dotazu pak definují jednotlivé metody, které dotazy zpracovávají. Logika kontrolerů by měla být velmi jednoduchá. Základním pravidlem pro jejich vytváření je provádět na této úrovni minimum úkonů a delegovat většinu práce na služby v pozadí.

5.2.1 ObjectController

`ObjectController` kopíruje základní „api/objects“ url z návrhu komunikačního API v kapitole 4.2.2. Kontroler se stará o práci s world objekty a jejich vlastnostmi. Všechny metody uvnitř `ObjectController` jsou asynchronní a vracejí generický `Task<T>`. Typ používaný pro odpovědi z webového API, který se v tasku vrací, se nazývá `IActionResult`. Specifikuje například HTTP stavový kód, zprávu pro klienta nebo získaná data ze serveru. U dotazů s vícero různými chybovými stavy by vytvoření `IActionResult` a nastavení návratových kódů mohlo být složitější a kvůli snahám o jednoduchost kontrolerů se dá očekávat, že by bylo vyjmuto a přesunuto do samostatné třídy. V případě `ObjectController` to však není nutné.

5.2.1.1 World objekty

GetAllObjects Metoda dostupná přes GET „api/objects“ endpoint vrací úplně všechny objekty uložené na serveru. Metoda nemá žádný očekávaný chybový stav, proto vrací pouze odpověď OK se stavovým kódem 200. Tělo odpovědi je serializovaný JSON z kolekce `IEnumerable<WorldObjectDto>`.

AddObject Na ukázkovém zdrojovém kódu v příkladu 5.4 je vidět dotaz typu POST, který nahrává objekt na server. Novinkou zde je parametr `[FromBody]`, který z těla HTTP zprávy deserializuje objekt typu `WorldObjectDto`. Jelikož jsou data posílána zvenčí a na server se může připojit téměř kdokoliv s kterýmkoliv klientem, může se stát, že obsah těla nebude odpovídat formátu objektu `WorldObjectDto`. Jelikož je nutné tyto chybové stavy ošetřit a vyskytují se u vícero různých dotazů, byla validace přesunuta do filtru nazývaného `ValidationFilterAttribute`, jeho použití lze vidět na řádce 2.

Úspěšnost přidání objektu závisí na jeho jménu. Jedná se totiž o primární klíč pro uložení world objektů. Kdyby se stalo, že se jméno opakuje a takový world objekt již existuje, pak se vrátí chybový stav Conflict se stavovým kódem 409 a chybovým hlášením „Object already exists“.

Zdrojový kód 5.4: HTTP add dotaz

```
/// <summary>
/// Adds a new object to the virtual world and informs other clients about it.
/// </summary>
/// <param name="worldObjectDto">The object.</param>
/// <returns>A task with a response.</returns>
[HttpPost]
[ServiceFilter(typeof(ValidationFilterAttribute))]
public async Task<IActionResult> AddObject([FromBody] WorldObjectDto worldObjectDto)
{
    var success = await _worldObjectService.AddObjectAsync(GetConnectionId(), worldObjectDto);
    return success ? Ok("Object added") : Conflict("Object already exists");
}
```

GetObject Získat world objekt ze serveru jde dotazem GET „api/objects/{name}“. Metoda, která ho reprezentuje se jmenuje `GetObject`. Na serveru je vyhledán world objekt podle zadaného jména. Když existuje, v odpovědi OK bude serializován `WorldObjectDto`. Může se stát, že objekt se zadaným jménem se na serveru nevyskytuje, což způsobí vrácení odpovědi typu Not Found s chybovým hlášením.

Zdrojový kód 5.5: HTTP get dotaz

```
/// <summary>
/// Gets a world object.
/// </summary>
/// <param name="name">A name of the world object.</param>
/// <returns>A task with a response.</returns>
[HttpGet("{name}")]
public async Task<IActionResult> GetObject(string name)
{
    var worldObjectDto = await _worldObjectService.GetObjectAsync(name);
    return worldObjectDto is not null ? Ok(worldObjectDto) : NotFound($"Object \"{name}\" has not been found");
}
```

ContainsObject Pro kontrolu existence objektu na serveru je dostupný zdrojový kód 5.6. Na řádce 1 se nastavuje typ dotazu a zbylá část cesty. Jediné volání uvnitř metody je do `IWorldObjectService`. V případě kódu v 5.6 lze očekávat pouze dva výsledné stavy. Buď hledaný objekt existuje a ze serveru se vrátí stavový kód 200, jinak známý jako OK, se zprávou "World object exists", nebo hledaný objekt není na serveru dostupný, pak lze očekávat stavový kód 404, neboli Not Found.

Zdrojový kód 5.6: HTTP contains dotaz

```
1  /// <summary>
2  /// Checks whether an objects exists on the server.
3  /// </summary>
4  /// <param name="name">Name of the object.</param>
5  /// <returns>A task with a response.</returns>
6  [HttpGet("contains/{name}")]
7  public async Task<IActionResult> ContainsObject(string name)
8  {
9      var success = await _worldObjectService.
       ContainsObjectAsync(name);
10     return success ? Ok("World object exists") : NotFound($"
       Object \"{name}\" has not been found");
11 }
```

RemoveObject Odstranění objektu ze serveru je v podstatě totožné jako metoda `ContainsObject`. Jediný významnější rozdíl je volání metody `RemoveObjectAsync` z `IWorldObjectService`.

UpdateObject Úprava objektu nepřináší z hlediska implementace nic nového. Kombinuje všechny přístupy popsané výše.

5.2.1.2 Vlastnosti world objektů

GetObjectProperties Vrácení vlastností world objektu se provádí přes `IWorldObjectService`, stejně jako dotazy směřované na world objekty. Může se stát, že server neobsahuje objekt se zadaným jménem a vrátí odpověď Not Found se zněním chybové zprávy.

UpdateObjectProperties Úprava vlastností se chová totožně jako úprava world objektů.

5.2.2 Validace dat

Filtry v ASP.NET umožňují provádět kód těsně před nebo po vykonání metody v kontroleru. První ze jmenovaných případů je pro validaci ideální, jelikož se filtr

provádí až po deserializaci parametrů. Ukázka validačního filtru je v kódu 5.7. Kontrolou property `ModelState.IsValid` se dá zjistit výsledek deserializace. Když došlo k selhání vrátí se stavový kód 422, čili Unprocessable Entity.

Zdrojový kód 5.7: Validace dat u kontroleru

```
/// <summary>
/// Called before the action executes, after model binding is complete.
/// </summary>
/// <param name="context">The <see cref="ActionExecutingContext"/>.</param>
public void OnActionExecuting(ActionExecutingContext context)
{
    if (!context.ModelState.IsValid)
    {
        context.Result = new UnprocessableEntityObjectResult(context.ModelState);
    }
}
```

Validita `ModelState` se určuje z definice typu deserializovaných objektů. V případě přidání objektů ve zdrojovém kódu 5.4 je typ `WorldObjectDto`. Na ukázce 5.8 je k nahlédnutí jeho properta `Name`. Její součástí je atribut `Required`, ten definuje, že properta musí být v deserializovaných datech dostupná a nemůže chybět. Pokud by chyběla, deserializace selže a `ModelState` v kódu 5.7 nebude validní. Dostupných atributů, kterými lze deserializaci dále modifikovat, je více například délka textu, rozsah číselných hodnot a další.

Zdrojový kód 5.8: Properta WorldObjectDto

```
/// <summary>
/// Name of a world object.
/// </summary>
/// <remarks>
/// Name is needed to communicate with a server.
/// It is used to reference a world object in multiple calls.
/// </remarks>
[Required(ErrorMessage = $"{nameof(Name)} cannot be null or empty")]
public string? Name { get; init; }
```

5.2.3 Zachytávání výjimek

V předchozích částech bylo popsáno, jak lze klienta informovat o očekávaných chybových stavech a kontrolovat přicházející data. Při zpracování na serverovém backendu se však může přihodit nějaká neočekávaná situace a nastane chybový stav,

který není registrován. V takovém případě by kód vyhodil výjimku. Ani v jedné z předcházejících ukázek zdrojového kódu, však není ani zmínka o zachytávání výjimek a přeposílání chybových hlášení na klienta. Jedná se o kód, který by bylo nutné provádět kompletně ve všech dotazech napříč všemi kontrolery. Z toho důvodu je funkční kód separován do samostatné třídy. V ASP.NET existují konstrukce zvané middleware. Jejich účelem je pracovat s dotazy, které na server přicházejí. Než dotaz doputuje do kontroleru může procházet řadou middlewarů. Příkladem může být automatické převedení z HTTP protokolu na HTTPS, autentizace a autorizace, směrování (rozhoduje, která metoda v kontroleru dotaz dostane) a tak dále.

Jak bylo popsáno, dotaz se middlewary pouze provolává. Funguje pro to delegát `RequestDelegate`, který HTTP dotaz pošle do dalšího middlewaru v řadě. Jediné užitečné, co tedy lze v middlewaru dělat, je předzpracování nebo reakce na provedený dotaz. Nelze přímo ovlivnit jeho chování, které se provádí v kontroleru. V kusu kódu 5.9 o zachytávání výjimek metoda `InvokeAsync` pouze zavolá delegáta `_request` a obalí ho try-catch blokem. Pokud výjimka nastane, do odpovědi se nastaví stavový kód 500 a chybové hlášení.

Zdrojový kód 5.9: Zachytávání výjimek u kontroleru

```
/// <summary>
/// Catches exceptions thrown in a HTTP request.
/// This method must be implemented for middleware to monitor HTTP requests.
/// </summary>
/// <param name="httpContext">Encapsulation of HTTP request.</param>
/// <returns>A task.</returns>
public async Task InvokeAsync(HttpContext httpContext)
{
    try
    {
        await _request(httpContext);
    }
    catch (Exception ex)
    {
        _logger.LogError("Something went wrong: {Exception}", ex);
        await HandleExceptionAsync(httpContext);
    }
}

/// <summary>
/// Writes response to HTTP request about a thrown exception.
/// </summary>
/// <param name="context">Encapsulation of HTTP request</param>
```

```
24 /// <returns>A task.</returns>
25 private static async Task HandleExceptionAsync(HttpContext
       context)
26 {
27     context.Response.ContentType = "application/json";
28     context.Response.StatusCode = (int)HttpStatusCode.
       InternalServerError;
29
30     await context.Response.WriteAsync(new ErrorDetails()
31     {
32         StatusCode = context.Response.StatusCode,
33         Message = "Internal Server Error"
34     }.ToString());
35 }
```

5.3 Huby

Nové huby se vytvářejí děděním od třídy `Hub`. Existuje také generická verze této třídy. Je příjemnější na použití, protože umožňuje zadat rozhraní do generického parametru, pomocí kterého se popisují metody klienta, k nimž má server přístup. Tento způsob se používá i uvnitř zdrojového kódu této práce. Je praktičtější, protože není nutné při každém volání uvádět metody pomocí řetězce se jménem, což může být náchylnější na chyby.

5.3.1 VirtualWorldHub

Klient pro `VirtualWorldHub` je popsán rozhraním `IVirtualWorldClient` ve zdrojovém kódu 5.10.

Zdrojový kód 5.10: Hub klient

```
1  /// <summary>
2  /// Interface describes virtualWorld endpoint callbacks.
3  /// </summary>
4  public interface IVirtualWorldClient
5  {
6      /// <summary>
7      /// Informs a client about added world object.
8      /// </summary>
9      /// <param name="name">A name of the added world object
       .</param>
10     /// <returns>A task.</returns>
11     Task AddWorldObject(string name);
12
13     /// <summary>
14     /// Informs a client about updated world object.
```

```
15      /// </summary>
16      /// <param name="name">A name of the updated world object
        .</param>
17      /// <returns>A task.</returns>
18      Task UpdateWorldObject(string name);
19
20      /// <summary>
21      /// Informs a client about removed world object.
22      /// </summary>
23      /// <param name="name">A name of the removed world object
        .</param>
24      /// <returns>A task.</returns>
25      Task RemoveWorldObject(string name);
26
27      /// <summary>
28      /// Informs a client about changed properties of a world
        object.
29      /// </summary>
30      /// <param name="name">A name of the object.</param>
31      /// <returns>A task.</returns>
32      Task UpdateWorldObjectProperties(string name);
33
34      /// <summary>
35      /// Informs a client about transformed world object.
36      /// </summary>
37      /// <param name="transform">Transformation of a world
        object.</param>
38      /// <returns></returns>
39      Task TransformWorldObject(WorldObjectTransformDto
        transform);
40  }
41
42  /// <summary>
43  /// The VirtualWorldHub class handles requests on a
        virtualWorld endpoint.
44  /// </summary>
45  public class VirtualWorldHub : Hub<IVirtualWorldClient> {...}
```

TransformWorldObject Samotný hub vystavuje pouze jednu metodu pro přístup z klientské strany. Metoda v 5.11 umožňuje transformovat objekt v reálném čase. Hlavní činnost je prováděna uvnitř `IWorldObjectService`. Metody uvnitř hubu by měly být tvořeny se stejným cílem jednoduchosti, jako v případě kontrolerů. Transformace je uložena v objektu `WorldObjectTransformDto`.

5.3.2 Validace dat

U SignalR nefunguje validace dat přes `ModelState` jako u kontrolerů. Z toho důvodu se validují data uvnitř metody v hubu, která vykonává požadovanou akci. Existuje ekvivalent filtrů, jenž je možné použít a vyhnout se dlouhé hub metodě. Aplikují se však na celý hub. V případě různých parametrů s různými typy, může být cílená validace složitá. Bylo by nutné vytvořit složitou logiku pro validaci. V kódu 5.11 jsou validovány všechny chybové případy, které mohou u vstupního parametru nastat. Jak si lze povšimnout, tak všechny chybové stavy vracejí výjimku `HubException`. Jedná se o způsob, jak odeslat na klienta informaci o chybě, která na serveru při zpracování dotazu nastala.

Zdrojový kód 5.11: Metoda hubu pro transformaci

```
1  /// <summary>
2  /// Transforms a world object and informs other clients about
       it.
3  /// </summary>
4  /// <param name="transform">A transformation of a world
       object.</param>
5  /// <returns>A task.</returns>
6  public async Task TransformWorldObject(
       WorldObjectTransformDto? transform)
7  {
8      if (transform == null)
9      {
10         throw new HubException("Transform cannot be null");
11     }
12
13     if (string.IsNullOrEmpty(transform.ObjectName))
14     {
15         throw new HubException("ObjectName cannot be null or
       empty");
16     }
17
18     if (transform.Position == null)
19     {
20         throw new HubException("Position cannot be null");
21     }
22
23     if (transform.Rotation == null)
24     {
25         throw new HubException("Rotation cannot be null");
26     }
27
28     if (transform.Scale == null)
29     {
30         throw new HubException("Scale cannot be null");
```

```
31      }
32
33      if (!await _worldObjectService.TransformObjectAsync(
    Context.ConnectionId, transform))
34      {
35          throw new HubException($"Object␣\"{transform.
    ObjectName}\"␣is␣not␣present␣on␣the␣server.");
36      }
37  }
```

5.3.3 Zachytávání výjimek

Ohledně téma zpracování výjimek ještě zbývá zachytit neočekávané situace při zpracování dotazu. Existuje filtr vytvořený přesně k tomuto účelu. Způsob zpracování je zde teoreticky totožný jako u kontroleru ve zdrojovém kódu 5.9. Dotaz se postupně filtry provolává a odchytávají se výjimky. Filtr zachytává také `HubException`. Aby bylo možné `HubException` zpropagovat na klienta, používá se pouze `throw;` pro přeposlání výjimky. Všechny ostatní se logují a na klienta se odešle nově vytvořená `HubException` obsahující zprávu o interní serverové chybě.

Zdrojový kód 5.12: Zachytávání výjimek u hubu

```
1  // <summary>
2  /// Allows handling of all Hub method invocations.
3  /// </summary>
4  /// <param name="invocationContext">The context for the
       method invocation that holds all the important information
        about the invoke.</param>
5  /// <param name="next">The next filter to run, and for the
       final one, the Hub invocation.</param>
6  /// <returns>Returns the result of the Hub method invoke.</
       returns>
7  public async ValueTask<object?> InvokeMethodAsync(
       HubInvocationContext invocationContext, Func<
       HubInvocationContext, ValueTask<object?>> next)
8  {
9      try
10     {
11         return await next(invocationContext);
12     }
13     catch (HubException)
14     {
15         // Pass HubException to a client
16         throw;
17     }
18     catch (Exception ex)
19     {
```

```
20          _logger.LogError("Exception␣calling␣'{MethodName}':␣{
    Exception}", invocationContext.HubMethodName, ex);
21          throw new HubException("Internal␣Server␣Error");
22      }
23  }
```

5.4 Služby

Služby, neboli services, se starají o provádění hlavních serverových úkonů. Zpracovávají, transformují a ukládají data.

5.4.1 Správa objektů

Centrální služba z diagramu tříd 5.1 se nazývá `WorldObjectService`. Implementuje rozhraní `IWorldObjectService`, které již bylo několikrát zmíněno výše. Většina metod v `WorldObjectService` si je navzájem velmi podobná. Všechny služby se logují, aby bylo možné lehce zkontrolovat příchozí komunikaci na serveru.

Na ukázce 5.13 je k vidění metoda pro přidání objektu na server. Na řádku 5 se převádí `WorldObjectDto` na `WorldObject`. Jak bylo zmíněno v kapitole o architektuře, je na převod mezi objekty využita knihovna AutoMapper. Konfigurace mapování objektů se provádí ve třídě `MappingProfile`. Následně se zavolá funkce z úložiště `IWorldObjectRepository` a objekt se přidá. Když je přidání úspěšné, reportuje se změna na připojené klienty pomocí služby `IReportingService`.

Ostatní metody v `WorldObjectService` nutně nepoužívají stejné funkce, ale typ jednotlivých volání je stejný. Ve všech případech dojde k nějakému dotazu do úložiště a pokud se provedla na serveru nějaká změna, tak se odešle na všechny klienty, kteří poslouchají. Když žádná změna nenastane, například v případě, že klient chce pouze vrátit nějaký objekt, řádky osm až jedenáct se ve zdrojovém kódu vynechají.

Zdrojový kód 5.13: Přidání objektu v service

```
1 /// <summary>
2 /// Remaps world object dto and adds it to a repository.
3 /// When the addition is successful the change is reported to
      listeners.
4 /// </summary>
5 /// <param name="sourceId">Id of a client that added the
    object.</param>
6 /// <param name="worldObjectDto">Added world object dto.</
    param>
7 /// <returns>A task that returns true - if the addition was
    successful, false otherwise.</returns>
```

```
8  public async Task<bool> AddObjectAsync(string? sourceId,
       WorldObjectDto worldObjectDto)
9  {
10     _logger.LogInformation("Adding␣object␣{Name}␣to␣the␣
       server", worldObjectDto.Name);
11
12     var worldObject = _mapper.Map<WorldObjectDto, WorldObject
       >(worldObjectDto);
13     var success = await _repository.AddWorldObjectAsync(
       worldObject);
14
15     if (success)
16     {
17         await _reportingService.ReportWorldObjectAddedAsync(
       sourceId, worldObject.Name!);
18     }
19
20     return success;
21 }
```

5.4.2 Reportování změn

Služba `IReportingService` je závislá na SignalR hubu, což je viditelné také z diagramu 5.1. Již bylo zmíněno, že se stará o přeposílání změn na připojené klienty. Jedna z možností by byla změnu posílat na všechny připojené klienty. Toto řešení by znamenalo, že zdrojový klient, který dotaz zaslal, musí rozpoznat, že dostal zprávu o změně, kterou sám udělal a ignorovat ji. Vyvodit pouze z dat, kdo zprávu poslal může být problematické. Druhou možností by bylo již na serveru odesílat změnu jen ostatním klientům a neposílat zprávu zpět na zdroj. Výhodou by bylo také snížení objemu komunikace. ASP.NET však neposkytuje vazbu, která by umožnila z dotazu na HTTP API vyvodit klienta, který dotaz poslal a identifikovat jeho spojení přes SignalR. SignalR používá pro odlišení klientských relací ConnectionId. Jedná se o jedinečný identifikátor dostupný na klientském i serverovém konci spojení. Když je nutné poslat cílenou zprávu ze serveru na klienta, používá se právě ConnectionId. Je možné také posílat zprávu všem kromě zadaného ConnectionId.

Bylo nutné spojit dohromady tyto dva světy. Do hlavičky HTTP dotazu se na klientovi přidá ConnectionId. REST API musí být bezstavové, takže je nutné posílat ConnectionId při každém dotazu na server. ConnectionId se z hlavičky získá v kontroleru. Metoda už byla zmíněna na řádce 5 v kódu 5.4. `GetConnectionId` vrátí ConnectionId, které se následně přepošle až do `HubReportingService`. Kód 5.14 představuje způsob oznamování změn. V případě, že klient není připojený přes SignalR nebo mu nevadí, že se mu zpráva přepošle, může nechat hlavičku prázdnou.

Zdrojový kód 5.14: Reportování změny

```
1  /// <summary>
2  /// Reports, that a new world object has been added to the
       server, to all listener except the source.
3  /// </summary>
4  /// <remarks>
5  /// When <paramref name="sourceId"/> is null or empty, the
       message is reported to all listeners.
6  /// </remarks>
7  /// <param name="sourceId">Id of the source that triggered
       the event.</param>
8  /// <param name="worldObjectName">Name of the added world
       object.</param>
9  /// <returns>A task.</returns>
10 public Task ReportWorldObjectAddedAsync(string? sourceId,
       string worldObjectName)
11 {
12     return ReportActionAsync(sourceId, client => client.
       AddWorldObject(worldObjectName));
13 }
14
15 private Task ReportActionAsync(string? sourceId, Func<
       IVirtualWorldClient, Task> action)
16 {
17     var virtualWorldClient = string.IsNullOrEmpty(sourceId) ?
        _virtualWorldHub.Clients.All : _virtualWorldHub.Clients.
       AllExcept(sourceId);
18     return action(virtualWorldClient);
19 }
```

5.5 Úložiště

Server nevyžaduje perzistentní úložiště, protože se předpokládá, že bude ukládat data jenom v průběhu běhu klientské aplikace. Doba, po kterou aplikace poběží, je mnohem kratší než doba běhu serveru. Data se ukládají pouze do operační paměti. Rozhraní úložiště je popsáno v `IWorldObjectRepository`.

5.5.1 Slovník

Implementace `WorldObjectMapRepository` využívá pro uložení dat slovník, jehož klíčem je jméno nahraného `WorldObject`. Metody třídy `WorldObjectMapRepository` v zásadě pouze obalují metody slovníku. Všechny zprostředkované funkce zamykají přístup vláken ke slovníku, jak lze sledovat na ukázce 5.15.

Zdrojový kód 5.15: Přidání objektu v repository

```
/// <summary>
/// Lock object for synchronization.
/// </summary>
private readonly object _lockObject = new();

/// <summary>
/// Adds world object to a repository.
/// </summary>
/// <param name="worldObject">The world object.</param>
/// <returns>True - the world object was added to the
    repository, false otherwise (object is already in a
    repository).</returns>
public Task<bool> AddWorldObjectAsync(WorldObject worldObject
    )
{
    lock (_lockObject)
    {
        return Task.FromResult(_dictionary.TryAdd(worldObject
    .Name!, worldObject));
    }
}
```

5.6 Testování

Server obsahuje celkem 273 testů. Integrační testy vynechané z důvodu nízkého class coupling, jak ukazuje obr. 4.6. Pokrytí testy je 100%. Uživatelské explorativní testování proběhlo v technických laboratořích na Západočeské univerzitě v Plzni a Gymnáziu Sokolov. Jejich výsledky však nebyly nijak kvantifikovány.

Implementace knihoven

6

6.1 .NET knihovna

6.1.1 Připojení

`IDataClient` poskytuje rozhraní typické pro REST API. Umožňuje volat metody typu GET, POST, PUT, DELETE. Rozhraní je implementováno třídou `RestDataClient`. K odeslání dat na server využívá `HttpClient` z System.Net.Http. Adresa serveru se nastavuje v konstruktoru `RestDataClient`. Je možné ji kdykoliv změnit. Při dotazování na server se proto očekává, že zadaná cesta v metodách `DeleteAsync`, `GetAsync`, `PostAsync` a `PutAsync` bude obsahovat jen cestu konkrétního serverového endpointu. Všechny objekty posílané přes tyto metody jsou převáděni z/do JSONu statickou třídou `JsonSerializer` z jmenného prostoru System.Text.Json.

`ISessionClient` obsahuje eventy

- `Disconnected`,
- `Reconnecting`,
- `Reconnected`,

které reportují o změnách stavu aktivní relace. Kromě eventů je možné použít propertu `State`, která vrací aktuální stav. `SignalRSession` používá na pozadí `HubConnection` z SignalR knihovny. Relaci je nutné před spuštěním inicializovat metodou `InitializeAsync`. Metodu lze zavolat pouze jednou. V průběhu volání se relaci nastaví adresa serveru a hub, na který se má připojit. Po inicializaci již nejde cíl připojení upravit a je nutné v takovém případě vytvořit novou instanci. Po inicializaci už je možné relaci libovolně startovat a stopovat. Chování je řízeno propertou `State`, tak aby se relace nedostala do nekonzistentního stavu. Relace se může přerušit vlivem neočekávaných okolností, například vypnutí serveru, přerušení spojení

mezi klientem a serverem atd. V takovém případě se změní stav připojení automaticky.

`ServerDataAdapter` je postavený na `IDataClient` a poskytuje připravené dotazy na server popsané v této práci. Třída obsahuje předem připravené cesty k serverovému kontroleru z kapitoly 5.2. Uživatel knihovny tak nemusí znát konkrétní cesty.

`ServerSessionAdapter` využívá `ISessionClient`, u kterého registruje akce na metody, o nichž je známo, že mohou být volané ze serveru. SignalR umožňuje registrovat libovolné množství akcí na jednu metodu. Potíž je v tom, že při zrušení registrované akce se odstraní úplně všechny. Není možné u metody odstranit pouze jednu zaregistrovanou akci. Může se však jednat o důležitou funkci. K jejímu podpoření obsahuje podle diagramu 6.1 `ServerSessionAdapter` dvojici akcí - privátní a veřejné.

Do `ISessionClient` se pro každou metodu zaregistruje privátní akce. Privátní akce provolávají veřejné a tím reportují ve veřejném rozhraní příchozí volání metod ze serveru. Privátní akce se do `ISessionClient` nastaví pouze v případě, že někdo přidá první novou veřejnou akci. Není nutné naslouchat příchozím metodám, které nikdo nepotřebuje. Když veřejnou akci uživatel odstraní, s privátními akcemi ani s `ISessionClient` se nic nedělá. Pouze tehdy, když se odstraní poslední veřejná akce, tak se zruší registrace jejího privátního protějšku z `ISessionClient`. Ve zdrojovém kódu 6.1 lze vidět ukázkovou implementaci popsaného problému.

Zdrojový kód 6.1: Duplicita akcí v `ServerSessionAdapter`

```
1  private const string AddWorldObjectMethodName = "
       AddWorldObject";
2  private Action<string>? _worldObjectAdded;
3
4  /// <summary>
5  /// The event triggered when other client adds a world object
       to a repository.
6  /// </summary>
7  /// <remarks>
8  /// Name of the object is passed from the repository.
9  /// </remarks>
10 /// <seealso cref="IAddWorldObject.AddWorldObjectAsync(
       WorldObjectDto)"/>
11 public event Action<string> WorldObjectAdded
12 {
13     add => AddEvent(ref _worldObjectAdded, value,
       AddWorldObjectMethodName, OnWorldObjectAdded);
14     remove => RemoveEvent(ref _worldObjectAdded, value,
       AddWorldObjectMethodName);
15 }
16
```

```
17 private void OnWorldObjectAdded(string objectName)
18 {
19     _logger.LogTrace("New world object {Name} has been added
       on a server.", objectName);
20     _worldObjectAdded?.Invoke(objectName);
21 }
22
23 /// <summary>
24 /// Assigns a new action to a local event.
25 /// When the event is empty it also registers a callback to a
       session which triggers the event.
26 /// </summary>
27 /// <typeparam name="T">Type of arguments of the event.</
       typeparam>
28 /// <param name="localEvent">Local event.</param>
29 /// <param name="newAction">New added action.</param>
30 /// <param name="serverMethod">Name of a method in a server
       API.</param>
31 /// <param name="callback">The callback registered to a
       session client.</param>
32 private void AddEvent<T>(ref Action<T>? localEvent, Action<T>
       newAction, string serverMethod, Action<T> callback)
33 {
34     if (localEvent == null)
35     {
36         _sessionClient.RegisterCallback(serverMethod,
       callback);
37     }
38
39     localEvent += newAction;
40 }
41
42 /// <summary>
43 /// Removes an action from a local event.
44 /// When it removes last action, a callback that triggers the
       event is removed from a session client.
45 /// </summary>
46 /// <typeparam name="T">Type of arguments of the event.</
       typeparam>
47 /// <param name="localEvent">Local event.</param>
48 /// <param name="removedAction">Removed action.</param>
49 /// <param name="serverMethod">Name of a method in a server
       API.</param>
50 private void RemoveEvent<T>(ref Action<T>? localEvent, Action
       <T> removedAction, string serverMethod)
51 {
52     localEvent -= removedAction;
53     if (localEvent == null)
```

```
54      {
55          _sessionClient.RemoveCallbacks(serverMethod);
56      }
57  }
```

6.1.2 Serializace

Převod celých čísel v `IntSerializer` se provádí pomocí `BitConverter`, jak ukazuje zdrojový kód 6.2. Vrací pole bytů s endianem, který je na dané platformě aktivní. Nepředpokládá se, že by se knihovna používala na jiné architektuře než x64, která používá little-endian. Z toho důvodu jsou konverze mezi endiany ignorovány.

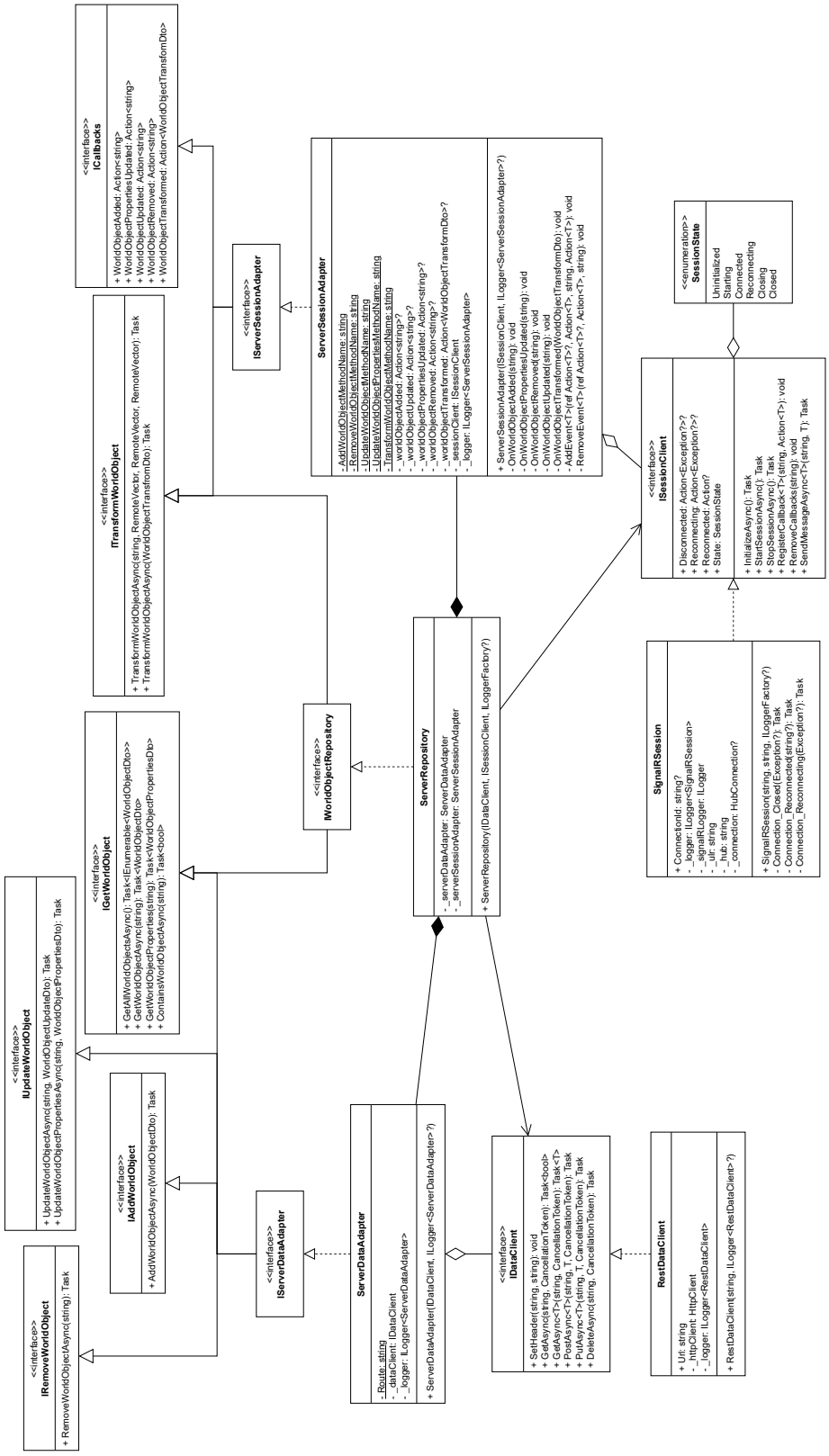
Zdrojový kód 6.2: Převod celých čísel

```
1  /// <summary>
2  /// Deserializes a property from a byte array.
3  /// </summary>
4  /// <param name="property">Byte array property.</param>
5  /// <returns>Deserialized property to a concrete type.</
       returns>
6  public override int Deserialize(byte[] property)
7  {
8      _logger.LogDebug("Deserializing␣property␣{Property}.",
       property);
9      return BitConverter.ToInt32(property, 0);
10 }
11
12 /// <summary>
13 /// Serializes a property to a byte array.
14 /// </summary>
15 /// <param name="property">The property.</param>
16 /// <returns>Byte array representation of a property.</
       returns>
17 public override byte[] Serialize(int property)
18 {
19     _logger.LogDebug("Serializing␣property␣{Property}.",
       property);
20     return BitConverter.GetBytes(property);
21 }
```

`StringSerializer` se stará o převod mezi řetězcem a bytovým polem. Pro podporu většího množství znaku se převod provádí přes UTF-8 kódování.

Zdrojový kód 6.3: Převod řetězce

```
1  /// <summary>
2  /// Deserializes a property from a byte array.
3  /// </summary>
```



Obrázek 6.1: Implementace připojení v knihovně

```
4 /// <param name="property">Byte array property.</param>
5 /// <returns>Deserialized property to a concrete type.</
    returns>
6 public override string Deserialize(byte[] property)
7 {
8     _logger.LogDebug("Deserializing␣property␣{Property}.",
    property);
9     return Encoding.UTF8.GetString(property);
10 }
11
12 /// <summary>
13 /// Serializes a property to a byte array.
14 /// </summary>
15 /// <param name="property">The property.</param>
16 /// <returns>Byte array representation of a property.</
    returns>
17 public override byte[] Serialize(string property)
18 {
19     _logger.LogDebug("Serializing␣property␣{Property}.",
    property);
20     return Encoding.UTF8.GetBytes(property);
21 }
```

Serializace pole v `ArraySerializer` se dělá pomocí blokového kopírování do nového pole s cílovým typem. Když se serializuje bytové pole do bytového pole, tak se pouze přetypuje a vrátí. Tím se ušetří kopírování dat například při zpracování pixelů u obrázku.

Zdrojový kód 6.4: Převod pole

```
1 /// <summary>
2 /// Serializes a property to a byte array.
3 /// </summary>
4 /// <remarks>
5 /// When generic type of this serializer is set to byte, the
    method just casts a reference to the array.
6 /// When types do not match the array is copied, data are
    retyped and returned.
7 /// </remarks>
8 /// <param name="property">The property.</param>
9 /// <returns>Byte array representation of a property.</
    returns>
10 public override byte[] Serialize(T[] property)
11 {
12     byte[]? byteArray;
13
14     if (typeof(T) == typeof(byte))
15     {
16         /*
```

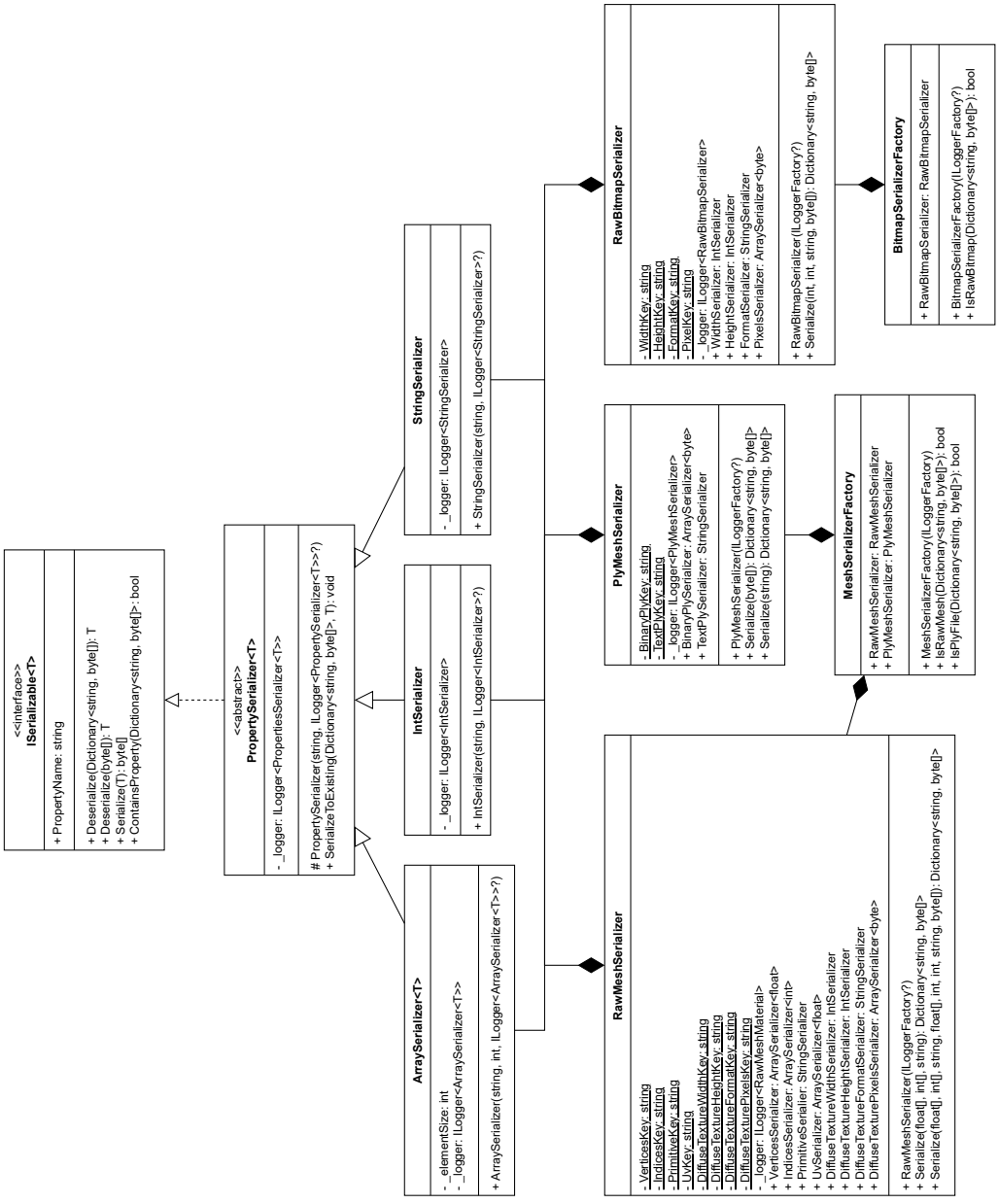
```
17          * There is no need to copy the entire array when
    serializing byte array to byte array.
18          * The generic array is just casted to byte array.
19          */
20         _logger.LogTrace("Serializing␣byte␣array␣to␣byte␣
    array.");
21         byteArray = property as byte[];
22     }
23     else
24     {
25         // When types of arrays differs, bytes has to be
    copied to a new array.
26         _logger.LogTrace("Block-copying␣byte␣array␣to␣typed␣
    array.");
27         byteArray = new byte[property.Length * _elementSize];
28         Buffer.BlockCopy(property, 0, byteArray, 0, byteArray
    .Length);
29     }
30
31     return byteArray!;
32 }
```

Továrny pro serializaci world objektů obsahují tovární metody pro vytvoření serializátorů. Implementace nemusí být bezvýhradně řešena metodami. V případě C# mohou být použity i property. Žádná z tříd, které provádí serializaci, nemění svůj vnitřní stav, takže je možné vždy vracet stejnou instanci. Ušetření zdrojů na vytváření nových instancí je značné, jelikož by jinak musely být vytvářeny znovu pro každý dotaz. `BitmapSerializerFactory` dokáže zkontrolovat, zda vlastnosti uložené ve slovníku odpovídají raw bitmapě. Pokud tato metoda vrátí `True`, může si uživatel vybrat instanci `RawBitmapSerializer`. `MeshSerializerFactory` může rozhodovat mezi vícero typy. Rozděluje vlastnosti na raw a ply mesh. Opět si může uživatel přes property vybrat instance `RawMeshSerializer` nebo `PlyMeshSerializer`.

`RawBitmapSerializer` využívá pro každou vlastnost vlastní serializátor. Potřebné typy serializátorů lze odvodit od datových typů vlastností jednotlivých typů world objektů. Pro serializaci šířky a výšky se používá `IntSerializer`. Formát je možné serializovat přes `StringSerializer` a pixely pomocí `ArraySerializer<byte>`.

`RawMeshSerializer` používá pro serializaci difuzní textury stejné serilizátory jako `RawBitmapSerializer`. Pro pole vrcholů a jejich indexů se užívá `ArraySerializer` s primitivními datovými typy `float` a `int`. Primitive je datového typu `string`, proto se pro jeho serializaci využívá `StringSerializer`.

`PlyMeshSerializer` využívá `ArraySerializer<byte>` pro binární PLY soubor a `StringSerializer` pro textový PLY soubor.



Obrázek 6.2: Implementace serializace v knihovně

6.1.3 Testování

Knihovna obsahuje celkem 222 unit testů, které plně pokrývají serializaci objektů.

6.2 Unity knihovna

6.2.1 Připojení

Abstraktní `DataClientWrapper` a `SessionClientWrapper` obsahují privátní atribut obalovaného typu. Jeho instance je nastavena v `Awake` přes abstraktní metodu `CreateClient`. Většina instančních metod je pouze přesměrována na nižší vrstvu. Jen v `SessionClientWrapper` dochází k obalení volání dalším kódem, který se stará například o Unity eventy nebo logování výjimek.

Situace je však u `SessionClientWrapper` složitější než se může na první pohled zdát. Vzhledem k popsanému způsobu inicializace `SignalRSession` v kapitole 6.1.1 může být problematické měnit adresu serveru za běhu aplikace. U normálního .NET programu stačí vytvořit novou instanci, ale v Unity, kde objekty jsou součástí scény a nastavené reference při návrhu nemusí být možné při běhu změnit, je vytváření nového `SignalRSessionWrapper` nepravděpodobné. Proto bylo potřeba funkčnost `SignalRSessionWrapper` upravit. Při změně adresy se vytváří nová instance `SignalRSession`. To narušuje kontrakt `SignalRSession`, který by měl wrapper dodržet. Tato úprava má vliv i na další části aplikace, protože registrované akce metodou `RegisterCallback` budou po změně adresy serveru ztraceny.

Bylo nutné přizpůsobit `ServerSessionAdapterWrapper`, čímž se odstranila výhoda vnitřní implementace `ServerSessionAdapter`, která registrovala pouze aktuálně potřebné akce. V Unity implementaci se při startu `ServerSessionAdapterWrapper` automaticky nastaví akce pro všechny metody a eventy jsou přesměrovány do veřejného rozhraní.

Dříve v textu bylo zmíněno, že na server může být odesílán ConnectionId v hlavičkách HTTP dotazů. K propojení `RestDataClientWrapper` a `SignalRSessionWrapper` v Unity slouží třída `TechnologyLabServerConnection`. Při startu nebo návázání nového připojení se ConnectionId nastaví do hlavičky `RestDataClientWrapper`.

6.2.2 Práce s objekty

`WorldObjectManager` propojuje lokální uložiště uvnitř `WorldObjectStorageWrapper` a `PrefabStorageWrapper` se serverem, který je dostupný přes `ServerDataAdapterWrapper`. Práce s lokálním úložištěm se vždy provádí až po odeslání změny na server. Kdyby dotaz na serveru selhal, zpracování se přeruší a lokální

úložiště zůstane nezměněno. Jak lze vidět na ukázce přidání nového objektu ve zdrojovém kódu 6.5.

Zdrojový kód 6.5: Přidání objektu v Unity

```
/// <summary>
/// Adds an object to the manager and to a server.
/// </summary>
/// <param name="worldObject">Added game object.</param>
/// <returns>A task.</returns>
public async Task AddObjectAsync(GameObject worldObject)
{
    var propertiesManager = WorldObjectUtils.
    GetPropertiesManager(worldObject);

    var worldObjectDto = CreateWorldObjectDto(worldObject,
    propertiesManager);
    await this.serverDataConnection.Data.AddWorldObjectAsync(
    worldObjectDto);

    if(this.worldObjectEventsHandler != null)
    {
        this.worldObjectEventsHandler.AssignEventHandlers(
    worldObject);
    }

    if (!this.worldObjectStorage.Store(worldObject))
    {
        throw new ArgumentException($"GameObject␣with␣name␣{
    worldObject.name}␣cannot␣be␣added");
    }
}
```

Kromě přidání podporuje `WorldObjectManager` i další operace. Stažení obsahu serveru metodou `LoadServerContentAsync` nebo odstranění všech lokálních objektů `ClearLocalContent`.

`WorldObjectMemoryStorageWrapper` ukládá lokální world objekty v operační paměti. Stejně jako na serveru jsou objekty ukládány do slovníku, jehož klíčem je jméno world objektu. Rozhraní `IWorldObjectStorage` pouze obaluje základní funkce zmíněného slovníku. Implementace je tedy velmi jednoduchá. Provádí pouze volání slovníkových metod. `PrefabStorageWrapper` skladuje vzorové objekty ve slovníku, jehož klíčem je typ objektu, který se posílá na server. Ve výchozí implementaci se tedy jedná o „Mesh“ nebo „Bitmap“. Hodnotou ve slovníku je pak instance `GameObject` reprezentující Unity prefab.

`ServerEventsHandler` se stará o zpracování dotazů, které chodí ze serveru na klienta. Pro připojení závisí na `ServerDataAdapterWrapper` a `ServerSession`-

`AdapterWrapper`. Když mu například dojde zpráva, že nějaký world objekt byl přidaný na server, tak se `ServerEventsHandler` postará o celý proces. Nejprve ze serveru získá objekt se jménem, které ve zprávě dostal. Následně vezme jeho vzor z `PrefabStorageWrapper` a vytvoří ho v Unity scéně, nastaví vlastnosti přes správce vlastností a uloží do lokálního úložiště v `WorldObjectStorageWrapper`. Obsluha ostatních serverových akcí funguje na stejném principu.

6.2.3 Správa vlastností

Správci vlastností se starají o převod world objektů ze serverové reprezentace do Unity objektů. Za pomoci serializátorů z .NET knihovny nastaví vlastnosti do obrázků a meshí, které lze zobrazit uvnitř Unity scény.

6.2.3.1 Mesh

`MeshPropertiesManager` reprezentuje mesh pomocí Unity komponent `MeshFilter` a `MeshRenderer`. První jmenovaný udržuje informace o struktuře meshe - její vrcholy, trojúhelníky, texturovací souřadnice a další. `MeshRenderer` obsahuje data o materiálu, který mesh používá. Z něho je možné získat informace o barvě nebo texturách. `MeshPropertiesManager` poskytuje metody pro přístup k těmto komponentám. Při nastavení nových hodnot reportuje změny přes veřejný event, aby je bylo možné zpropagovat přes `WorldObjectEventsHandler` až na server. Díky serializátorům z .NET knihovny se stará také o serializaci hodnot z komponent do slovníku pro serverovou komunikaci.

6.2.3.2 Bitmapa

`BitmapPropertiesManager` používá `MeshFilter` a `MeshRenderer` stejně jako `MeshPropertiesManager`. Mesh se však nemění. Specifikuje se pevná mesh se čtyřmi vrcholy v jedné rovině. Poměr stran meshe je závislý na poměru stran posílané bitmapy. Texturovací souřadnice (0,0), (1,0), (0,1) a (1,1) se nastaví na vrcholy meshe a bitmapa se pak vykreslí jako textura v materiálu. Jelikož je mesh závislá na bitmapě, musejí se při každé změně upravit její vrcholy. `BitmapPropertiesManager` sdílí s `MeshPropertiesManager` nejen používané Unity komponenty, ale také způsob práce. Umožňuje manipulovat s bitmapou a dávat ostatním objektům vědět o provedených změnách.

6.2.3.3 Volitelné vlastnosti

Kromě definovaných vlastností pro bitmapu a mesh se přes `OptionalPropertiesManager` dají serializovat i další vlastnosti, které nejsou součástí specifikace objektů. Aby bylo možné vlastnosti z `OptionalPropertiesManager` získat, musí

se `OptionalPropertiesManager` přiřadit buďto k `MeshPropertiesManager` nebo `BitmapPropertiesManager`. `OptionalPropertiesManager` obsahuje seznam objektů typu `OptionalProperty`. Každá `OptionalProperty` definuje vlastní způsob serializace a deserializace z `Dictionary`. `OptionalPropertiesManager` prochází jednu `OptionalProperty` za druhou a volá jejich serializační a deserializační metody.

6.2.4 Načítání souborů

Třída `FileLoader` se používá pro načítání souborů z adresářové struktury. Při volání metody `OpenLocalFile` automaticky otevírá dialogové okno z knihovny Runtime File Browser. V okně lze vybrat soubor, který se má načíst. Dialog se otevírá v coroutine, aby neblokoval hlavní Unity vlákno. `FileLoader` se však o čtení obsahu souborů nestará. K tomu vyžaduje instance `FileReader`, které se rozdělují podle typu souboru. Připravenými implementacemi jsou `MeshFileReader` a `BitmapFileReader`. Obě třídy dědí od `FileReader`. Ke čtení meshí se používá knihovna Runtime OBJ Importer. Po načtení objektů se na ně přidají komponenty pro správu vlastností - `MeshPropertiesManager` respektive `BitmapPropertiesManager`.

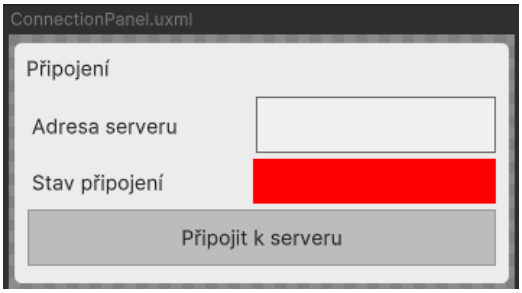
6.2.5 Uživatelské rozhraní

V architektuře Unity knihovny bylo zmíněno o existenci společných částí uživatelského rozhraní. Dialogová okna se hodí téměř všude a jsou obecně rozšířenou formou předávání zpráv uživateli. Snímek dialogového okna je na obr. 6.3. V knihovně je okno reprezentováno abstraktní třídou `MessageBox`. Třída je abstraktní, protože Unity využívá několik různých UI systémů. Každý z nich má jiné rozhraní a jen výjimečně jsou použity současně. Pro každý UI systém se proto musí vytvořit vlastní implementace. Podpořeny jsou `MessageBoxUGUI` a `MessageBoxToolkit`. První jmenovaný využívá UGUI, který vytváří uživatelské rozhraní přes Unity game objekty. Novější systém UI Toolkit staví na UXML souborech a CSS.

Dialogové okno se velmi často vytváří statickou metodou, aby ho bylo možné využít odkudkoliv. Příkladem může být například System.Windows.Forms s metodou `MessageBox.Show`. Dialogové okno uvnitř Unity knihovny se inspiruje právě touto verzí. `MessageBox` obsahuje statickou metodu `Show`, která nainstancuje poskytnutý prefab typu `MessageBox` a nastaví mu titulek, obsah a typ zprávy. Způsob zobrazení závisí na konkrétním typu prefabu. Nelze se bohužel zcela vyhnout předávání prefabu v parametru funkce, ale i tak se jedná o elegantní řešení.

Připojení na server se provádí ve všech úlohách stejně. Proto i panel pro připojení nazývaný `ConnectionPanel` (na obr. 6.4) je součástí knihovny. Stejně tak také jeho kontroler. V uživatelském rozhraní umožňují nastavit adresu serveru a

Obrázek 6.3: Snímek dialogového okna



Obrázek 6.4: Snímek panelu pro připojení

zobrazit stav připojení barevným indikátorem. `ConnectionPanelController` při kliknutí na tlačítko „Připojit k serveru“ inicializuje `SignalRSessionWrapper` a metodou `StartSessionAsync` vytvoří spojení k serveru.

Seznam objektů se také může hodit pro více aplikací. Propojení se zbytkem systému se však může lišit od aplikace k aplikaci, proto `ObjectList` nemá v knihovně žádný kontroler. Různé způsoby výběru objektů mohou vyžadovat jinou implementaci v kontroleru. `ObjectList` umožňuje vybrat objekt přímo v seznamu a vyvolat událost o změně vybraného objektu. Jinak jsou funkce typicky seznamového charakteru, například přidání objektu, odstranění objektu a tak dále.

Referenční řešení úloh

7

7.1 Aplikace pro správu serveru

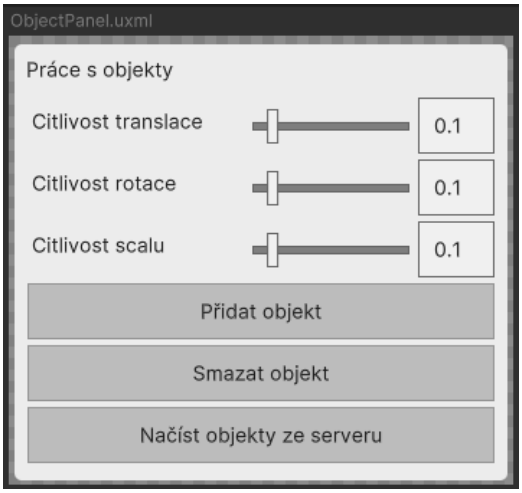
7.1.1 Implementace

Pro tvorbu uživatelského rozhraní byl použit UI Toolkit. Jedná se o nový balík pro tvorbu uživatelského rozhraní v Unity. Snaží se přejít k často využívanému řešení WISIWYG editoru a XML formátu pro reprezentaci UI scény. Uživatelské rozhraní je rozděleno do devíti souborů, kde každý definuje jednu část a postupně se skládají do výsledné scény.

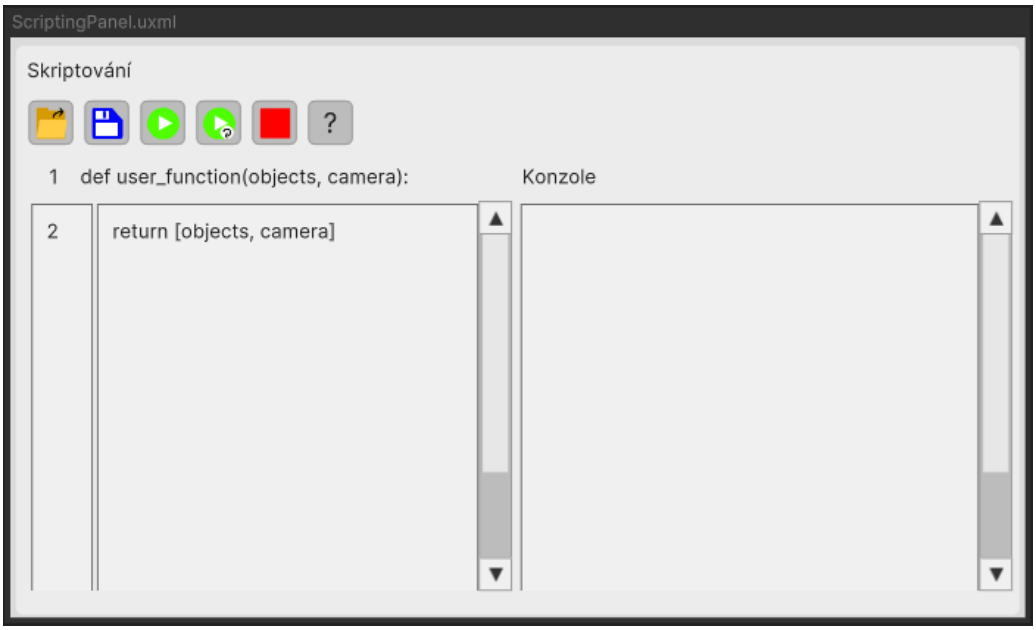
- CameraPanel - obsahuje nastavení a ovládání kamery (viz obr. 7.1).
- ControlPanel - spojení ConnectionPanel, ObjectPanel, CameraPanel.
- MenuBar - tlačítka pro zobrazování a schovávání ostatních panelů,
- ObjectListPanel - obsahuje ObjectList a PropertyList.
- ObjectPanel - pracuje s world objekty. Umožňuje měnit nastavení manipulace a načítat, mazat nebo stáhnout objekty ze serveru. (viz obr. 7.2)
- Property - vzorový objekt pro vytváření property. Obsahuje jméno a hodnotu.
- PropertyList - panel s vlastnostmi (jméno, pozice, rotace, scale),
- Settings - panel měnící globální nastavení.
- ScriptingPanel - zjednodušené vývojové prostředí. Horní část panelu obsahuje tlačítka na otevření, uložení, spuštění a zastavení skriptu. Na jedné polovině je místo pro psaní kódu na druhé konzole. (viz obr. 7.3)

Výsledná scéna je v souboru GameUI.uxml. Každá část rozhraní kromě GameUI má svoji vlastní stejnojmennou třídu, aby byla dostupná ve zdrojovém kódu, a často

Obrázek 7.1: Snímek ovládacího panelu kamery



Obrázek 7.2: Snímek panelu pro práci s objekty



Obrázek 7.3: Snímek skriptovacího panelu

i svůj vlastní kontroler. Třídy z View vrstvy pomáhají s přístupem k prvkům uživatelského rozhraní. Například umožňují kontrolerům schovávat části UI, reagovat na stisknutí tlačítka nebo na změnu vstupu v zadávacím políčku. Kontrolery se starají o propojení různých částí uživatelského rozhraní a o propojení backendu s View vrstvou. Příkladem může být například `MenuBarController`, který při kliknutí na položku v menu schovává či zobrazuje části uživatelského rozhraní. Dobrým příkladem může být také `ObjectListController`, který reaguje na většinu akcí spojených s world objekty. Když se vytvoří na backendu nový world objekt, přidá se do seznamu. Při odstranění objektu ze serveru nebo lokálně se smaže i ze seznamu.

Aplikace využívá Unity Input System, kterým snímá, zda došlo ke stisknutí tlačítka myši nebo klávesnice. Pro manipulaci je nutné definovat vstupní akce, na které mohou objekty reagovat. Mezi akce patří

- Move - pohyb kamery pomocí WASD nebo šipek na klávesnici.
- Look - rozhlížení pravým tlačítkem myši.
- Select - výběr world objektu stisknutím levého tlačítka myši.
- Translate - posun world objekt držením levého tlačítka myši.
- Rotate - rotace držením prostředního tlačítka myši.
- Scale - změna měřítka rotací prostředního kolečka myši.

7.1.2 Propojení s úlohami

V kapitolách o návrhu úloh a logické architektuře bylo popsáno, co musí klient vše umět. Při pohledu na use case diagram 4.3 je vidět, že většina funkčnosti už byla zmíněna někde dříve v textu práce. Většina backendu pro úlohy na správu serveru je totiž zajištěna přes vytvořenou Unity knihovnu. Uvnitř zdrojového kódu aplikace se nachází v podstatě pouze frontend celé úlohy. Ten byl popsaný v minulé kapitole. V této sekci bude probráno, jak jednotlivé části úloh navazují na provedenou implementaci.

Připojit k centrálnímu systému Připojení k centrálnímu systému se provádí přes uživatelské rozhraní ConnectionPanel z projektu ZCU.TechnologyLab.Common.Unity.UI z Unity knihovny. Prvním nutným úkonem je zadat správnou adresu serveru. Kliknutím na tlačítko „Připojit k serveru“ se přes `ConnectionPanelController` akce propaguje hlouběji do Unity knihovny. `SignalRSessionWrapper` dostane žádost o připojení až do .NET knihovny a `SignalRSession` připojí aplikaci k serveru. Při selhání připojení nebo zadání špatné adresy se objeví chybové hlášení.

Nahrát objekt Objekt je možné nahrát kliknutím na tlačítko „Přidat objekt“. Přes `ObjectPanelController` se zavolá čtení objektu ze souboru v Unity knihovně. Objeví se dialogové okno se soubory, z kterých je možné vybrat OBJ mesh nebo bitmapu v různých formátech. Po načtení souboru se world objekt dostane do třídy `WorldObjectManager`, kde se zavolá `ServerDataAdapterWrapper`. Ten pak předá world objekt do třídy `ServerDataAdapter` uvnitř .NET knihovny. Nakonec se world objekt odešle přes `RestDataClient` na server.

Vybrat objekt Aplikace poskytuje více možností, kterými lze objekt zvolit. `ObjectMouseSelection` reaguje na stisknutí levého tlačítka myši. Při výběru dojde k raycastu, jehož směr určí vybraný world objekt. Druhá možnost, jak objekt zvolit, je přes `ObjectListController`, a to výběrem world objektu ze seznamu. Když některá z těchto dvou tříd informuje o změně vybraného objektu, změní se properta `SelectedObject` ve třídě `ObjectSelection`. Jak bylo popsáno v architektuře, jedná se o mediátor, který rozesílá události o této změně. Na ní zareaguje `HighlightBox` a zvýrazní vybraný objekt v trojrozměrném prostoru.

Smazat objekt Pro smazání objektu existuje v uživatelském rozhraní tlačítko „Smazat objekt“. Cesta je velmi podobná jako u nahrání objektu. `ObjectPanelController` získá z `ObjectSelection` aktuálně vybraný objekt a jeho jméno předá do `WorldObjectManager` z Unity knihovny. Dále putuje dotaz přes `ServerDataAdapterWrapper` a `ServerDataAdapter` až do `RestDataClient`, odkud se po-

žadavek dostane na server.

Stáhnout všechny objekty V `ObjectPanel` existuje tlačítko „Načíst objekty ze serveru“, které v `ObjectPanelController` zavolá načtení serveru v `WorldObjectManager`. Jako první se odstraní všechny existující lokální objekty a pak se stáhne obsah serveru. Dotaz na stažení jde opět do `ServerDataAdapterWrapper`, následně `ServerDataAdapter` a nakonec `RestDataClient`. Stažené objekty se dostanou zpět do `WorldObjectManager`, kde se vytvoří jejich Unity reprezentace. Jako poslední se uloží objekty do lokálního úložiště v `WorldObjectMemoryStorageWrapper`.

Manipulovat s kamerou Manipulace s kamerou je součástí jen klienta pro správu serveru. Při provádění se nepřistupuje k žádné knihovně. Manipulace s kamerou závisí na výše popsaných akcích z Input Systemu. Na Move a Look reaguje třída `CameraMovement`.

Manipultovat s objektem Manipulaci s objektem zajišťuje třída `ObjectMouseManipulation`. Pokud nastane nějaká ze vstupních akcí Translate, Rotate nebo Scale, provede se metoda, která danou akci hlídá. Vybraný world objekt v `ObjectSelection` je následně přesunut nebo se změní jeho rotace či měřítko. Na změnu transformace reaguje třída `ReportTransformChange` z Unity knihovny. Třída je umístěna na každém world objektu. `WorldObjectEventsHandler` naslouchá jejím událostem a odesílá je přes `ServerSessionAdapterWrapper` do .NET knihovny. Tam je zachytí `ServerSessionAdapter` a pomocí `SignalRSession` odešle na server v reálném čase.

Skriptovat Ke skriptování lze přistupovat přes `ScriptingPanel`, o který se stará kontroler `ScriptingPanelController`. Při kliknutí na zelené tlačítko start, se skript automaticky spustí. `ScriptingPanelController` v tento okamžik vytvoří instance `DispatchedTextWriter` pro přesměrování výstupu do konzole. V konzoli se zobrazuje veškerý výstup, jak stdout, tak stderr, aby mohl uživatel při testování získávat zpětnou vazbu. Chybová hlášení v konzoli většinou obsahují číslo řádky, na kterém se chyba nachází. Pole pro skriptování má automaticky číslované řádky. Snadněji se díky tomu identifikují pozice chyb. Práce s Python prostředím probíhá ve třídě `ScriptingController`. Inicializace prostředí i spouštění skriptů funguje asynchronně, aby delší běh neblokoval hlavní vlákno. Skriptování povoluje vytvořit jednu Python metodu s předepsanou hlavičkou a návratovými typy. Parametry funkce jsou

- objects - slovník objektů, které popisují transformaci world objektů. Klíčem

ve slovníku je jméno world objektu, které lze zjistit například ze seznamu v pravém panelu `ObjectListPanel`.

Zdrojový kód 7.1: Transformace world objektu pro skriptování

```
public class ObjectTransform
{
    public float XPosition { get; set; }
    public float YPosition { get; set; }
    public float ZPosition { get; set; }
    public float XRotation { get; set; }
    public float YRotation { get; set; }
    public float ZRotation { get; set; }
    public float XScale { get; set; }
    public float YScale { get; set; }
    public float ZScale { get; set; }
}
```

- camera - transformace kamery

Zdrojový kód 7.2: Transformace kamery pro skriptování

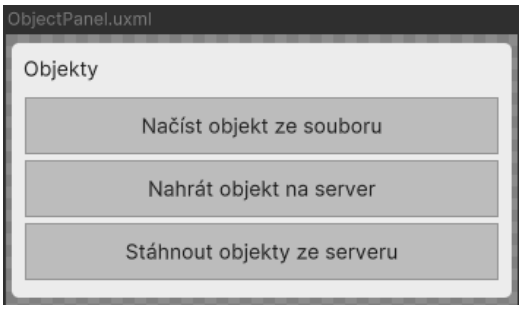
```
public class CameraTransform
{
    public float XPosition { get; set; }
    public float YPosition { get; set; }
    public float ZPosition { get; set; }
    public float XRotation { get; set; }
    public float YRotation { get; set; }
    public float ZRotation { get; set; }
}
```

V návratové hodnotě se očekává pole se stejnými typy objektů, jako v parametrech. To znamená pole se slovníkem a `CameraTransform`. Po běhu skriptu se pomocí mapování jmen přiřadí nové transformace k příslušným world objektům. Stejně jako v manipulaci pak bude změna transformace zachycena pomocí `ReportTransformChange` a odeslána na server.

7.2 Haptické kreslení

7.2.1 Implementace

Pro tvorbu uživatelského rozhraní byl opět použit UI Toolkit. Uživatelské rozhraní je rozděleno do tří souborů, kde každý definuje jednu část a postupně se skládají do výsledné scény.

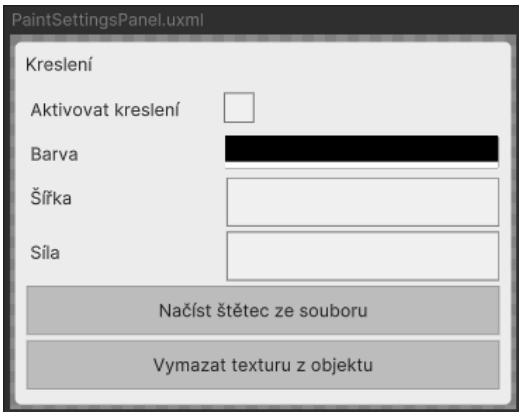


Obrázek 7.4: Snímek panelu pro práci s objekty

Obrázek 7.5: Snímek panelu pro nastavení haptiky

- ObjectPanel - pracuje s world objekty. Umožňuje načítat, mazat nebo stáhnout objekty ze serveru. (viz obr. 7.4)
- HapticSettingsPanel - nastavuje haptické vlastnosti na objektech (viz obr. 7.5)
- PaintSettingsPanel - nastavuje vlastnosti štětce a umožňuje štětec načíst z XML souboru. (viz obr. 7.6)

Spojení do jedné scény se provede v souboru GameUI.uxml. Stejně jako tomu bylo u programu na správu serveru, má každý panel příslušející třídu a kontroler. `HapticSettingsPanelController` vystavuje přes sdílené proměnné nastavení haptického zařízení třídě `ObjectManager`, která je posílá dále do `HapticManager`. Podobnou vazbu mezi sebou mají také `PaintSettingsPanelController` a `PaintManager`. Jediný rozdíl je ve sdílených datech. Sdílené proměnné jsou serializované



Obrázek 7.6: Snímek panelu pro nastavení kreslení

objekty, v Unity pojmenované `ScriptableObject`. Jedná se o objekty dostupné z adresářové struktury, ke kterým lze přes referenci přistupovat téměř odkudkoliv. Výhodou je, že se nevytvářejí nové instance. `ScriptableObject` funguje podobně jako návrhový vzor jedináček.

`PaintSettingsPanel` obsahuje tlačítko „Načíst štětec ze souboru“ na obr. 7.6. Díky `BrushReader`, který štětec z XML načítá, lze nastavit barvu a sílu štětce i z tohoto XML formátu. Formát štětce obsahuje i další vlastnosti, které ovšem nejsou v aplikaci podpořené. Pro vytvoření štětce se používá program třetí party vytvořený v rámci projektu IKAP [Kön23].

`ObjectManager` se stará o inicializaci tříd, které spravují haptickou knihovnu. Inicializace probíhá v rámci přidání world objektů do aplikace, ať už přes `ServerEventsHandler` nebo `FileLoader`. V aplikaci je možné pracovat pouze s jedním objektem naráz. Mezitím jsou ostatní deaktivovány. Přepínání aktivních objektů obstarává `ObjectManager` díky metodě `SetObject`. V metodě se deaktivuje aktuální viditelný world objekt a na jeho místo přijde nový. Při aktivaci je kontrolováno, zda world objekt obsahuje texturovací souřadnice. Poslední funkčnost, kterou `ObjectManager` zprostředkovává, je manipulace s world objektem v prostoru pomocí tlačítek na haptickém zařízení a myši.

`HapticManager` a `PaintManager` jsou přítomni na všech world objektech v kreslící aplikaci. `HapticManager` řídí nastavení haptických vlastností (statické tření, dynamické tření, viskozita, hmotnost, tvrdost, velikost síly působící kolem objektu a její směr), aby se všechny world objekty chovaly stejně. Naslouchá událostem na zmíněných sdílených proměnných a mění podle nich nastavení uvnitř tříd z haptické knihovny. `PaintManager` funguje totožně, pouze sleduje změny u barvy, velikost a tlaku štětce.

Ač to nebylo přímo specifikováno v požadavcích na úlohu zabývající se haptickým zařízením, umožňuje aplikace také pracovat s daty ze serveru. `ObjectPanel`

obsahuje tlačítka na načtení souboru, odeslání world objektu na serveru a stažení obsahu ze serveru. Společné použití desktopové aplikace na správu serveru a haptického kreslení demonstruje propojení mezi klienty přes centrální systém. Primárně by však měla být aplikace využívána pouze lokálně, proto se při načítání souboru objekty automaticky nenahrávají na server, jako tomu bylo v případě desktopového klienta na správu serveru. Nahrání na server se musí provést ručně. Nutno podotknout, že jakmile se aplikace připojí k serveru, bude získávat objekty ze serveru a odesílat lokální změny existujících objektů automaticky. Pozice, rotace a měřítko se na server z kreslící aplikace neposílají, protože všechny objekty jsou automaticky umístěny do počátku soustavy souřadnic, aby byly v dosahu haptického pera, které se nemůže zcela volně pohybovat po virtuálním prostoru, ale je svázáno svým fyzickým protějškem.

7.2.2 Propojení s úlohami

Hýbat s haptickým perem Většinu práce s haptickým perem provádí sama knihovna Haptics Direct for Unity V1. V knihovně jsou připravené Unity komponenty, která se vkládají do scény. Knihovna se stará o snímání pohybu ze zařízení a také o pohyb virtuálního pera. Knihovna poskytuje události o akcích typu stisknutí tlačítka, uvolnění tlačítka, držení tlačítka a dalších. Na ně je možné reagovat a přidat vlastní manipulaci.

Dotýkat se perem objektů Kolize mezi objekty je další část, kterou knihovna Haptics Direct for Unity V1 umí poskytnout. Na objekty, s kterými lze kolidovat, musí být umístěn `MeshCollider`.

Pociťovat tvrdost materiálu Tvrdost materiálu se nastavuje pomocí prvního posuvníku v `HapticSettingsPanel`. Když je velikost tvrdosti nulová, simulace je vypnuta. Tvrdost materiálu není brána absolutně, ale jedná se o relativní hodnotu ke schopnostem haptického zařízení. Maximální hodnota je rovna jedné, což představuje maximální možný odpor, který dokáže zařízení simulovat. Z `HapticSettingsPanel` se hodnota dostává přes `HapticSettingsPanelController` do sdílené proměnné `FloatVariable`. Proměnná s tvrdostí je referencována z `ObjectManager`, který ji předá do `HapticManager` při načtení world objektu. `HapticManager` reaguje na změny sdílené proměnné a aplikuje je do `HapticMaterial` z Haptics Direct for Unity V1. Tvrdost je v knihovně následně simulována při kolizi mezi perem a world objektem s `HapticMaterial`.

Pociťovat viskozitu materiálu Viskozita funguje totožně jako tvrdost v předchozím odstavci. Nastavuje se čtvrtým posuvníkem v `HapticSettingsPanel`. Způsob

propagace hodnoty do `HapticMaterial` je identický. Jediným rozdílem je chování, které lze očekávat, když se nastaví na větší hodnotu než nula. Zatímco tvrdost blokovala pohyb do objektu při kolizi, tak viskozita zpomaluje pohyb uvnitř objektu. Z toho je jasné, že vzájemné nastavení tvrdosti a viskozity nedává velký smysl, protože by nešlo do objektu vůbec vstoupit.

Pociťovat tření materiálu Tření sdílí s předchozími fyzikálními vlastnostmi stejný způsob zpracování. Chování se však liší. Tření se rozděluje na statické a dynamické. Statické tření působí v okamžiku, kdy se pero začíná hýbat z klidové polohy. Na druhou stranu dynamické tření zapříčiní obtížnější pohyb po objektu. Statické tření se mění pomocí druhého posuvníku v `HapticSettingsPanel` a dynamické pomocí třetího.

Pociťovat hmotnost materiálu Hmotnost se nastavuje stejně jako ostatní. Na rozdíl od všech předchozích vlastností však nepůsobí hmotnost tělesa vždy. Aplikuje se jen při manipulaci s objektem. Hodnota nastavovaná v `HapticSettingsPanel` je udávána v kilogramech.

Pociťovat přitažlivou/odpudivou sílu z objektu Síla se skládá ze dvou vlastností. Jako první je nutné určit velikost síly a pak směr. Úprava hodnot se provádí dvěma posledními editačními poli v `HapticSettingsPanel`. Směr síly je vždy relativní ke směru normály v bodu dotyku mezi objektem a haptickým perem. Směr síly je jako jediná z hodnot uložená v objektu `Vector3Variable`.

Manipulovat s objektem pomocí pera Manipulace začíná při kliknutí tlačítka na haptickém peru. Cílové haptické zařízení popsané v kapitole 2.1 obsahuje dvě tlačítka a knihovna Haptics Direct for Unity V1 je na to připravena. Při držení prvního tlačítka se aktivuje chycení world objektu. Třída `HapticPlugin` z knihovny Haptics Direct for Unity V1 poskytuje událost `OnHoldButton1`. Na ní je navěšena metoda `AllowGrab` z `ObjectManager`. Puštění tlačítka způsobí událost `OnReleaseButton1`. Reakce v `ObjectManager.CancelGrab` zruší zachycení world objektu a vrátí ho zpět na původní místo. Chycení world objektu ve výsledku mění pouze jeho rotaci.

Kromě chycení je možné manipulovat s world objektem tradičně rotací kolem os. Držení druhého tlačítka vyvolá `OnHoldButton2`, což způsobí provedení metody `ObjectManager.TransformObject`. Haptické pero více tlačítek neobsahuje. Pro změnu měřítka bylo tedy použito prostřední kolečko myši stejně jako u aplikace na správu serveru.

Kreslit perem na objekty Kreslení zprostředkovává opět knihovna Haptics Direct for Unity V1. Vlastnosti z `PaintSettingsPanel` se skrz kontroler nastavují do serializovaných proměnných. Šířka štětce je uložena v `IntVariable`, zatímco barva v `ColorVariable` a síla působící na štětec v již zmíněné `FloatVariable`. Podle nastavené barvy, šířky i síly štětce a tvrdosti materiálu se vykreslují různě velké, barevné kruhy kolem bodu dotyku. Aby bylo možné kreslit na world objekty, musí být tvrdost nenulová, jinak by kreslení nefungovalo. Další překážkou jsou texturovací souřadnice. Kdyby nebyly souřadnice dostupné, nebylo by možné zjistit, do jaké části textury kreslit. Kreslení proto nepodporuje mesh bez texturovacích souřadnic. Kdyby byl do programu nahrán takovýto trojrozměrný model, ukáže se chybové hlášení.

Načíst objekt Objekt lze nahrát kliknutím na tlačítko „Načíst objekt ze souboru“ v uživatelském rozhraní `ObjectPanel`. `ObjectPanelController` volá souborové čtení z Unity knihovny. Tam se otevře dialogové okno pro výběr meshe. Načtený world object se přesune do třídy `WorldObjectManager`, která volá `ServerDataAdapterWrapper`. Wrapper závisí na `ServerDataAdapter`, jež odesílá world objekty přes `RestDataClient` na server.

7.3 Testování

Kromě knihoven nejsou v klientských aplikacích žádné další automatické testy. Aplikace na správu serveru byla otestováno uživatelským explorativním testováním v technologické laboratoři na Západočeské univerzitě v Plzni i Gymnáziu Sokolov. Testování haptického kreslení bylo provedeno pouze v technologické laboratoři na Západočeské univerzitě v Plzni.

Závěr 8

Cílem práce bylo vytvořit demonstrační aplikace a metodické materiály pro informatickou laboratoř techniky. K demonstraci různých zařízení v laboratoři byl navržen centrální systém, který je spojuje do jedné komplexní aplikace. Díky tomu lze uskutečnit spolupráci více zařízení, které spolu nejde běžně využívat - například více brýlí pro virtuální realitu. Podle popsaných požadavků byly navrženy úlohy na stolní počítač a haptické zařízení, jež demonstrují schopnosti centrálního systému a představují uživatelům, jak používat haptické zařízení pomocí nastudovaného API.

Kompletní systém včetně klientských aplikací byl navržen s cílem obecnosti, stability a jednoduchosti, aby bylo možné bez větších obtíží přidávat další klienty, kteří již mohou být vytvářeni přímo studenty gymnázia v Sokolově, kde se laboratoř nachází. Na podporu navržených úloh byly vytvořeny aplikace, které uživatelům umožní ovládat serverovou část systému a haptické zařízení. Aplikace byly postaveny na dvou knihovnách, které zprostředkovávají práci s centrálním serverem a poskytují tak studentů základ pro tvorbu vlastních aplikací na platformě .NET a herním enginu Unity.

Na závěr byly vytvořeny metodické materiály, které popisují zadání úkolů jednotlivých úloh připravených pro různé skupiny uživatelů včetně poskytnutého řešení s obrazovou dokumentací. Tyto metodické materiály figurují zároveň jako návody na použití centrálního serveru a haptického zařízení.

Přínosem této práce je vytvoření metodických materiálů pro zařízení, kde jich existuje jen velmi omezené množství. Haptické zařízení je velmi specializované a na trhu se příliš nevyskytuje. Kromě podkladů od výrobce existuje jen velmi málo softwaru, který by šlo při výuce použít, natož s vypracovanými úlohami a řešením. Práce také přispěje k podpoře tvůrčí činnosti studentů, kteří mohou pro své maturitní a jiné práce vytvářet vlastní programy pro server a tím obohacovat schopnosti celého systému, který gymnázium provozuje. Postupem času se může jedna o komplexní systém spolupracujících aplikací s velmi rozsáhlým použitím pro různé školní předměty. Záleží na budoucím zájmu a schopnostech studentů.

Existuje řada věcí, které by bylo možné dále upravit nebo rozšířit. Po domluvě s gymnáziem běží server uvnitř laboratoře na jednom ze stolních počítačů. Dalším

krokem by mohl být jeho přesun na školský server, díky čemuž by mohl být studentům přístupný i mimo laboratoř. V takovém případě by muselo být přidané perzistentní úložiště, aby všechna data nebyla v operační paměti, protože se dá očekávat větší zatížení. Zároveň by úložiště chránilo před ztrátou dat. Veřejné zpřístupnění serveru by pravděpodobně vyžadovalo přidat autentizaci a autorizaci připojených klientů. Pro robustnější práci s API by šlo přidat přístup k jednotlivým vlastnostem objektů, čímž by se snížil objem komunikace. Momentálně lze přistupovat pouze ke všem najednou. S objemem komunikace souvisí také přidání komprese posílaných dat, která by však snížila jednoduchost použití API v klientských aplikacích. Přidaná složitost by byla patrná především v programovacích jazycích, které nebyly podpořeny knihovnami.

Tato práce se soustředila primárně na vytvoření centrálního systému a podpůrných knihoven, na kterých lze dále stavět. Úlohy byly vytvořeny až jako následující. Z nasbíraných zkušeností by bylo pravděpodobně lepší v další práci pořadí otočit a server vytvářet až v závislosti na úlohách. Mohla by tím však utrpět obecnost vytvořeného řešení.

Příloha 1 - Uživatelská dokumentace serveru

A

Serverová aplikace dokáže pomocí definovaného API propojit programy ovládající různá zařízení. Interakci zajišťuje přes sdílené úložiště dat. Je schopná

- nahrávat data,
- mazat data,
- vracet data,
- upravovat data,
- hlásit změny
- a další.

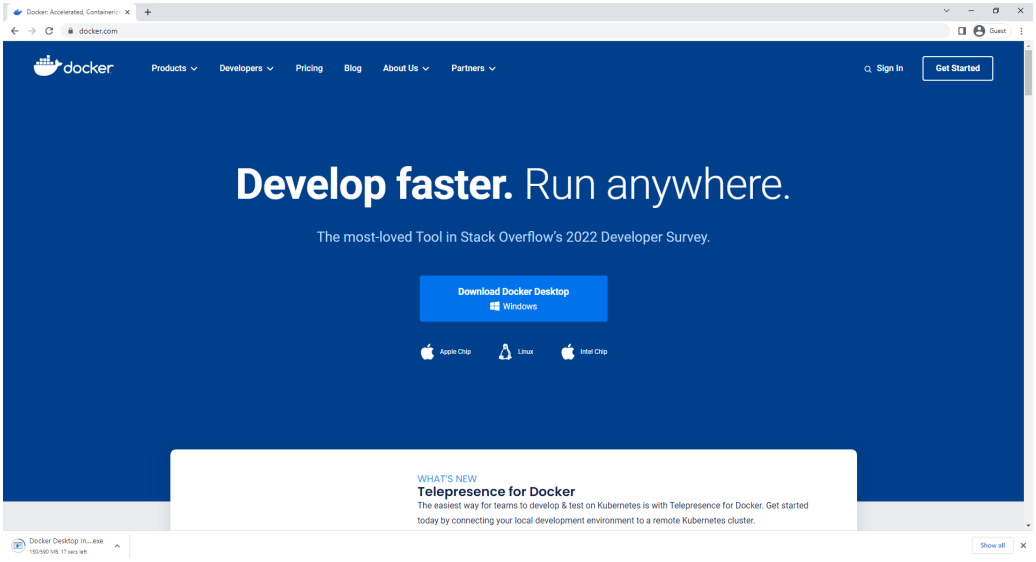
Server je distribuován jako zip archiv se zdrojovým kódem včetně nástrojů potřebných pro sestavení aplikace. Archiv je nutné rozbalit. Serverovou aplikaci je možné sestavit a spustit v Docker kontejneru.

A.1 Docker

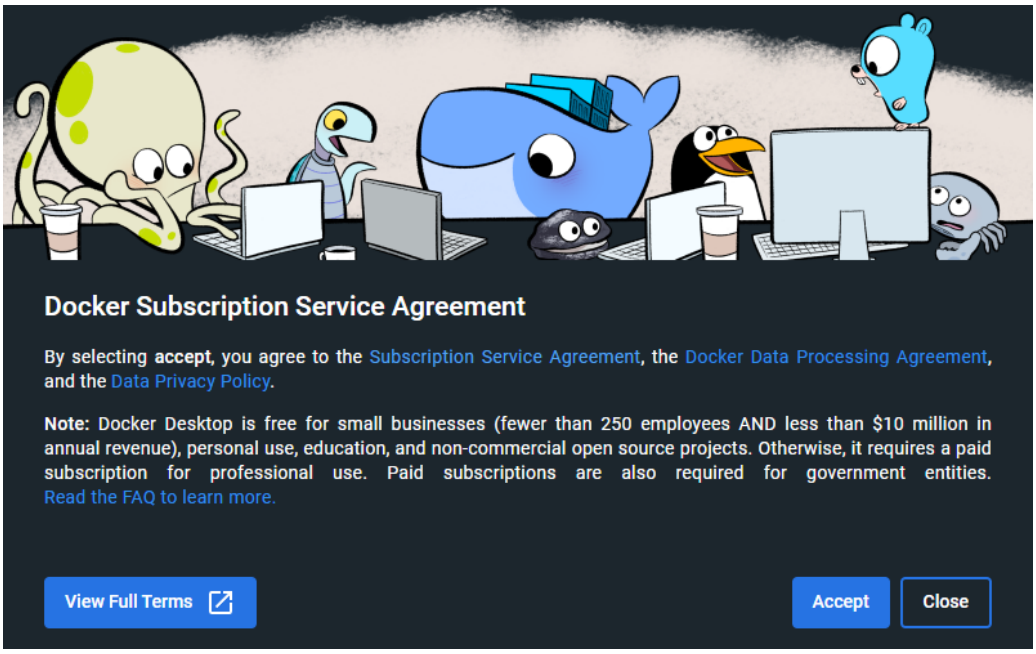
Serverová aplikace je primárně navržena pro běh v Dockeru. Pokud není Docker na počítači nainstalovaný, je nutné ho nejprve stáhnout. Instalační program je dostupný na stránkách `https://www.docker.com/`. Pro osobní, edukativní nebo open-source použití je Docker zdarma, takže není potřeba mít obavy o licenční podmínky.

Instalační program provede celou instalací programu Docker Desktop. Popis instalace v tomto dokumentu je závislý na platformě Windows, pokud instalace probíhá na jinou platformu, bližší popis je k nahlédnutí na stránce `https://docs.docker.com/desktop/`.

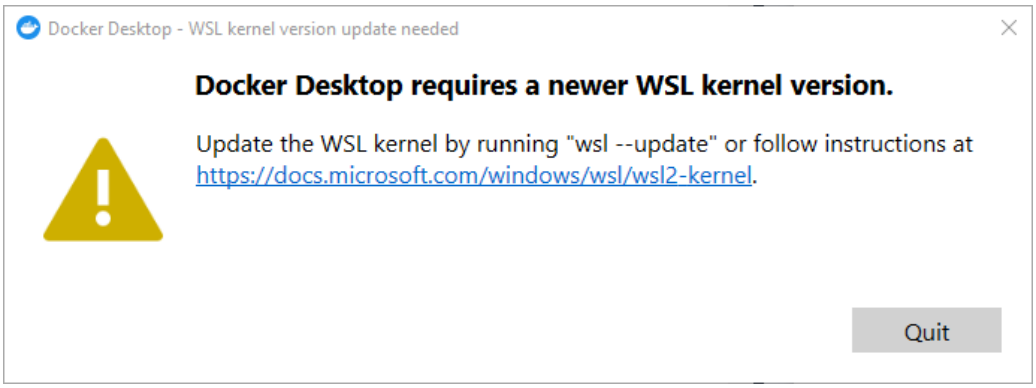
V průběhu instalace je vyžadováno provést restart operačního systému. Po něm stačí pouze potvrdit obchodní podmínky (viz obr. A.2) a může se Docker využívat.



Obrázek A.1: Stažení Docker Desktop



Obrázek A.2: Obchodní podmínky Docker Desktop



Obrázek A.3: Chybějící WSL

Po instalaci dojde k automatickému spuštění. Může se stát, že se ukáže chybová hláška o nenainstalovaném nebo zastaralém WSL jádře jako na obr. A.3.

Nainstalování Subsystem Windows pro Linux na Windows 10 verze 2004 a výš (Build 19041 a výš) nebo Windows 11 se provádí přes příkazovou řádku cmd.exe v režimu administrátora. V ní stačí spustit příkaz `wsl --install`. Konzolový výpis A.1 představuje ukázkový běh příkazu. Po dokončení instalace WSL lze Docker spustit. V případě starší verze Windows je instalace složitější a vyžaduje stažení instalačního souboru na adrese `https://learn.microsoft.com/en-us/windows/wsl/install-manual`.

Výpis A.1: Příkaz na instalaci WSL

```
C:\WINDOWS\system32>wsl --install
Installing: Virtual Machine Platform
Virtual Machine Platform has been installed.
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
Downloading: WSL Kernel
Installing: WSL Kernel
WSL Kernel has been installed.
Downloading: Ubuntu
The requested operation is successful. Changes will not be
    effective until the system is rebooted.

C:\WINDOWS\system32>
```

A.1.1 Docker compose

Uvnitř adresářové struktury je k dispozici soubor ZCU.TechnologyLab.VirtualWorld-Server/docker-compose.yml. Obsahuje příkazy, které popisují způsob sestavení a

popis vlastností prostředí pro běh aplikace v Dockeru. S docker compose se pracuje v příkazové řádce. Příkazy automaticky použijí nejbližší docker-compose soubor.

1. Přesunout aktivní složku příkazové řádky do složky ZCU.TechnologyLab.VirtualWorldServer příkazem `cd`
2. Vytvořit a sestavit kontejner v dockeru příkazem `docker compose create`
3. Spustit kontejner se serverem příkazem `docker compose start`

Výpis A.2: Příkaz na vytvoření kontejneru

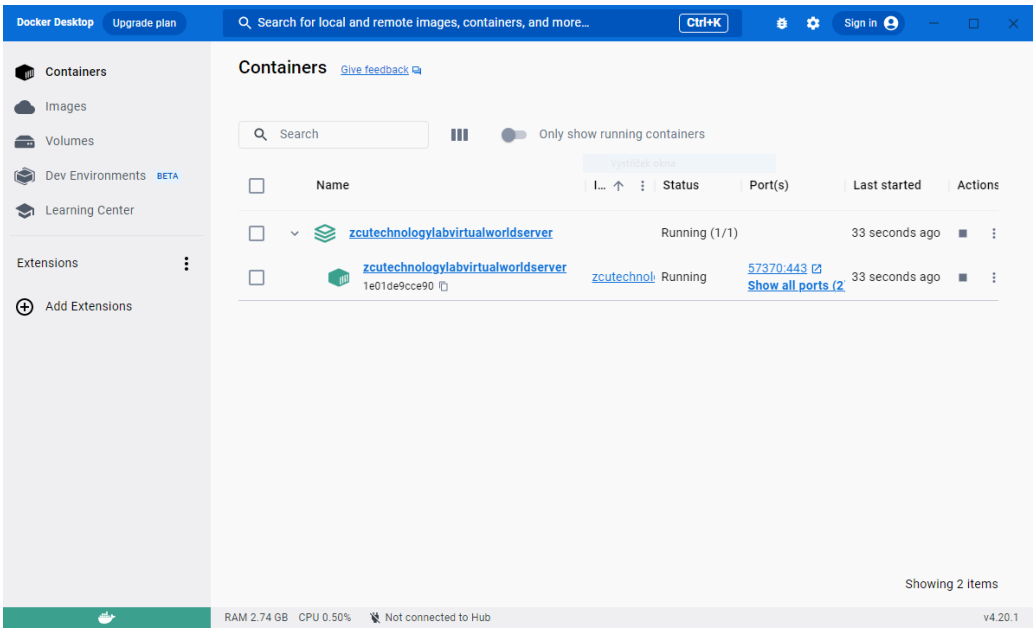
```
C:\server\ZCU.TechnologyLab.VirtualWorldServer>docker compose create
[+] Running 0/0
 - zcu.technologylab.virtualworldserver Pulling                    0.1s
[+] Building 3.9s (5/17)
 => [internal] load metadata for
    mcr.microsoft.com/dotnet/sdk:6.0                               0.4s
[+] Building 28.4s (18/18) FINISHED
 => [internal] load build definition from Dockerfile               0.1s
 => => transferring dockerfile: 971B                               0.0s
 => [internal] load .dockerignore                                  0.0s
 => => transferring context: 382B                                  0.0s
 => [internal] load metadata for
    mcr.microsoft.com/dotnet/sdk:6.0                               0.4s
 => [internal] load metadata for
    mcr.microsoft.com/dotnet/aspnet:6.0                            0.5s
 => [internal] load build context                                  0.2s
 => => transferring context: 72.71kB                               0.2s
 => [build 1/7] FROM
    mcr.microsoft.com/dotnet/sdk:6.0@sha256:a3bbff689a86ba7f3d
    dcee5089a729b20e20e3b4dbfb9d0a43b                             14.9s
 => => resolve
    mcr.microsoft.com/dotnet/sdk:6.0@sha256:a3bbff689a86ba7f3d
    dcee5089a729b20e20e3b4dbfb9d0a43bb3284d9                       0.0s

 ... hashování a extrakce dat jsou z výpisu vynechány

 => [base 1/2] FROM
    mcr.microsoft.com/dotnet/aspnet:6.0@sha256:f76f95813a87d71
    1928c6b02335614f328cc1c9c53315137fd                            7.1s
 => => resolve
    mcr.microsoft.com/dotnet/aspnet:6.0@sha256:f76f95813a87d71
    1928c6b02335614f328cc1c9c53315137fd64ea1                       0.0s

 ... hashování a extrakce dat jsou z výpisu vynechány

 => [base 2/2] WORKDIR /app                                        1.6s
```

```
 => [final 1/2] WORKDIR /app                                    0.1s
 => [build 2/7] WORKDIR /src                                    1.5s
 => [build 3/7] COPY
   [ZCU.TechnologyLab.VirtualWorldServer/ZCU.TechnologyLab.Vi
   rtualWorldServer.csproj, ZCU.Techn                           0.1s
 => [build 4/7] RUN dotnet restore
   "ZCU.TechnologyLab.VirtualWorldServer/ZCU.TechnologyLab.Vi
   rtualWorldServer.csp                                         4.4s
 => [build 5/7] COPY . .                                        0.1s
 => [build 6/7] WORKDIR
   /src/ZCU.TechnologyLab.VirtualWorldServer                    0.1s
 => [build 7/7] RUN dotnet build
   "ZCU.TechnologyLab.VirtualWorldServer.csproj" -c Release
   -o /app/build                                                4.3s
 => [publish 1/1] RUN dotnet publish
   "ZCU.TechnologyLab.VirtualWorldServer.csproj" -c Release
   -o /app/publish                                              2.4s
 => [final 2/2] COPY --from=publish /app/publish .              0.1s
 => exporting to image                                          0.1s
 => => exporting layers                                         0.1s
 => => writing image
   sha256:3cf6ceeced767b120937ad7463a7e9f757780ca4fea34142d76
   43d15cbafc562                                                0.0s
 => => naming to
   docker.io/library/zcutechnologylabvirtualworldserver         0.0s
[+] Running 2/2
 ✓ Network zcutechnologylabvirtualworldserver_default
   Cre...                                                       0.1s
 ✓ Container zcutechnologylabvirtualworldserver
   Created                                                      0.1s
C:\server\ZCU.TechnologyLab.VirtualWorldServer>docker compose start
[+] Running 1/1
 ✓ Container zcutechnologylabvirtualworldserver  Started 0.4s
C:\server\ZCU.TechnologyLab.VirtualWorldServer>
```

A.1.2 Práce s kontejnerem

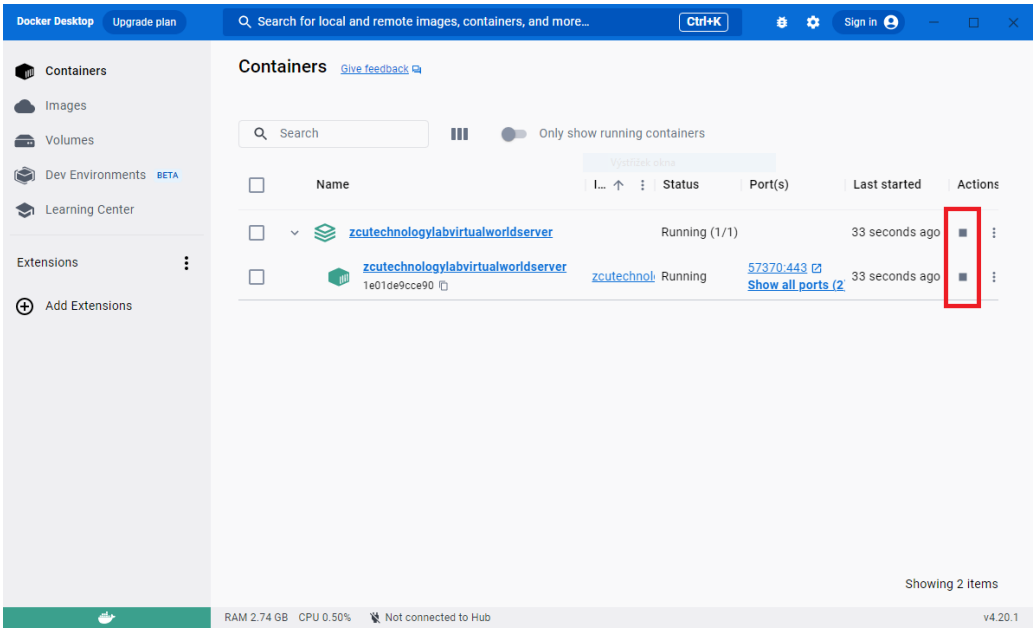
Kromě příkazové řádky lze přistupovat ke kontejneru také z aplikace Docker Desktop. Na hlavní obrazovce A.4 lze vidět seznam nainstalovaných kontejnerů. Běžící kontejner je možné zastavit tlačítkem se symbolem čtverce v aplikaci. Na obr. A.5 jsou tlačítka vyznačena. Na stejném místě se nalézá také tlačítko pro spuštění kontejneru, které se zobrazuje, když je kontejner zastavený.

Při kliknutí na vyznačený kontejner v obr. A.7 se zobrazí logovací obrazovka A.8. Jedná se o dobré místo na kontrolu funkčnosti serveru v případě nějakého neočekávaného chování. V pozdější části dokumentace bude práce s logy ukázána

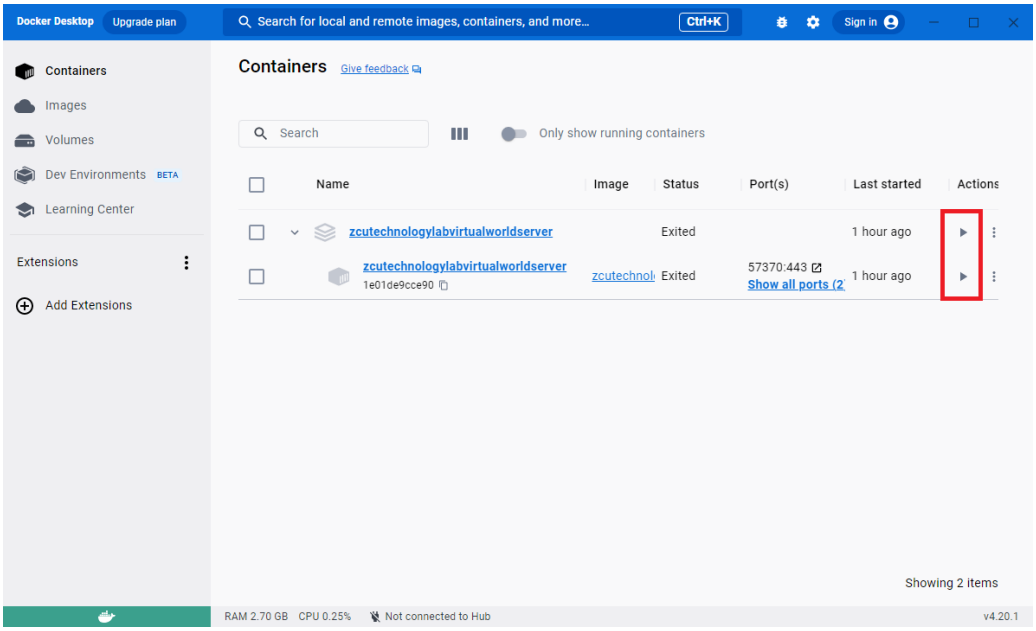


Obrázek A.4: Seznam kontejnerů v Docker Desktop

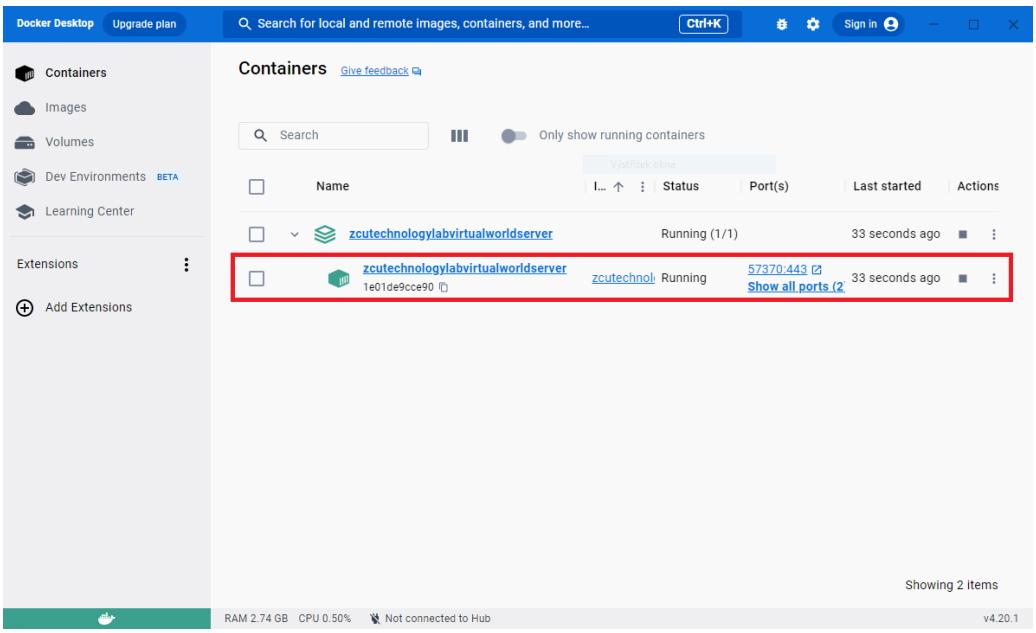
blíže. Významná je také obrazovka s vlastnostmi kontejneru pod záložkou „Inspect“. K nahlédnutí na obr. A.9 jsou například verze jazyka a frameworku, nastavené porty nebo adresářová struktura uvnitř kontejneru.



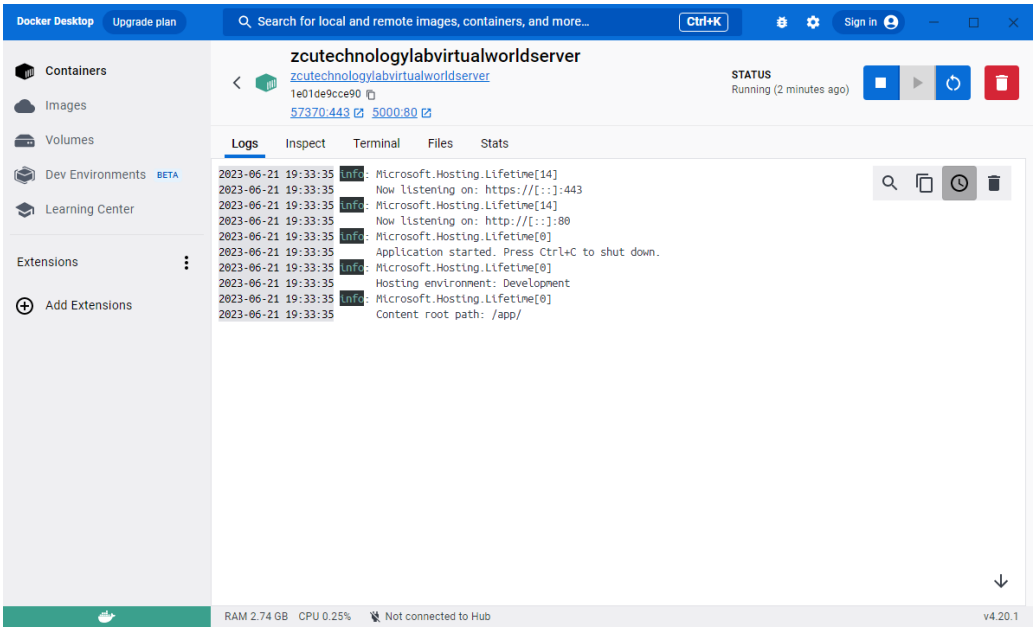
Obrázek A.5: Zastavení kontejneru v Docker Desktop



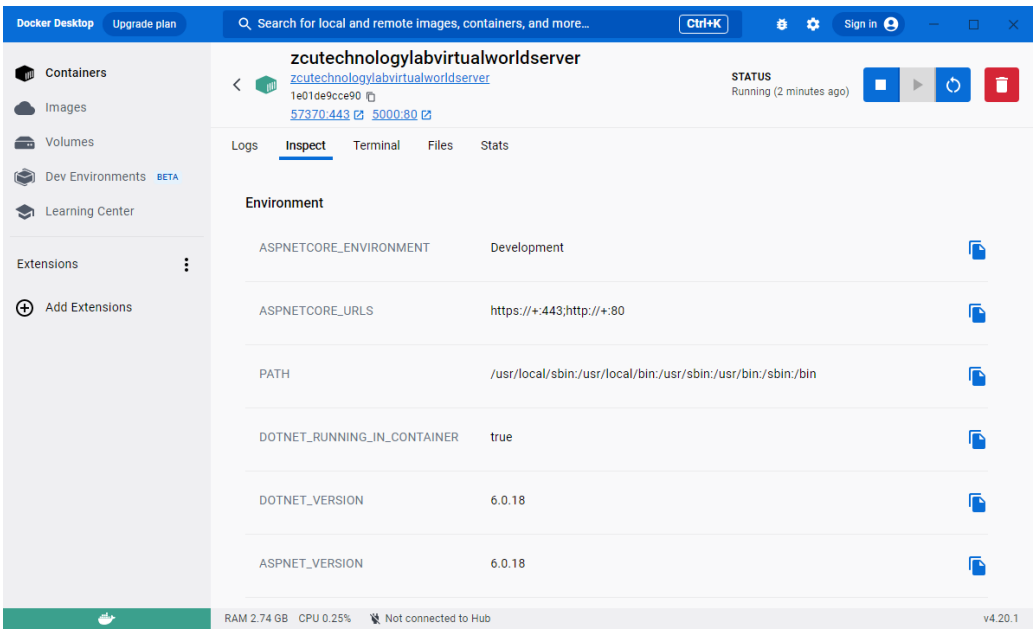
Obrázek A.6: Spuštění kontejneru v Docker Desktop



Obrázek A.7: Výběr kontejneru v Docker Desktop



Obrázek A.8: Logy v Docker Desktop



Obrázek A.9: Vlastnosti kontejneru v Docker Desktop

A.2 Komunikační API

Server vystavuje komunikační rozhraní veřejně. Nástrojem Swagger je možné na rozhraní nahlédnou a vykoušet odeslání dotazů na server. Na stejném zařízení, kde běží Docker, otevřete prohlížeč a do adresy zadejte `http://localhost:5000/swagger`. Pokud je vše správně nastaveno, měla by se otevřít webová stránka se Swaggerem na obr. A.10.

Otevřete POST /api/objects, zde uvidíte formát objektu v těle zprávy a možnost vyzkoušet dotaz. Klikněte na „Try it out“ a nastavte hodnoty na

Zdrojový kód A.3: Testovací JSON

```
{
  "Name": "testWorldObject",
  "Position": {
    "X": 1,
    "Y": 2,
    "Z": 3
  },
  "Rotation": {
    "X": 4,
    "Y": 5,
    "Z": 6
  },
  "Scale": {
    "X": 7,
    "Y": 8,
    "Z": 9
  },
  "Type": "Bitmap",
  "Properties": {
  }
}
```

Klikněte na tlačítko „Execute“. Jděte do Docker Desktop, otevřete logy a zkontrolujte, že výpis v A.4 odpovídá.

Výpis A.4: Logy

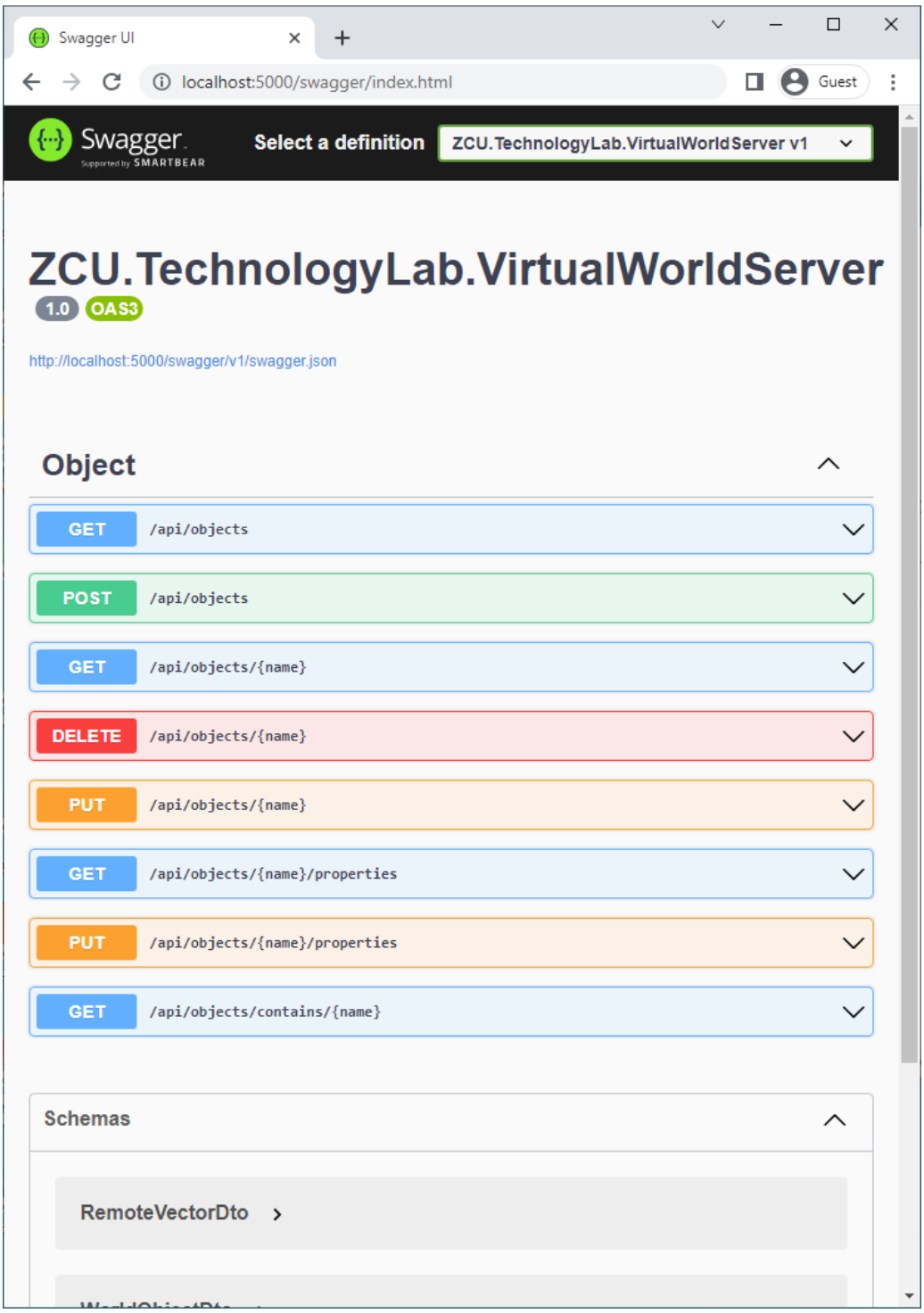
```
2023-06-22 00:41:40 info: Microsoft.Hosting.Lifetime[0]
2023-06-22 00:41:40       Application is shutting down...
2023-06-22 00:41:43 info: Microsoft.Hosting.Lifetime[14]
2023-06-22 00:41:43       Now listening on: https://[::]:443
2023-06-22 00:41:43 info: Microsoft.Hosting.Lifetime[14]
2023-06-22 00:41:43       Now listening on: http://[::]:80
2023-06-22 00:41:43 info: Microsoft.Hosting.Lifetime[0]
2023-06-22 00:41:43       Application started. Press Ctrl+C
   to shut down.
```

```
2023-06-22 00:41:43 info: Microsoft.Hosting.Lifetime[0]
2023-06-22 00:41:43       Hosting environment: Development
2023-06-22 00:41:43 info: Microsoft.Hosting.Lifetime[0]
2023-06-22 00:41:43       Content root path: /app/
2023-06-22 00:54:41 info: ZCU.TechnologyLab.VirtualWorldServ
   er.Services.WorldObjectService[0]
2023-06-22 00:54:41       Adding object testWorldObject to
    the server
```

Logy potvrzují, že byl objekt na server přidaný. Nyní ho zkusíme získat zpět. Otevřete GET /api/objects/{name}. Klikněte na „Try it out“ a do parametru name zadejte testWorldObject. Klikněte na tlačítko „Execute“. Tělo odpovědi by mělo být stejné jako testovací JSON v ukázce A.3.

Smažte objekt ze serveru. Otevřete DELETE /api/objects/{name}. Klikněte na „Try it out“. Do jména zadejte testWorldObject. Klikněte na „Execute“ a zkontrolujte, že vrácený stavový kód je 200 a tělo odpovědi obsahuje text „Object removed“.

Můžete znovu zkontrolovat logy. Budou v nich všechny prováděné akce.



Obrázek A.10: Swagger

Příloha 2 - Metodický materiál základní úlohy pro správu serveru

B

Úloha Vás provede základním ovládáním aplikace pro správu centrálního systému a nechá Vás také nahlédnout, jak pracovat přímo s centrálním systémem vně aplikace.

B.1 Úkol 1

Otevřete webový prohlížeč a do vstupního políčka pro webovou adresu zadejte `http://adresa-serveru:5000/swagger`. Pokud neznáte adresu serveru, tak na počítači, kde server běží, otevřete příkazovou řádku (program cmd.exe) a zadejte příkaz `ipconfig`. Najděte ve výpisu „Ethernet adapter Ethernet“ nebo „Wireless LAN adapter WiFi“. V nich je uvedena „IPv4 Address“. Tu je nutné zadat do prohlížeče. Když se Vám webová stránka se Swaggerem neotevře, tak v Dockeru server zapněte.

Řešení Příklad na vyhledání IP adresy v příkazové řádce je zobrazen ve výpisu B.1. Adaptérů může být ve výpisu více. Pokud se nepovede nalézt „Ethernet adapter Ethernet“ ani „Wireless LAN adapter WiFi“ je možné, že zařízení není připojené k síti.

Výpis B.1: IP konfigurace

```
C:\WINDOWS\system32>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : home
```

```
   Link-local IPv6 Address . . . . . :
   fe80::4ce0:5820:252e:d786%18
   IPv4 Address. . . . . . . . . . . : 192.168.0.149
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.0.1

C:\WINDOWS\system32>
```

Po zadání správné IP adresy do webového prohlížeče by se v případě spuštěného serveru zobrazila stránka na obr. B.1.

Pokud se stránka nezobrazila, zkontrolujte, zda kontejner zcutechnologylabvirtualworldserver uvnitř Dockeru běží. Pro návod na spuštění využijte Přílohu 1 - Uživatelská dokumentace serveru, kde je vše o práci s Dockerem popsáno.

B.2 Úkol 2

Spusťte aplikaci pro správu serveru.

Řešení Po spuštění aplikace by se měla zobrazit obrazovka na obr. B.2. Je vidět pouze lišta hlavního menu.

B.3 Úkol 3

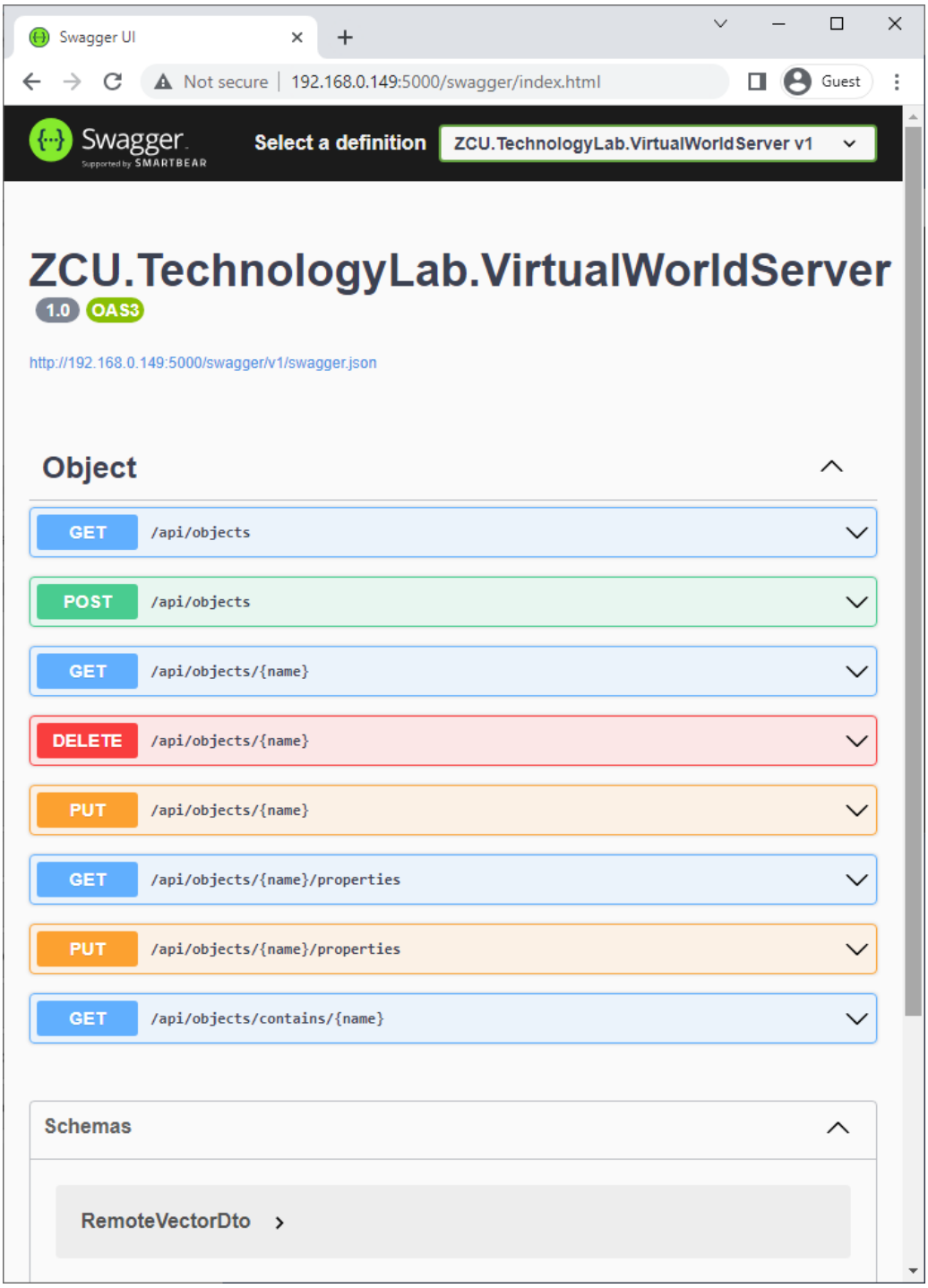
Zobrazte ovládací panel.

Řešení Kliknutím na tlačítko „Ovládací panel“ uvnitř lišty hlavního menu se panel zobrazí. Výsledný stav po otevření ovládacího panelu a ukazatel pozice tlačítka je k vidění na obr. B.3.

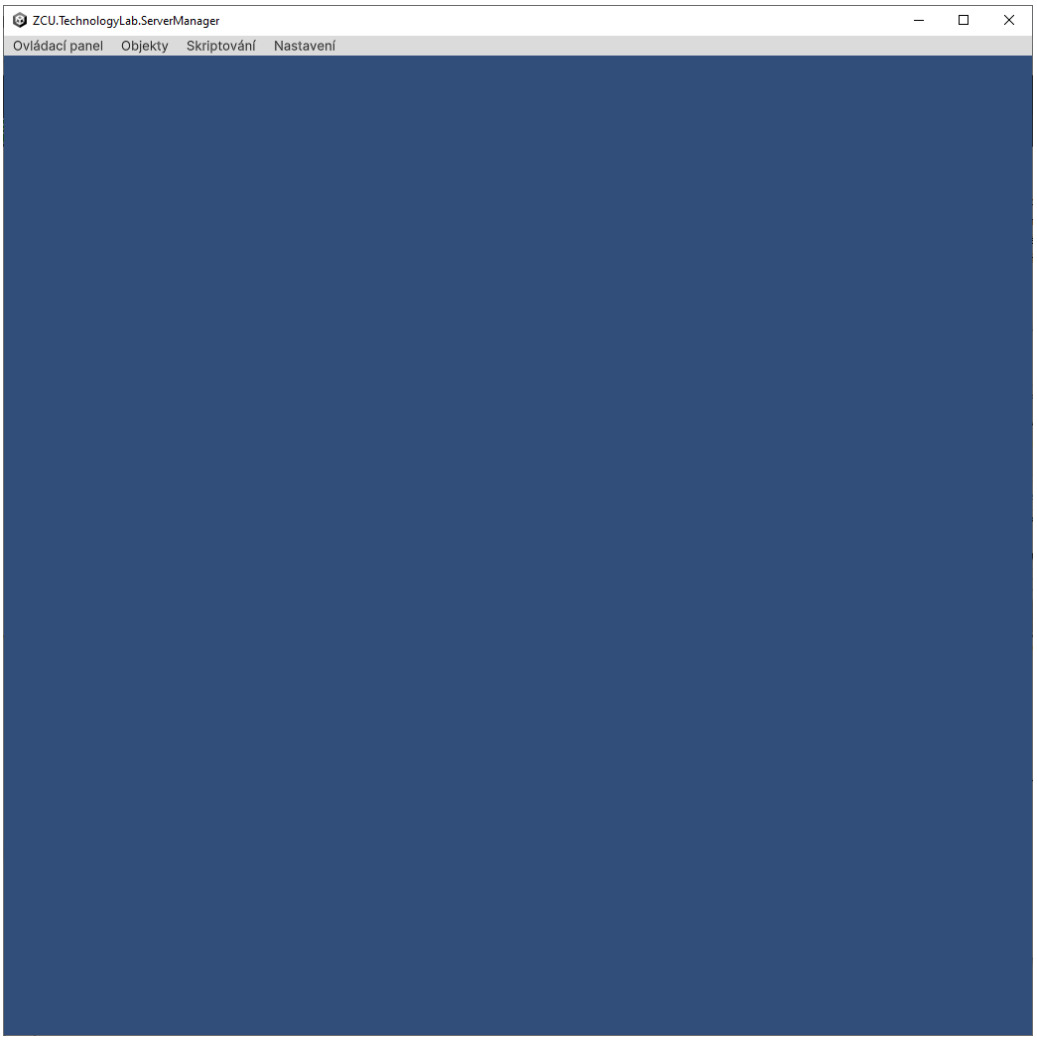
B.4 Úkol 4

Připojte se k serveru. Musíte zas zadat adresu (`http://adresa-serveru:5000/`), jako v úkolu jedna, když jste spouštěli Swagger v internetovém prohlížeči.

Řešení Připojení k serveru má tři kroky. V prvním kroku je nutné nastavit adresu serveru, na kterou se budete připojovat. Jedná se o stejnou adresu jako v úkolu 1. Jediný rozdíl je v tom, že tentokrát neobsahuje řetězec „swagger“. Na obr. B.4 je tento krok označený šipkou s číslem jedna. To je celá příprava potřebná před připojením. Nyní stačí kliknout na tlačítko „Připojit k serveru“. Pokud jste postupovali podle kroků návodu, mělo by přihlášení proběhnout bez problémů. Na obr. B.5 je zobrazen stav aplikace po navázání spojení.



Obrázek B.1: Řešení úkolu 1



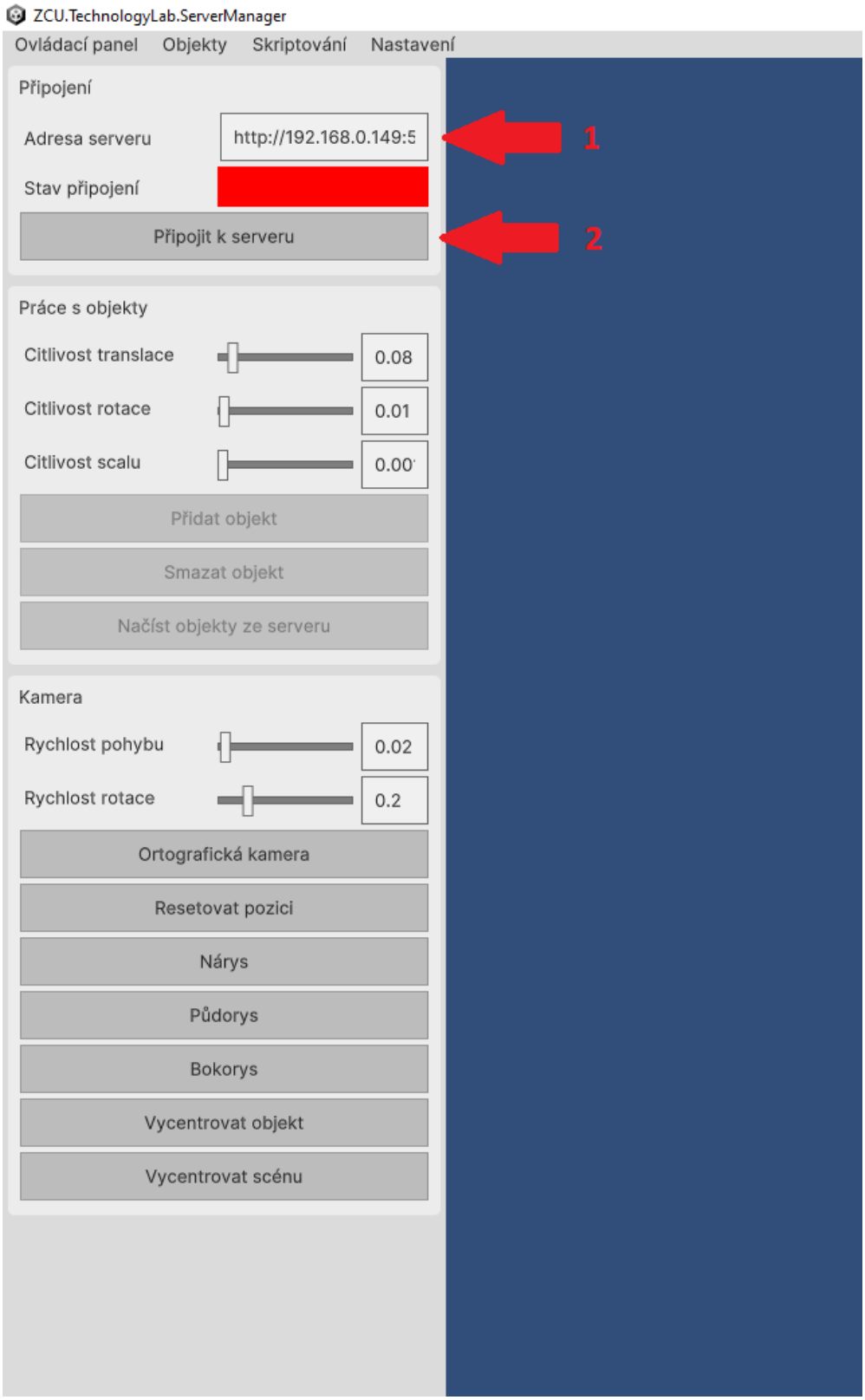
Obrázek B.2: Řešení úkolu 2

B.5 Úkol 5

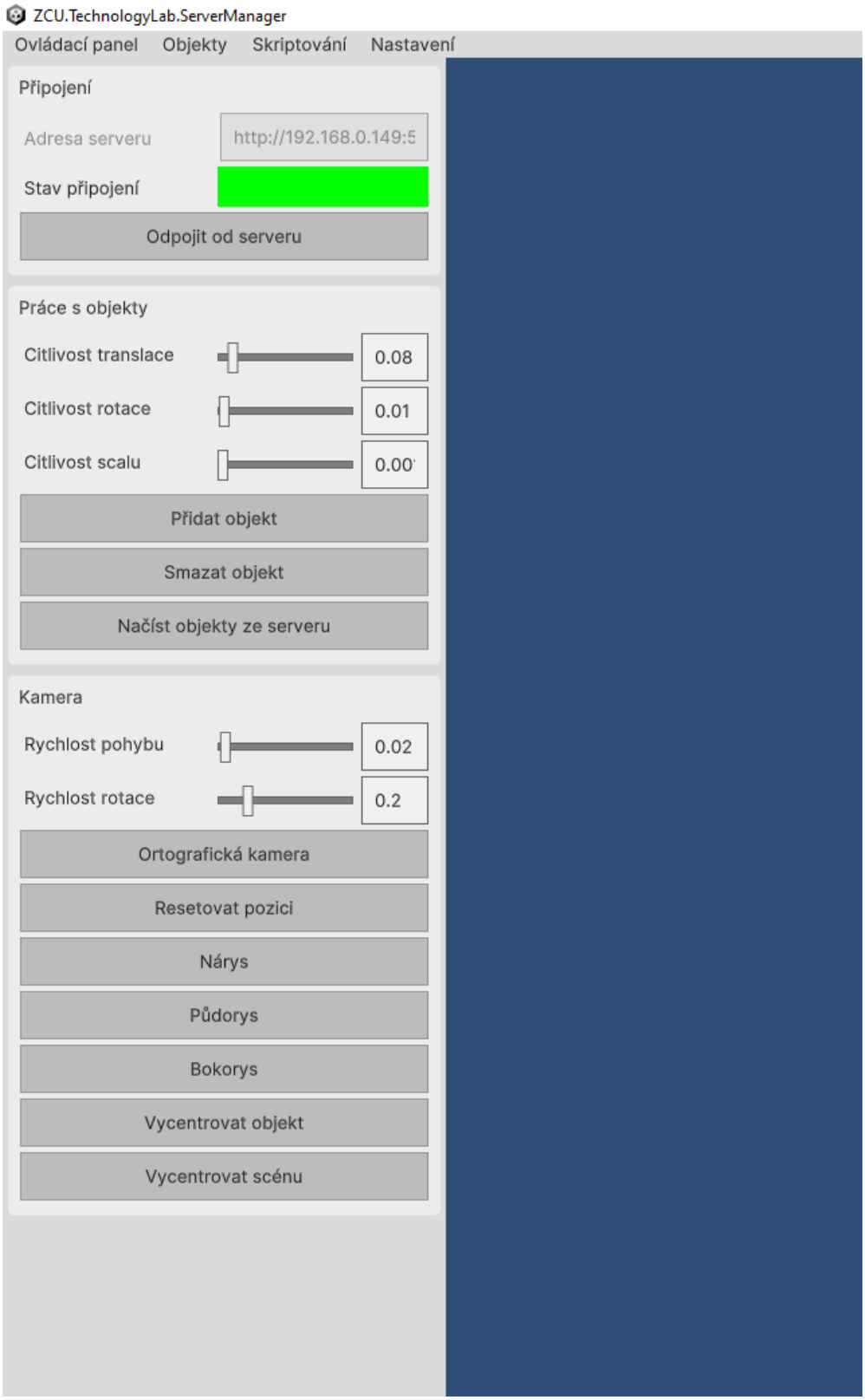
Přidejte do aplikace pro správu serveru novou mesh. Zkušební 3D model je uložený ve stejné složce, ze které jste aplikaci spouštěli, nebo můžete využít svůj vlastní.

Řešení Přidání objektu do aplikace se provádí tlačítkem „Přidat objekt“. To otevře dialogové okno pro výběr souboru. Zde je nutné nejprve přepnout typ na Mesh, jinak nebudou žádné 3D modely v adresářová struktuře viditelné. Když máte objekt vybraný kliknete na tlačítko „Otevřít“. Tento proces ukazuje obr. B.6. Po načtení se trojrozměrný model zobrazí jako na obr. B.7.

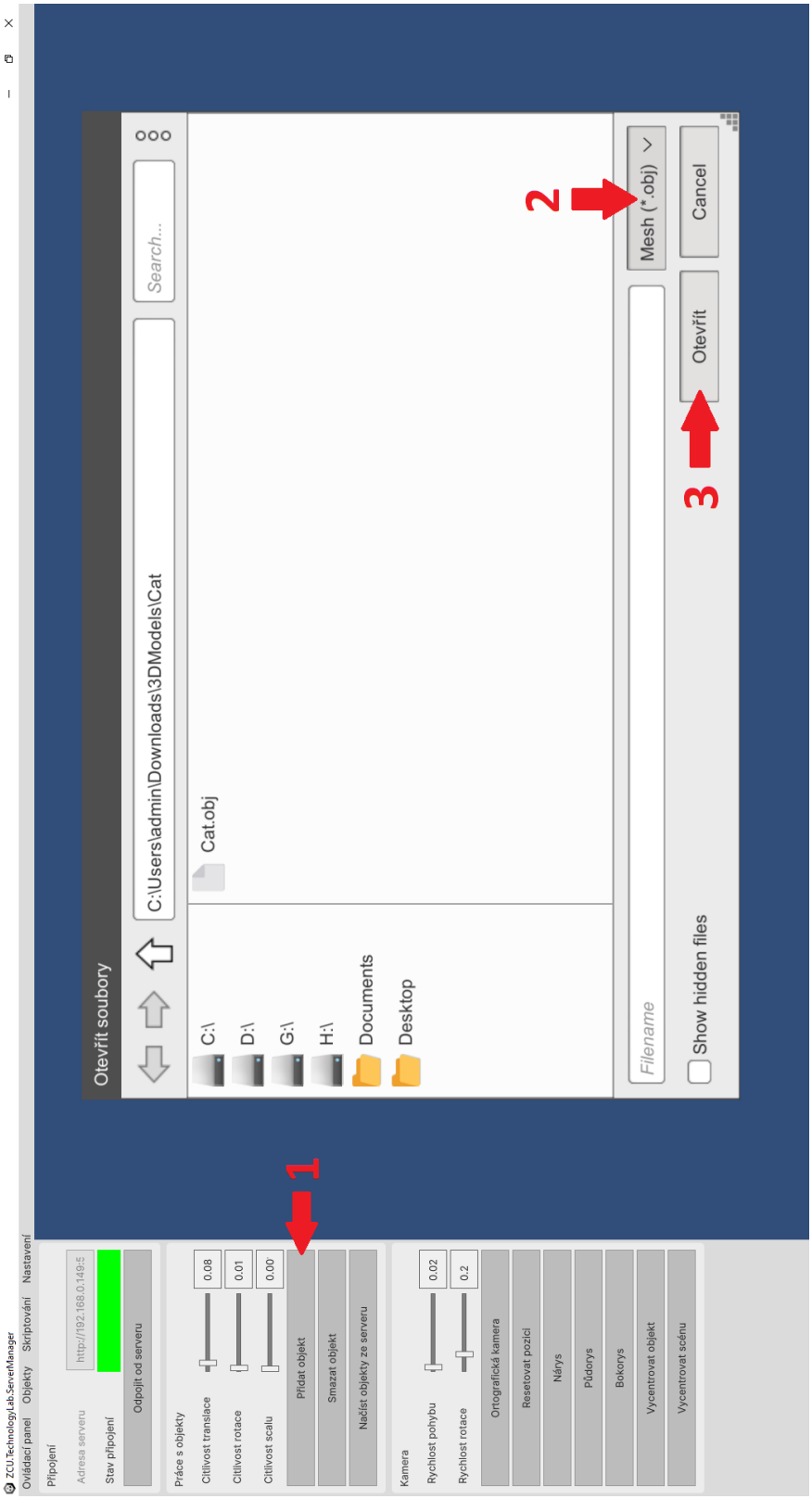
Obrázek B.3: Řešení úkolu 3



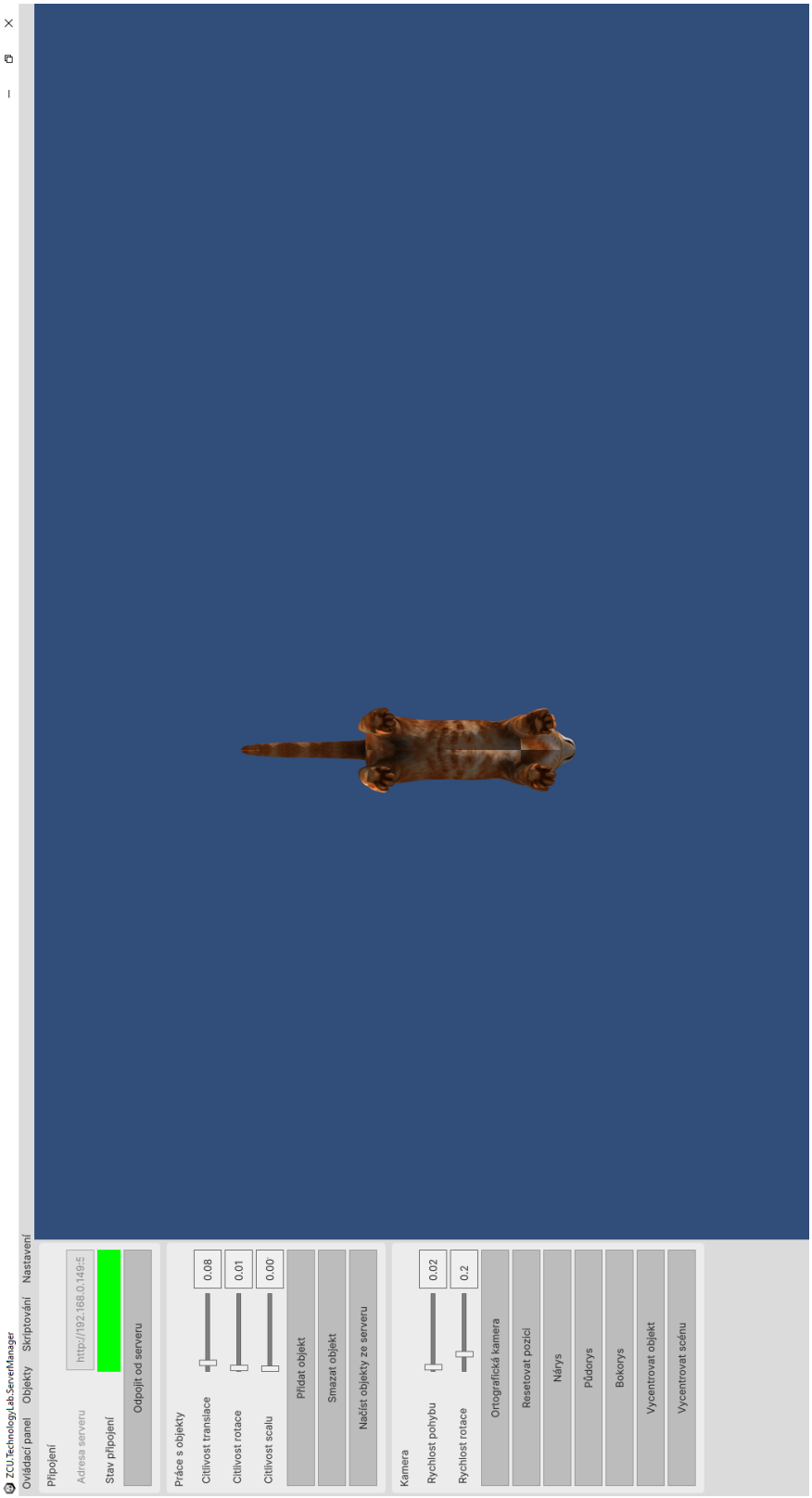
Obrázek B.4: Řešení úkolu 4 před připojením



Obrázek B.5: Řešení úkolu 4 po připojením



Obrázek B.6: Řešení úkolu 5 - otevření souboru



Obrázek B.7: Řešení úkolu 5 - zobrazení modelu

B.6 Úkol 6

Zobrazte si panel s objekty, aby jste zkontrolovali, že se mesh opravdu načetla a je v seznamu.

Řešení Panel s objekty jde zobrazit přes lištu hlavního menu. K zobrazení slouží položka „Objekty“. Načtený objekt se objeví v seznamu v pravém horním rohu. Na obr. B.8 je pozice tlačítka označena číslem jedna a načtený objekt v otevřeném panelu číslem dva.

B.7 Úkol 7

Vyberte objekt (můžete na něj kliknout nebo ho vybrat v seznamu) a přesuňte ho co nejblíže souřadnicím X = 0.5, Y = 0 a Z = 0. V ose Y objekt rotujte o 90°. Nakonec upravte měřítko (scale) na 1.5. V ovládacím panelu můžete upravit citlivost operací. Posunutí objektu se provádí pohybem myši při držení levého tlačítka na označeném objektu. K rotaci poslouží držení prostředního tlačítka myši. Pro změnu měřítka musíte k otáčet prostředním kolečkem myši.

Řešení Hodnoty pozice, rotace a scale jsou umístěny v pravém dolním rohu obrazovky v panelu s vlastnostmi objektu. Model by měl skončit zhruba na stejném místě jako na obr. B.9. Pokud používáte jiný objekt situace se může lišit, ale hodnoty v panelu s vlastnostmi musí být stejné.

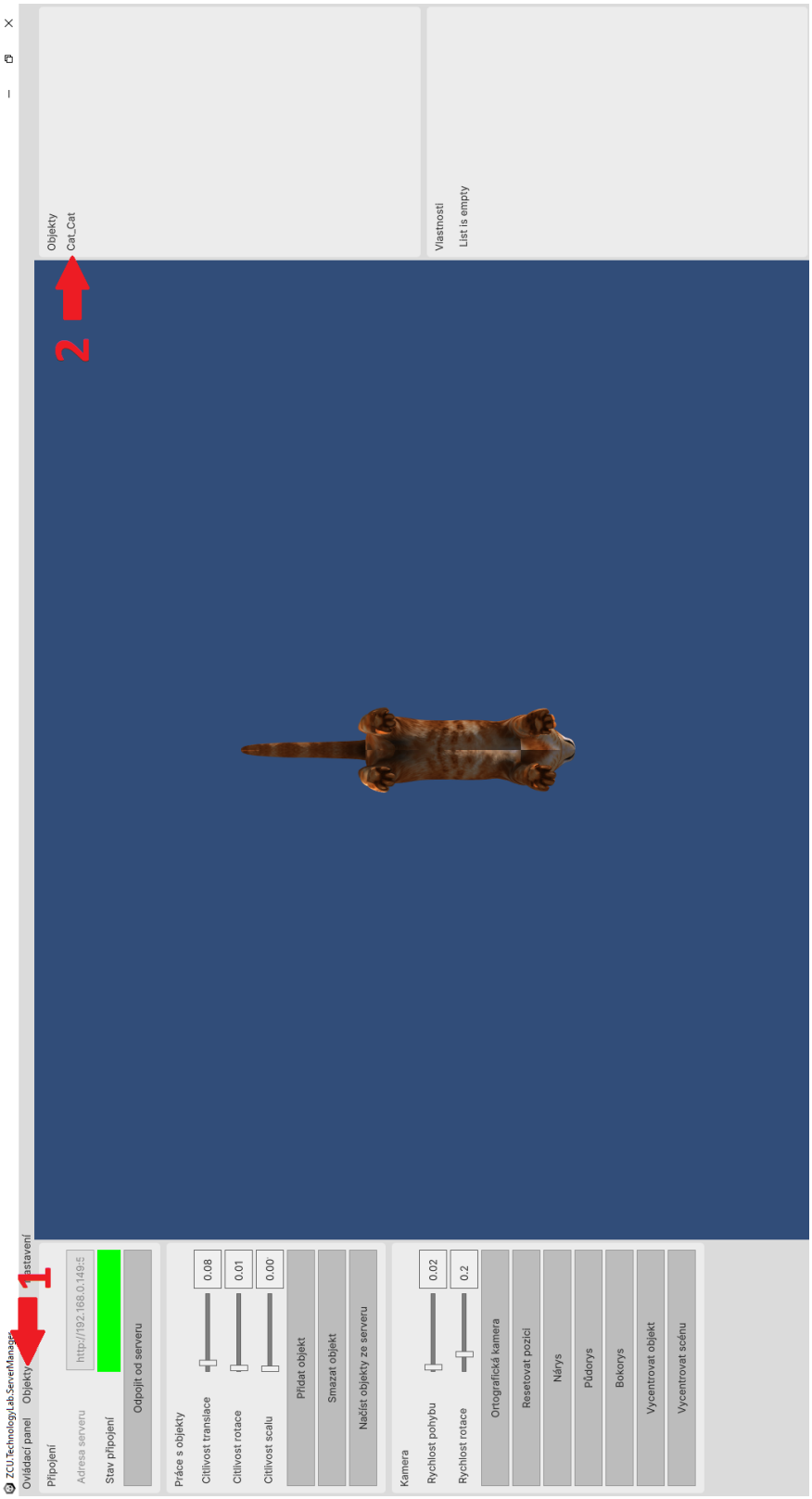
B.8 Úkol 8

Vycentrujte objekt na obrazovce (můžete pohybovat s kamerou ručně, nebo využít tlačítka v ovládacím panelu). Můžete hýbat pouze kamerou. Pozice, rotace a scale objektu musejí zůstat stejné. Pro pohyb kamery slouží klávesy WASD nebo šipky. Pokud se budete chtít rozhlížet po okolí přidržte pravé tlačítko myši.

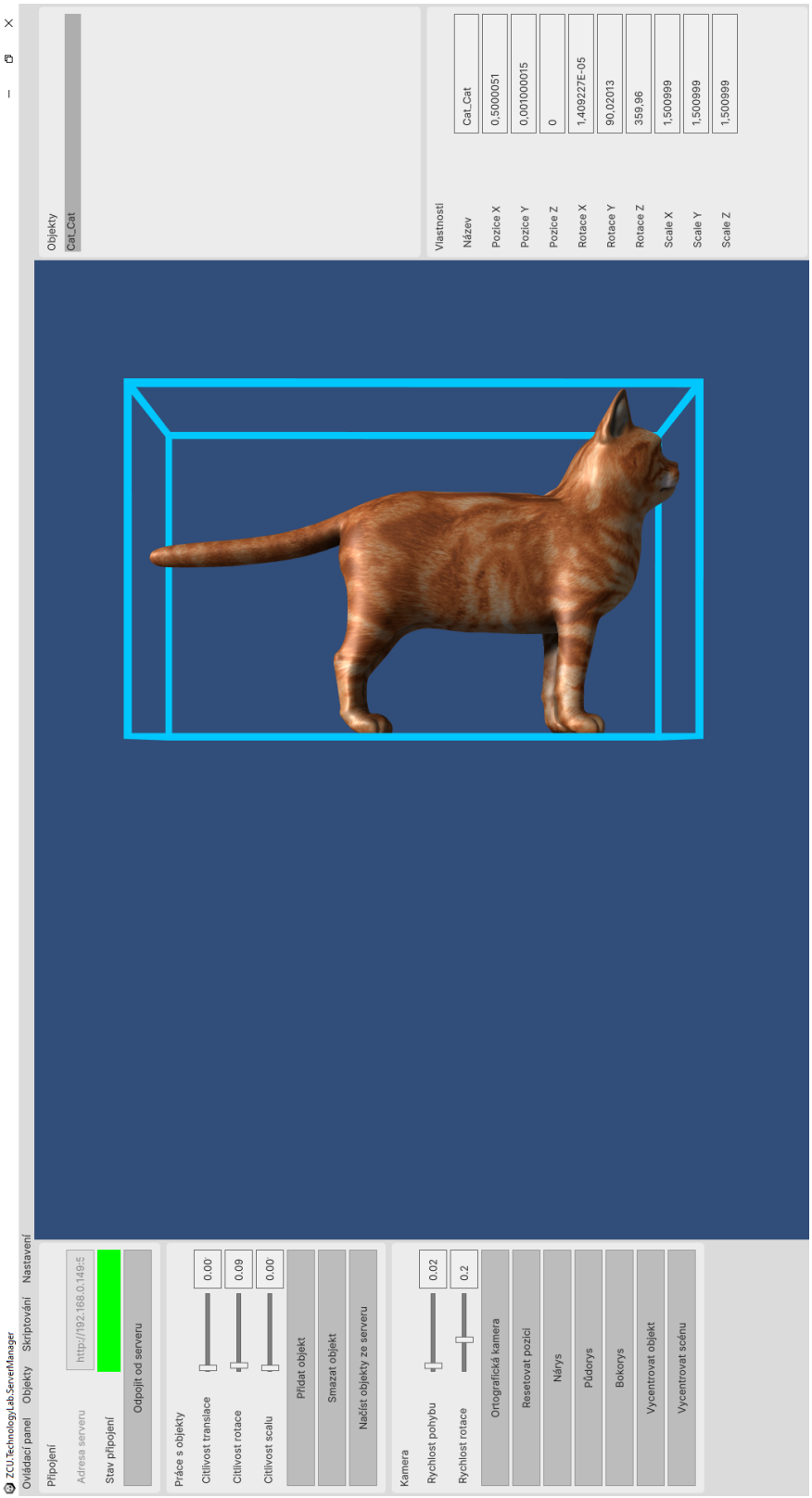
Řešení Výsledný stav by měl vypadat podobně jako na obr. B.10. Zda se změnila transformace nebo ne, se zkontroluje v panelu s vlastnostmi objektu.

B.9 Úkol 9

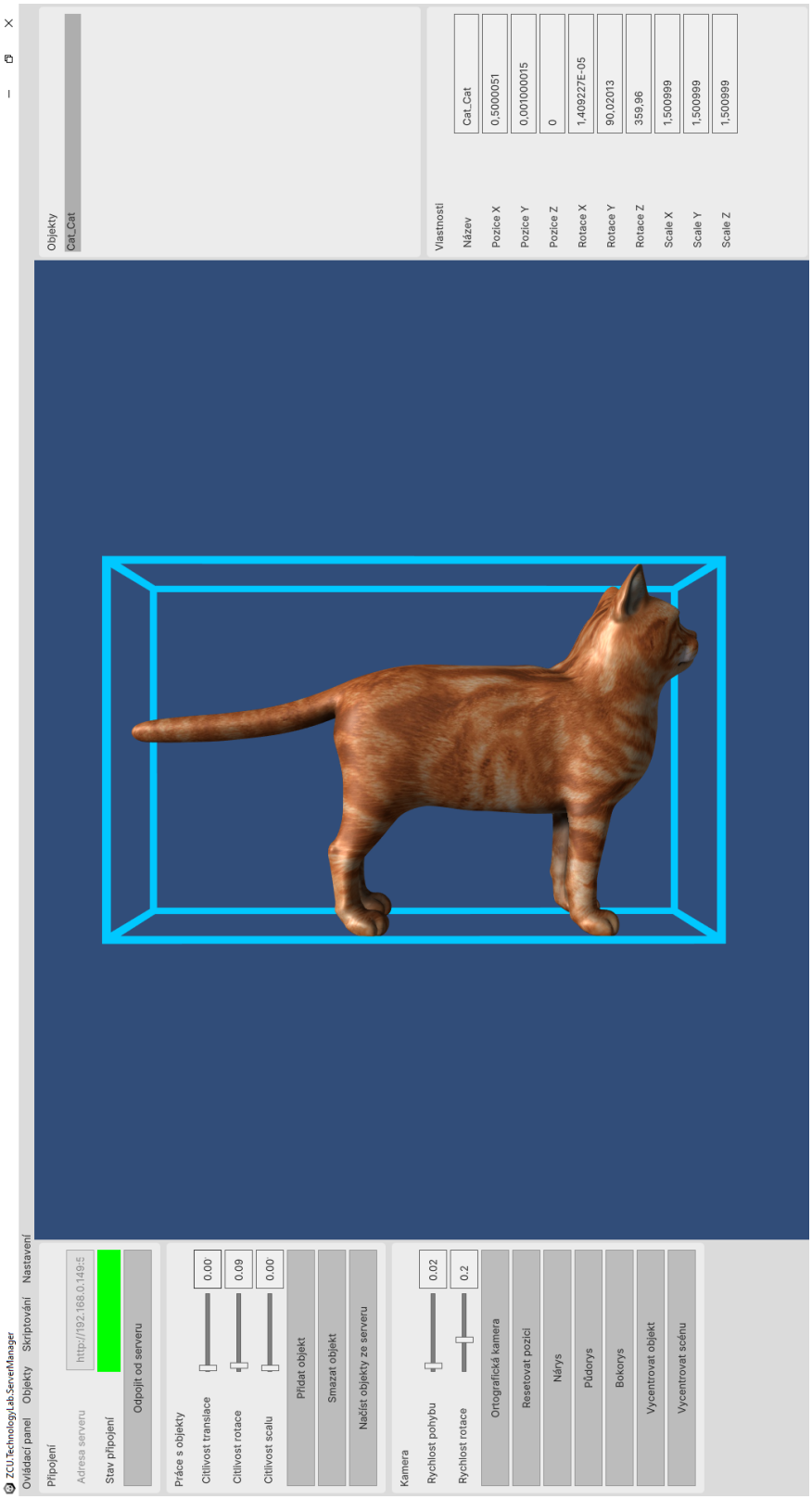
Vraťte se do Swaggeru a otevřete `GET /api/objects/{name}`. Zde můžete vidět nastavení požadavku do centrálního systému, který musejí programy vykonat, aby mohly stáhnout objekt ze systému. Dotaz má jeden parametr a je jím jméno objektu.



Obrázek B.8: Řešení úkolu 6



Obrázek B.9: Řešení úkolu 7



Obrázek B.10: Řešení úkolu 8

Vyzkoušejte dotaz pomocí tlačítka „Try it out“ ve Swaggeru a zadejte jméno objektu, který jste nahráli do aplikace. Můžete ho zjistit zpět v aplikaci v pravém panelu objektů. Pokud jste jméno zadali správně, dostanete odpověď. V těle odpovědi by měl být serializovaný objekt, který jste do aplikace nahráli. Zkontrolujte, že má změněnou pozici, rotaci a scale na hodnoty, které jste nastavovali v aplikaci. (Z důvodu zaokrouhlení a chyby přesnosti reálných čísel v počítači se může hodnota drobně lišit.)

Řešení Tento úkol má více mezikroků. Pro každý z nich je připravena ukázková obrazovka. Na obr. B.11 je přesně ukázáno, který z dostupných dotazů se má v úkolu otevřít. Na dalším obr. B.12 je vidět nastavení dotazu `GET /api/objects/{name}` a pomocí šipky je zvýrazněné tlačítko „Try it out“ zmíněné v zadání, aby se na stránce snadno identifikovalo. Po kliknutí na zvýrazněné tlačítko se přejde na obr. B.13. Pomocí šipek s čísly je popsána posloupnost kroků na dané obrazovce. Jméno „Cat_Cat“ je pouze ilustrativní, pokud použijete vlastní objekt, můžete mít jiné jméno. Po dokončení druhého kroku - kliknutí na tlačítko „Execute“, se vrátí objekt ze serveru. Jeho tvar je zvýrazněná šipkou. Když porovnáte obr. B.9 a B.14, je zřejmé že vlastnosti objektu se shodují.

B.10 Úkol 10

Nyní zavřete aplikaci pro správu serveru. Znovu ji spusťte a opětovně připojte k serveru. Otevřete panel s objekty a zkontrolujte, že je prázdný.

Řešení Kroky se pouze opakují z předchozích úkolů. Pro zapnutí zkontrolujte Úkol 2. Připojení je popsané v Úkolu 4. Otevření panelu s objekty je v Úkolu 6.

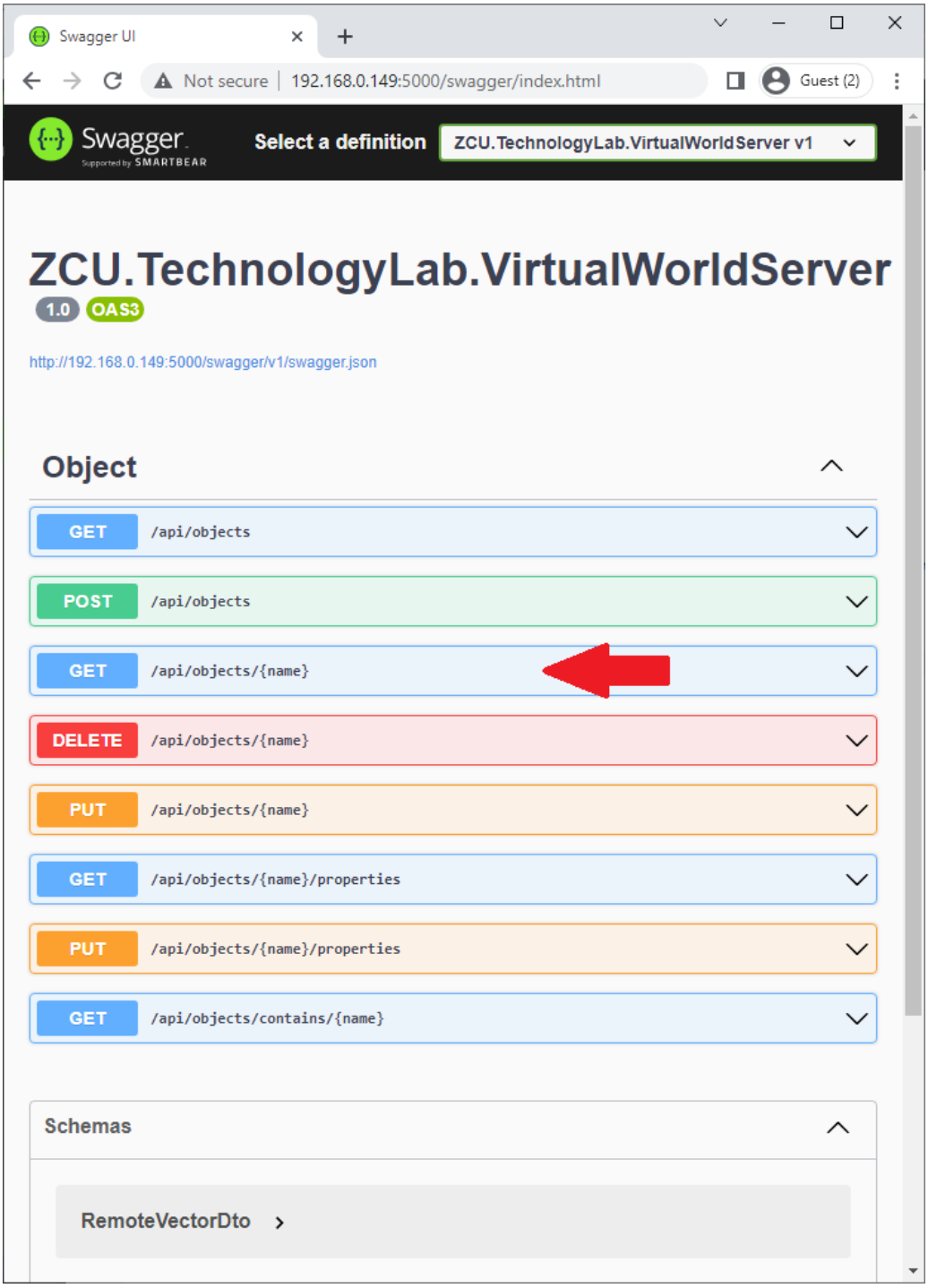
B.11 Úkol 11

Nahrajte do aplikace pro správu serveru stejnou mesh jako minule. Objeví se Vám chybové hlášení, protože stejná mesh v systému již existuje.

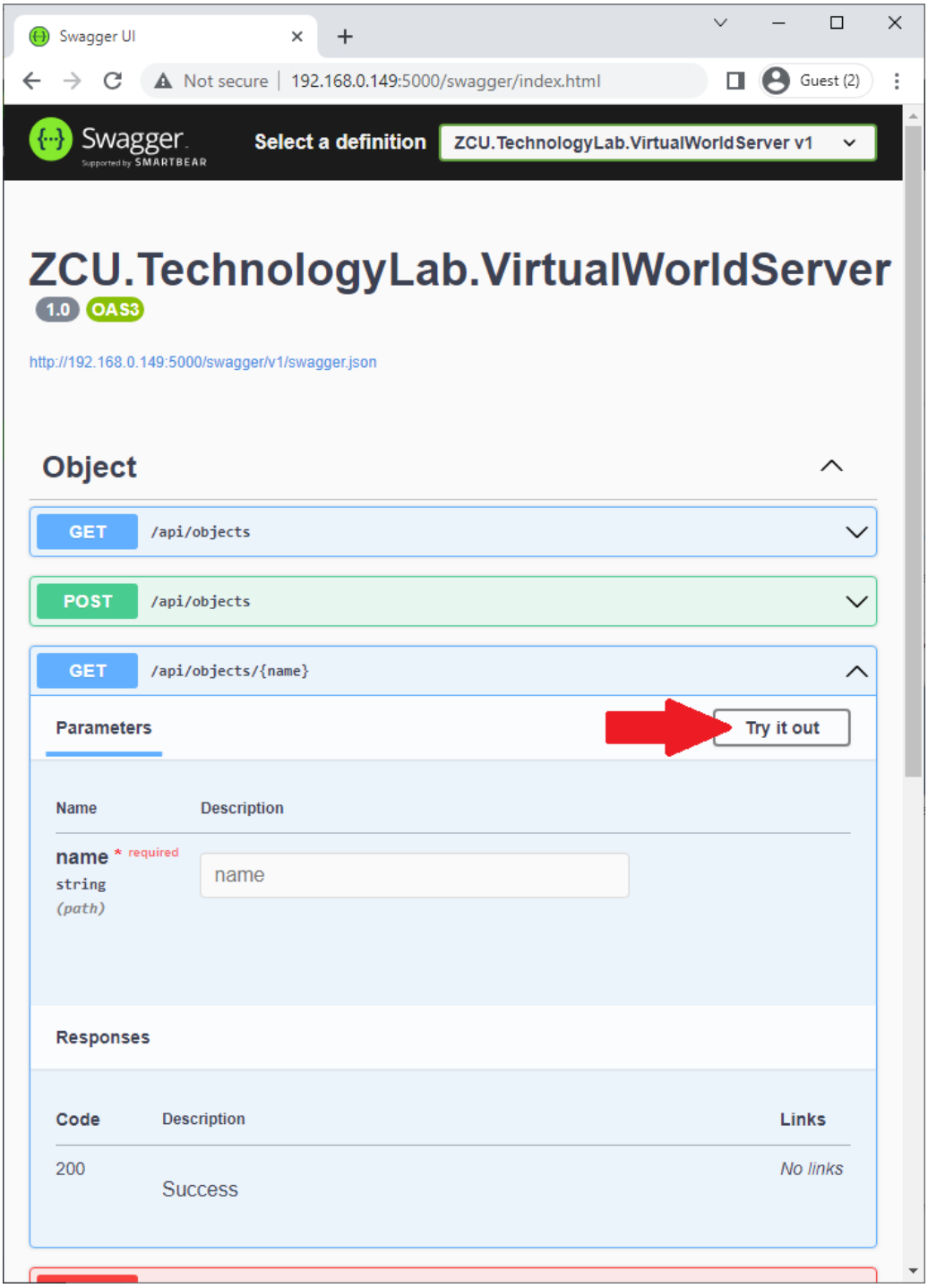
Řešení Chybové hlášení je zobrazené na obr. B.15.

B.12 Úkol 12

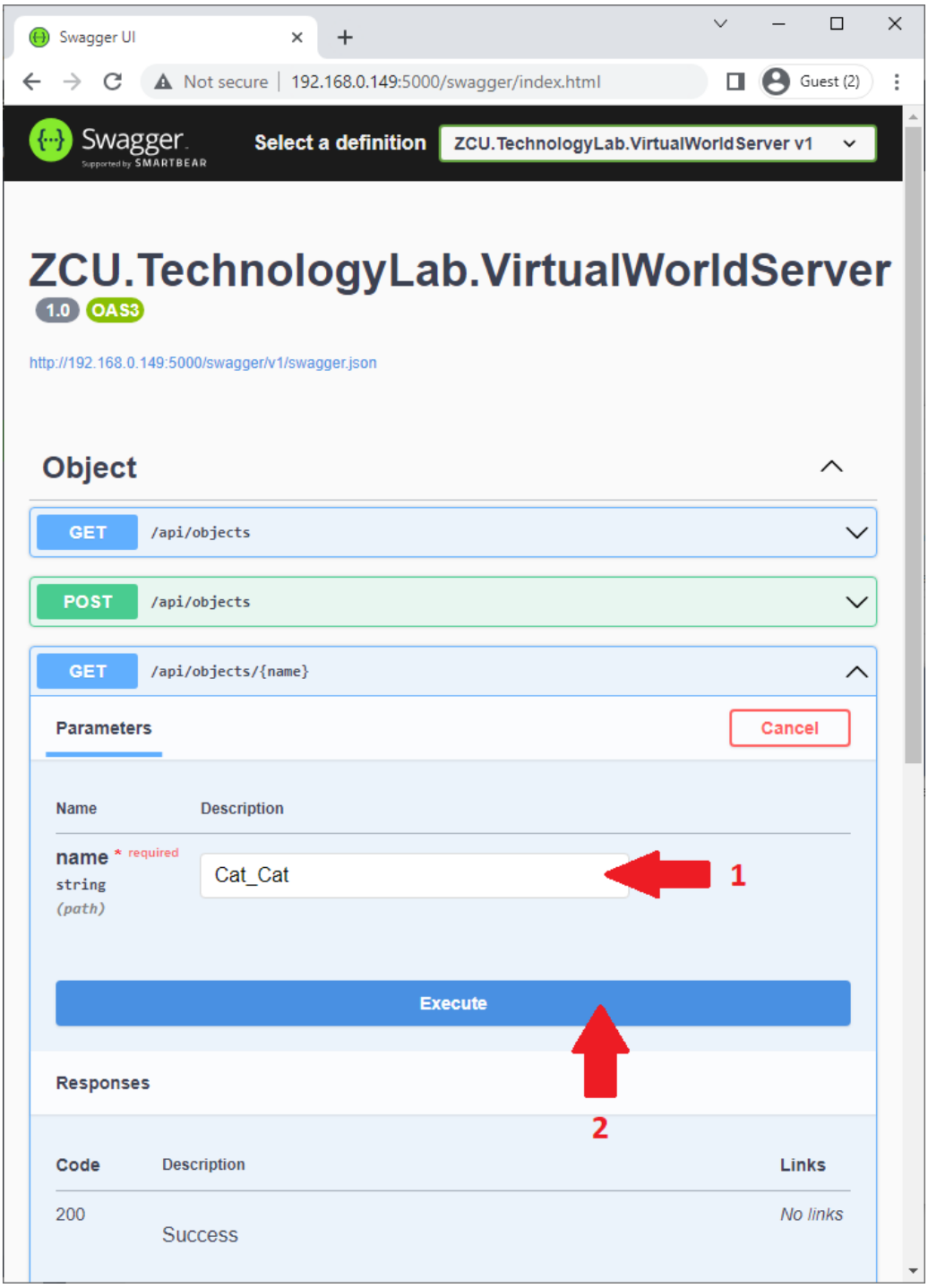
Jelikož mesh již existuje, můžeme ji v kontrolním panelu načíst ze serveru. Klikněte na tlačítko a okamžik počkejte. Měl by se Vám na obrazovce zobrazit nahraný objekt.



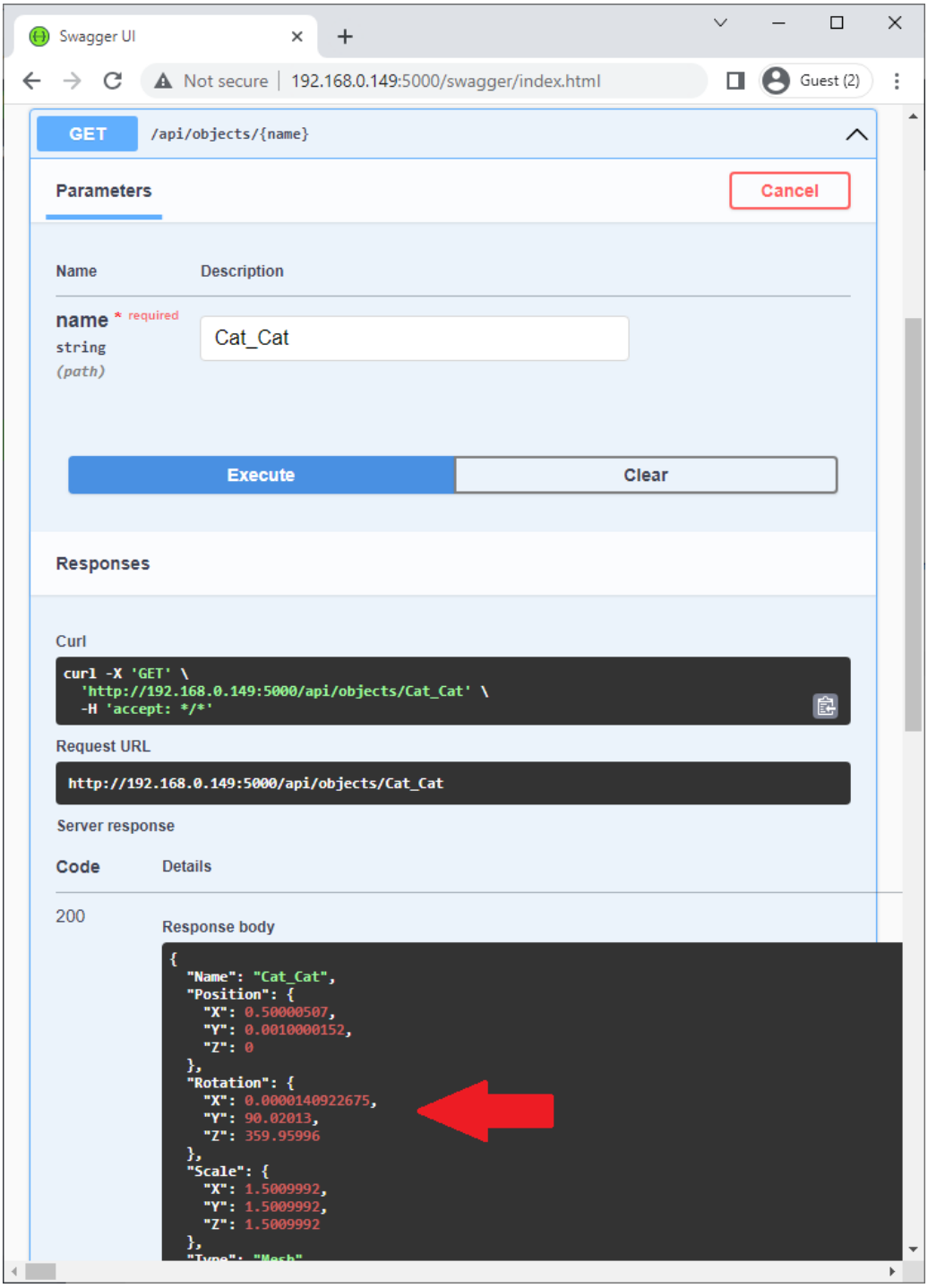
Obrázek B.11: Řešení úkolu 9 - Vybrat GET



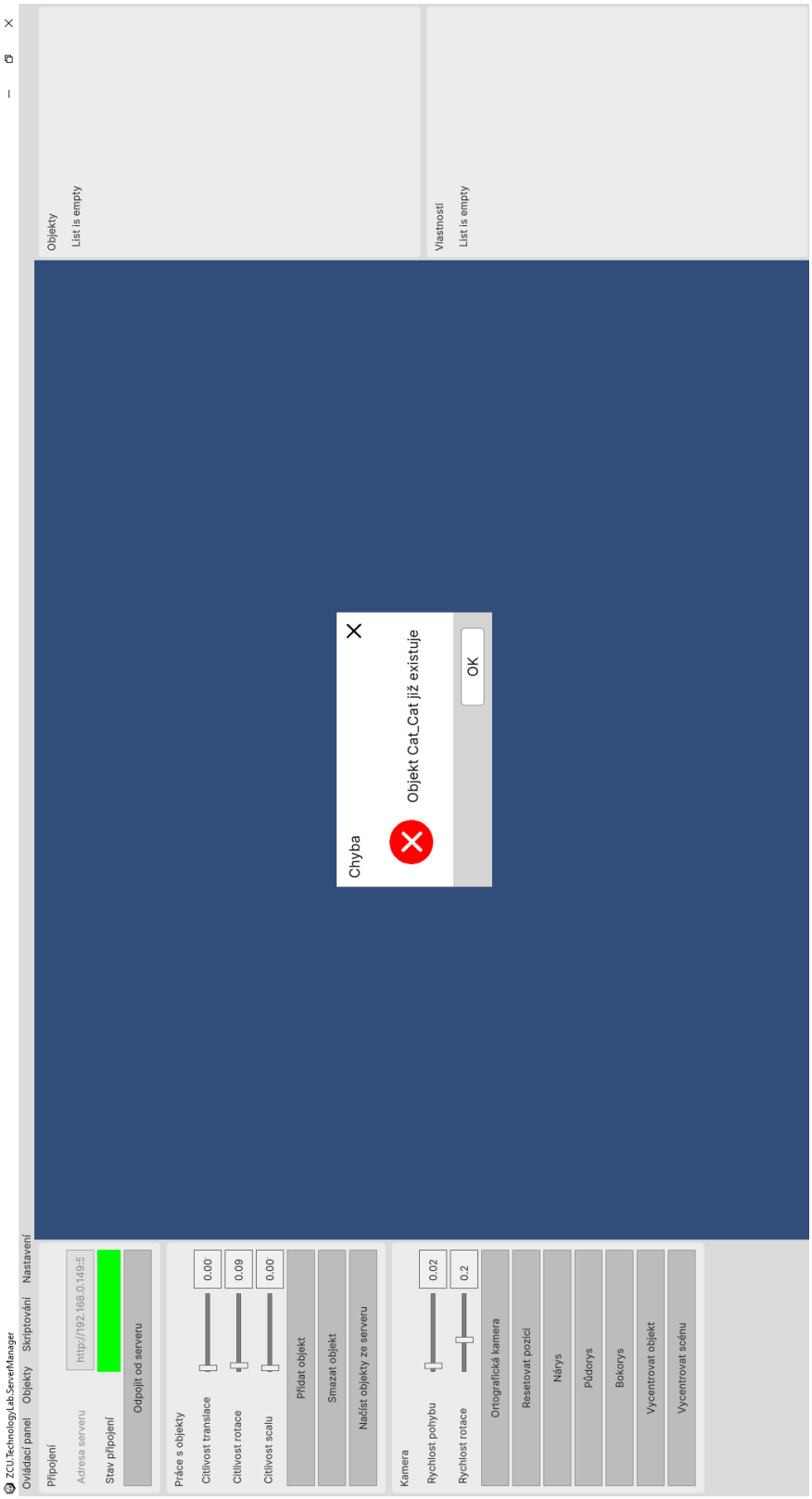
Obrázek B.12: Řešení úkolu 9 - Tlačítko Try it out



Obrázek B.13: Řešení úkolu 9 - Tlačítko Execute



Obrázek B.14: Řešení úkolu 9 - Výsledek



Obrázek B.15: Řešení úkolu 11

Zkontrolujte, že přibyl také v seznamu objektů a že má stejnou pozici jakou jste nastavili.

Řešení Výsledná obrazovka v úkolu 12 musí být totožná s obr. B.9. Mezi poslední změnou objektu a jeho následným stažením se nesmí nic změnit.

B.13 Úkol 13

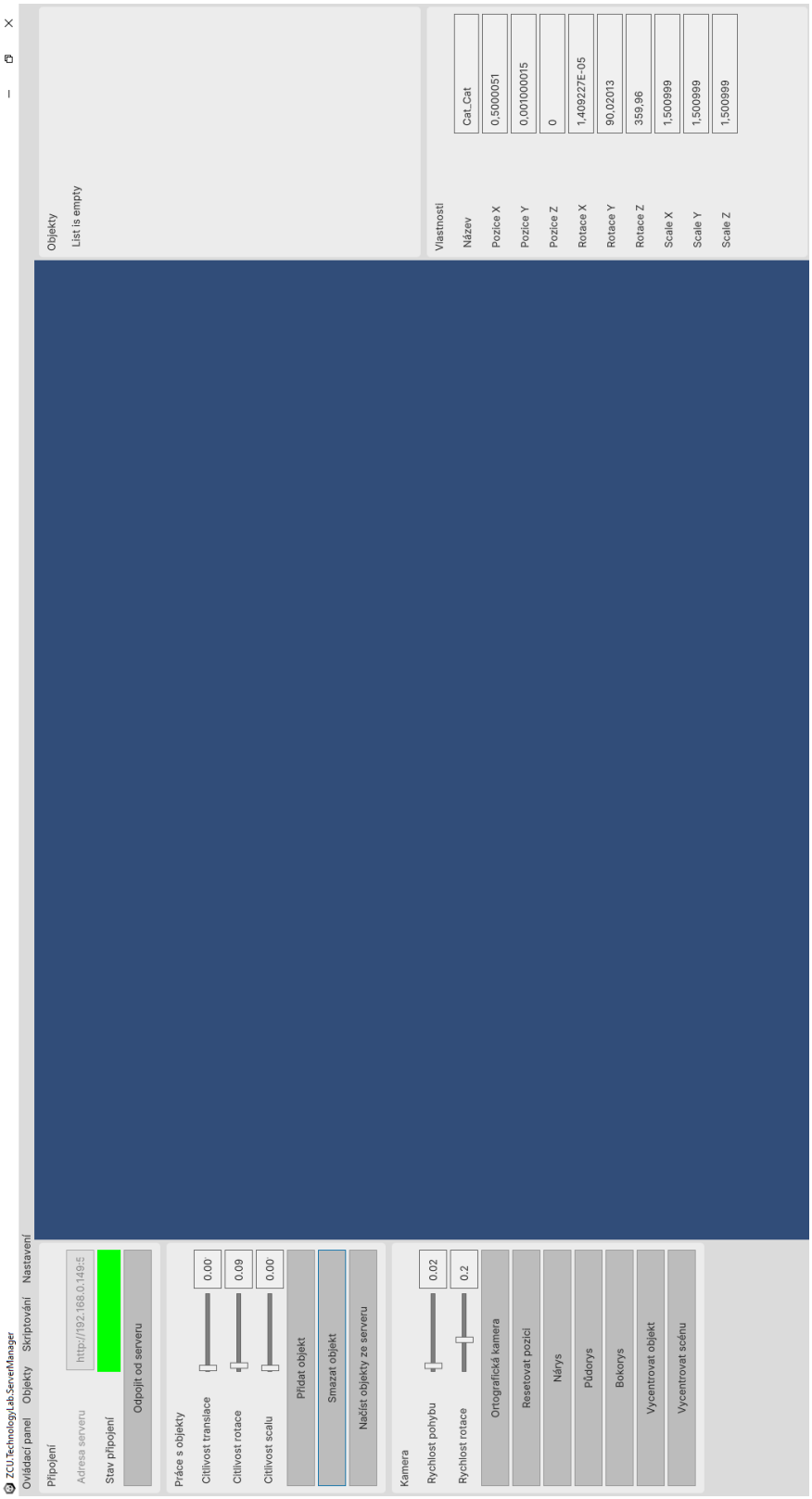
Objekt vyberte a smažte ho pomocí tlačítka v kontrolním panelu.

Řešení Objekt musí po smazání zmizet z obrazovky a ze seznamu objektů. Stav aplikace po smazání objektu je k vidění na ukázce B.16. To, že seznam vlastností zůstal i po odstranění nezměněn je v pořádku. Díky tomu lze zpětně zjistit, jaký objekt byl smazán, alespoň do doby než se vybere jiný objekt. Při tom se vlastnosti přepíší.

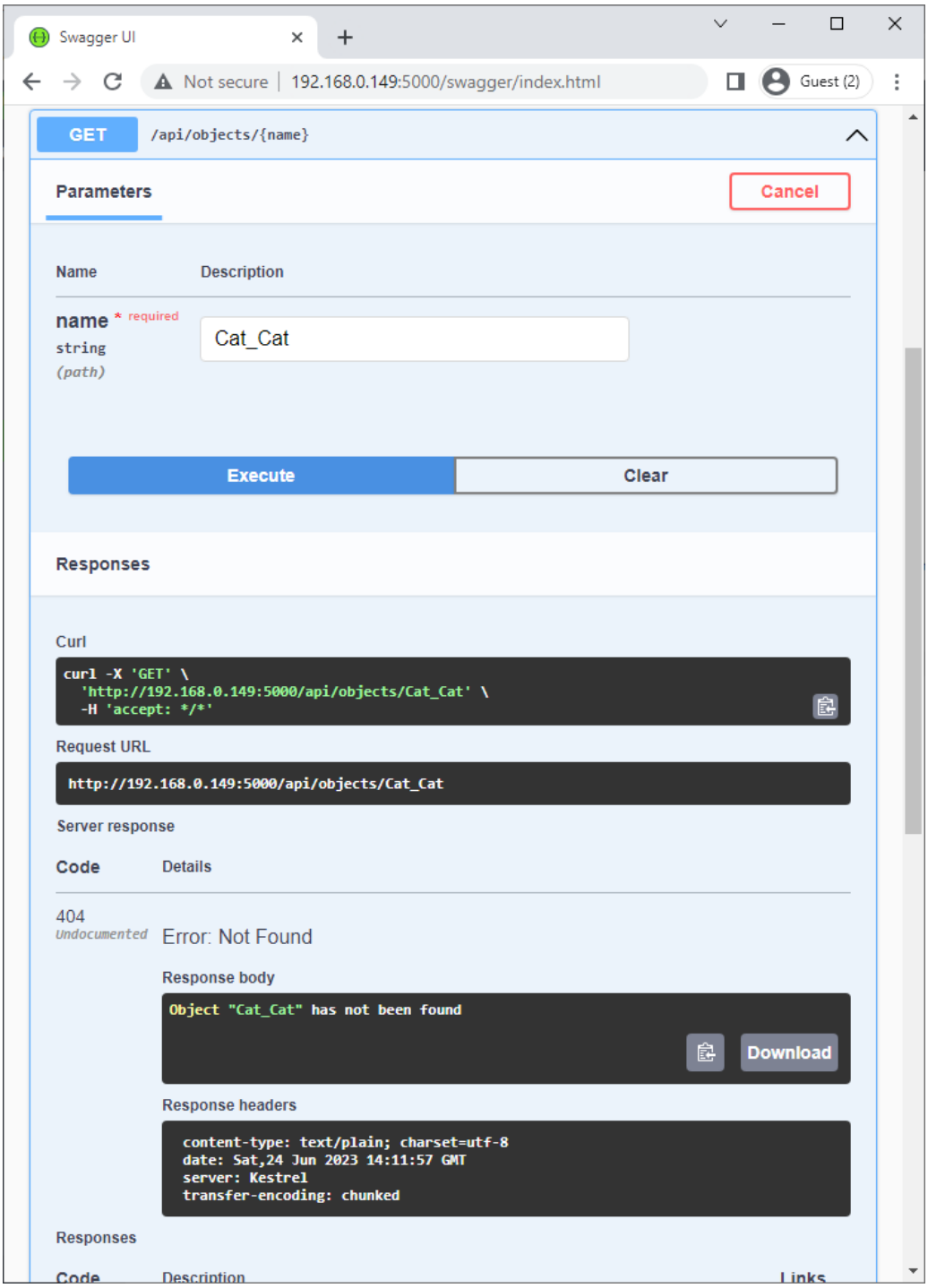
B.14 Úkol 14

Vraťte se zpět do Swaggeru a opět spusťte stejný příkaz `GET /api/objects/{name}`. Měli byste v odpovědi dostat chybové hlášení, což znamená, že objekt se ze serveru smazal správně.

Řešení Tvar získaného chybového hlášení se musí shodovat s ukázkou na obr. B.17.



Obrázek B.16: Řešení úkolu 13



Obrázek B.17: Řešení úkolu 14

Příloha 3 - Metodický materiál navazující úlohy pro správu serveru — C

C.1 Úkol 1

Zapněte aplikaci pro správu systému. Připojte se k serveru a přidejte několik různých objektů. Pokud si nejste jistí, jak se to dělá, vraťte se k základní úloze.

Řešení Jak může taková obrazovka vypadat vidíte na obr. C.1.

C.2 Úkol 2

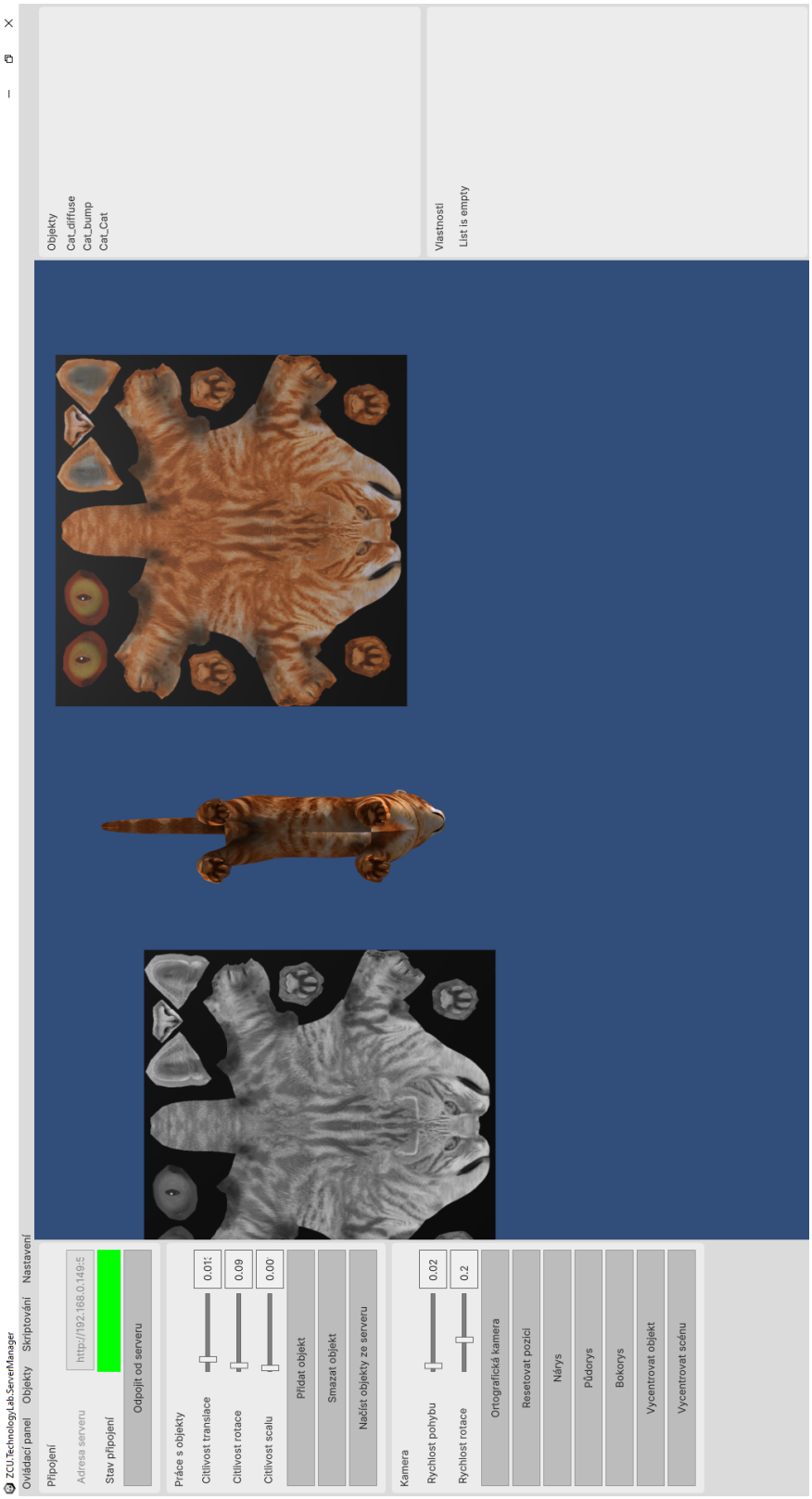
Otevřete panel pro skriptování.

Řešení V obr. C.2 je tlačítko, které skriptování zpřístupní, označeno šipkou. Na obrazovce lze také vidět již otevřený skriptovací panel.

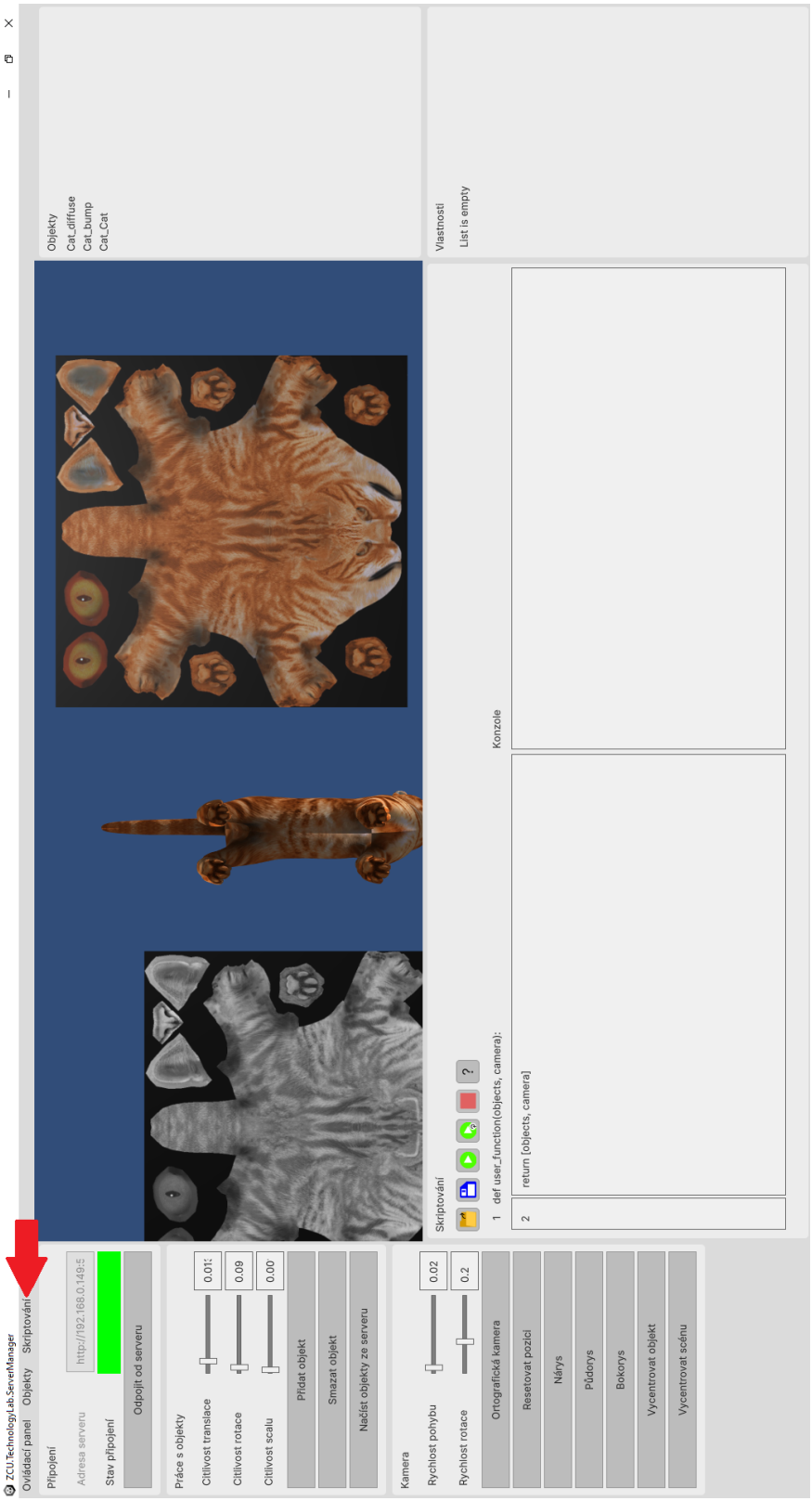
C.3 Úkol 3

Ve skriptovacím panelu lze vytvářet skripty v jazyce Python. Skript umožňuje vytvořit metodu, která pracuje s načtenými objekty a kamerou. Spusťte ukázkový kód pomocí třetího tlačítka v panelu nástrojů uvnitř skriptovacího okna. V konzoli by se měla objevit zpráva o úspěšném provedení kódu. Kód zatím nic nedělá a pouze vrací vstupní parametry.

Řešení Ukázkové řešení na obr. C.3 obsahuje očíslované kroky, které je nutné pro splnění Úkolu 3 podniknout.



Obrázek C.1: Řešení úkolu 1



Obrázek C.2: Řešení úkolu 2

Obrázek C.3: Řešení úkolu 3

C.4 Úkol 4

Záměrně udělejte chybu například odstraněním ukončovací závorky v návratové hodnotě a spusťte kód znovu. V konzoli by mělo přibýt chybové hlášení.

Řešení Chybové hlášení by mělo znít „'[' was never closed (<string>, line 2)". Jak výpis do konzole vypadá se můžete přesvědčit na obr. C.4.

C.5 Úkol 5

Pracujte s parametry funkce. `Objects` reprezentuje slovník, jehož klíčem jsou jména načtených objektů a hodnotou je instance typu `ObjectTransform`.

Zdrojový kód C.1: Transformace world objektu pro skriptování

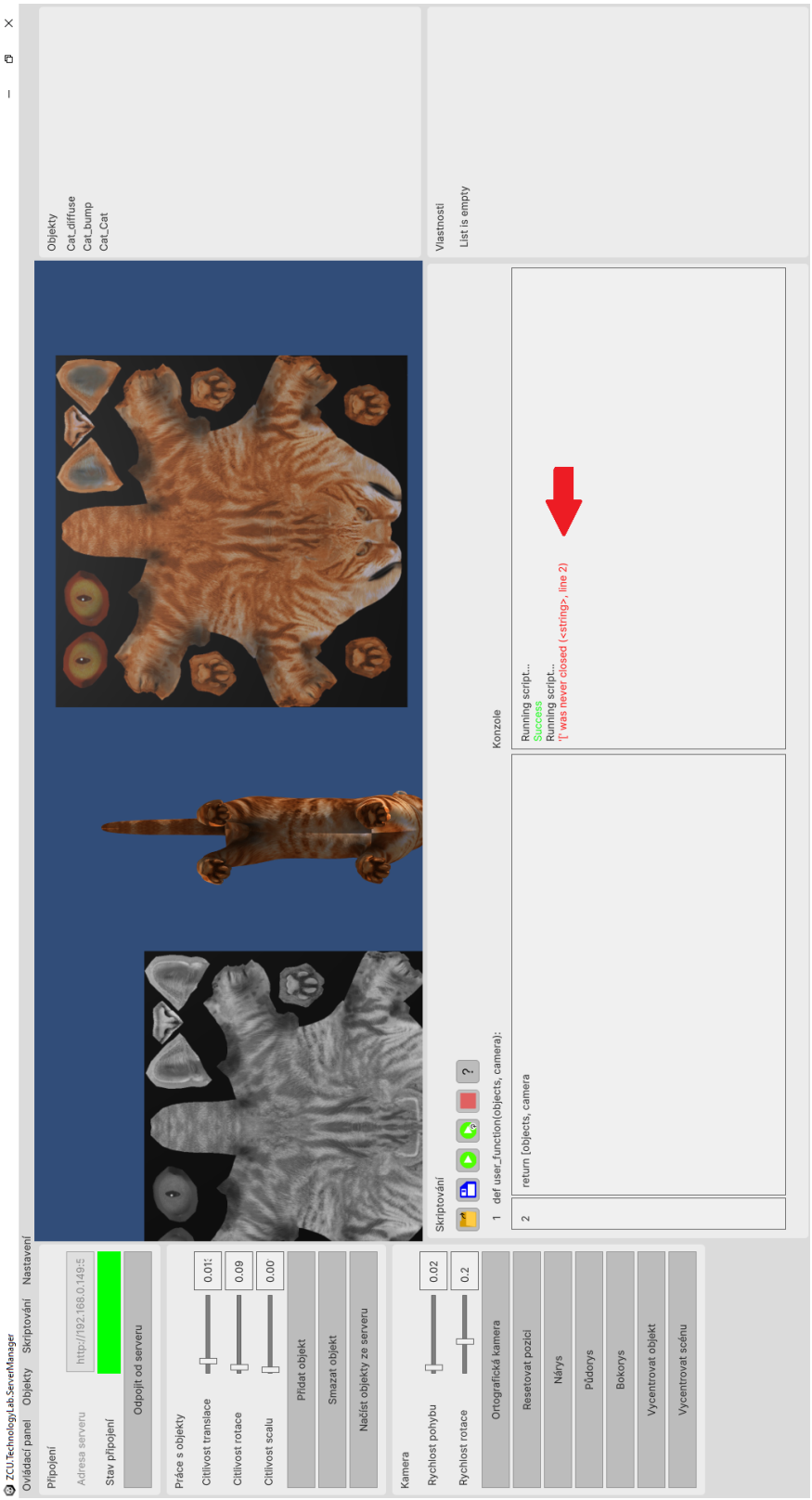
```
class ObjectTransform:
    XPosition: float
    YPosition: float
    ZPosition: float
    XRotation: float
    YRotation: float
    ZRotation: float
    XScale: float
    YScale: float
    ZScale: float
```

Zdrojový kód C.2: Transformace kamery pro skriptování

```
class CameraTransform:
    XPosition: float
    YPosition: float
    ZPosition: float
    XRotation: float
    YRotation: float
    ZRotation: float
```

Pomocí těchto parametrů upravte hodnoty u jednoho z objektů. Objekt ve slovníku vyhledejte pomocí jeho jména. Seznam dostupných objektů můžete dohledat v aplikaci v pravém panelu. Nastavte pozici objektu na hodnotu X = -0.4, Y = 1, Z = 2, rotaci na X = 60, Y = 30, Z = 45 a scale na X = 2, Y = 2, Z = 2. Pokud byste potřebovali nahlédnout na strukturu objektů znovu, klikněte na poslední tlačítko v panelu nástrojů se symbolem otazníku. Otevře se Vám pomocné okno.

Řešení Kód, co změní hodnoty vlastností u objektu se jménem „Cat_Cat" je v ukázce C.3.



Obrázek C.4: Řešení úkolu 4

Zdrojový kód C.3: Změna transformace objektu ve skriptování

```
cat = objects["Cat_Cat"]
cat.XPosition = -0.4
cat.YPosition = 1
cat.ZPosition = 2
cat.XRotation = 60
cat.YRotation = 30
cat.ZRotation = 45
cat.XScale = 2
cat.YScale = 2
cat.ZScale = 2

return [objects, camera]
```

Porovnejte obr. C.4 a C.5 a uvidíte, že transformace modelu kočky se změnila.

C.6 Úkol 6

Skript je možné spouštět časovaně v pravidelných intervalech. Pro nastavení časování přejděte do lišty hlavního menu a klikněte na tlačítko „Nastavení“. Zobrazí se okno, ve kterém lze nastavit interval mezi voláními vytvářeného kódu. Interval se uvádí v milisekundách. Nastavte, aby se kód spouštěl každých 100 milisekund. S využitím časovaného spuštění (čtvrté tlačítko v panelu nástrojů) napište kód, který bude v ose Y rotovat sudé objekty po směru hodinových ručiček a liché proti směru hodinových ručiček. Rotaci měňte o 10 stupňů. Vykonávání kódu můžete zastavit červeným tlačítkem ve tvaru čtverce.

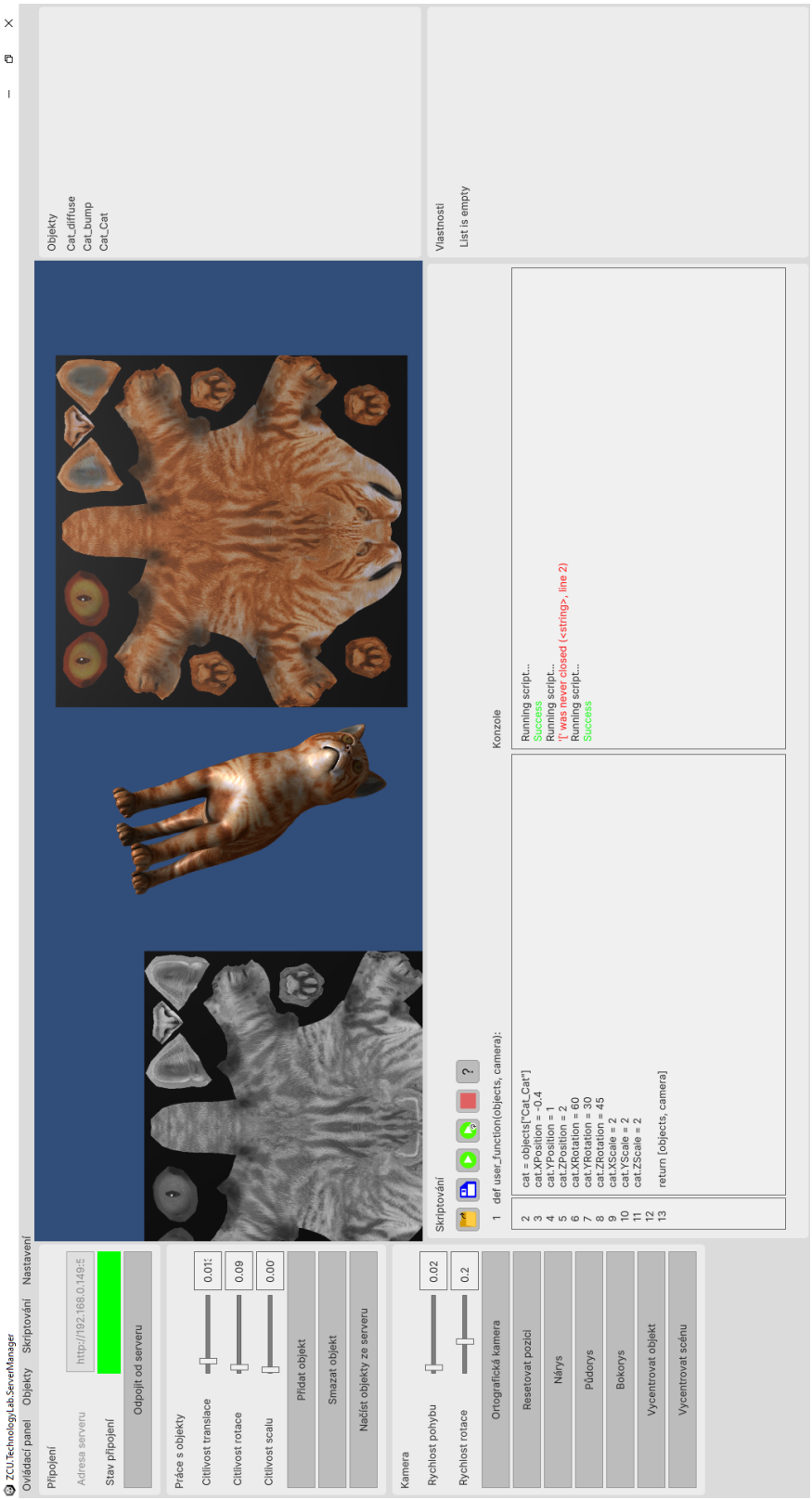
Řešení Řešení algoritmu pro opakovanou rotaci sudých a lichých objektů je v kódu C.4.

Zdrojový kód C.4: Rotace objektu po a proti směru hodinových ručiček

```
for index, (key, value) in enumerate(objects.items()):
  if index % 2 == 0:
    value.YRotation += 10
  else:
    value.YRotation -= 10

return [objects, camera]
```

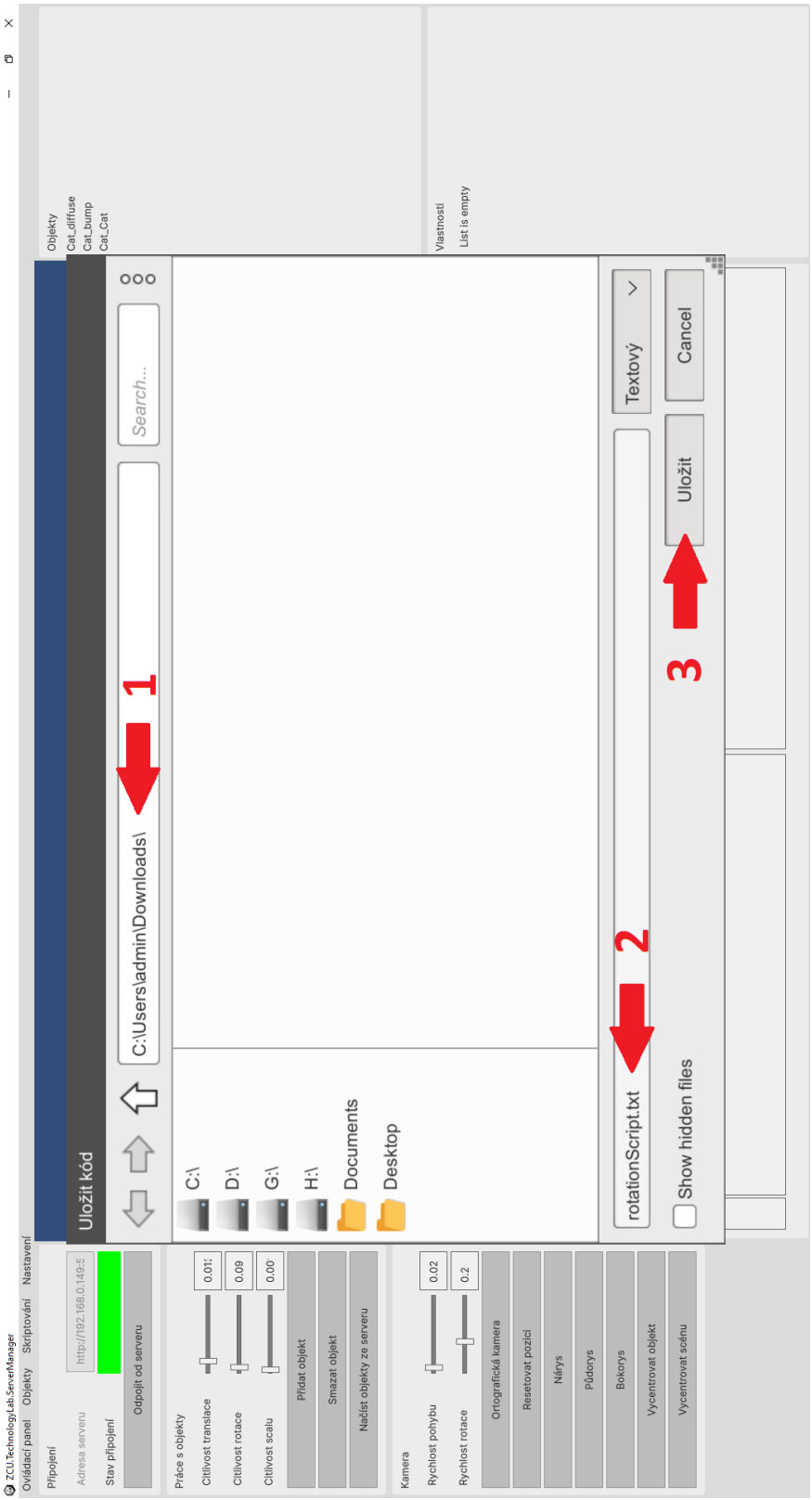
C.7 Úkol 7

Výsledný kód si uložte do souboru. Využijte druhé tlačítko s ikonou diskety uvnitř skriptovacího panelu nástrojů.



Obrázek C.5: Řešení úkolu 5

Řešení Průběh ukládání je k vidění na obr. C.6. Jako první určete cestu, kam se má soubor uložit. Následně zadejte jeho jméno a typ. Když máte vše hotovo klikněte na tlačítko „Uložit“.



Obrázek C.6: Řešení úkolu 7

Příloha 4 - Metodický materiál základní úlohy pro haptické zařízení

D

D.1 Úkol 1

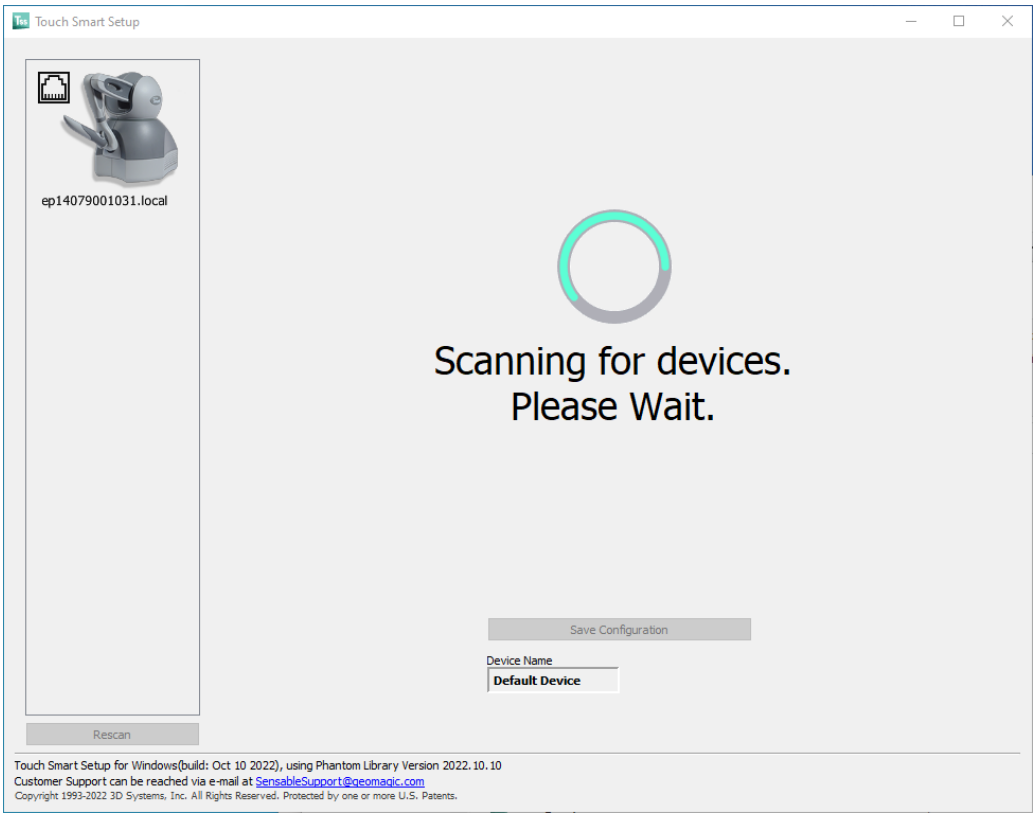
Zkontrolujte, že je haptické zařízení rozpoznané operačním systémem. Jděte do `C:\Program Files\3D Systems\Touch Device Drivers\Touch\_SmartSetup.exe`. Když tuto aplikaci spustíte, připojené haptické zařízení se automaticky rozpozná a spáruje. Pro jeho kalibraci postupujte podle pokynů na obrazovce. Nakonec konfiguraci uložte.

Řešení Jednotlivé obrazovky rozpoznání, párování a kalibrace jsou na obr. D.1, D.2, D.3, D.4. Skenování je automatické a nevyžaduje žádnou akci od uživatele. Pro párování je nutné zmáčknout tlačítko na zadní straně haptického zařízení. Po párování zkontrolujte, že se haptické pero hýbe stejně v reálném světě jako na obrazovce. Poté konfiguraci uložte kliknutím na tlačítko „Save configuration“. Neměňte výchozí jméno.

D.2 Úkol 2

Spusťte aplikaci HapticPaint pro haptické zařízení. Na startovní obrazovce klikněte na tlačítko „Pokračovat“, abyste přešli na hlavní obrazovku.

Řešení Otevřená aplikace ve výchozím stavu je na obr. D.5.



Obrázek D.1: Řešení úkolu 1 - Hledání zařízení

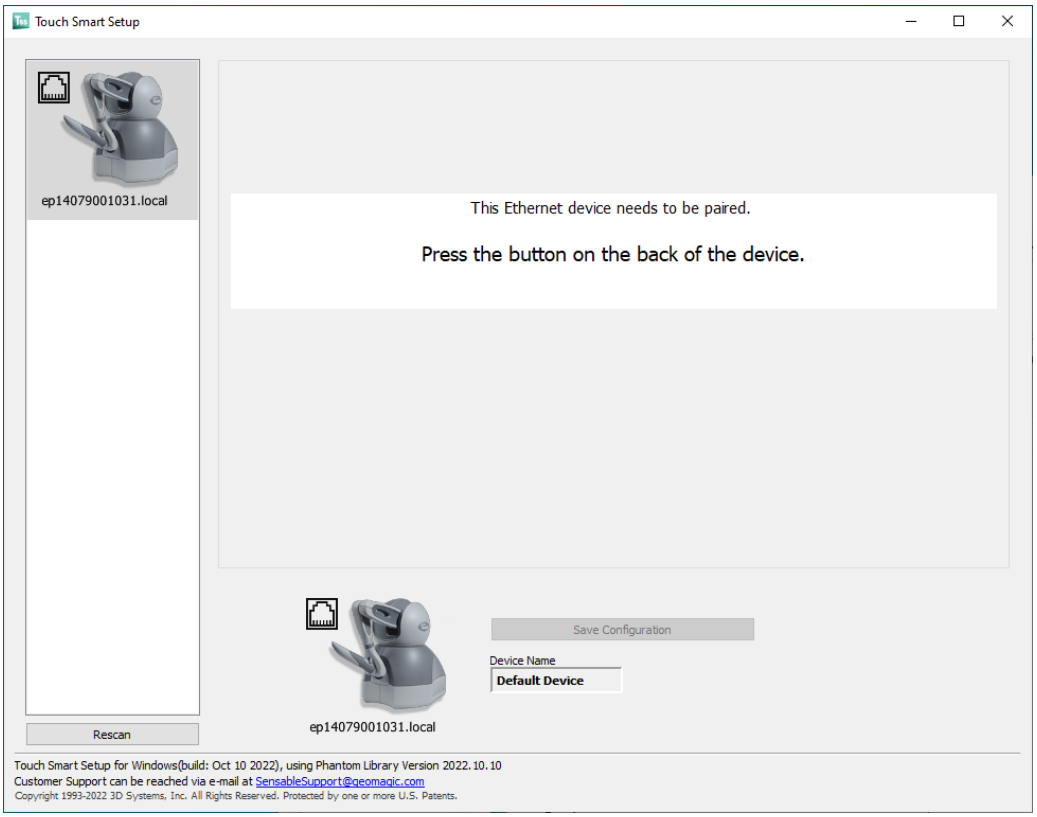
D.3 Úkol 3

Načtěte ze souboru ico sféru. Složka s 3D modely by měla být umístěna ve stejném adresáři, kde máte nainstalovanou aplikaci pro haptické zařízení.

Řešení Klikněte na tlačítko „Načíst objekt ze souboru“ v pravém panelu pojmenovaném „Manipulace s objekty“. To Vám otevře dialog pro výběr souboru. Vyhledejte v adresářové struktuře složku, kde máte nainstalovanou aplikaci pro haptické zařízení. V ní je umístěna složka Vstupni_data_na_testovani, ve které naleznete 3D_modely/Icosphere/. OBJ soubor vyberte a klikněte na tlačítko „Otevřít“. Ukázka obrazovky s dialogem je k vidění na obr. D.6. Načtený model by měl vypadat stejně jako na obr. D.7.

D.4 Úkol 4

Vyzkoušejte si různé hodnoty u haptických veličin.



Obrázek D.2: Řešení úkolu 1 - Párování zařízení

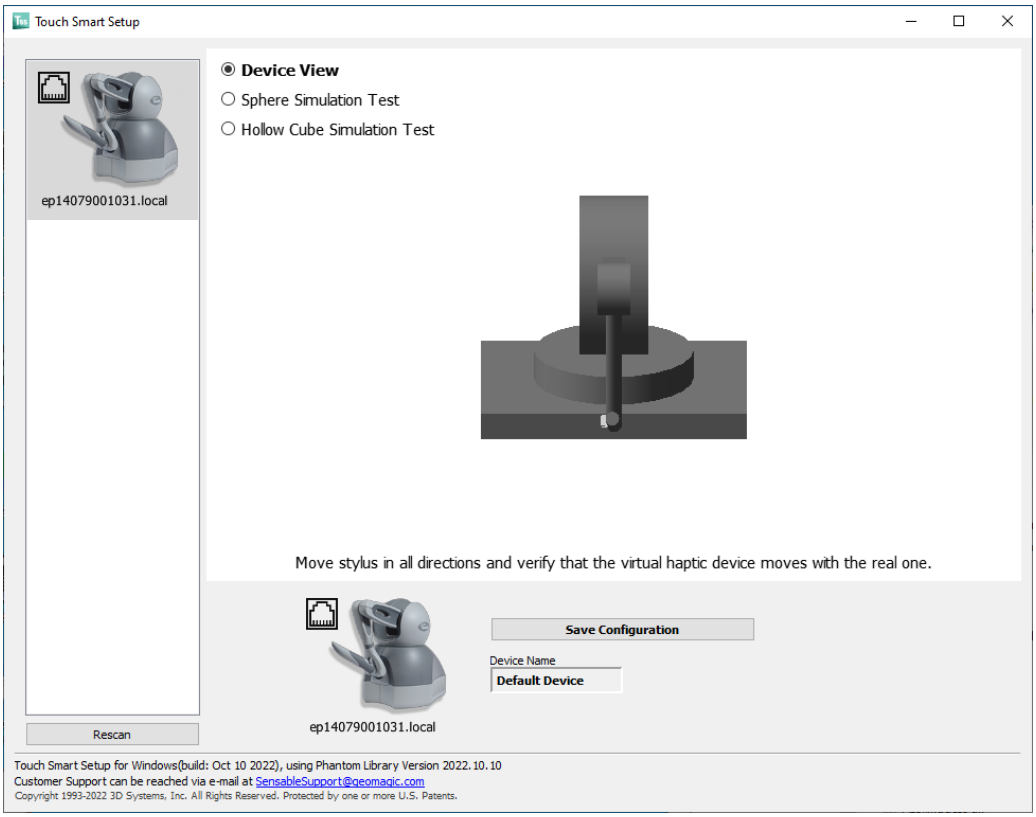
1. Nastavte tvrdost na hodnotu větší než nula a haptickým perem přejíždějte po sféře. Vyzkoušejte různé hodnoty. Čím více se budete sféře blížit, tím náročnější bude haptickým perem dále pohybovat, až vás zastaví stěna modelu.

2. Přidejte statické tření a nastavte ho na libovolnou hodnotu. Statické tření působí v okamžiku, kdy se pero začíná hýbat z klidové polohy. Budete cítit lehké škubnutí, když se budete rozjíždět perem po objektu. Aby tření fungovalo, musíte nechat nenulovou tvrdost.

3. Vynulujte statické tření a zvětšete dynamické tření na větší hodnotu než nula. Dynamické tření zapříčiní obtížnější pohyb po objektu. Když budete pohybovat perem po objektu, haptické zařízení Vám bude klást odpor.

4. Zkombinujte různé hodnoty statického a dynamického tření.

5. Vynulujte všechny předchozí hodnoty. Nastavte libovolnou viskozitu a pokuste se haptickým perem projít skrz model sféry. Viskozita bude naopak od tření působit uvnitř objektu, kde bude klást odpor při průchodu.



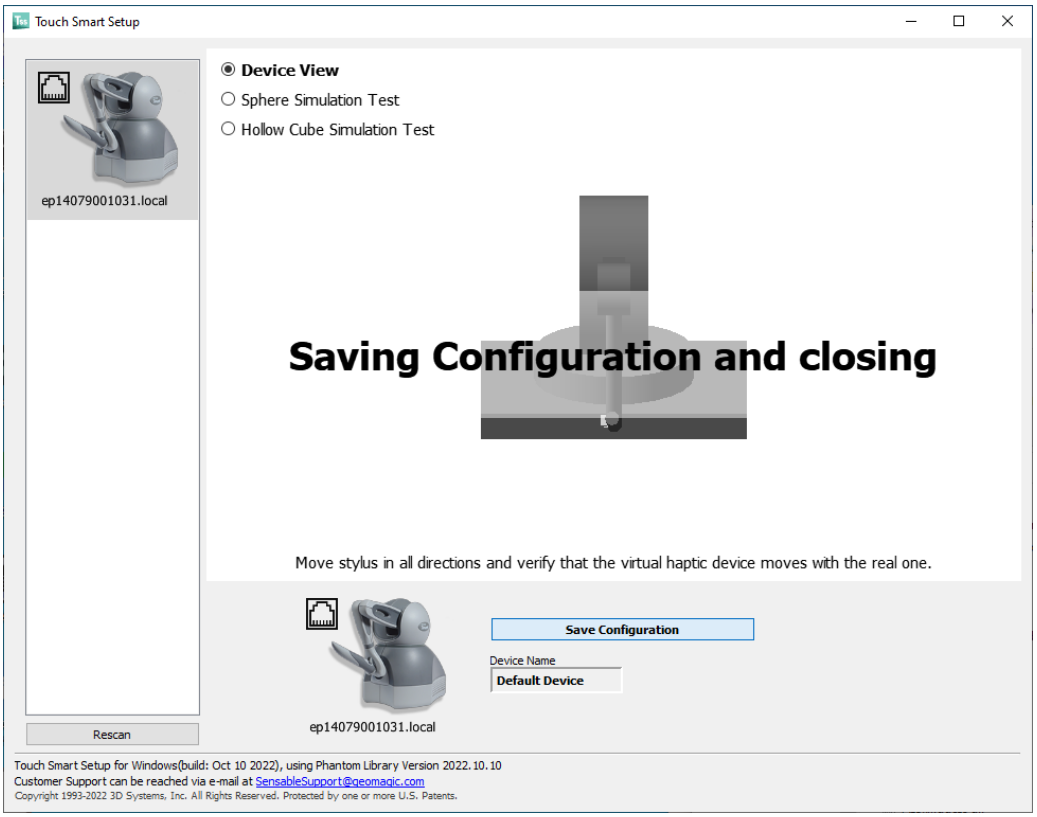
Obrázek D.3: Řešení úkolu 1 - Kalibrace zařízení

6. Vynulujte viskozitu. Nastavte velikost síly na kladnou hodnotu. Ve směru změňte Z-ovou souřadnici na -1. Když se přiblížíte k modelu sféry, haptika Vás odrazí pryč ve směru normály. Vyzkoušejte i další směry, abyste zjistili, jak ovlivňují odraz.

7. Vyzkoušejte různé kombinace vlastností. Při větším počtu vlastností volte raději nižší hodnoty, abyste tolik nezatěžovali přístroj.

8. Vyzkoušejte si další 3D modely, které jsou uložené ve stejné složce jako ico sféra.

Řešení Všechny posuvníky v panelu „Haptika“ umožňují nastavit hodnoty v rozsahu nula až jedna. Nula znamená, že daná vlastnost je vypnutá, zatímco jednička přepisuje maximální možnou úroveň, kterou je zařízení schopno vyvinout. Tvrdost, statické i dynamické tření, viskozita a velikost síly lze nastavit posuvníky s popisky, které odpovídají jejich názvům, v levém panelu „Haptika“. Směr síly obsahuje tři složky X, Y, Z. Každá ze složek udává směr v jedné souřadnicové ose.



Obrázek D.4: Řešení úkolu 1 - Uložení konfigurace

D.5 Úkol 5

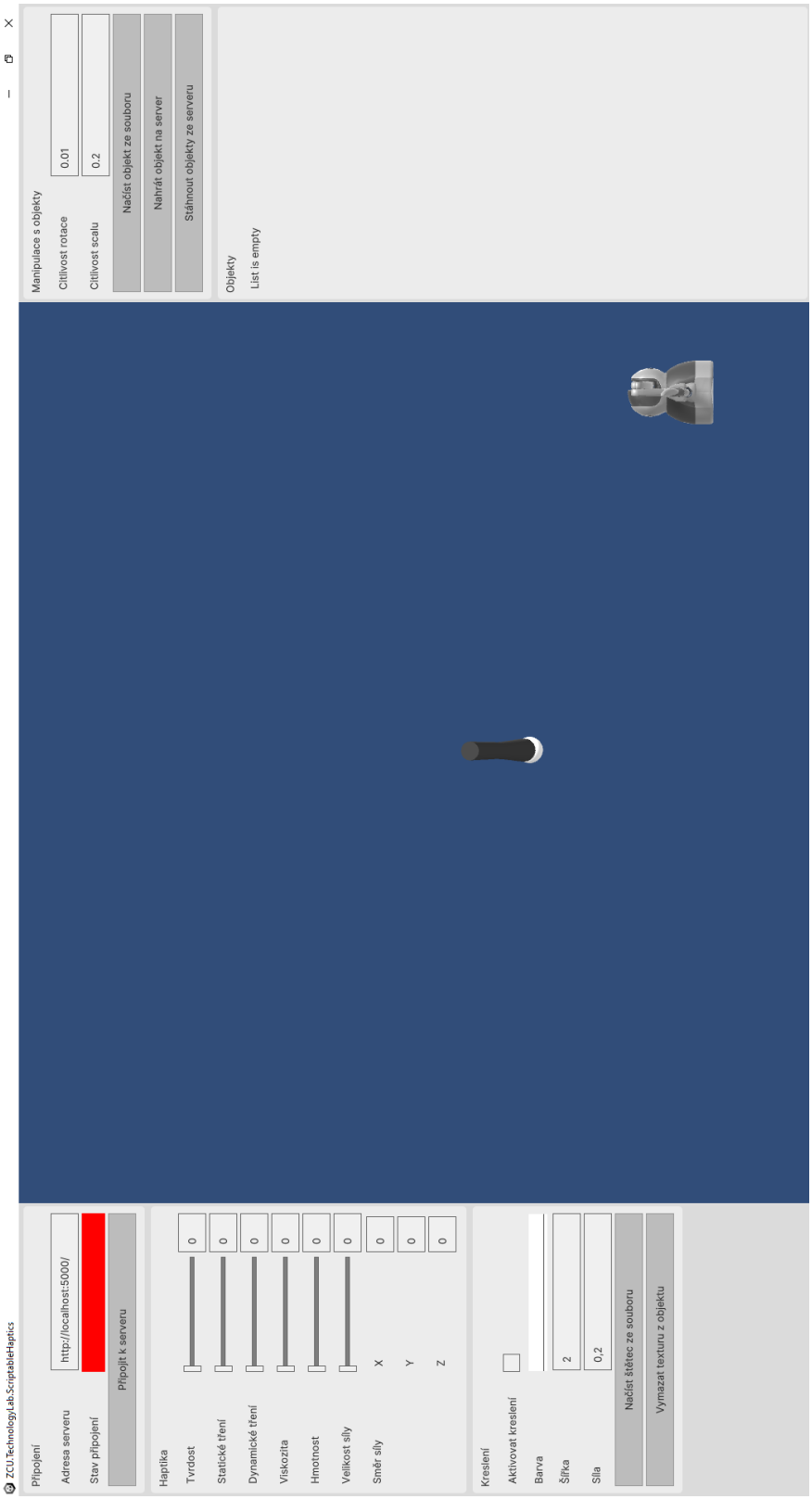
Zvyšte hmotnost objektu. Hmotnost je aktivní pouze ve chvíli držení objektu. Objekt je možné chytit pomocí hlavního tlačítka na haptickém peru. Při kliknutí se musíte dotýkat objektu, aby ho bylo možné zachytit. Zvyšujte hmotnost postupně.

Řešení Ukázka způsobu manipulace pomocí držení je na obr. D.8.

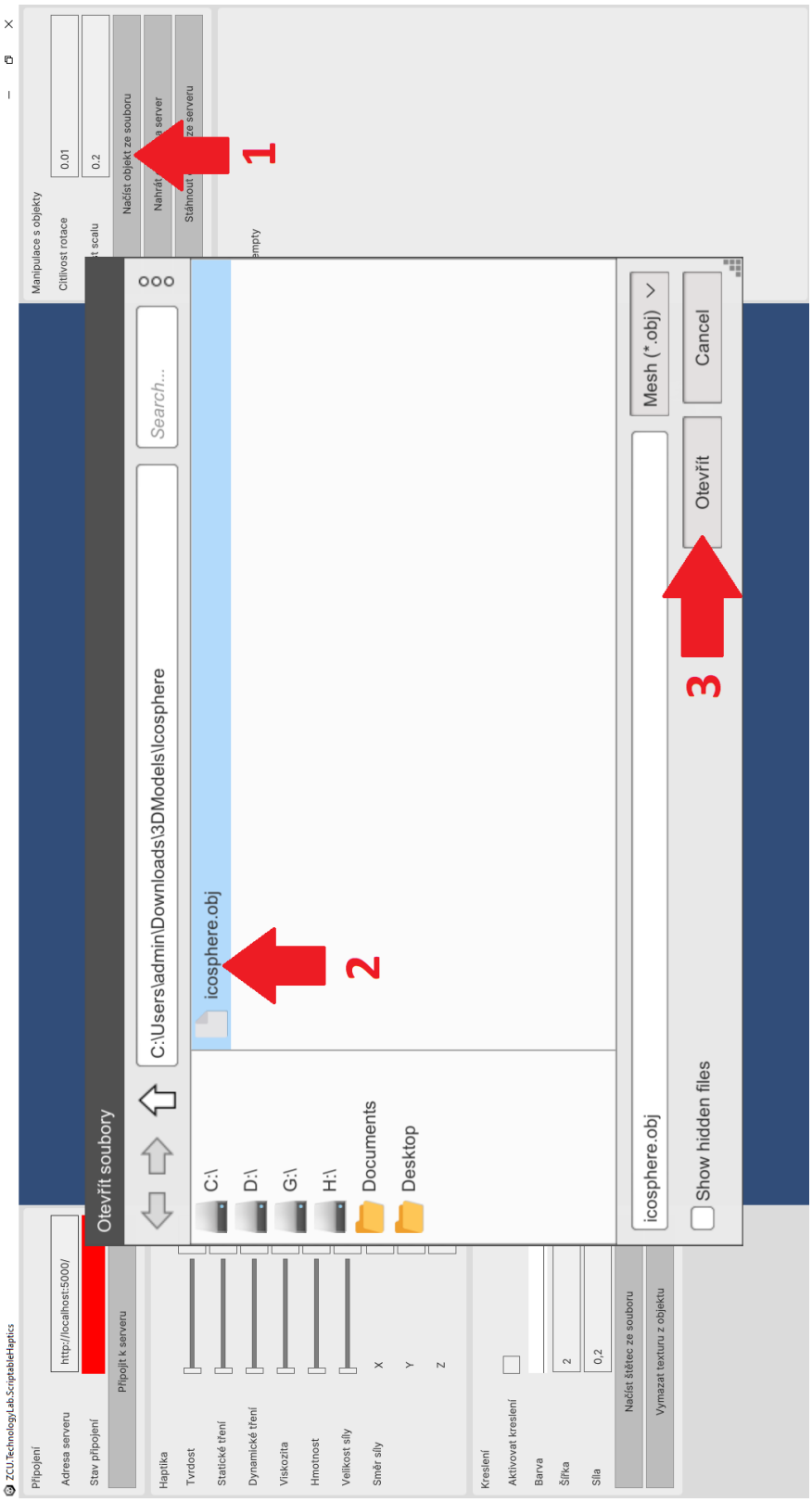
D.6 Úkol 6

Nastavte tvrdost například na hodnotu 0.5. Aktivujte kreslení. Změňte barvu štětce na červenou a pokuste se vytvořit na sféře oblouk z jedné strany na druhou. Kreslení probíhá dotykem pera s modelem. Můžete změnit šířku a sílu štětce pro jeho zvětšení/zmenšení. Pro větší cit z kreslení můžete také nastavit dynamické tření.

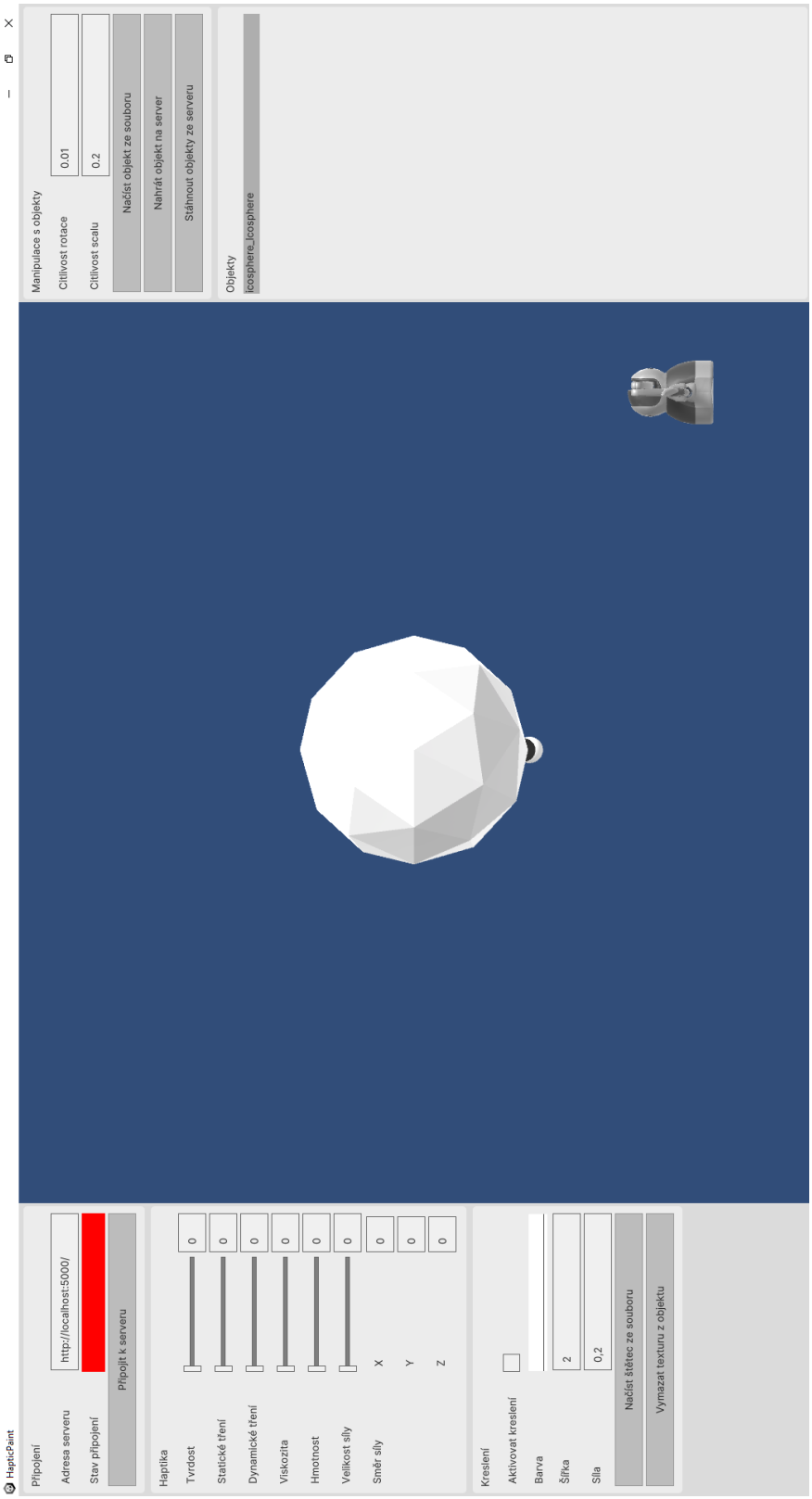
Řešení Kreslení se aktivuje zaškrtávacím polem „Aktivovat kreslení“. Ukázkový výsledek je k vidění na obr. D.9.



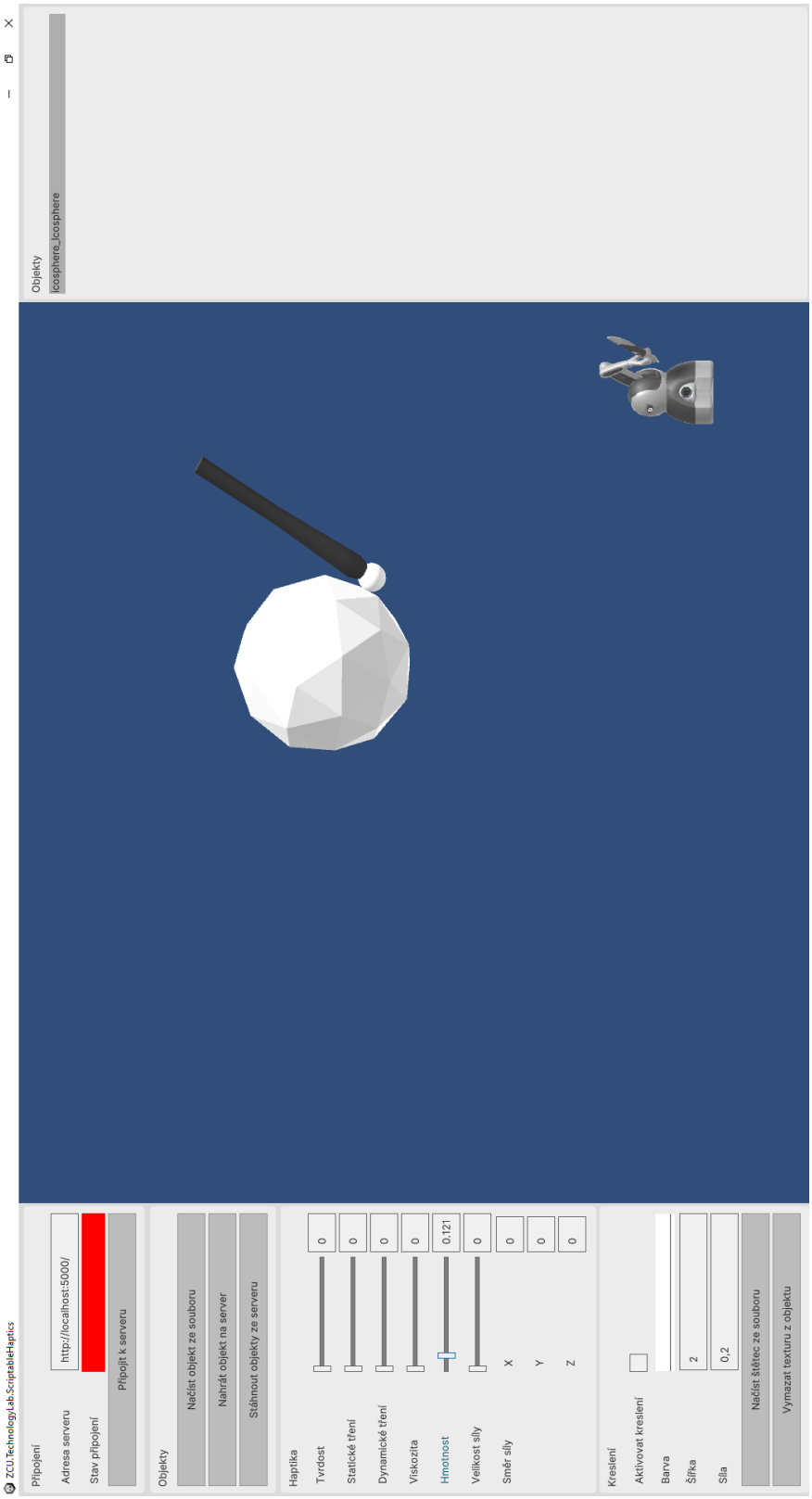
Obrázek D.5: Řešení úkolu 2



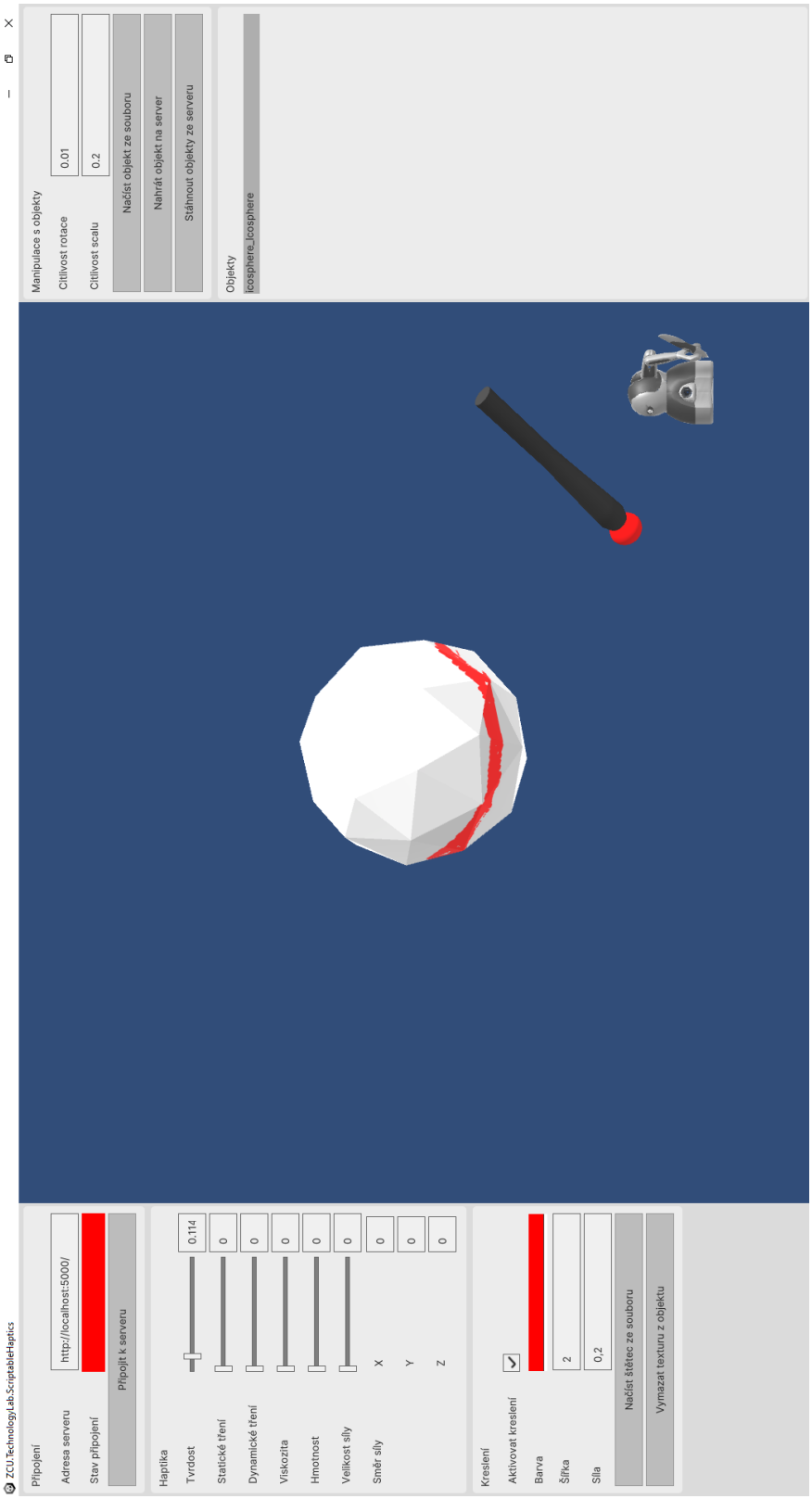
Obrázek D.6: Řešení úkolu 3 - Dialog



Obrázek D.7: Řešení úkolu 3 - Ico sféra



Obrázek D.8: Řešení úkolu 5



Obrázek D.9: Řešení úkolu 6

D.7 Úkol 7

V adresáři aplikace je kromě 3D modelů také ukázkový soubor se štětcem. Načtěte štětec ze souboru. Otočte sféru o 180° v ose Y a spojte vytvořený oblouk z minulého úkolu načteným štětcem. Rotace s modelem se provádí držením druhého tlačítka na haptickém peru. V pravém panelu lze upravit citlivost rotace.

Řešení Štětec se načítá pomocí tlačítka „Načíst štětec ze souboru“. Je k vidění na obr. D.10. Část výsledného oblouku je v ukázce na obr. D.11.

D.8 Úkol 8

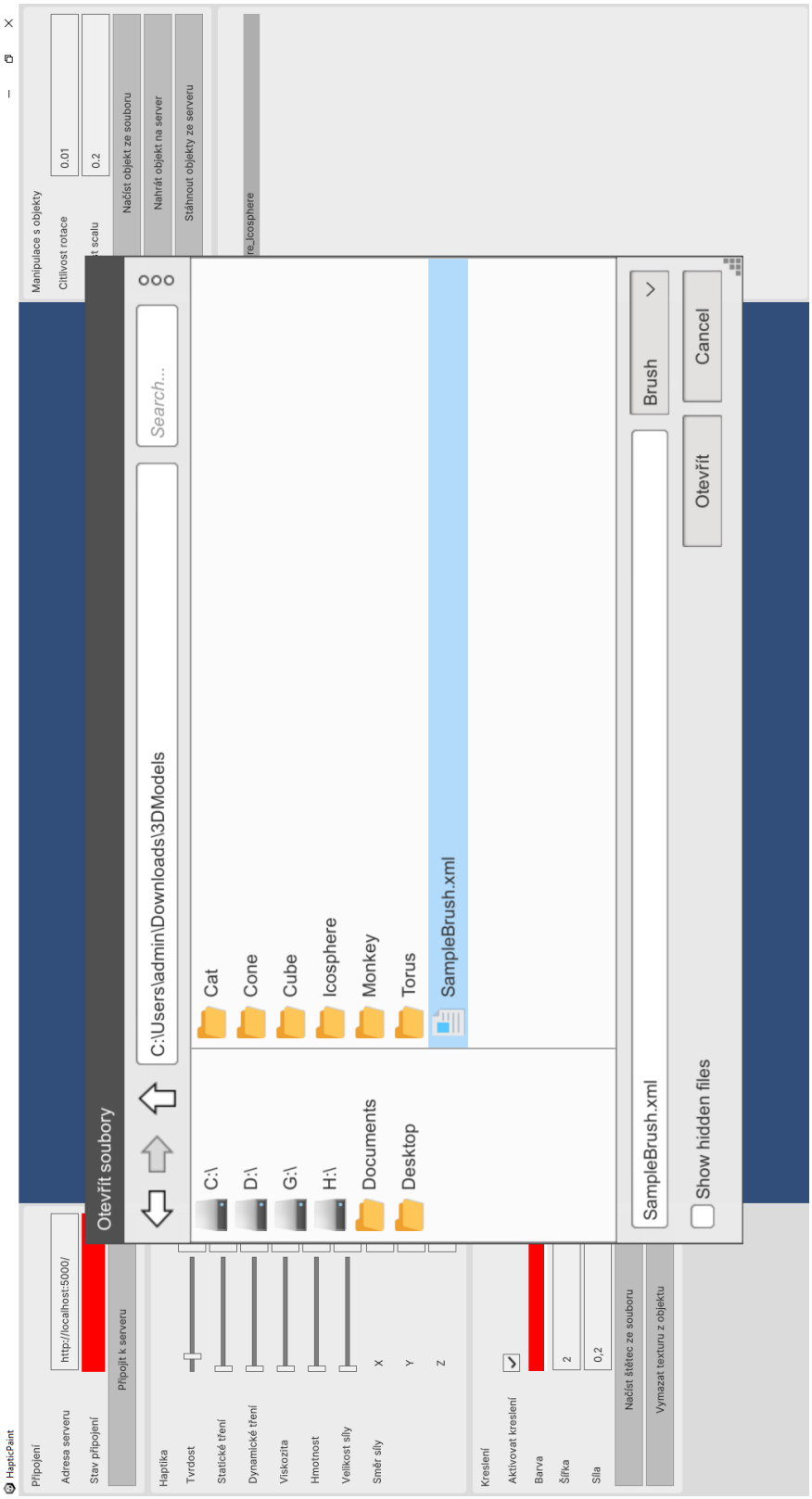
Načtěte jiný z ukázkových modelů v adresáři. Vymažte texturu z objektu. Pokuste se objekt co nejlépe vybarvit. Když se budete potřebovat dostat do menších záhybů, ale haptické pero bude příliš velké, můžete změnit měřítko objektu pomocí kolečka myši. V pravém panelu lze upravit citlivost změny měřítka (scalu).

Řešení Na načtení slouží tlačítko „Načíst objekt ze souboru“. Výsledek po načtení lze vidět na obr. D.12. Odstranění textury se vzorem lze udělat pomocí tlačítka „Vymazat texturu z objektu“. Model bude mít po odstranění textury bílou barvu. Je však možné kreslit i bez mazání přímo do textury. Model kočky po odstranění textury je k nahlédnutí na obr. D.13. Při vybarvení můžete popustit uzdu své fantazie. Na obr. D.14 je k vidění ukázka, která se snaží přiblížit originálu.

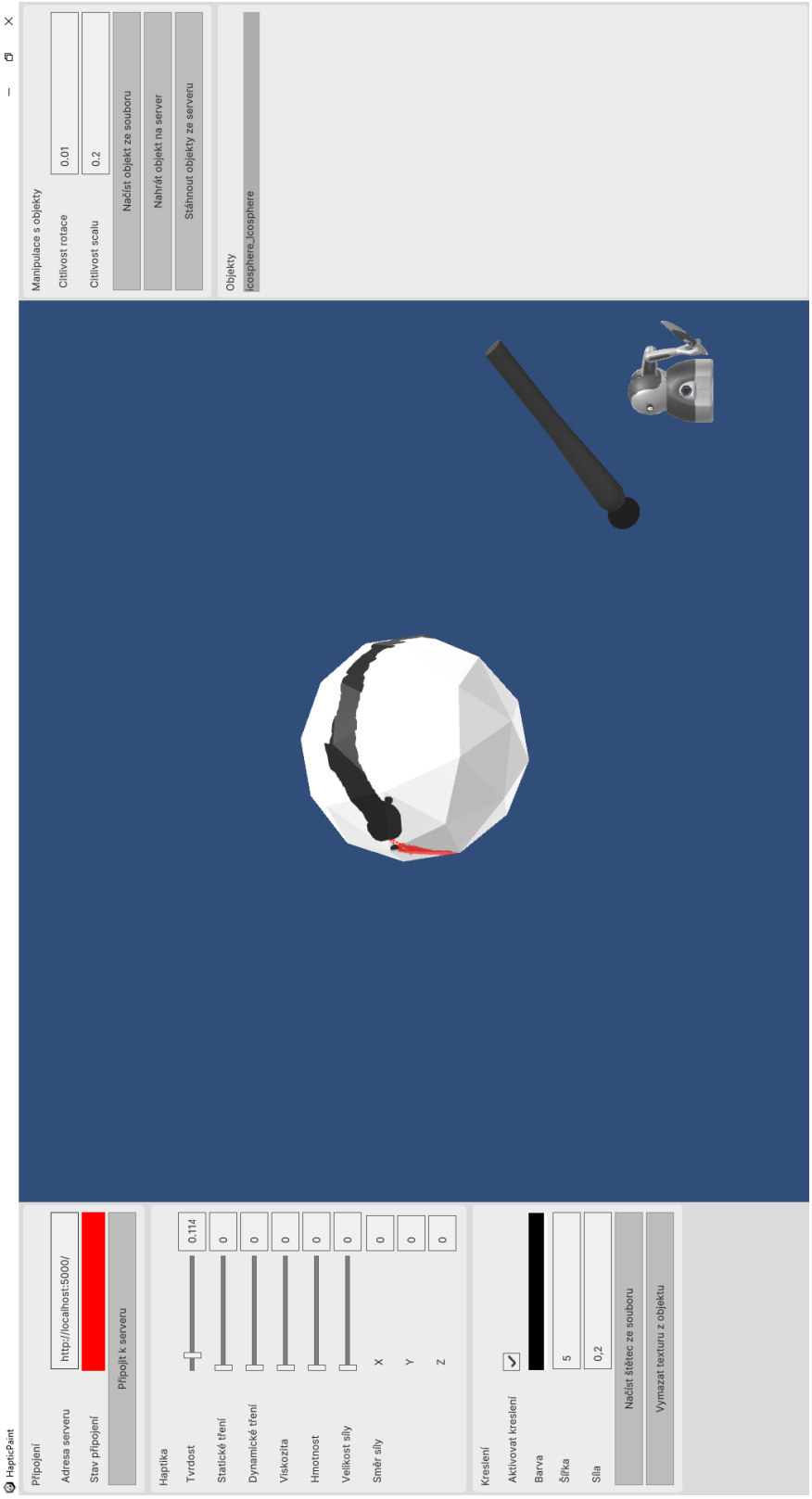
D.9 Bonusový úkol

Spusťte program pro správu serveru a přihlaste ho k systému. Přejděte zpět do aplikace pro haptické kreslení. Ze stejného adresáře jako v předchozích úkolech načtěte model opice. Po načtení modelu se připojte k serveru v aplikaci pro haptické zařízení. Nahrajte opici na server. Podívejte se do programu pro správu serveru a uvidíte opici také tam. V haptickém programu začněte kreslit. Změny se budou projevovat i ve správci serveru. Nakonec můžete objekt v programu pro správu serveru vymazat a ztratí se i z programu pro kreslení.

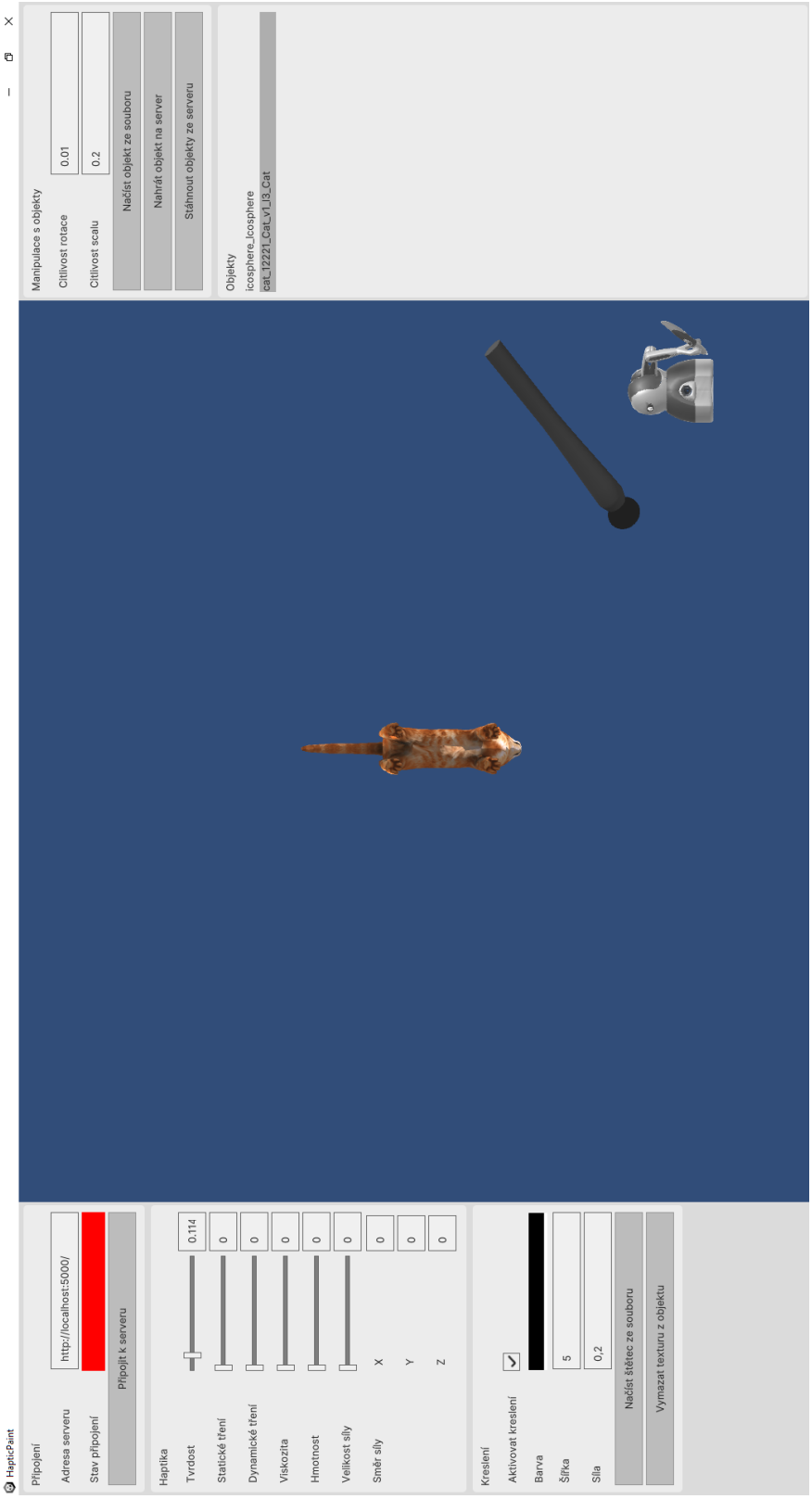
Řešení Načtený model opice je k vidění na obr. D.15. Po načtení připojte aplikaci k serveru. Postup lze vidět na obr. D.16. Díky tomu, že jsou oba programy připojené ke stejnému serveru, dojde k přeposlání nahraného trojrozměrného modelu do aplikace pro správu serveru. Když se následně provádějí nějaké úpravy druhá aplikace se o nich dozví. To lze vidět na obr. D.18 a D.19.



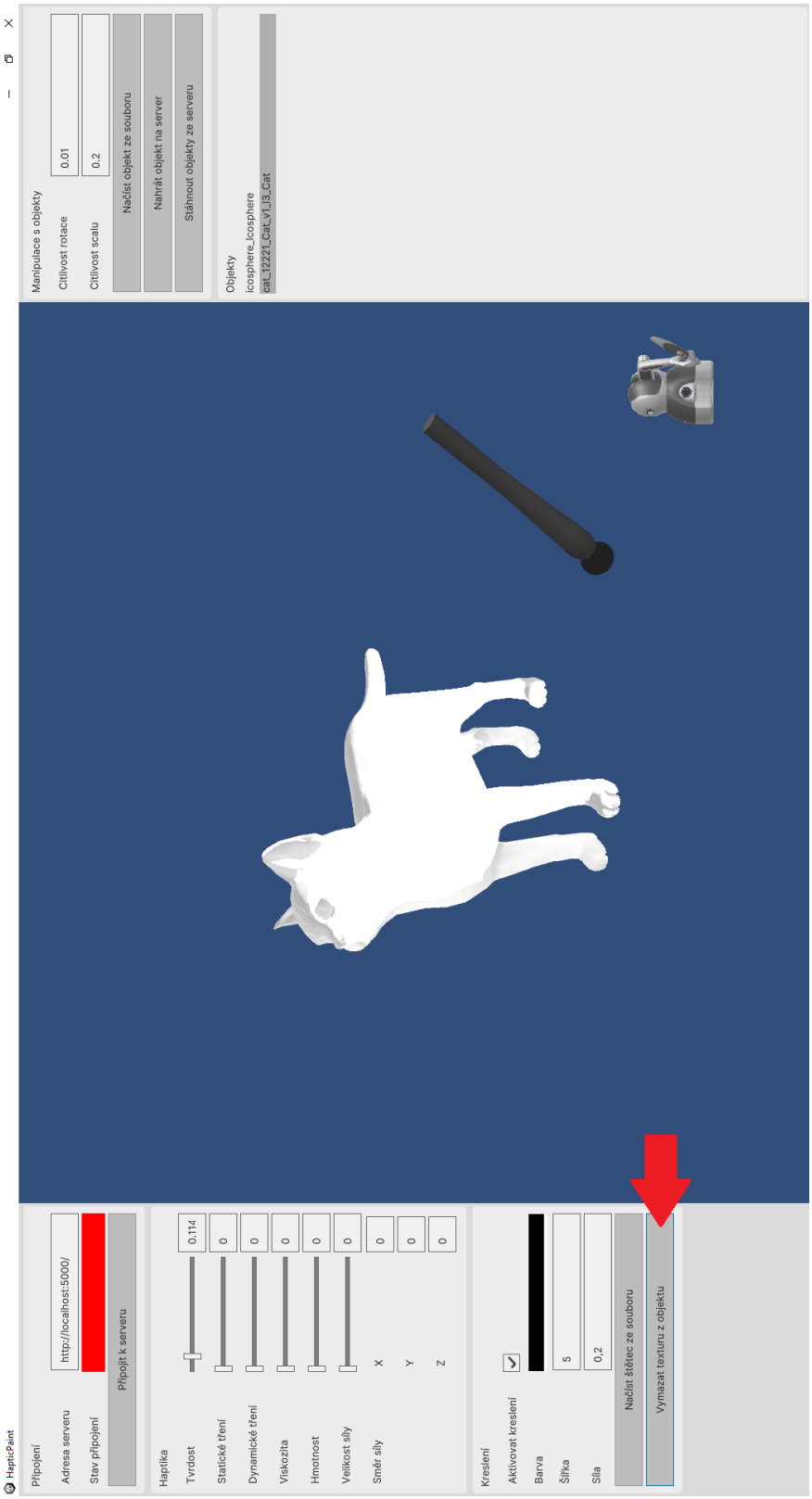
Obrázek D.10: Řešení úkolu 7 - Načtení štětce



Obrázek D.11: Řešení úkolu 7 - Dokončený oblouk

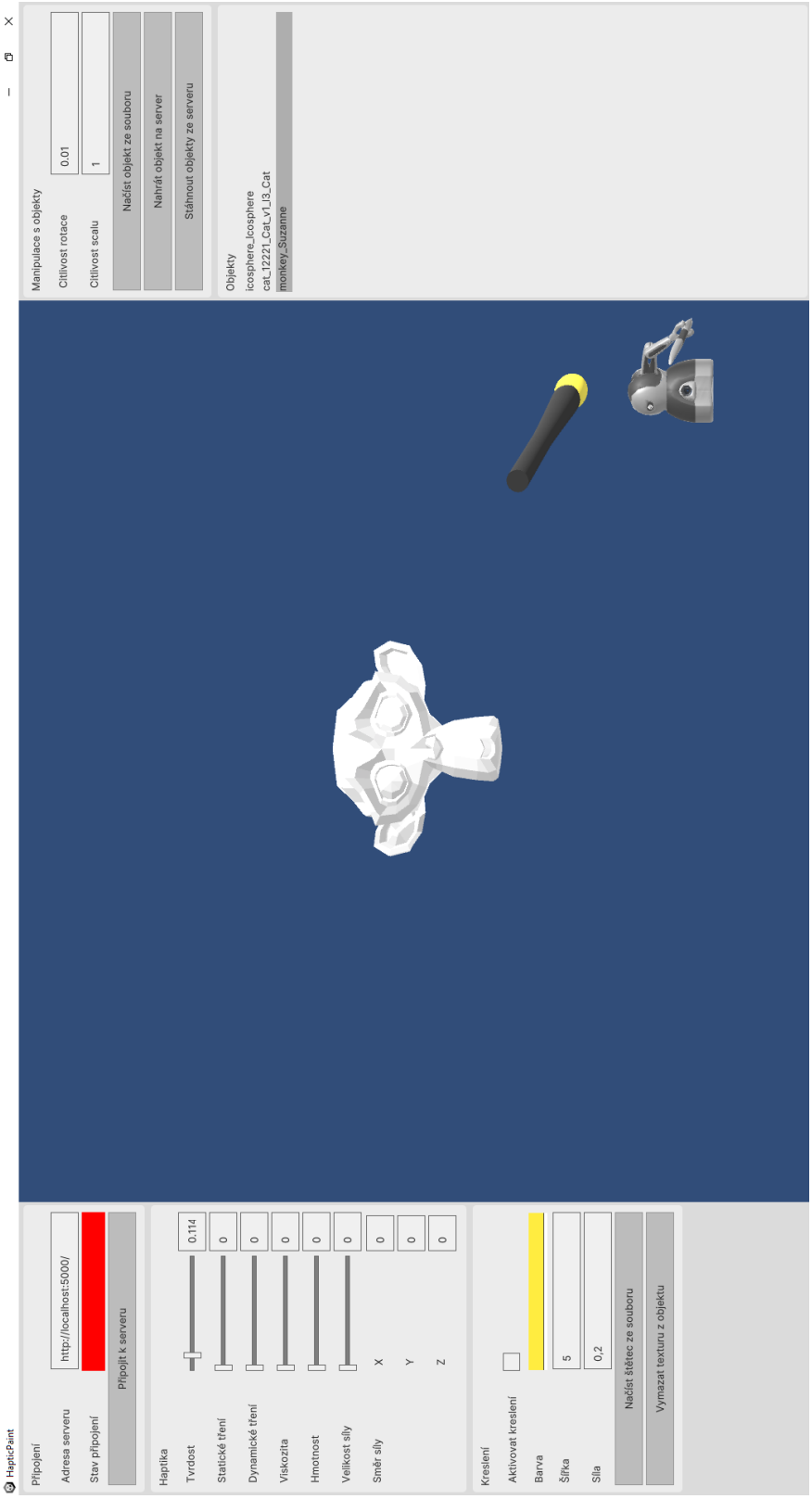


Obrázek D.12: Řešení úkolu 8 - Model kočky načtený

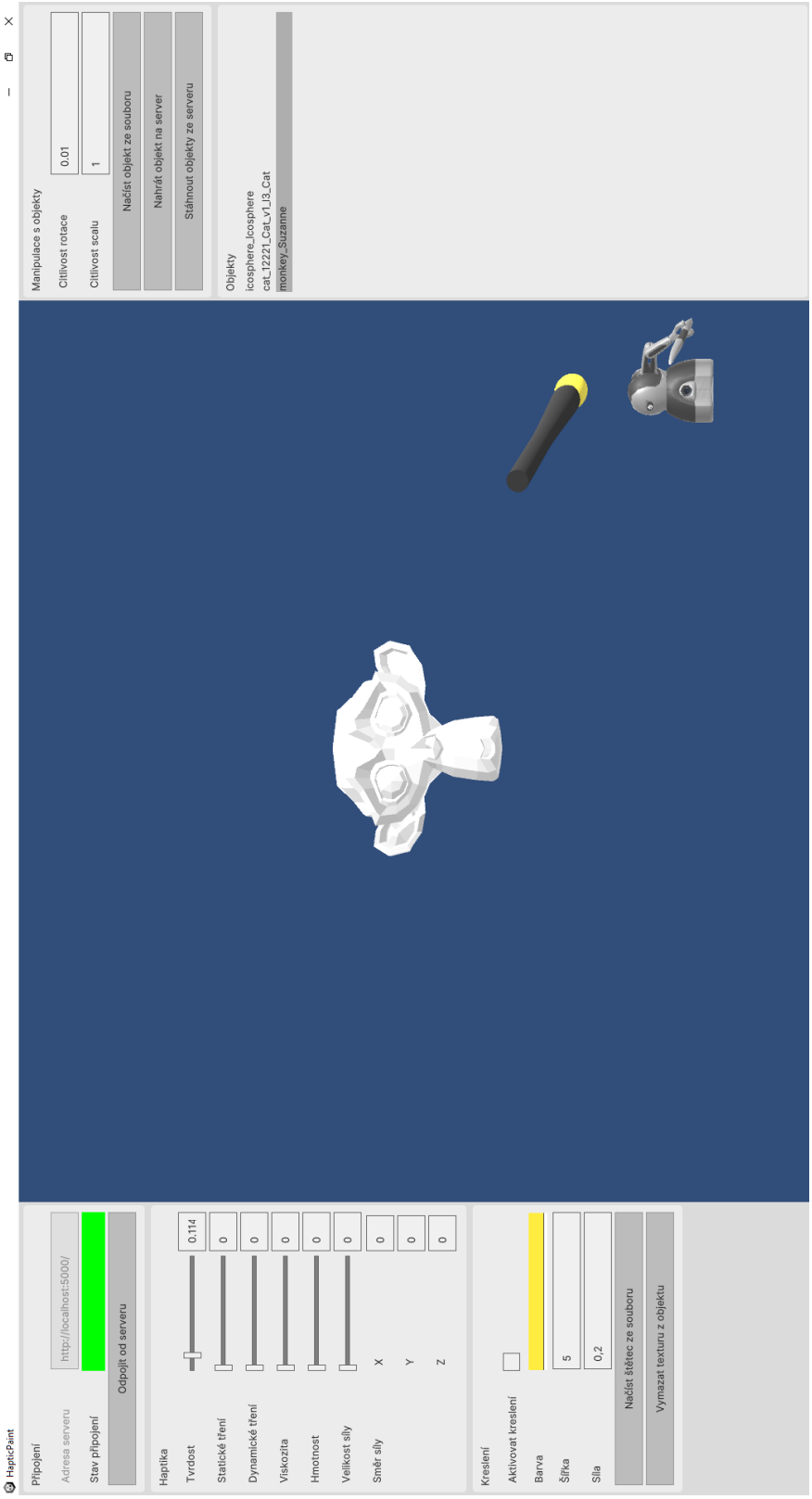


Obrázek D.13: Řešení úkolu 8 - Odstraněná textura

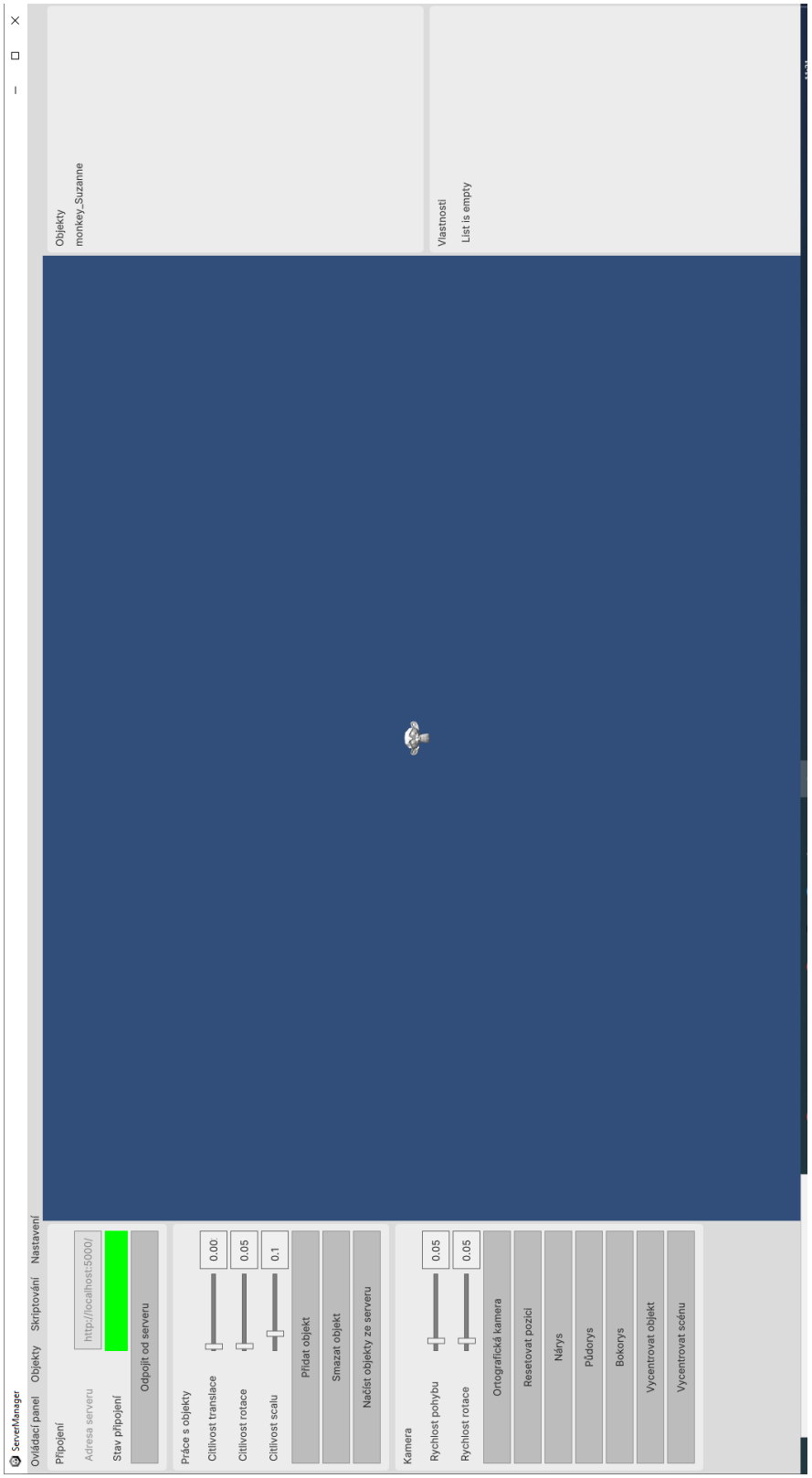
Obrázek D.14: Řešení úkolu 8 - Vybarvený model kočky



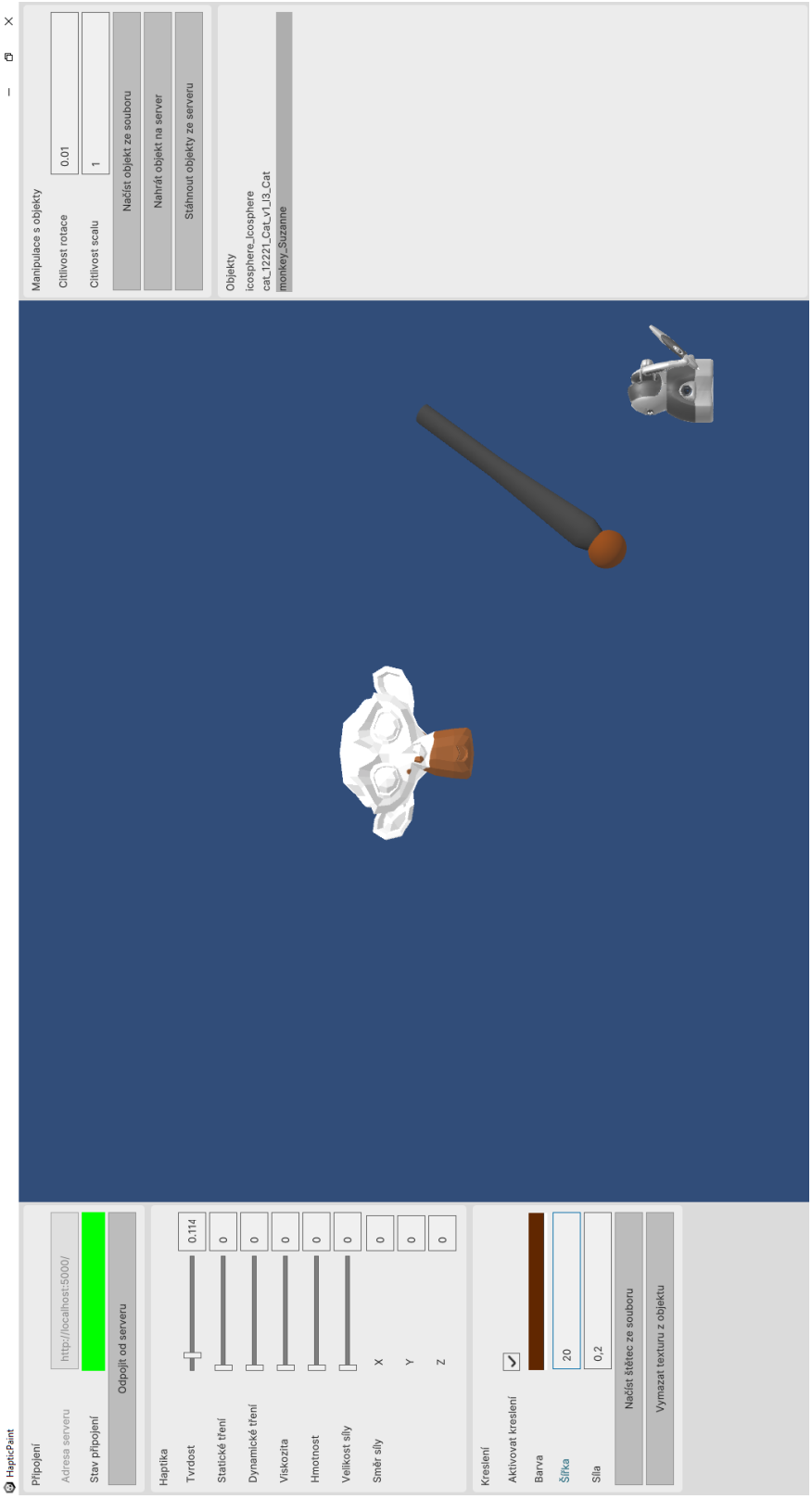
Obrázek D.15: Řešení bonusového úkolu - Model opice načtený



Obrázek D.16: Řešení bonusového úkolu - Připojení k serveru a odeslání



Obrázek D.17: Řešení bonusového úkolu - Získání modelu ze serveru



Obrázek D.18: Řešení bonusového úkolu - Úprava opice

Obrázek D.19: Řešení bonusového úkolu - Úprava opice

Příloha 5 - Metodický materiál navazující úlohy pro haptické zařízení

E

E.1 Úkol 1

Otevřete si projekt ZCU.TechnologyLab.ScriptableHaptics v Unity Hubu. K otevření potřebujete verzi Unity 2022.3.0f1.

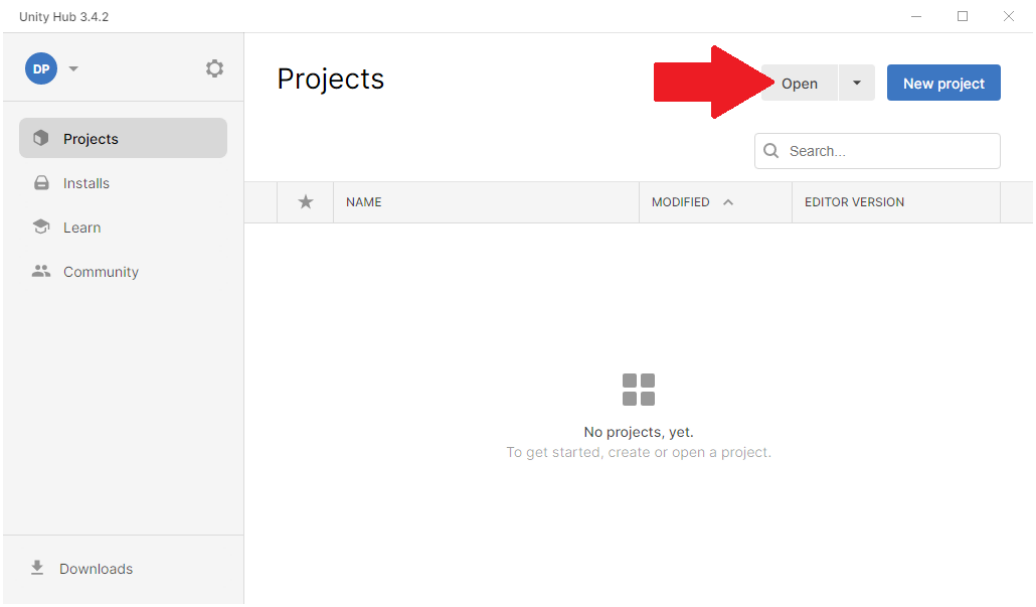
Řešení Otevřete Unity Hub. V pravém horním rohu klikněte na tlačítko „Open". Unity Hub máte vyobrazený na obr. E.1. V adresářové struktuře vyhledejte a vyberte projekt ZCU.TechnologyLab.ScriptableHaptics. Po otevření počkejte než se Unity Editor načte do stavu na obr. E.2.

E.2 Úkol 2

Zkuste zapnout Play mode. Bohužel Vám však nepůjde hýbat s haptickým perem.

Haptický plugin ve scéně není aktivní. Snažte se ho najít a aktivujte ho. Znovu spusťte scénu. Malé haptické zařízení v rohu již funguje, ale stylus se stále nehýbe. Haptický plugin musí znát, co ve scéně představuje stylus, aby ho mohl ovládat. Najděte, kde na haptickém pluginu stylus chybí a doplňte ho přetažením z hierarchie scény.

Řešení Haptický plugin je nastavený na game objektu se jménem „HapticActor_DefaultDevice", jenž je potomkem game objektu „HapticInput". Game objekt je však deaktivovaný, proto se haptický plugin nespustí. Aktivujte game objekt zaškrtávacím políčkem u jména v panelu „Inspektor". Na obr. E.3 máte označenou pozici game objektu a zaškrtávacího tlačítka. Teď když spustíte Play mode, objeví se Vám chybové hlášení v konzoli, které by Vás mělo navést na chybějící haptické pero v pluginu.



Obrázek E.1: Řešení úkolu 1 - Unity Hub

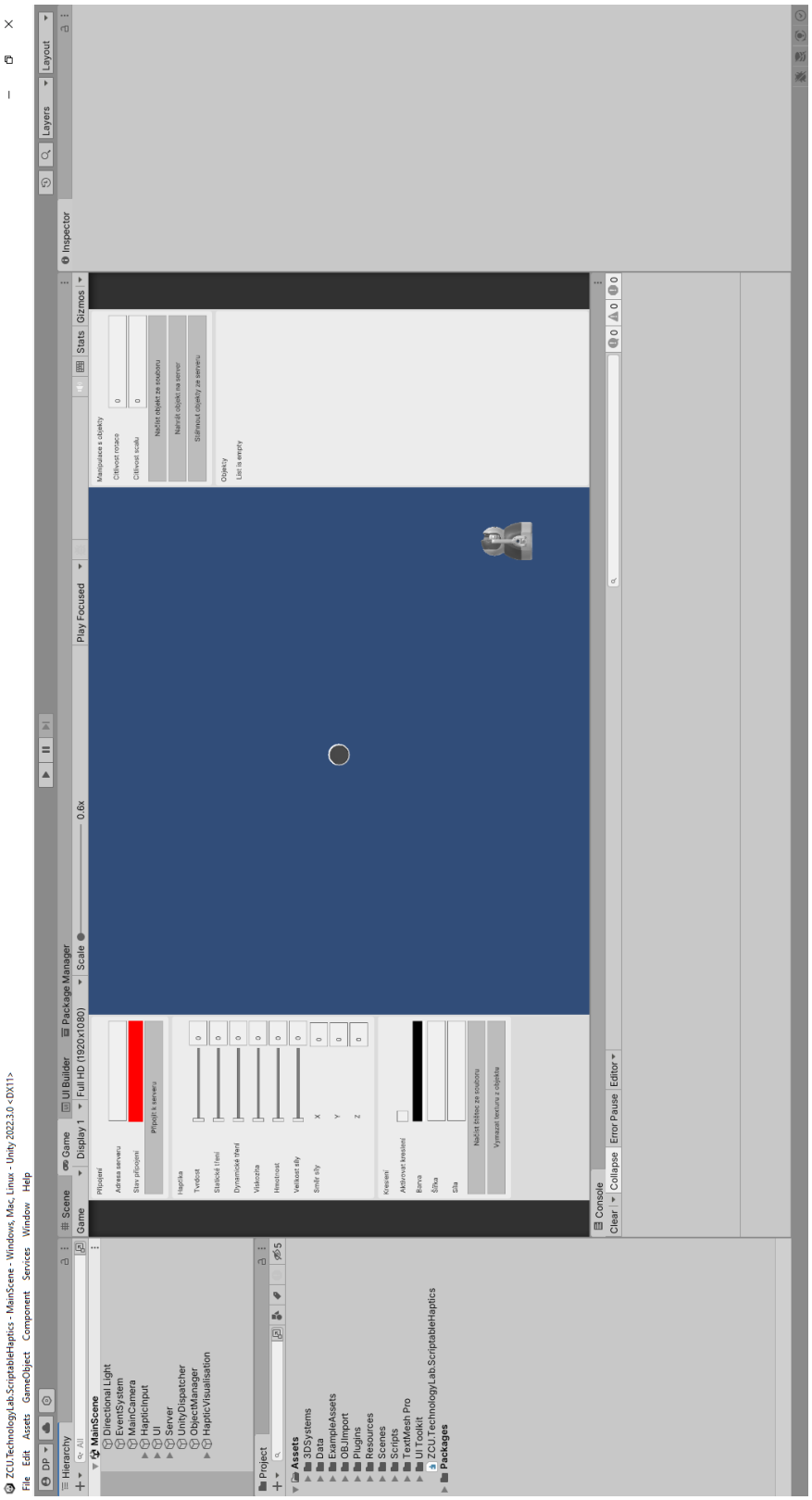
Stačí nastavit haptický collider a stylus do atributů „Collision Mesh“ repspektive „VisualMesh“. Správné párování je ukázané na obr. E.4. Znovu spusťte Play mode a vyzkoušejte, že virtuální stylus se pohybuje souhlasně s haptickým perem.

E.3 Úkol 3

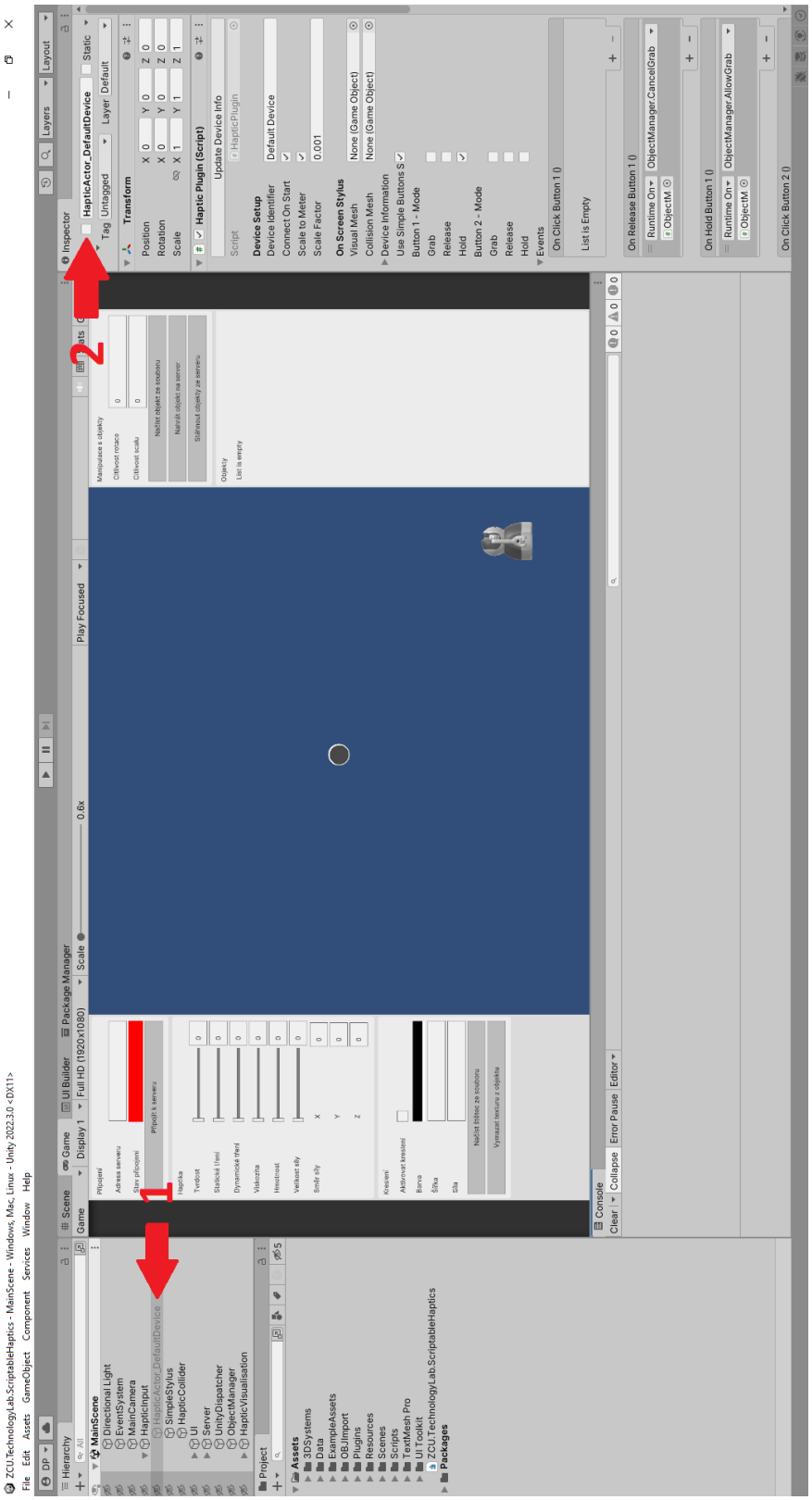
Když máte hotový úkol 2, spusťte scénu a načtěte libovolný objekt ze souboru. Zkuste chytit objekt pomocí haptického pera. Při dotyku s objektem stiskněte tlačítko 1. Pero Vám pouze projde objektem a nic dalšího se nestane. Odkud se nastavuje uchycení, zjistíte ve scéně z `HapticPlugin.OnHoldButton1`.

Otevřete kód ve vývojovém prostředí a podle návodu ve zdrojovém kódu doplňte chybějící část.

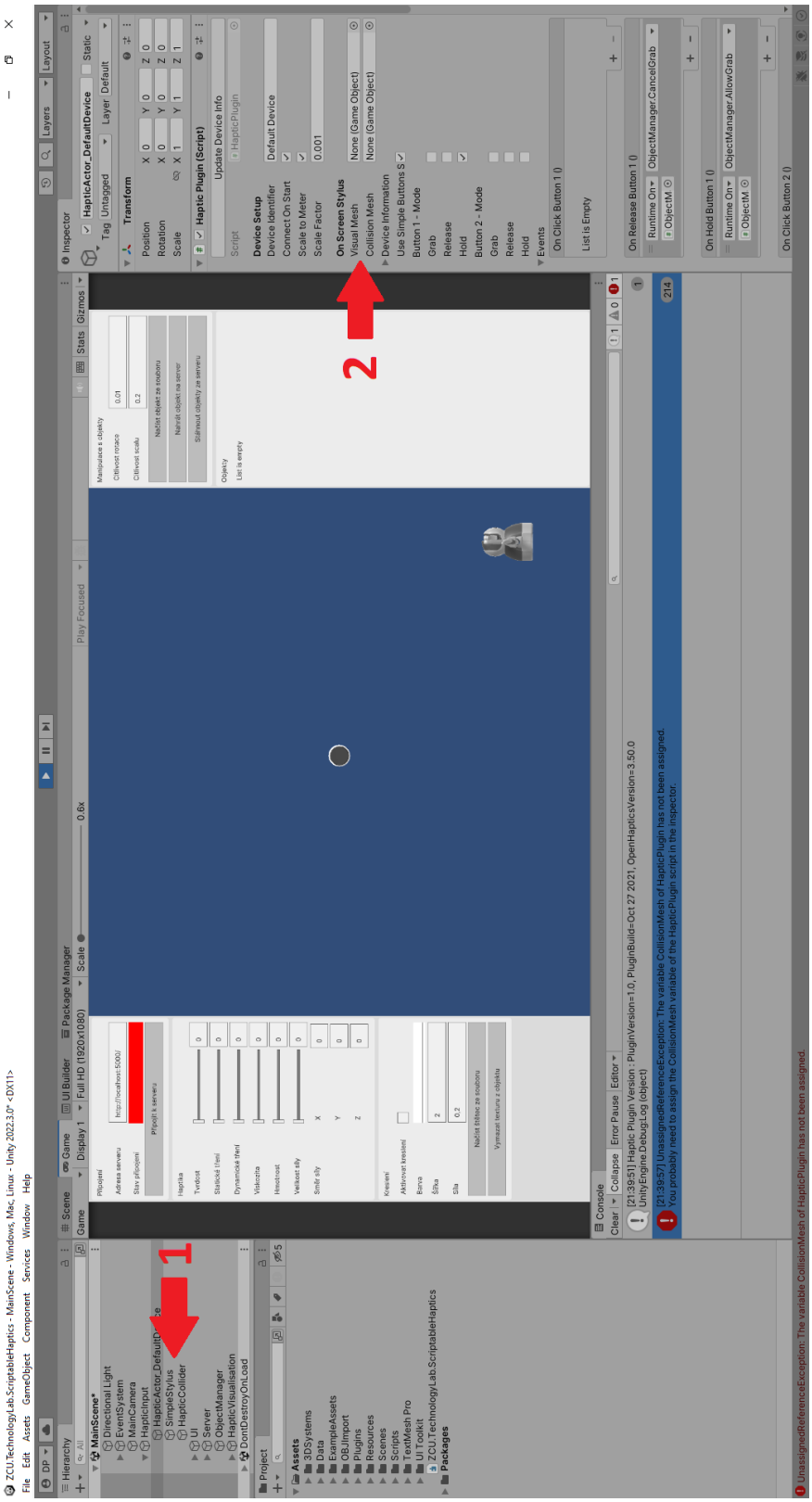
Řešení Obr. E.5 ukazuje nefunkční chycení objektu a také volanou metodu v události `HapticPlugin.OnHoldButton1`. Chybějící kód uvnitř metody `AllowGrab` nastavuje nezbytné předpoklady pro funkční zachycení objektu haptickým perem. Na ukázce kódu E.1 je možné řešení. Podmínkou pro zachycení objektu pomocí haptického pera je nekinematičnost manipulovaného rigid body. Aby dotyk s objektem kopíroval tvar, je collider nekonvexní. Unity však nekonvexní a zároveň nekinematické objekty nepodporuje. Proto je před kinematičností nutno změnit i konvexitu.



Obrázek E.2: Řešení úkolu 1 - Unity Editor



Obrázek E.3: Řešení úkolu 2 - Aktivace pluginu

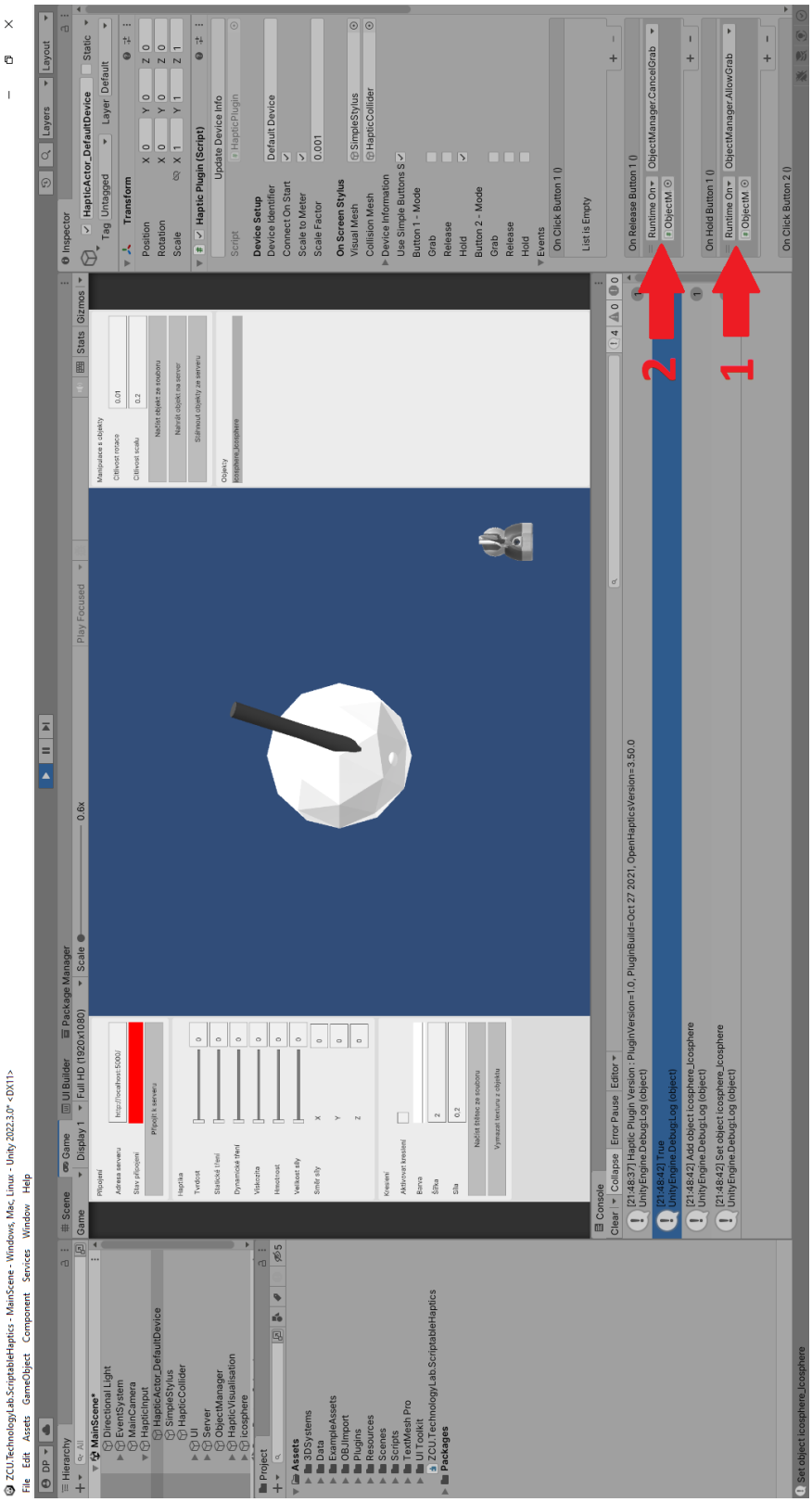


Obrázek E.4: Řešení úkolu 2 - Nastavení stylusu

Ve zrušení úchytu se změněné vlastnosti vracejí do původních hodnot. Aby nebylo možné vynést objekt z dosahu haptického pera, nastavuje se pozice objektu po skončení držení na (0, 0, 0).

Zdrojový kód E.1: Doplnění chybějícího kódu pro chycení objektu

```
/// <summary>
/// Allows to grab a visible game object with haptic stylus.
/// </summary>
public void AllowGrab()
{
    if (TargetGameObject != null)
    {
        if (_hapticPlugin.bIsGrabbingActive)
        {
            // Unity cannot use non-convex mesh collider with gravity
            var meshCollider = TargetGameObject.GetComponent<MeshCollider>();
            meshCollider.convex = true;

            // To grab an object, it needs to have non-kinematic rigidbody, that is affected by gravity
            var rigidBody = TargetGameObject.GetComponent<Rigidbody>();
            rigidBody.isKinematic = false;
        }
    }
}

/// <summary>
/// Stops grabbing a visible game object.
/// </summary>
public void CancelGrab()
{
    if (TargetGameObject != null)
    {
        // When grabbing stops, disable effects of gravity so the object stays in one place.
        var rigidBody = TargetGameObject.GetComponent<Rigidbody>();
        rigidBody.isKinematic = true;

        // While grabbed object can be moved, which is undesirable.
        TargetGameObject.transform.position = Vector3.zero;

        // Disable convex mesh collider so a mesh can be touched directly with haptic stylus.
```



Obrázek E.5: Řešení úkolu 3 - Nefunkční chycení objektu

```
36          var meshCollider = TargetGameObject.GetComponent<
    MeshCollider>();
37          meshCollider.convex = false;
38      }
39  }
```

E.4 Úkol 4

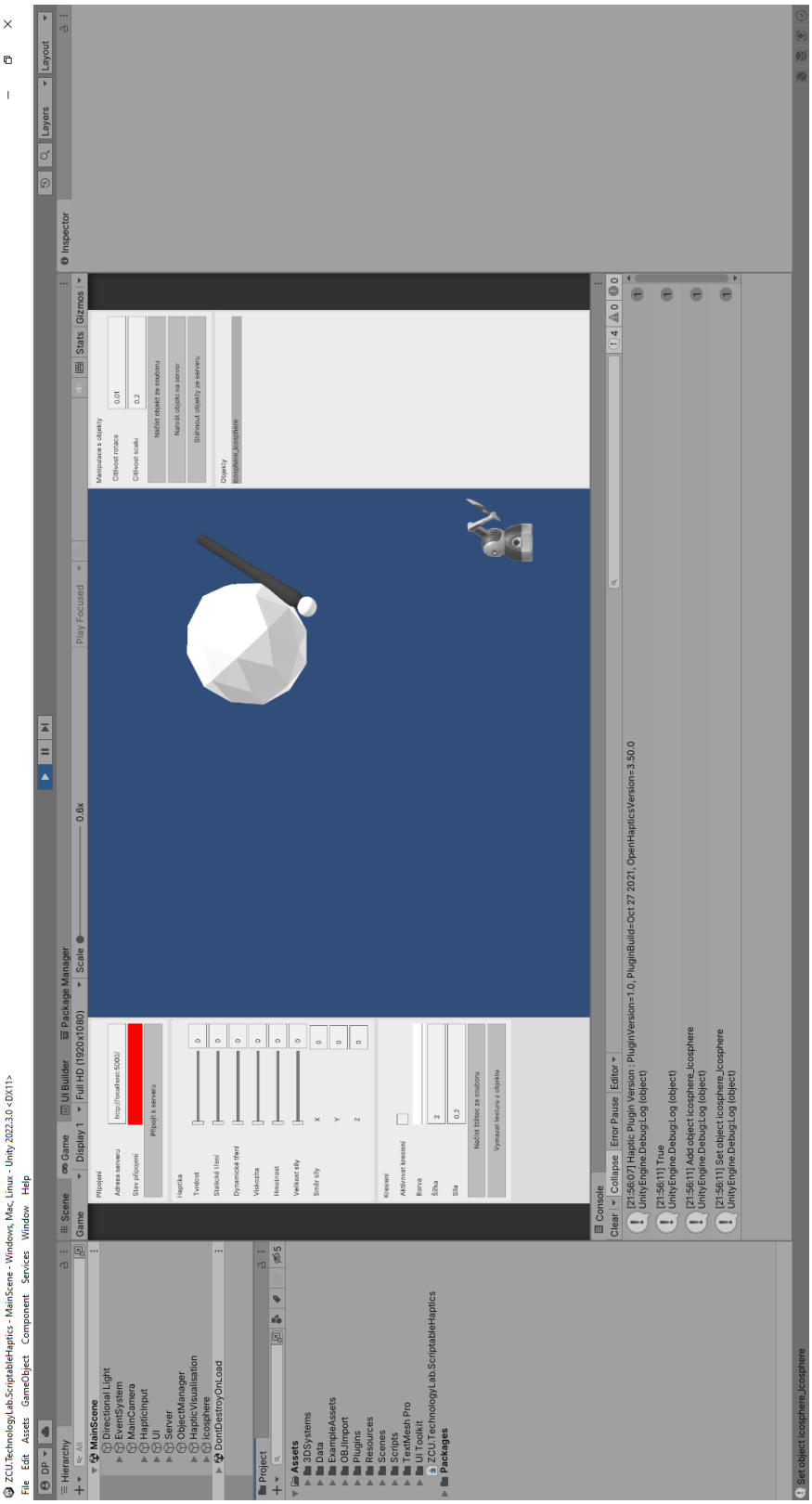
Nastavte tvrdost a povolte kreslení. Když budete přejíždět po objektu, kreslení nebude fungovat.

Projděte posloupnost volání `PaintSettingsPanel` -> `PaintSettingsPanelController` -> `ObjectManager` -> `PaintManager` -> `HapticPainter`. Na jednom místě je vynechaná část kódu s komentářem k úloze. Podle popsaného návodu doplňte chybějící kód. Spusťte aplikaci a zkontrolujte, že nastavení vlastností pro kreslení z uživatelského rozhraní funguje.

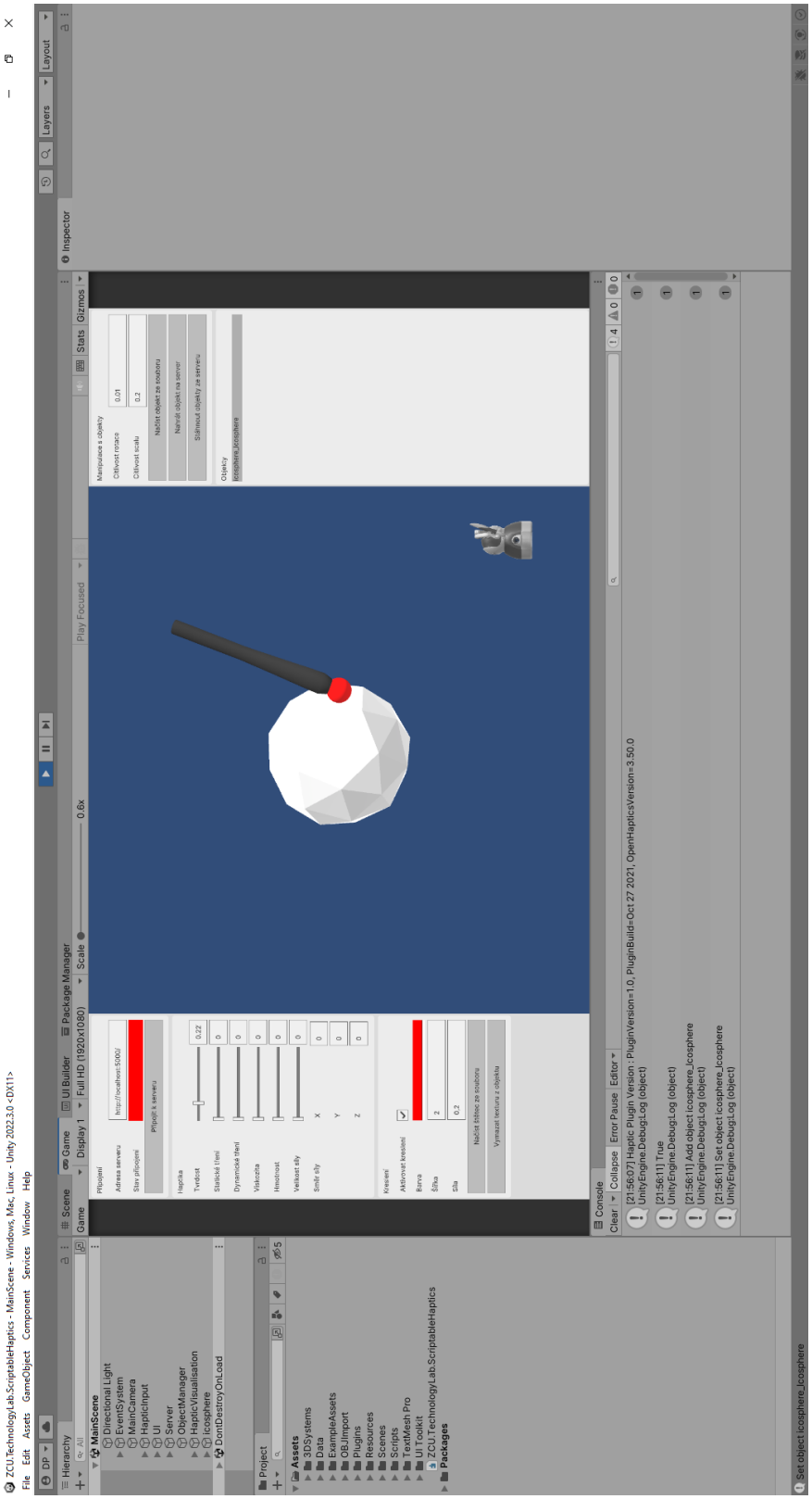
Řešení Díky obr. E.7 můžete vidět, že i když se stylus dotýká s objektem, tak se nic nekreslí. Úprava vyžaduje doplnit předávání vlastností ze sdílených proměnných do haptického kreslení. Na řešení můžete nahlédnout ve zdrojovém kódu E.2. Inicializace všech vlastností je přímočará. Stačí do `HapticPainter` nastavit hodnoty všech vlastností, které jsou v metodě připraveny. Malým chytákem v úloze je, že když změníte nějakou z hodnot v uživatelském rozhraní, už je dávno po inicializaci v metodě `Start`. Upravené hodnoty se Vám sem nedostanou. Z toho důvodu je nutné reagovat na události `ValueChanged` u jednotlivých vlastností. Že kód funguje můžete vidět na obr. E.8.

Zdrojový kód E.2: Doplnění chybějícího kódu pro kreslení

```
1  /// <summary>
2  /// Does one time initialization when <see cref="PaintManager
       "/> is started.
3  /// </summary>
4  private void Start()
5  {
6      // Painting is better with mesh collider.
7      if (gameObject.GetComponent<MeshCollider>() == null)
8      {
9          gameObject.AddComponent<MeshCollider>();
10     }
11
12     // Start is called only one time, but it is needed to
       update painter everytime values change in UI.
13     IsActive.ValueChanged += IsActive_OnValueChanged;
14     BrushColor.ValueChanged += BrushColor_OnValueChanged;
```



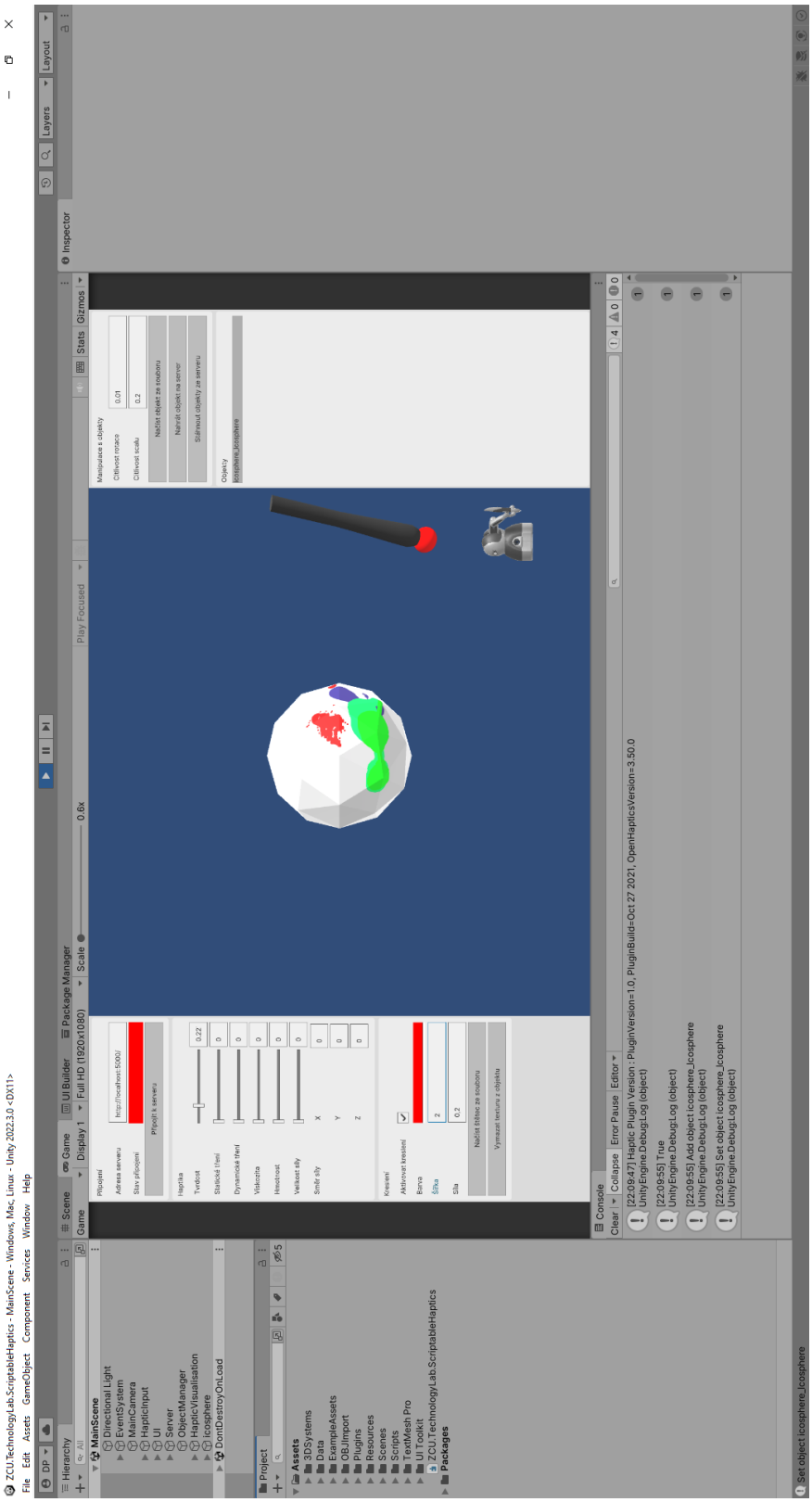
Obrázek E.6: Řešení úkolu 3 - Opravené chycení objektu



Obrázek E.7: Řešení úkolu 4 - Nefunkční kreslení

```
15      BrushSize.ValueChanged += BrushSize_OnValueChanged;
16      BrushForce.ValueChanged += BrushForce_OnValueChanged;
17
18      // Initialize values in painter.
19      _painter = gameObject.AddComponent<HapticPainter>();
20      _painter.hapticPlugin = HapticPlugin;
21      _painter.IsActive = IsActive.Value;
22      _painter.DrawColor = BrushColor.Value;
23      _painter.Size = BrushSize.Value;
24      _painter.forceImpact = BrushForce.Value;
25  }
26
27  /// <summary>
28  /// Sets new active indicator to a painter if its value
        changes.
29  /// </summary>
30  /// <param name="active">New value.</param>
31  private void IsActive_OnValueChanged(bool active)
32  {
33      if (_painter != null)
34      {
35          _painter.IsActive = active;
36      }
37  }
38
39  /// <summary>
40  /// Sets new brush force to a painter if its value changes.
41  /// </summary>
42  /// <param name="force">New value.</param>
43  private void BrushForce_OnValueChanged(float force)
44  {
45      if (_painter != null)
46      {
47          _painter.forceImpact = force;
48      }
49  }
50
51  /// <summary>
52  /// Sets new brush size to a painter if its value changes.
53  /// </summary>
54  /// <param name="size">New value.</param>
55  private void BrushSize_OnValueChanged(int size)
56  {
57      if (_painter != null)
58      {
59          _painter.Size = size;
60      }
61  }
```

```
62
63 /// <summary>
64 /// Sets new brush color to a painter if its value changes.
65 /// </summary>
66 /// <param name="color">New value.</param>
67 private void BrushColor_OnValueChanged(Color color)
68 {
69     if (_painter != null)
70     {
71         _painter.DrawColor = color;
72     }
73 }
```



Obrázek E.8: Řešení úkolu 4 - Oprava kreslení

Bibliografie

[23a] *API Documentation & Design Tools for Teams | Swagger* [online]. Swagger, 2023 [cit. 2023-06-28]. Dostupné z: https://swagger.io/.

[23b] *AutoMapper* [online]. AutoMapper, 2023 [cit. 2023-06-28]. Dostupné z: https://automapper.org/.

[23c] *Azure Kinect DK – vývoj modelů AI | Microsoft Azure* [online]. Microsoft, 2023 [cit. 2023-06-28]. Dostupné z: https://azure.microsoft.com/cs-cz/products/kinect-dk.

[16a] *Camera Tools for GoPro Heros* [online]. Toolsforgopro, 2016 [cit. 2023-06-28]. Dostupné z: https://www.toolsforgopro.com/cameratools.

[Dek16] DEKKER, Marcel Douwe. *Global illustration of the 4+1 Architectural View Model — Wikipedia, The Free Encyclopedia*. 2016. Dostupné také z: https://commons.wikimedia.org/wiki/File:4+1_Architectural_View_Model.svg. [Online; accessed 12-June-2023] Tato práce je licencovaná pod Creative Commons Attribution 3.0 International License. Kopie licence je dostupná na http://creativecommons.org/licenses/by/3.0/.

[23d] *Depth Camera D415 – Intel® RealSense™ Depth and Tracking Cameras* [online]. Intel® RealSenseTM Depth a Tracking Cameras, 2023-04 [cit. 2023-06-28]. Dostupné z: https://www.intelrealsense.com/depth-camera-d415/.

[23e] *Docker Compose overview | Docker Documentation* [online]. Docker, 2023-06 [cit. 2023-06-28]. Dostupné z: https://docs.docker.com/compose/.

[Duc21] DUCHEK, Vladimír. *Strategický záměr Západočeské univerzity v Plzni pro období 2021–2025*. Plzeň, 2021.

[Gas+23] GASTER, Brady; ANDERSON, Rick; BUCK, Alex; LATHAM, Luke; PICKETT, Wade et al. *Přehled ASP.NET Core SignalR* [online]. Microsoft, 2023-05 [cit. 2023-06-28]. Dostupné z: https://learn.microsoft.com/cs-cz/aspnet/core/signalr/introduction?view=aspnetcore-7.0.

[GSW18] GRUNNET-JEPSEN, Anders; SWEETSER, John N.; WOODFILL, John. *Introducing the Intel® RealSenseTM D400 Product Family* [online]. Intel® RealSenseTM Depth a Tracking Cameras, 2018-02 [cit. 2023-06-28]. Dostupné z: https://www.intelrealsense.com/introducing-intel-realsense-d400-product-family/.

[23f] *Haptics Direct for Unity V1* [online]. Unity Asset Store, 2023 [cit. 2023-06-28]. Dostupné z: https://assetstore.unity.com/packages/tools/integration/haptics-direct-for-unity-v1-197034.

[21a] *HapticsDirect for Unreal* [online]. Unreal Engine, 2021 [cit. 2023-06-28]. Dostupné z: https://unrealengine.com/marketplace/en-US/product/hapticsdirect-for-unreal.

[23g] *HTTP Specification v2.0 : Open GoPro* [online]. GoPro, 2023 [cit. 2023-06-28]. Dostupné z: https://gopro.github.io/OpenGoPro/http_2_0.

[21b] *Intel RealSense SDK 2.0 – Intel RealSense Depth and Tracking cameras* [online]. Intel® RealSenseTM Depth a Tracking Cameras, 2021-07 [cit. 2023-06-28]. Dostupné z: https://www.intelrealsense.com/sdk-2/.

[Itu22] ITURBE, Konrad. *GoPro API for Python* [online]. GitHub, 2022 [cit. 2023-06-28]. Dostupné z: https://github.com/KonradIT/gopro-py-api.

[Itu23a] ITURBE, Konrad. *GoPro Wifi Hack* [online]. GitHub, 2023 [cit. 2023-06-28]. Dostupné z: https://github.com/KonradIT/goprowifihack.

[Itu23b] ITURBE, Konrad. *Python Bluetooth controller for GoPro cameras with BLE connection* [online]. GitHub, 2023 [cit. 2023-06-28]. Dostupné z: https://github.com/konradit/gopro-ble-py.

[Kön23] KÖNIG, Alex. *Software pro demonstrační laboratoř techniky*. Plzeň, 2023. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Doc. Ing. Libor Váša, Ph.D.

[Kru95] KRUCHTEN, Philippe. The 4+1 View Model of architecture. *IEEE Software*. 1995, roč. 12, č. 6, s. 42–50. Dostupné z DOI: 10.1109/52.469759.

[Leb23] LEBENDIG, Usnea. *GoPro Comparison Chart & Buyers Guide* [online]. Shotkit, 2023-06 [cit. 2023-06-28]. Dostupné z: https://shotkit.com/gopro-comparison/.

[13] *MessagePack: It's like JSON. but fast and small* [online]. MessagePack, 2013 [cit. 2023-06-28]. Dostupné z: https://msgpack.org/.

[Mor23] MORRIS, Richard. *Swashbuckle.AspNetCore: Swagger tools for documenting API's built on ASP.NET Core* [online]. GitHub, 2023-01 [cit. 2023-06-28]. Dostupné z: https://github.com/domaindrivendev/Swashbuckle.AspNetCore.

[23h] *Open GoPro* [online]. GoPro, 2023 [cit. 2023-06-28]. Dostupné z: https://gopro.com/en/us/info/open-gopro.

[16b] *OpenHaptics Developer Software* [online]. 3D Systems, 2016-10 [cit. 2023-06-28]. Dostupné z: https://www.3dsystems.com/haptics-devices/openhaptics.

[18a] *Qualcomm and GoPro Collaborate on HERO7 Silver and White Cameras* [online]. Hongkong: Qualcomm, 2018-10 [cit. 2023-06-28]. Dostupné z: https://www.qualcomm.com/news/releases/2018/10/qualcomm-and-gopro-collaborate-hero7-silver-and-white-cameras.

[20] *Runtime File Browser* [online]. Unity Asset Store, 2020 [cit. 2023-06-28]. Dostupné z: https://assetstore.unity.com/packages/tools/gui/runtime-file-browser-113006.

[15] *Runtime OBJ Importer* [online]. Unity Asset Store, 2015 [cit. 2023-06-28]. Dostupné z: https://assetstore.unity.com/packages/tools/modeling/runtime-obj-importer-49547.

[23i] *Technické údaje PS VR2 | Displej, nastavení a kompatibilita PlayStation VR2 (Česká republika)* [online]. Playstation, 2023 [cit. 2023-06-28]. Dostupné z: https://www.playstation.com/cs-cz/ps-vr2/ps-vr2-tech-specs/.

[16c] *Touch Haptic Device* [online]. 3D Systems, 2016-06 [cit. 2023-06-28]. Dostupné z: https://www.3dsystems.com/haptics-devices/touch.

[17] *Under the Hood of HERO6 BLACK* [online]. GoPro, 2017-10 [cit. 2023-06-28]. Dostupné z: https://gopro.com/en/us/news/what-is-the-gp1-chip-in-hero6-black.

[18b] *VIVE Pro | VIVE European Union* [online]. Vive, 2018 [cit. 2023-06-28]. Dostupné z: https://www.vive.com/eu/product/vive-pro/.

[23j] *VIVE Pro 2 Specs | VIVE United States* [online]. Vive, 2023 [cit. 2023-06-28]. Dostupné z: https://www.vive.com/us/product/vive-pro2/overview/.

[23k] *VIVE Wave - Developer Resources* [online]. Vive, 2023 [cit. 2023-06-28]. Dostupné z: https://developer.vive.com/resources/vive-wave/.

[Wag+23] WAGNER, Bill; PEDRI, Sergio; KULIKOV, Petr; WARREN, Genevieve et al. *Common C# Coding Conventions* [online]. Microsoft, 2023-03 [cit. 2023-06-28]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions.

[Wei18] WEINBERGER, Matt. *The rise and fall of Kinect: Why Microsoft gave up on its most promising product* [online]. Yahoo Finance, 2018 [cit. 2023-06-28]. Dostupné z: https://finance.yahoo.com/news/downfall-kinect-why-microsoft-gave-183900710.html?guccounter=1.

[21c] *What is a Container? | Docker* [online]. Docker, 2021-11 [cit. 2023-06-28]. Dostupné z: https://www.docker.com/resources/what-container/.

[You21] YOUNG, Leslie. *Various scripts related to Unity UI Toolkit (UIElements)* [online]. GitHub, 2021-07 [cit. 2023-06-28]. Dostupné z: https://github.com/plyoung/UIElements.

Seznam obrázků

Seznam výpisů