Univerzita Palackého v Olomouci
Přírodovědecká fakulta
Katedra geoinformatiky

VIZUALIZACE AKTUÁLNÍCH FYZICKOGEOGRAFICKÝCH PODMÍNEK V DETAILNÍM 3D MODELU ÚZEMÍ

Diplomová práce

Radim HOLUB

Vedoucí práce RNDr. Jan BRUS, Ph.D

Olomouc 2024
Geoinformatika a kartografie

ANOTACE

Diplomová práce se zaměřuje na vytvoření desktopové aplikace, která vizualizuje a simuluje prostředí vybrané lokality v Litovelském Pomoraví. Aplikace využívá reálná data získaná pomocí dvou senzorů umístěných v tůni a v řece Morava, a také ze webové služby weatherbit.io. Po spuštění aplikace se uživateli zobrazí hlavní menu, které slouží jako rozcestník a zahrnuje digitálního průvodce, statistiky dat a možnost přístupu do hlavní scény. V hlavní scéně se prostředí dynamicky mění na základě telemetrických dat, což zahrnuje změny oblačnosti, rychlosti větru, výšce hladiny řeky a tůně, a další. Uživatel má možnost prozkoumat prostředí v různých ročních obdobích a sledovat historické události, jako jsou povodně. Uživatel si dále může zvolit vlastní datum, na základě kterého se dynamicky mění prostředí aplikace. Aplikace umožňuje individuální nastavení ovládání a grafických parametrů podle výkonu počítače uživatele. Optimalizace aplikace zahrnuje využití různých úrovní detailu pro objekty, optimalizaci počtu drawcalls pomocí metody CombineMesh a postprocessing v HDRP verzi 14.0.1. Nakonec bylo provedeno testování webového řešení aplikace.

KLÍČOVÁ SLOVA

Digital twins; telemetrická data; vizualizace; herní engine; environment architecture

Počet stran práce: 64

Počet příloh: 4 (z toho 4 elektronické)

ANOTATION

The thesis focuses on the creation of a desktop application that visualizes and simulates the environment of a selected location in Litovelské Pomoraví. The application uses real data obtained from two sensors located in a pond and in the Morava River, as well as from the weatherbit.io web service. After launching the application, the user is presented with a main menu that serves as a signpost and includes a digital guide, data statistics and the possibility to access the main scene. In the main scene, the environment changes dynamically based on telemetry data, which includes changes in cloud cover, wind speed, river and pool elevations, and more. The user has the ability to explore the environment in different seasons and track historical events such as floods. The user can also choose a custom date based on which the application environment changes dynamically. The application allows individual adjustment of the control and graphic parameters according to the user's computer performance. Optimization of the application includes the use of different levels of detail for objects, optimization of the number of drawcalls using the CombineMesh method and postprocessing in HDRP version 14.0.1. Finally, testing of the web-based application was performed.

KEYWORDS

Digital twins; telemetry data; visualition; game engine; environment architecture

Number of pages: 64
Number of appendixes: 4

Prohlašuji, že

- diplomovou práci včetně příloh, jsem vypracoval samostatně a uvedl jsem všechny použité podklady a literaturu.

- jsem si vědom, že na moji diplomovou práci se plně vztahuje zákon č.121/2000 Sb. - autorský zákon, zejména § 35 – využití díla v rámci občanských a náboženských obřadů, v rámci školních představení a využití díla školního a § 60 – školní dílo,

- beru na vědomí, že Univerzita Palackého v Olomouci (dále UP Olomouc) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užívat (§ 35 odst. 3),

- souhlasím, že údaje o mé diplomové práci budou zveřejněny ve Studijním informačním systému UP,

- v případě zájmu UP Olomouc uzavřu licenční smlouvu s oprávněním užít výsledky a výstupy mé diplomové práce v rozsahu § 12 odst. 4 autorského zákona,

- použít výsledky a výstupy mé diplomové práce nebo poskytnout licenci k jejímu využití mohu jen se souhlasem UP Olomouc, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly UP Olomouc na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Olomouci dne: 25.04.2024 Radim Holub

Děkuji vedoucímu práce RNDr. Janu Brusovi, Ph.D a Mgr. Radku Barvířovi, Ph.D za připomínky, náměty a cenné rady při vypracování diplomové práce. Také děkuji všem, co se podíleli na testování aplikace. Děkuji KGI za to, že po pět let studia mi poskytla nejen vzdělání, ale představuje i místo, kam se rád budu vracet. Děkuji také všem, kteří se mnou sdíleli své myšlenky, zkušenosti, rady a znalosti během studia, jež budu moct využít v praxi. V neposlední řadě děkuji svojí mamince, tatínkovi, sourozencům, nejbližší rodině a mojí Petře a přátelům za podporu během studia.

Diplomová práce vznikla za podpory projektu MOSPREMA: Predikce a management kalamitních stavů komárů pro zachování biodiverzity v lužních lesích podpořeného Norskem prostřednictvím Norských fondů.

UNIVERZITA PALACKÉHO V OLOMOUCI

Přírodovědecká fakulta

Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Radim HOLUB**
Osobní číslo:	**R220008**
Studijní program:	**N0532A330009 Geoinformatika a kartografie**
Téma práce:	**Vizualizace aktuálních fyzickogeografických podmínek v detailním 3D modelu území**
Zadávající katedra:	**Katedra geoinformatiky**

Zásady pro vypracování

Cílem diplomové práce je vizualizovat aktuální fyzickogeografické podmínky s využitím integrace meteorologických a hydrologických dat ve 3D modelu území. Student vytvoří detailní 3D model vybraného území s cílem v něm vizualizovat aktuální dynamicky se měnící podmínky prostředí. Data budou získávána ze senzorové sítě a od poskytovatelů informací o počasí. Výsledné vizualizace budou plně využívat aktuální grafické možnosti, které přináší tvorba virtuálních prostředí v programu Unity. Vedlejším cílem práce je optimalizace takto vytvořených vizualizací pro publikování ve webovém prostředí a testované možnosti vizualizace historických záznamů.
Text práce student zpracuje v souladu se závaznou šablonou pro kvalifikační práce KGI. O diplomové práci student vytvoří webovou stránku a poster. Celou práci (text, přílohy, výstupy, zdrojová a vytvořená data, poster a web) odevzdá student v digitální podobě na datové úložiště katedry. Do evidence STAG student odevzdá úplný text práce s přílohami, které určí vedoucí práce. Fyzicky student odevzdá výtisk posteru ve formátu A2 a přílohy určené vedoucím práce.

Rozsah pracovní zprávy:	**max. 50 stran**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování diplomové práce:	**elektronická**

Seznam doporučené literatury:

BAUER, Peter; STEVENS, Bjorn; HAZELEGER, Wilco. A digital twin of Earth for the green transition. Nature Climate Change, 2021, 11.2: 80-83.

HAM, Youngjib; KIM, Jaeyoon. Participatory sensing and digital twin city: Updating virtual city models for enhanced risk-informed decision-making. Journal of Management in Engineering, 2020, 36.3: 04020005.

HELBIG, Carolin, et al. Concept and workflow for 3D visualization of atmospheric data in a virtual reality environment for analytical approaches. Environmental earth sciences, 2014, 72: 3767-3780.

NOUEIHED, Hamza; HARB, Heba; TEKLI, Joe. Knowledge-based virtual outdoor weather event simulator using unity 3D. The Journal of Supercomputing, 2022, 78.8: 10620-10655.

PEDERSEN, Agnethe N., et al. Living and prototyping digital twins for urban water systems: towards multi-purpose value creation using models and sensors. Water, 2021, 13.5: 592.

LI, David, et al. Meteovis: Visualizing meteorological events in virtual reality. In: extended abstracts of the 2020 CHI conference on human factors in computing systems. 2020. p. 1-9.

Vedoucí diplomové práce: **RNDr. Jan Brus, Ph.D.**
Katedra geoinformatiky

Datum zadání diplomové práce: **9. prosince 2022**
Termín odevzdání diplomové práce: **9. května 2024**

L.S.

doc. RNDr. Martin Kubala, Ph.D.
děkan

prof. RNDr. Vilém Pechanec, Ph.D.
vedoucí katedry

V Olomouci dne 1. září 2023

OBSAH

SEZNAM POUŽITÝCH ZKRATEK

Zkratka	Význam
BIM	Building information modeling
C3S	Copernicus Climate Change Service
CAD	Computer-aided design
CoAP	Constrained Application Protocol
CUZK	Český úřad zeměměřický a katastrální
DMK	Digitální model krajiny
DMP	Digitální model povrchu
DMR	Digitální model reliéfu
DTM	Digitální model terénu
ECMWF	European Centre for Medium-Range Weather Forecasts
ESA	European Space Agency
EU	European Union
FMI	Finnish Meteorological Institute
FPS	Frames per second
GIS	Geographic information system
GPU	Graphics processing unit
GUI	Graphical user interface
GTX	Giga Texel Shader eXtreme
HDRP	High definition render pipeline
HTTP	Hypertext Transfer Protocol
IOT	Internet of things
LI2CPP	Intermediate Language To C++
LIDAR	Light Detection and Ranging
LOD	Level of detail
MQTT	Message Queuing Telemetry Transport
NASA	National Aeronautics and Space Administration
PLM	Product lifecycle management
REST API	Representational State Transfer application programming interface
RGB	Red Green Blue
RPAS	Remotely piloted aircraft system
RRF	Recovery and Resilience Facility
SDK	Software development kit
SGI	Screen Space Global Illumination
SPA	Stupně povodňové aktivity
SSAO	Screen Space Ambient Occlusion
SSR	Screen Space Reflection
UI	User interface
UX	User experience
WebGL	Web Graphics Library
WYSIWYP	What you see is what you get

ÚVOD

V posledních letech svět čelí narůstajícímu úbytku vody a nárůstu extrémních přírodních jevů, jako jsou častější a intenzivnější sucho, záplavy, povodně a další katastrofy. Tyto změny mají značný dopad na životní prostředí, lidské aktivity a celkovou udržitelnost naší planety. Je stále naléhavější monitorovat a porozumět těmto změnám, abychom mohli efektivněji předcházet jejich negativním dopadům.

V tomto kontextu hraje klíčovou roli využití digitálních dvojčat. Princip digitálních dvojčat umožňuje vizualizaci a simulaci historických dat o vybraném prostředí a jeho klimatu. Díky těmto nástrojům mohou uživatelé sledovat a analyzovat změny v prostředí v průběhu času a lépe porozumět dynamice klimatických jevů. Zaznamenávání těchto extrémních událostí a sledování historických dat má klíčový význam pro identifikaci dlouhodobých trendů a pro přijímání informovaných rozhodnutí ohledně ochrany životního prostředí a adaptace na klimatické změny.

Interaktivní aplikace postavená na principech digitálních dvojčat umožní mladé generaci lépe porozumět problematice klimatických změn než tradiční výuka ve školních učebnicích. Díky integraci herních principů a prostředí ve 3D světě si široká veřejnost snadněji představí vliv klimatických změn a extrémních přírodních jevů na životní prostředí i naše každodenní životy. Možnost využívat aplikaci jako historický deník umožňuje uživatelům sledovat, jak se prostředí mění v reálném čase, což je klíčové pro efektivní řízení prostředí a adaptaci na klimatické změny.

1 CÍLE PRÁCE

Cílem diplomové práce je vizualizovat aktuální fyzickogeografické podmínky s využitím integrace meteorologických a hydrologických dat ve 3D modelu území. Vytvořit detailní 3D model vybraného území s cílem vizualizovat aktuální dynamicky se měnící podmínky prostředí. Data budou získávána ze senzorové sítě a od poskytovatelů informací o počasí. Výsledné vizualizace budou plně využívat aktuální grafické možnosti, které přináší tvorba virtuálních prostředí v programu Unity. Vedlejším cílem práce je optimalizace takto vytvořených vizualizací pro publikování ve webovém prostředí a testování možností vizualizace historických záznamů.

2 SOUČASNÝ STAV ŘEŠENÉ PROBLEMATIKY

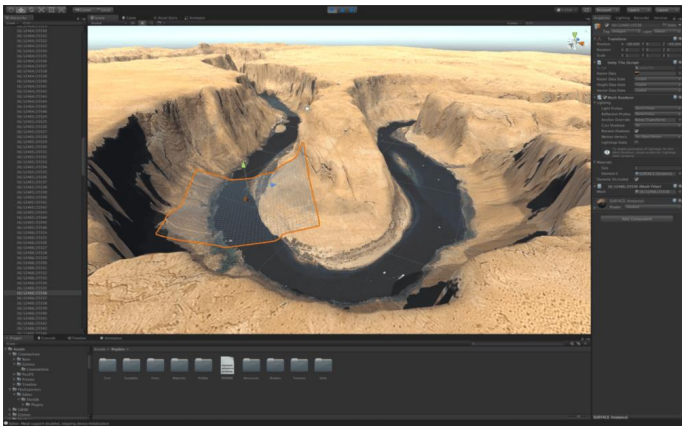
Lidé od pradávna projevovali snahu zaznamenat a vizualizovat prostředí, ve kterém žijí. Jednou z nejstarších dochovaných památek kartografického charakteru s fyzickogeografickým tématem je rytina pavlovského tábořiště lovců mamutů. Tato mapa byla vyryta na hrotu mamutího klu a byla objevena roku 1962 v Pavlově na jižní Moravě. Památka pochází z mladšího paleolitu, z doby asi před 28 000 až 24 800 lety. Další významné nálezy pochází ze švýcarských jeskyní poblíž města Schaffhausen, povodí řeky Jenisej a Ladožského jezera, zachycují vodní toky, tábořiště a lovecké stezky našich předků (Bauerová, 2019).

V dnešní době existuje mnoho forem vizualizace fyzickogeografických dat. Stále se můžeme setkat s analogovými mapami, jako jsou školní atlasy, nebo státní mapová díla České republiky – fyzickogeografická mapa České republiky v měřítku 1 : 500 000, zobrazující celé území České republiky na jednom papírovém listě (ČÚZK). Dále to mohou být digitální webové mapy, které se dělí podle druhu dat na statické a dynamické. Fyzickogeografická data se dají vizualizovat i pomocí interaktivních grafů bez použití map, příkladem je ThingsBoard. Jedná se o open-source platformu pro sběr, zpracování, vizualizaci a správu zařízení. ThingsBoard umožňuje připojení zařízení prostřednictvím standardních protokolů IoT - MQTT, CoAP a HTTP (ThingsBoard).

S rozvojem mobilních dotykových zařízení, jako jsou telefony a tablety, se otevřely nové možnosti pro vizualizaci fyzickogeografických dat prostřednictvím mobilních aplikací. Tyto aplikace nejenomže umožňují uživatelům pohodlněji procházet a analyzovat geografická data, ale také poskytují jednoduché a přehledné nástroje pro okamžitou interpretaci informací. Jedním z významných příkladů takové mobilní aplikace je Windy, která byla vyvinuta českým zakladatelem Seznamu, Ivem Lukačovičem (Lukačovič, 2015). Dalším příkladem mobilní aplikace zaměřené na vizualizaci fyzickogeografických dat je Google Earth for mobile. Tato aplikace umožňuje uživatelům prozkoumávat virtuální globus a přiblížit se k libovolnému místu na Zemi. S využitím satelitních snímků a 3D modelů terénu poskytuje Google Earth for mobile poutavý způsob, jak procházet a zkoumat geografická data (Anon).

V posledních letech vývoje se čím dál častěji setkáváme s řešeními využívajícími virtuální a rozšířenou realitu postavenými nad herními enginy využívajícími principy digital twins. Příkladem vizualizace prostředí pomocí herních enginů zahrnuje využití SDK. Jednou z možností je ArcGIS Maps pro Unity, které vytvořila společnost Esri. Toto SDK umožňuje přístup k datům vedeným pod účtem ArcGIS a také slouží jako API pro vytváření geografických aplikací. ArcGIS Maps SDK for Unity umožňuje vizualizovat obrovské množství dat přímo v herním enginu Unity. Po propojení s Esri účtem umožňuje uživatelům měřit geodetické vzdálenosti ve 3D sférickém prostoru na svých datech. Toto SDK funguje pomocí REST API, které umožňuje dotazování dat a získávání seznamů prvků s jejich polohou a dalšími atributy. Jednou z hlavních funkcionalit SDK je rychlé a detailní načítání světa pomocí dlaždic. Dále poskytuje možnost přímého geokódování, což je proces převodu adres nebo místních jmen na geografické souřadnice (Esri). Dalším příkladem je SDK Mapbox for Unity, které podobně jako ArcGIS Maps SDK for Unity umožňuje vizualizaci GIS dat a poskytuje sadu nástrojů pro vytváření GEO/GIS aplikací z reálných dat. Toto SDK také umožňuje připojení přes REST API a poskytuje možnost generování jednotlivých dílků světa pro tvorbu 2D / 3D map. Mapbox SDK for Unity rovněž podporuje rozšířenou realitu a virtuální realitu, což umožňuje vytvářet interaktivní prostředí s geografickými daty. Uživatel má také plný přístup k webovým službám poskytovaným společností Mapbox, což zahrnuje API pro vektorové dlaždice, rastrové dlaždice, statické obrázky,

geokódování, směrování a mapování tras. Díky těmto funkcím mohou vývojáři využívat SDK Mapbox for Unity k vytváření komplexních geoaplikací s možností distribuce a šíření pomocí různých API poskytovaných společností Mapbox (Mapbox).



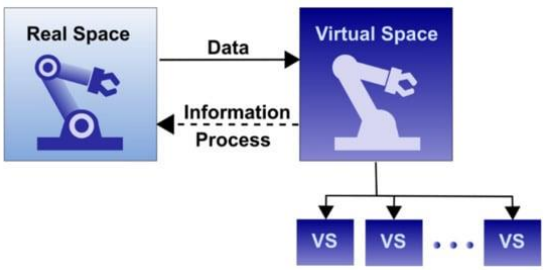
Obr. 2.1.2 SDK Mapbox for Unity (zdroj: https://www.mapbox.com/unity)

Jednou z nejpokročilejších aplikací, zahrnující nejmodernější trendy a principy digital twins, je vGIS (vGIS). Aplikace vGIS je primárně navržena pro zobrazení inženýrských sítí a umožňuje uživatelům pracovat s geografickými informačními systémy přímo v terénu. Jednou z hlavních výhod této aplikace je možnost zobrazit GIS data přímo na místě práce a provádět úpravy a zpřesňování polohy těchto sítí v reálném čase – její architektura je založena na principech digital twins. Tímto způsobem je možné efektivněji a přesněji pracovat s geografickými daty a provádět úpravy potřebné k optimalizaci inženýrských sítí. Jednou z unikátních funkcí aplikace vGIS je schopnost zobrazovat také BIM data, což jsou informační modely budov. To umožňuje uživatelům zobrazit nejen samotné inženýrské sítě, ale také prostorové informace o budovách, což může být užitečné například při plánování rozšíření sítí v blízkosti staveb. Další výhodou použití aplikace vGIS je možnost provádět záznamy přímo na místě. To znamená, že uživatelé mohou snadno přistupovat k informacím o konkrétních částech sítí, jako jsou například plynovody, vodovody nebo kabely, a provádět úpravy nebo doplňovat potřebné záznamy přímo na místě. Celé řešení funguje na principu načítání BIM podkladů, které jsou nahrány do aplikace. Pro používání aplikace je nezbytný chytrý mobilní telefon s operačními systémy Android nebo iOS. Je však důležité mít stabilní internetové připojení, aby bylo možné efektivně pracovat s daty v reálném čase (vGIS). Tento vývoj od pravěku až po moderní technologie ukazuje, jak lidé odjakživa touží po pochopení a vizualizaci prostředí, ve kterém žijí…

Obr. 2.1.2 Aplikace vGIS (zdroj: https://www.vgis.io)

2.1 Digital Twins

V současném světě chápeme význam digitálního dvojčete jako: Digitální odraz fyzického objektu, procesu, služby či prostředí, který emuluje chování a vizuální podobu svého reálného protějšku (Twi-Global). Digitální dvojče je v podstatě počítačový program, který využívá data z reálného světa k vytvoření simulací, které mohou předpovědět, jak bude objekt nebo proces fungovat. Simulace je založena jak na aktuálním stavu, tak na historických datech. Tyto programy mohou integrovat umělou inteligenci a softwarovou analytiku, aby vylepšily výstupy. Využití IoT umožňuje přenášet data z reálného světa a vytvářet tak virtuální reprezentaci v digitálním prostředí (Twi-Global). Nebo digitální dvojčata můžeme chápat jako: Virtuální repliku fyzického aktiva nebo procesu. Digitální dvojčata jsou vytvářena pomocí inženýrského simulačního softwaru a simulují a zrcadlí chování, výkon a prostředí svého reálného protějšku v reálném čase. Při vytváření digitálního dvojčete aktiva je nejprve vytvořen podrobný model a poté je spárován s daty, včetně všech dostupných údajů a údajů ze senzorů. Tyto informace jsou pak vloženy do modelu a vytváří digitální dvojče (Akselos). Původ pojetí digitálního dvojčete, tak jak ho chápeme a známe dnes, vychází ze souvislostí s vývojem řízením životního cyklu výrobku PLM v roce 2002 (Grieves 2016). Navržený model měl tři složky: reálný prostor, virtuální prostor a propojovací mechanismus pro tok dat/informací mezi nimi - model byl tehdy označen jako "Mirrored Spaces Model" (Grieves 2005). Nicméně samotná technologie digitálního dvojčete je ve skutečnosti konceptem praktikovaným již od 60. let 20. století. Poprvé byla použita kosmickou agenturou NASA. NASA vytvořila repliky svých kosmických lodí na zemské úrovni s cílem simulovat podmínky vesmírných misí a umožnit efektivnější průzkumné mise. Tato technologie byla výrazně uplatněna během mise Apollo 13, kde propojená digitální dvojčata umožnila řídicím střediskům rychle adaptovat simulace na aktuální stav poškozené kosmické lodě a vyvinout strategie pro bezpečný návrat astronautů na Zemi (Miskinis 2019). V počátcích sedmdesátých let byly hlavní rámce počítačů využívány jako digitální dvojčata pro monitorování rozsáhlých zařízení, jako například elektráren. V osmdesátých letech vznikly 2D CAD systémy, jako například AutoCAD, umožňující tvorbu technických výkresů a revolučně změnily způsob, jakými bylo nahlíženo na provázanost mezi realitou a digitálním otiskem (Unity). Podobnou koncepci, v níž softwarové modely napodobují realitu na základě vstupních informací z fyzického světa, vymyslel David Gelernter v roce 1993 a nazval ji "Mirror Worlds" (Gelernter 1993). V roce 2003 Kary Främling a kol. rovněž navrhli "architekturu založenou na agentech, kde každá položka výrobku má odpovídající "virtuální protějšek" nebo agenta, který je s ní spojen", jako řešení neefektivity přenosu výrobních informací prostřednictvím papíru pro PLM (Främling 2003). Do roku 2006 byl název koncepčního modelu navrženého Grievesem změněn z "Mirrored Spaces Model" na "Information Mirroring Model" (Grieves 2009. Model kladl důraz na to, aby mechanismus propojení dvou prostorů byl obousměrný a aby pro jeden reálný prostor existovalo více virtuálních prostorů, v nichž lze zkoumat alternativní nápady nebo návrhy.



Obr. 2.1.2.1 Mirrored Spaces Mode / Information Mirroring Model (zdroj: Michael W. Grieves)

Vzhledem k omezeným technologiím, jako byl nízký výpočetní výkon, nízká nebo žádná konektivita zařízení s internetem, ukládání a správa dat, nedostatečně vyvinuté strojové algoritmy atd. neměly digitální dvojčata v té době praktické využití.

To ovšem v současnosti neplatí a digitální dvojčata mají široké uplatnění:

1. Výroba, technologie

   Digitální dvojčata se stále více používají ke zvýšení efektivity ve výrobě. Vytvořením virtuální repliky průmyslového procesu mohou výrobci rychleji vyhodnocovat data, analyzovat trendy výkonnosti v čase a optimalizovat své operace. Jedním ze způsobů, jak se digitální dvojčata uplatňují, je monitorování výrobních linek. Společnosti je mohou využívat ke sledování klíčových ukazatelů výkonu, jako je teplota, tlak a rychlost. Mohou také sledovat podmínky prostředí a odhalovat případné anomálie ve výrobě. Tyto anomálie mohou indikovat možný problém nebo závadu. Digitální dvojčata v průmyslu mohou být využita k poskytování pokynů pro prediktivní údržbu zařízení. Díky tomu mohou identifikovat potenciální problémy dříve, než se z nich stanou nákladné poruchy. To pomáhá zkrátit prostoje a zvýšit spolehlivost jejich výrobků (Ranjan 2023).

2. Automobilový průmysl

   Pokud jde o automobilový průmysl, digitální dvojčata jsou užitečná při navrhování, výrobě a poprodejních službách. Inženýři mohou testovat mnoho návrhů a vybrat ty nejvýkonnější, aniž by museli vyvíjet jediný fyzický prototyp. Výrobní proces lze optimalizovat, čímž se zkrátí prostoje a zvýší rychlost výroby. Prediktivní údržba pomůže servisovat vozidla dříve, než by mohl nastat jakýkoli problém. Digitální dvojčata v automobilovém průmyslu mají řadu výhod a s přechodem trhu na elektromobily se potřeba digitálních dvojčat stává klíčovou složkou. Například Volvo používá digitální dvojčata k testování nových konstrukcí. Vytváří virtuální repliky k testování a zkoušení různých materiálů a aerodynamiky nových konstrukcí vozidel. Tímto způsobem jsou schopni vybrat ideální konstrukci, která by zlepšila výkon a vytvořila úsporné modely (Ranjan 2023).

Obr. 2.1.2.2 Příklad digitálního dvojčete v automobilovém průmyslu – Volvo (zdroj https://www.toobler.com/blog/digital-twin-examples)

3. Energie

   V důsledku nedávné pandemie COVID-19 se v odvětví energetiky a veřejných služeb výrazně zrychlily digitální technologie, včetně technologie digitálních dvojčat. Tato technologie umožňuje společnostem vytvářet virtuální modely svých energetických aktiv a simulovat jejich výkon v reálném čase. Díky tomu může výrazně snížit potřebu fyzického testování a snížit náklady. Navíc může pomoci zlepšit celkovou kvalitu a výkonnost ochranných systémů (Ranjan 2023). Technologie digitálních dvojčat může také umožnit, aby testovací činnosti probíhaly na dálku a kdykoli. Nemusí sice zcela nahradit testování na místě, ale může je výrazně omezit. Například společnost Shell používá digitální dvojčata pro své námořní plošiny k optimalizaci výroby a zvýšení bezpečnosti. Shromažďuje data v reálném čase z různých senzorů na plošině a pomocí digitálních dvojčat vizualizuje a simuluje různé scénáře. To pomáhá identifikovat problémy dříve, než se stanou kritickými, a optimalizovat provoz platformy (Oilandgasadvancement). Kromě zvýšení efektivity, produktivity a bezpečnosti mohou digitální dvojčata také zlepšit školení pracovníků.

4. Smart City

   Chytré město je městské prostředí, kde se technologie, data a konektivita spojují, aby zvýšily kvalitu života jeho obyvatel. Zaměřuje se také na optimalizaci správy zdrojů a zlepšení udržitelnosti. Mezi používané technologie patří internet věcí a pokročilá analýza dat. Pomocí těchto technologií můžeme vytvářet inteligentní systémy a infrastrukturu pro různá odvětví. Mezi tato odvětví patří mimo jiné energetika, zdravotnictví a správa věcí veřejných (Twi-Global). Digitální dvojče chytrého města lze použít k simulaci a analýze různých scénářů, jako je tok dopravy, spotřeba energie a reakce na mimořádné události. Města mohou například využít technologii digitálního dvojčete k simulaci a analýze dopravního toku, identifikaci úzkých míst a optimalizaci časování dopravních signálů. Výsledkem může být efektivnější využívání silnic a snížení dopravních zácp. Chytrá města mohou simulovat a analyzovat spotřebu energie, identifikovat oblasti, kde lze zlepšit energetickou účinnost, a optimalizovat využívání obnovitelných zdrojů energie. Výsledkem mohou být úspory nákladů a snížení emisí uhlíku. Kromě toho mohou města využívat technologii digitálních dvojčat k simulaci a analýze scénářů reakce na mimořádné události, jako jsou přírodní katastrofy a teroristické útoky. Městům to může pomoci lépe se připravit na mimořádné události a účinněji reagovat, když nastanou. Například platforma Virtual Singapur využila digitální dvojče ke zdokonalení digitální repliky městského státu (Geospatialworld). Platforma slouží jako komplexní replika Singapuru a pomáhá řešit a vyřešit problémy městského plánování (Walker 2023). S využitím digitálních dvojčat může simulovat a testovat různá řešení pro zlepšení města. Zlepšení se může týkat čehokoli od infrastruktury a dopravních systémů až po městské prostředí. Jedna z klíčových předností platformy Virtual Singapore spočívá ve schopnosti zahrnout aktuální data z různých zdrojů. Díky integraci těchto dat může platforma přesně simulovat různé scénáře. Například urbanisté mohou posoudit dopad navrhovaných infrastrukturních projektů na dopravní zácpy, identifikovat potenciální úzká místa a podle toho optimalizovat silniční sítě. Mohou také analyzovat a předpovídat dopady přívalových dešťů na odvodňovací systémy a zmírnit tak riziko záplav v ohrožených oblastech (Esri).

Obr. 2.1.2.3 Příklad digitálního dvojčete - virtuální singapurská platforma (https://www.geospatialworld.net/prime/case-study/national-mapping/virtual-singapore-building-a-3d-empowered-smart-nation/)

Koncept digitálních dvojčat se původně objevil, ale i převládá v oblasti inženýrství a osvědčil se jako virtuální reprezentace fyzického objektu či procesu, který je neustále aktualizován tak, aby reprezentoval aktuální strukturu a chování tohoto objektu či procesu (Blair, 2021). Koncept digitálních dvojčat lze aplikovat také na přírodní prostředí, což nám poskytuje nové důležité nástroje pro pochopení a správu přírodního prostředí ve všech jeho aspektech a v různých měřítkách (Blair a kol. 2020). Můžeme si například představit digitální dvojče pro povodí řeky, které se zabývá otázkami povodní, sucho a kvality vody; stejně tak bychom se mohli zabývat vývojem digitálních dvojčat pro zdravou půdu, která by fungovala na úrovni polí/farem, na regionální nebo národní úrovni a odpovídala by na různé otázky týkající se využívání půdy, například v souvislosti s ambicemi Net Zero (Netzeroclimate). Mohli bychom vyvinout digitální dvojče pro celý zemský systém a interakce mezi atmosférou, půdou, oceány a biologickou rozmanitostí v rámci různých klimatických scénářů. U některých z nich mohou dvojčata fungovat v reálném čase nebo téměř v reálném čase, u jiných se snažíme podpořit rozhodování v potenciálně velmi dlouhém časovém horizontu. V konečném důsledku by digitální dvojče mělo odrážet otázky a scénáře, na které bude odpovídat (Henrys 2023). Podle Henrys jsou Digitální dvojčata v oblasti životního prostředí poháněna nebývalým množstvím dostupných dat o přírodním prostředí z nejrůznějších zdrojů, včetně dat dálkového průzkumu ze satelitů, letadel a dronů až po potenciálně husté rozmístění pozemních senzorů. Významným zdrojem dat o životním prostředí jsou také výstupy modelů, jejichž výsledky z předchozích běhů modelů jsou často uchovávány pro budoucí analýzy. Každý z těchto zdrojů dat se může v mnoha ohledech lišit (např. v nejistotách nebo pokrytí a rozlišení v prostoru či čase) a úkolem je tyto rozdíly překonat, aby bylo možné vytvořit ucelený obraz. Datová věda hraje důležitou roli v tom, aby výsledná vysoce komplexní data dávala smysl. Vzhledem k povaze těchto dat je však zapotřebí, aby existovaly techniky datové vědy šité na míru, které by se s takovou složitostí vypořádaly (Blair, Henrys a kol. 2019).

Příkladem může být projekt https://www.freshwatercompetencecentre.com/digital-twin/, kde se věnují monitoringu povodí. Digitální dvojče poskytuje aktuální virtuální reprezentaci stavu povodí, která propojuje vlastnosti a funkce povodí s hydroklimatickým prostředím. Digitální dvojče povodí by mělo především podporovat transformaci

udržitelnosti moderní společnosti. Fyzikální scénáře modelování povodí jako takové, jsou díky aplikaci různých řešení přírodního základu relevantní při poskytování informací o nejúčinnějších opatřeních ke snížení zatížení vodních ploch. Digitální dvojče pro říční prostředí je vyvíjeno v rámci projektu Green-Digi-Basin, financovaného Evropskou unií z prostředků Finské akademie RRF (Freshwatercompetencecentre). Základní prvky digitálních dvojčat lze popsat pěti entitami, mezi něž patří fyzické, digitální, datové úložiště, připojení a entity služeb (Deren a kol 2021). Fyzická entita zahrnuje přístroje a zařízení pro mapování a monitorování prostředí. Digitální entita může obsahovat komponenty, jako je umělá inteligence, neuronové sítě, fyzikální modely nebo nástroje pro zpracování velkých objemů dat a fúzi dat, pro analýzu dat souvisejících s povodím (Gonzales-Inca a kol 2022). Pro takový systém náročný na data je třeba dobře naplánovat a nastavit ukládání dat a připojení. Jádrem výsledků jsou služby, informace a scénáře pro lepší rozhodování, lepší nástroje plánování a řízení. V současné fázi by měly být lépe odhadnuty účinky změn ve využívání půdy. Například je stále zapotřebí více informací o účincích řešení založených na přírodě blízkých opatřeních. Klíčovou roli při rozvoji nových služeb pro povodí hrají nové technologie monitorování a modelování. Pro testování byly vybrány tři tzv. supermoderní povodí Vantaanjoki, Oulankajoki a Tenojoki, která budou fungovat jako prototypy a platformy pro vytvoření nové generace digitálních dvojčat pro povodí (Freshwatercompetencecentre).

Další firma https://www.storm-geomatics.com/ se zabývá vytváření komponent pro digitální dvojčata povodí, která zlepšují navrhování, řízení a předpovídání projektů na zmírnění povodní a správu stanovišť. Postupem je spojení dat z různých senzorů a poskytnutí podrobných geoprostorových a environmentálních dat, která se používají pro informování o povodňových a environmentálních projektech. Digitální dvojče řeky umožňuje správcům pozemků, inženýrům a ekologům lépe znát geometrii celého ekosystému, řeky a záplavového území, a proto přijímat vyváženější rozhodnutí při navrhování zásahů, které ovlivňují hydrologii a ekologii říčního koridoru. Geomatic současně monitorují detaily nad i pod vodní hladinou, aby poskytli úplné informace v barevných mračnech bodů, fotogrammetrii, ortomozaických snímcích s vysokým rozlišením a digitálních modelech terénu DTM. Říční digitální dvojčata se obvykle používají pro extrakci prvků v multidisciplinárních projektech, kde se extrahovaná data exportují do softwaru třetích stran pro předpovídání povodní a vytváření biotopů. Říční digitální dvojče se stává přínosem pro zainteresované strany napojené na danou lokalitu, které jej využívají k plánování projektů, analýze typů a výšek vegetace, extrakci geometrie říčního koryta, extrakci geometrie topografických měření, záznamu dat o půdních a vodních vzorcích, záznamu časových 2D ortomozaických snímků ve vrstvách, návrhu biotopů, mapování podzemních služeb, přidávání údajů o počasí a mnoha dalších. Toho lze dosáhnout pomocí průzkumů s využitím dronů a RPAS (Geomatics).

Dalším příkladem může být projekt, kdy Evropská kosmická agentura (ESA) spustila digitální dvojče Země, což je digitální replika naší planety řízená umělou inteligencí, která převádí veškerou sílu umělé inteligence, cloud computingu, vědy o Zemi a modelování a environmentálních, společenských a ekonomických dat v měřítku Země do použitelných poznatků pro vědce a rozhodovací orgány. ESA zahájila činnosti, jejichž cílem je vytvořit prototyp příkladu digitálního dvojčete Země pro určitou tematickou oblast s využitím stávajících schopností, vědeckých poznatků a infrastruktury. Tyto aktivity tvoří portfolio projektů, které je pojmenováno jako Digital Twin Earth Precursors (Foresttwin). Foresttwin se zaměřují na vytvoření prototypu digitálního dvojčete lesů s využitím stávajících vědeckých poznatků a kapacit. Kdy projektové konsorcium bude pracovat s daty z pozorování Země, aby určilo strukturální a biologické vlastnosti lesů. Tyto informace

konsorcium využije k inicializaci souboru modelů lesních ekosystémů, které zahrnují procesy nad zemí i pod ní. Dále konsorcium vyvine nová rozhraní a vizualizační nástroje, díky nimž bude cloudová platforma snadno přístupná koncovým uživatelům. Cílem systému vyvinutého v rámci projektu je spojit lesnická data a uživatele se zájmem o lesy (Foresttwin).

Dalším velkým krokem k využívání digital twins se zavázal Finský meteorologický institut (FMI) a Evropské centrum pro střednědobou předpověď počasí (ECMWF), kdy podepsaly novou smlouvu o využití digitálních dvojčat Destination Earth pro zlepšení lesnických činností. Cílem případu užití pro lesnictví je zlepšit rozsah a výkonnost stávající aplikace Harvester Seasons, která je v provozu od května 2020, neboť byla původně vyvinuta jako případ užití pro službu Copernicus Climate Change Service (C3S). Aplikace Harvester Seasons, vyvinutá v rámci C3S a projektu E-shape programu EU Horizont 2020, již pomáhá více než 200 uživatelům ve Finsku lépe plánovat využití těžké techniky ve složitém finském lesním terénu, který je často nepřístupný kvůli sněhu, blátu a komplikovaném terénu. Aplikace poskytuje informace o vlhkosti půdy, teplotě půdy a výšce sněhu a informuje tak lesníky o vhodných okamžicích pro vjezd těžké techniky na jednotlivé lesní pozemky. Harvester Seasons má za cíl poskytovat jak dlouhodobé informace, tak krátkodobé předpovědi pro efektivnější plánování. Přesnější údaje by například umožnily přesun prostředků v případě rozsáhlé lesní plochy zasažené vichřicí, kdy je třeba včas odstranit poškozené stromy (Ecmvf).

Využití technologie digitálních dvojčat může být vhodné k monitorování stavu prostředí v důsledku sledování komářích kalamit, kterým se zabývá již zmiňující projekt MOSPREMA https://mosprema.upol.cz. Může sloužit jako historický deník vybrané lokality a z pohodlí domova přináší plnohodnotnou reálnou kopii vybraného území. Díky ní si lidé mohou uvědomit, jak často dochází k extrémním výkyvům v přírodě.

3 METODY A POSTUP ZPRACOVÁNÍ

3.1 Použité metody

Druhy metod použité v této práci se lišily na základě postupu práce. Pro urbanistické modelování byla zvolena metoda Hard-Surface. Zvolená metoda využívá pro tvorbu 3D modelů tzv. tvrdé hrany. Tedy hrany s přesnou a často uspořádanou geometrií, která se pouze minimálně ohýbá (Holub 2022). Pro dosažení co nejlepšího výkonu bylo v aplikaci provedeno několik metod. Jednou z nich je použití různých úrovní detailu (LOD) pro objekty podle jejich velikosti a vzdálenosti od kamery. Dále byla aplikována metoda CombineMesh k optimalizaci počtu drawcalls. Dále byla použita metoda Occlusion Culling, jež renderuje pouze objekty, leží ve směru pohledu kamery. Funguje na principu, že objekt typu kamera posílá virtuální paprsky směrem k herním objektům, pokud splňují předem nastavené podmínky, daný herní objekt se vykreslí. Zbylé herní objekty zůstanou nevykresleny a šetří tak výkon uživatele. Pro detekci 3D modelu byla použita metoda Physics.Raycast. Dále byly použity metody (zde vypsány bez parametrů), které se podílejí na funkčnosti celé aplikace a na dynamické změně prostředí: ToggleLeft(), ToggleRight(), Start(), ToggleUI(int), Quit(), Update(), ResetDefaultSpeed(), ResetSensitivity(), ResetFast(), ResetSlow(), LockCursor(), Resume(), Pause(), MenuScene(), OpenGLT(string), ToggleAllUI(), ButtoCheck(), ConvertToTimestamp(), ButtonSpring(), ButtonSummer(), ButtonAutumn(), ButtonWinter(), ButtonSPA3(), ButtonSPA2(), ButtonSPA1(), PostRequest(), GetDataFromSenzor1(), GetDataFromSenzor2(), GetDataFromWeather(), StatisticsUpdate(), ShowDatum(), SetTerrainPalette(TerrainLayer[]), ChangeTrees(), OnApplicationQuit(), LoadSceneSenzorDefault(), LoadSceneSenzor1(), LoadSceneSenzor2(), LateUpdate(), FollowTarget(), StartHeight(), SetTerrainPalette(), Map(), StartVolumeSky(), UpgradeHeightAfterCall(), UpgradeVolumetricClouds(), JWT_Height(), ChangeSunSettings(), ChangeSunSettingsUpdate(), SunParametreSettings(), This(), GetNazevMesice(), ChangeDefaultSpeed(), ChangeSensitivity(), ChangeFastFactor(), ChangeSlowFactor(), DisplayMessage(), StartTypewriterEffect(), ControlPostRequest(), PostRequest(), RefreshData(), GetDataFromSenzor1(), GetDataFromSenzor2(), GetDataFromSenzor3() GetDataFromWeather(), StatisticsUpdate(), ShowDatum(), LogMessage(), DeleteMessage(), QualityLow(), QualityMedium() a QualityHigh().

3.2 Použitá data

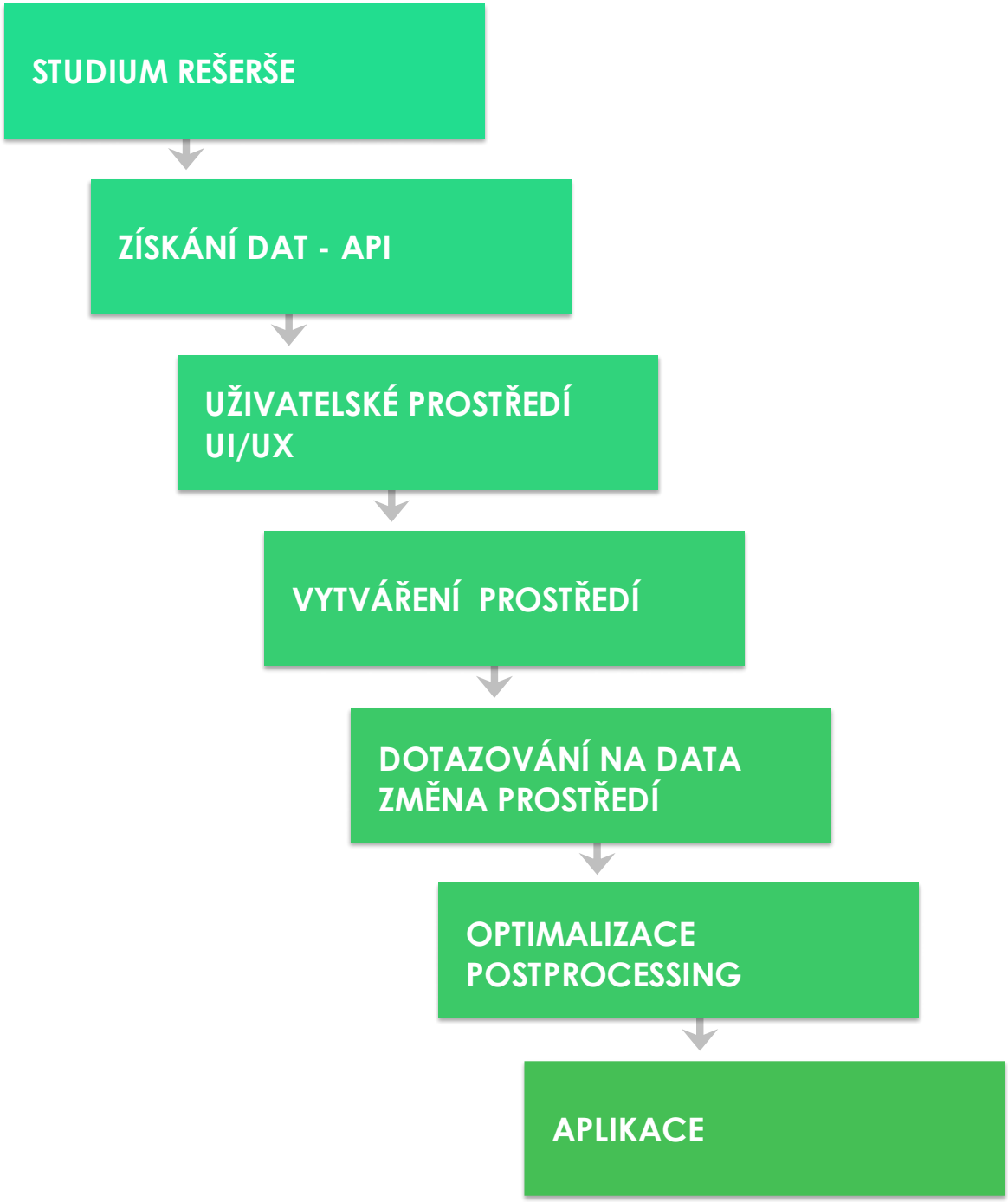
Diplomová práce vznikla za podpory projektu MOSPREMA: Predikce a management kalamitních stavů komárů pro zachování biodiverzity v lužních lesích, a proto téměř všechna data použita v diplomové práci, vychází z tohoto projektu. Pro telemetrické záznamy byly využity dva senzory. Telemetrická data jsou dále doplněna o webovou službu weatherbit.io. V diplomové práci byly použity záplavové území, ortofota, digitální model reliéfu a povrchu. Dále byly použity nejrůznější herní assety, textury apod. a to od: NatureManufacrue, EasyRoad, Polyhaven, Ambientcg, Shakeel, D.F.Y a romullus.

3.3 Použité programy

V diplomové práci bylo použito několik rozdílných softwarů. Většina práce probíhala v softwaru Unity 3D, verze 2022.3.20.f1, 2021.3.20f1 a 2022.3.16f1. Pro psaní skriptu byl používán Visual Studio verze 2022 a Visual Studio Code. Pro práci s DMR a DMP byl využit CloudCompare a Agisoft Metashape. Pro modelování 3D objektů byly použity programy

Sketchup, a to konkrétně verze 2022 a 2023. Vytváření chybějících map pro textury se provádělo v softwaru CrazyBump verze 1.22. Pro 2D grafické úpravy byl použit Inkscape verze 1.3.2 a Gimp verze 2.10.36 a 2.10.30. Pro zobrazování záplavových území byl použit QGIS verze 3.30.0 a pro dílčí práce byl využit Blender verze 3.4.

3.4 Postup zpracování

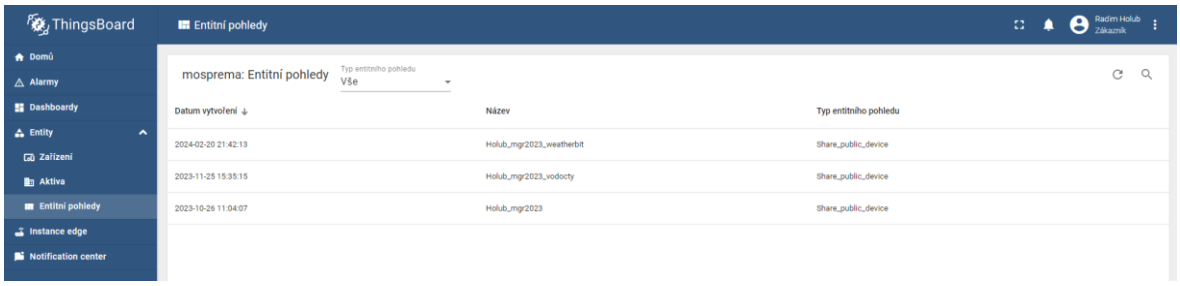


4 VLASTNÍ ŘEŠENÍ I

Cílem vlastního řešení bylo vytvořit desktopovou aplikaci s využitím integrace meteorologických a hydrologických dat ve 3D modelu území. Kde výsledné vizualizace budou plně využívat aktuální grafické možnosti, které přináší tvorba virtuálních prostředí v programu Unity. Vedlejším cílem práce byla poté optimalizace těchto vytvořených vizualizací pro publikování ve webovém prostředí a testování možnosti vizualizace historických záznamů. Všechny skripty, jež obsahuje diplomová práce, jsou k dispozici ke stažení v příloze práce.

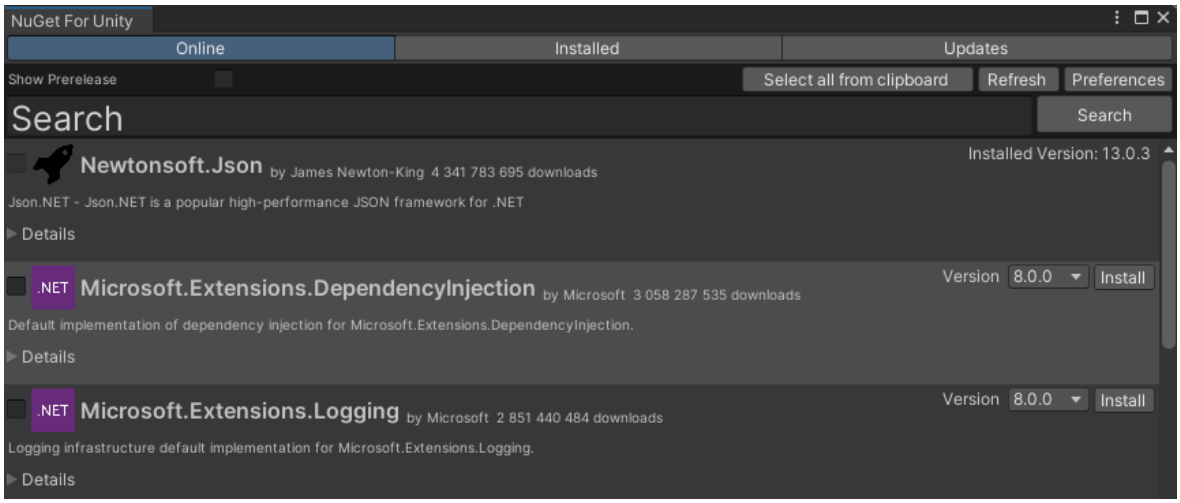
4.1 API ThingsBoard

Pro diplomovou práci byla využívána telemetrická data ze senzorů a webové služby. Telemetrická data z Litovelského Pomoraví byla ukládána na platformě ThingsBoard. Pro získání dat bylo proto potřebné navázat komunikaci mezi klientem a serverem. V platformě ThingsBoard byly ukládány průměrné hodinové hodnoty dat, které se získávaly ze senzorů a webové služby, jež byly instalovány v Litovelském Pomoraví, a následně byly zpracovány v prostorové databázi. V diplomové práci byla získávána data ze dvou senzorů a jedné webové služby. První senzor byl umístěn v tůni za fotbalovým hřištěm u obce Střeň. Tento senzor získává data, jako je teplota půdy, výška hladiny tůně, teplota vzduchu v 20, 40 a 80 cm nad povrchem a mnoho dalších. Druhý senzor, ke kterému poskytla přístup Obec Střeň je umístěn v řece Morava pod mostem spojujícím obce Střeň a Lhota nad Moravou. Tento senzor slouží k získávání aktuální výšky hladiny řeky Moravy. Posledním zdrojem dat byla webová služba weatherbit.io, která obsahuje data, jako je rychlost větru, teplota, pocitová teplota, viditelnost, oblačnost atd. Platforma ThingsBoard poté sloužila jako prostředník mezi databází a klientem, odkud se přes entitní pohledy s jedinečným ID každého senzoru či webové služby získávala data. Pro správnou funkcionalitu byla používána dokumentace od ThingsBoard k REST API.



Obr. 4.1.1 Entitní pohledy v platformě ThingsBoard (zdroj: autor)

V softwaru Unity bylo nejprve nutné stáhnout klienta *NuGetForUnity*. *NuGet* je systém pro správu balíčků distribuovaných na serveru a využívaných uživateli. *NuGet* je zároveň plně kompatibilní s editorem Unity a poskytuje rozhraní pro instalaci dalších knihoven (Glitchenzo). Instalace *NuGetForUnity* probíhala pomocí *PackageManageru* a volby "*Add package from git URL*". Následně bylo nutné najít a nainstalovat knihovnu *Newtonsoft.Json* ve verzi 13.0.3. Knihovna *Newtonsoft.Json* slouží pro práci s formátem JSON v jazyce C#. Tato knihovna poskytuje sadu nástrojů pro serializaci (převod objektů na JSON) a deserializaci (převod JSON zpět na objekty) dat (Glitchenzo).



Obr. 4.1.2 Editor NugetForUnity (zdroj: autor)

Po instalaci knihovny *Newtonsoft.Json* byl vytvořen skript *JWT.cs* a přiřazen prázdnému objektu v okně *Hierarchy*. Pro získání dat z platformy ThingsBoard bylo nejprve nutné sestavit URL pro přihlášení a vytvořit JSON řetězec obsahující uživatelské jméno a heslo pro přihlášení:

```
private string loginUrl = "https://mospremasens.upol.cz:443/api/auth/login";
private string jsonRequestBody = "{\"username\": \"---\", \"password\": \"---\"}";
```

Po spuštění metody *Start()* se provedla metoda *coroutine StartCoroutine(PostRequest()),* která zajišťovala odeslání POST požadavku na server pro autentizaci a další nastavení hlavičky. Vytváří se zde objekt *UnityWebRequest* pro odeslání *POST* požadavku na specifikovanou URL. Dále byl JSON řetězec *jsonRequestBody* převeden na pole bitů, které bylo přeneseno v těle HTTP požadavku, a tělo požadavku bylo nastaveno na toto pole bitů obsahující JSON řetězec. Dále byl nastaven *handler* pro stahování odpovědi ze serveru. V tomto případě byl použit *DownloadHandlerBuffer* pro uchování odpovědi v paměti a byla nastavena hlavička požadavku pro specifikaci, že obsah požadavku je typu JSON a že klient akceptuje odpověď ve stejném formátu. Nakonec byl odeslán asynchronní požadavek na server a čekalo se, dokud nebyl dokončen.

```
private IEnumerator PostRequest()
{
    UnityWebRequest request = new UnityWebRequest(loginUrl, "POST");
    byte[] jsonBytes = new System.Text.UTF8Encoding().GetBytes(jsonRequestBody);
    request.uploadHandler = (UploadHandler)new UploadHandlerRaw(jsonBytes);
    request.downloadHandler = (DownloadHandler)new DownloadHandlerBuffer();
    request.SetRequestHeader("Content-Type", "application/json");
    request.SetRequestHeader("accept", "application/json");
    yield return request.SendWebRequest();
```

Další část kódu kontroluje, zda byl požadavek úspěšný (HTTP status kód 200). Pokud ano, zpracovává odpověď získanou od serveru. Následně deserializuje JSON odpověď do objektu *tokenResponse*, k čemuž byla použita knihovna *Newtonsoft.Json*. Po úspěšné autentizaci se získává přístupový a obnovovací token *JWT* z odpovědi serveru.

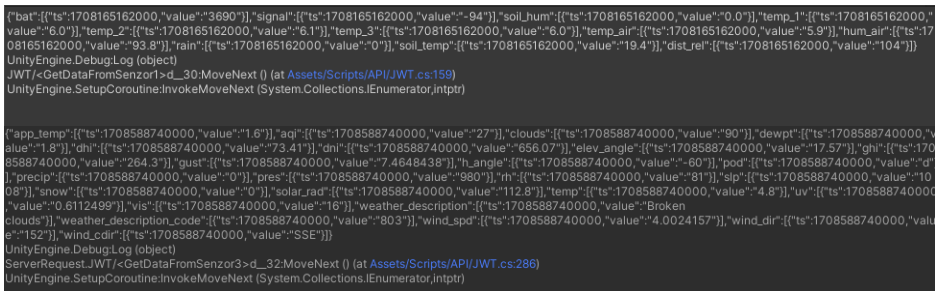
```
if (request.result == UnityWebRequest.Result.Success)
        {
            string jsonResponse = request.downloadHandler.text;
            tokenResponse = JsonConvert.DeserializeObject<TokenResponse>(jsonResponse);
            accessToken = tokenResponse.token;
            refreshToken = tokenResponse.refreshToken;
```

Podle hodnoty proměnné *checkJWT* se skript rozhoduje, zda se použije nový přístupový token nebo se použije obnovovací token pro další autentizaci a následné spuštění dalších coroutine metod. Platnost *JWT* tokenu z platformy ThingsBoard je asi 15 minut.

```
JWT_Token = checkJWT ? accessToken : refreshToken;
StartCoroutine(GetDataFromSenzor1());
StartCoroutine(GetDataFromSenzor2());
StartCoroutine(GetDataFromSenzor3());
```

Po spuštění metody *StartCoroutine(GetDataFromSenzor1())* se provede odeslání *GET* požadavku na danou *URL* adresu pro získání telemetrických dat ze senzoru číslo 1, 2 a 3 (webová služba).

```
private IEnumerator GetDataFromSenzor1()
{
    string url = "https://mospremasens.upol.cz:443/api/plugins/telemetry/ENTITY_VIEW/---
/values/timeseries";

    using (UnityWebRequest webRequest = UnityWebRequest.Get(url))
    {
        webRequest.SetRequestHeader("accept", "application/json");
        webRequest.SetRequestHeader("X-Authorization", "Bearer " + JWT_Token);
        yield return webRequest.SendWebRequest();
        // Zpracování úspěšné odpovědi
        string responseText = webRequest.downloadHandler.text;
        // Deserializace JSON odpovědi
        TelemetryResponse responseData = JsonUtility.FromJson<TelemetryResponse>(responseText);
        Debug.Log(responseText);
        // Převod hodnoty vlhkosti na float (nebo dalších dat…)
        dist_relValue = float.Parse(responseData.dist_rel[0].value.Replace('.', '.'),
        CultureInfo.InvariantCulture);
```



Obr. 4.1.3 Vypsání dat ze senzoru č.1 a č.3 v consoli po deserializaci (zdroj: autor)

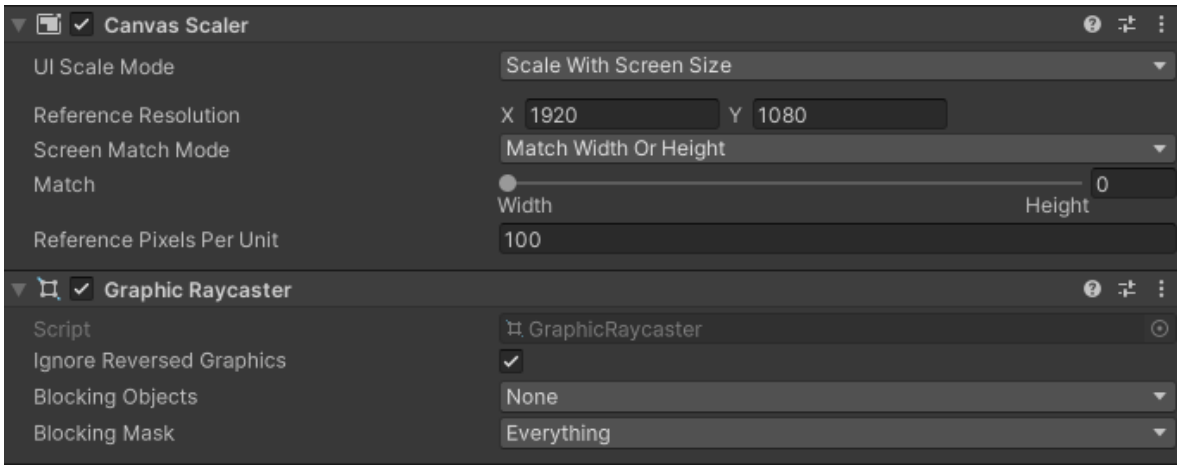
Totožný postup pro získání dat byl proveden u všech senzorů a jedné webové služby. Kontrola aktuálnosti dat se poté provádí každých 15 minut.

4.2 Tvorba uživatelského rozhraní

Pro celkový vzhled a dojem aplikace bylo potřeba navrhnout a implementovat uživatelské rozhraní (UI). Vzhledem k tomu, že aplikace cílí na platformu PC, je používán termín GUI, což znamená grafické uživatelské rozhraní. GUI se stalo nezbytným standardem ve vývoji různých aplikací, umožňuje přímou manipulaci s grafickými prvky, jako jsou tlačítka, posuvníky, okna, karty a nabídky (Holub 2022). Pro implementaci UI byl využíván výhradně software Unity ve verzi 2022.3.20.f1. Pro tvorbu grafických prvků byly použity programy Inkscape ve verzi 1.3.2 a Gimp ve verzi 2.10.36. Veškeré UI vycházelo, nebo bylo upravováno z assetů od D.F.Y STUDIO.

4.2.1 Hlavní menu UI/UX

Uživatel získá prvotní dojem o celkové aplikaci již při otevření defaultní scény či loading scény, která se objeví po spuštění aplikace. Scéna *Hlavní menu* obsahuje základní panely jako je: Průvodce, informace o aplikaci, statistiky dat, ukončení aplikace, nebo možnosti přístupu dál do další scény. Celá aplikace byla optimalizována pro FULL HD rozlišení, tedy 1920×1080 pixelů. Objekt reprezentující Hlavní menu se skládá z komponenty *Canvas Scaler,* jenž zajišťuje responzivní zobrazení a *Graphic Raycaster*. Dále obsahuje již umíněných pět objektů typu *Button* s komponentou *Horizontal Layer group*, který umožňuje pracovat s jednotlivými prvky, jako s celkem. Tedy vypočítává vzdálenosti v daném *Panelu* pro všechny komponenty dohromady, a ne pro každý zvlášť (Holub 2022). Vybrané objekty typu *Button* jsou *Interactable* a obsahují vytvořené efekty pro situace jako jsou: *Highlighted*, *Pressed, Selected* a *Disabled*.



Obr. 4.2.1.1 Nastavení komponent Canvas Scaler a Graphic Raycaster (zdroj: autor)

Pro přepínání mezi jednotlivými panely menu, byl naspán skript *ButtonController.cs.* Který vytváří propojení mezi objekty typu *Button* a *UI elemnety* a umožňuje přepínat mezi těmito prvky. Metoda *Start()* inicializuje tlačítka tím, že pro každé tlačítko přidává posluchače události kliknutí *AddListener*. Posluchači události jsou funkce *ToggleUI* s aktuálním indexem tlačítka. Metoda *ToggleUI()* je volána při kliknutí na tlačítko a přepíná mezi UI prvky. Pokud je kliknuté tlačítko stejné jako aktuálně aktivní UI prvek, UI prvek se deaktivuje. Pokud není kliknuté tlačítko stejné jako aktuálně aktivní UI prvek, deaktivuje se aktuálně aktivní prvek (pokud existuje) a aktivuje se vybraný UI prvek. Odkaz na aktuálně aktivní prvek se aktualizuje.

```
public class ButtonController : MonoBehaviour
{
    [SerializeField] private GameObject[] uiElements;
    [SerializeField] private Button[] buttons;
    private GameObject activeUI = null;

    private void Start()
    {
        for (int i = 0; i < buttons.Length; i++)
        {
            int buttonIndex = i;
            buttons[i].onClick.AddListener(() => ToggleUI(buttonIndex));
        }
    }

    private void ToggleUI(int buttonIndex)
    {
        if (activeUI != null && uiElements[buttonIndex] == activeUI)
        {
            activeUI.SetActive(false);
            activeUI = null;
        }
        else
        {
            if (activeUI != null)
            {
                activeUI.SetActive(false);
            }
            uiElements[buttonIndex].SetActive(true);
            activeUI = uiElements[buttonIndex];
        }
    }
```

Obr. 4.2.1.2 Scéna hlavní menu (zdroj: autor)

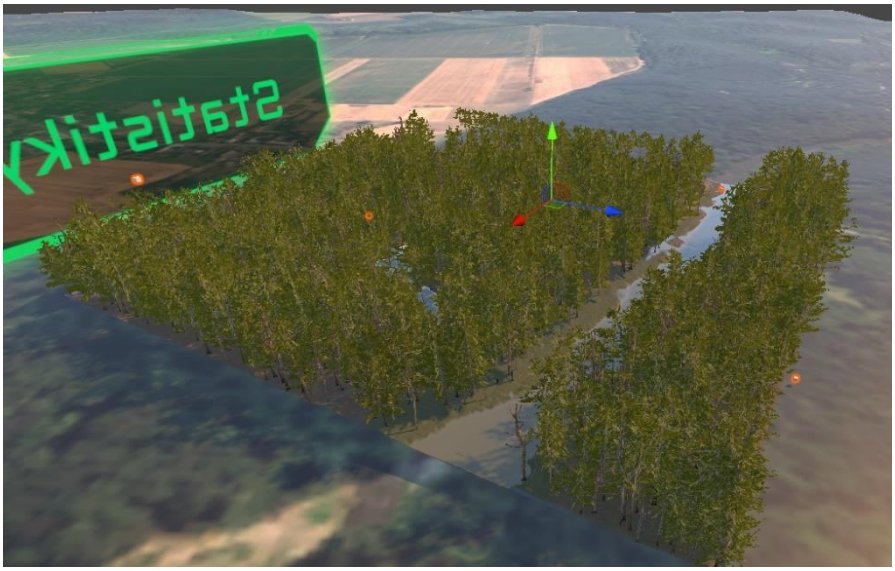
Jelikož scéna *Hlavní menu* slouží pouze jako rozcestník, bylo záměrem omezit uživateli zobrazená data. Zároveň bylo snahou, aby se uživatel rychle zorientoval a představil si, jak velké území je vizualizováno. Podkladová vrstva byla vybrána a získána z portálu ČUZK. Následně byla oříznuta a upravena v softwaru Gimp. Nejdříve bylo testováno využití podkladové vrstvy z již dostupných dat z projektu MOSPREMA. Nicméně z důvodu optimalizace a velikosti i přes převzorkování bylo rozhodnuto, že data nejsou přizpůsobená být využita v prostředí Unity. Taktéž pomocí pluginu BlenderGIS bylo v softwaru Blender zkoušeno úspěšně vyextrahovat a získat podkladová data dané oblasti v rozlišení srovnatelném od ČÚZK. Nicméně zde nastal problém s překryvem švů v okamžiku spojování jednotlivých dat, protože nebylo možné nastavit *bounding boxy* pro jednotlivé ořezy, jak tomu bylo u ČÚZK. DMK byl vytvořen v hlavní scéně, kterou popisuje kapitola 3.3 a do hlavního menu byl vložen jako *prefab*. Pro rozdělení vazeb navíc byly separovány data pro každý DMK zvlášť. Pro větší atraktivnost uživatel pozoruje vybrané území Litovelského Pomoraví z pozice „dronu“. Pro vytvoření tohoto efektu byl napsán skript *RotationCamera.cs*, který sleduje přiřazený objekt, kolem kterého se otáčí kamera. V Metodě *Update()* byla použita třída *transform* s metodou *RotateAround,* která otáčí přiřazenou kameru kolem objektu na ose Y, kdy rychlost udává proměnná *orbitSpeed*. Zároveň proměnná *desiredPosition* nastavuje pozici kamery na novou pozici, čímž udržuje konstantní vzdálenost od objektu *target*.

```
public class RotationCamera : MonoBehaviour
{
    public Transform target;
    public float orbitSpeed = 10.0f;
    public float distance = 5.0f;
    private float initialYPosition;
    void Start()
    {
        initialYPosition = transform.position.y;
    }
    void Update()
    {
        transform.RotateAround(target.position, Vector3.up, orbitSpeed * Time.deltaTime);
        Vector3 desiredPosition = target.position - transform.forward * distance;
        transform.position = desiredPosition;
        transform.LookAt(target);
    }
}
```

Aby bylo zajištěno správné otočení 2D prvků na terénu, bylo zapotřebí vytvořit čtyři objekty tzv. offset objekty bez *MeshRenderu*, které sloužily, jako výchozí body pro vybrané statistické ukazatele. Následovalo vytvoření skriptu *FlaotingText.cs* který umožňuje vizualizovat text ve scéně a jeho sledování za určeným cílem, zde předpokládaným *offset* objektem. Proměnné třídy obsahují informace o cíli *target*, ofsetu od cíle *offset* a uchovávají odkaz na komponentu *RectTransform* textového objektu pro manipulaci s jeho pozicí rectTransform. V metodě *Start* se při startu scény získává odkaz na komponentu *RectTransform* textového objektu a ukládá se do proměnné *rectTransform*. Metoda *LateUpdate* se vyvolává každý snímek po vykreslení prvků ve scéně. Kontroluje, zda je určený cíl nenulový a pokud ano, vypočítává jeho pozici s ohledem na ofset. Následně tuto pozici převádí na pozici ve screen space pomocí *Camera.main.WorldToScreenPoint()* a aktualizuje pozici *RectTransform* textového objektu na obrazovce tak, aby odpovídala

vypočítané screen space pozici. Celkově tedy tato třída umožňuje zobrazit textový objekt ve 3D prostoru s natočením na uživatele.

```
public class FloatingText : MonoBehaviour
{
    public Transform target;
    public Vector3 offset;
    private RectTransform rectTransform;

    private void Start()
    {
        rectTransform = GetComponent<RectTransform>();
    }

    private void LateUpdate()
    {
        if (target != null)
        {
            Vector3 worldPosition = target.position + offset;
            Vector3 screenPosition = Camera.main.WorldToScreenPoint(worldPosition);
            rectTransform.position = screenPosition;
        }
```



Obr. 4.2.1.3 Objekty offset (zdroj: autor)

Z již zmiňovaného skriptu *JWT.cs* se poté připojily vybrané UI prvky, které odkazovaly na vybrané hodnoty.

```
string text1 = "Vzduch\n" + temp_airValue + "\n°C\n";
TempAir_Text.text = text1;
string text2 = "Tůně\n" + dist_relValue + "\ncm\n";
PondsHigh_Text.text = text2;
string text3 = "Půda\n" + soil_tempValue + "\n°C\n";
TempSoil_Text.text = text3;
string text4 = "Řeka\n" + stavValue + "\ncm\n";
RiverHigh_Text.text = text4;
```

Obr. 4.2.1.4 Otočení UI na základě pohledu kamery s připojenými daty (zdroj: autor)

Zároveň vybrané UI prvky reprezentující výšku řeky a tůně, slouží jako prostředník pro přemístění do hlavní scény, k místům, kde jsou lokalizovány senzory.

```
public static void LoadSceneSenzorDefault()
{
    GlobalData.SenzorDefault = true;
    GlobalData.Senzor1 = false;
    GlobalData.Senzor2 = false;
    SceneManager.LoadScene("MainScene");
}
```

Dále byl vytvořen objekt *Console*, který je složen z jednotlivých UI prvků a dalších komponentů, jako je *Scrollbar*, *Hande*, *Button* a zobrazovací plocha. Objekt souží jako průvodce, zobrazuje postupně jednotlivé zprávy a akce, které se dají díky *Scrollbar* postupně zobrazovat a vracet se k nim nazpátek. Objekt *Console* obsahuje objekt typu *Button* pro mazání zpráv.

Obr. 4.2.1.5 Console a jeho hierarchie UI objektů (zdroj: autor)

Skript *Console.cs* slouží k logování a zobrazování zpráv v textovém panelu (Průvodci). Uchovává seznam zpráv, které mají být zobrazeny v Consoli. Přidává zprávu do seznamu a spustí efekt psacího stroje na textovém panelu pomocí dané metody. Zároveň obsahuje metodu pro smazání textu ze zobrazovací plochy, která je připojena přes komponentu *OnClick()* k danému objektu typu *Button*.

```
public class Console : MonoBehaviour
{
    public Text consoleText;
    public TypeWriteEffect_Scroll typeWriteEffect_Scroll;
    private List<string> messages = new List<string>();
    public void LogMessage(string message)
    {
        messages.Add(message);
        typeWriteEffect_Scroll.StartTypewriterEffect(message);
    }
    public void DeleteMessage()
    {
        consoleText.text = "";
    }
}
```

Zároveň veškeré texty, které se posílají do console obsahují efekt postupného načítání textu. Tedy *TypeWriteEffect_Scroll.cs* implementuje efekt psacího stroje, což je animovaný způsob zobrazování textu, kde se text postupně objevuje, jako by byl psán. Je zde využita fronta private *Queue<string> messageQueue = new Queue<string>(),* do které jsou ukládány zprávy, které mají být postupně zobrazeny. Fronta umožňuje systematické zpracování jednotlivých zpráv. Ve skriptu jsou použity proměnné, které slouží k řízení animace. *Timer* měří čas od posledního zobrazení písmene, *currentIndex* uchovává aktuální pozici v textu, a *isDisplayingMessage* indikuje, zda se právě zobrazuje zpráva. Klíčovou částí je coroutine *DisplayMessage(string message*), která postupně zobrazuje písmena z dané zprávy s časovým zpožděním mezi jednotlivými písmeny. Po dokončení zprávy se přidá nový řádek. Kód také obsahuje metodu *StartTypewriterEffect(string message),* která přidá novou zprávu do fronty, což spustí postupné zobrazování textu s efektem psacího stroje.

```
public class TypeWriteEffect_Scroll : MonoBehaviour
{
    private void Update()
    {
        if (messageQueue.Count > 0 && !isDisplayingMessage)
        {
            string currentMessage = messageQueue.Dequeue();
            StartCoroutine(DisplayMessage(currentMessage));
        }
    }

    private IEnumerator DisplayMessage(string message)
    {
        isDisplayingMessage = true;
        currentIndex = 0;

        while (currentIndex < message.Length)
        {
            timer += Time.deltaTime;

            if (timer >= letterDelay)
            {
                textObject.text += message[currentIndex];
                currentIndex++;
                timer = 0f;
            }

            yield return null;
```

```
        }

        textObject.text += "\n";
        isDisplayingMessage = false;
    }

    public void StartTypewriterEffect(string message)
    {
        messageQueue.Enqueue(message);
    }
}
```

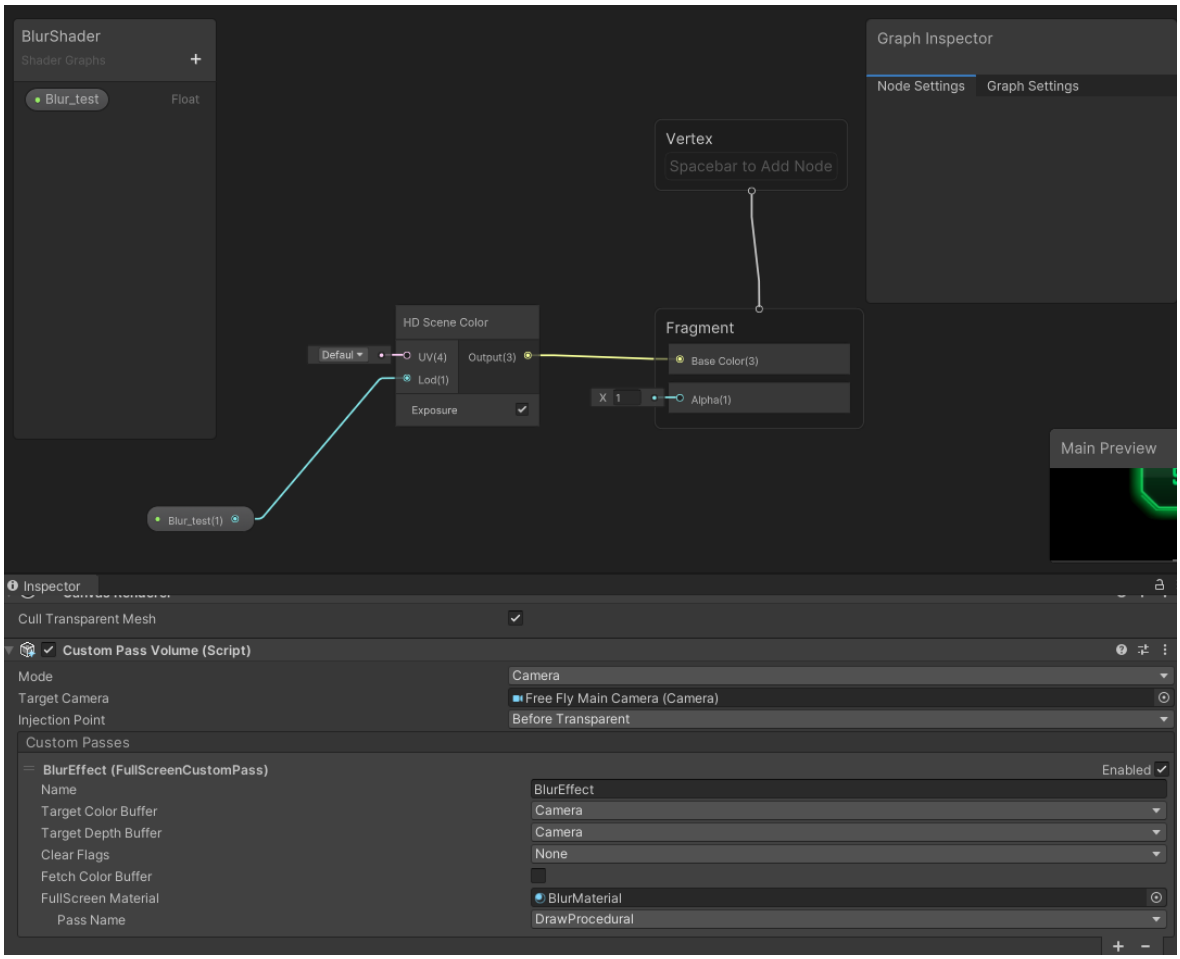
Dále byl vytvořen UI prvek pro zobrazení vybraných statistik. Aby se vybrané parametry zobrazovaly na UI prvku správně, byla napsána metoda *StatisticsUpdate(),* která přiřazuje výsledky do vybraného UI prvku. Pro zobrazení zaznamenaného data byla napsána metoda *ShowDatum(),* která se stará o zobrazení data a času. Nejprve kód dekóduje časovou značku *TimeStampMillis* z milisekundového formátu Unixového času na objekt typu *DateTime*. Poté tento objekt převádí na lokální čas, aby reflektoval aktuální časovou zónu. Následně formátuje tento lokální čas na řetězec ve specifickém formátu "dd.MM.yyyy HH:mm", kde dd představuje den, MM měsíc, yyyy rok, HH hodiny a mm minuty. Vytváří se řetězcová proměnná obsahující popisek "Datum zaznamenání:" a formátované datum a čas. Nakonec je tento text přiřazen do již zmiňovaného UI prvku, aby byl zobrazen uživateli.

```
private void StatisticsUpdate()
{
    string text = "Výška hladiny tůně: " + dist_relValue + " cm\n" +
     "Výška hladiny řeky: " + stavValue + " cm\n" +
     "Půdní vlhkost: " + soil_humValue + " %\n" +
     "Teplota půdy: " + soil_tempValue + " °C\n" +
     "Množství srážek: " + rain_tempValue + " mm\n" +


private void ShowDatum()
{
    DateTime dateTime = DateTimeOffset.FromUnixTimeMilliseconds(TimeStampMillis).DateTime;
    dateTime = dateTime.ToLocalTime();

    string formattedDateTime = dateTime.ToString("dd.MM.yyyy HH:mm");

    string text = "Datum zaznamenání: " + formattedDateTime + "\n";
    Datum.text = text;
}
```

Obr. 4.2.1.6 Zobrazení vybraných statistik a datumu zaznamenání (zdroj: autor)

4.2.2 Hlavní scéna UI/UX

V Hlavní scéně se využívá prefab z objektu ve scéně Hlavního menu, který je upraven a přizpůsoben potřebám Hlavní scény. Jedním z viditelných rozdílů je rozmazání obrazovky po aktivaci menu. Pro dosažení tohoto efektu byl vytvořen *BLUR shader* a materiál pomocí *FullScreen Shader Graph* a připojením komponenty *Custom Pass Volume* na volný objekt s nastavením viz obr. 4.2.1.1. Ve *FullScreen Shader Graph* se vytvořily node pro *HD Scene Color* s parametrem *Exposure* a dále byl připojen node *Slider*, který byl převeden na *float* pro lepší manipulaci.

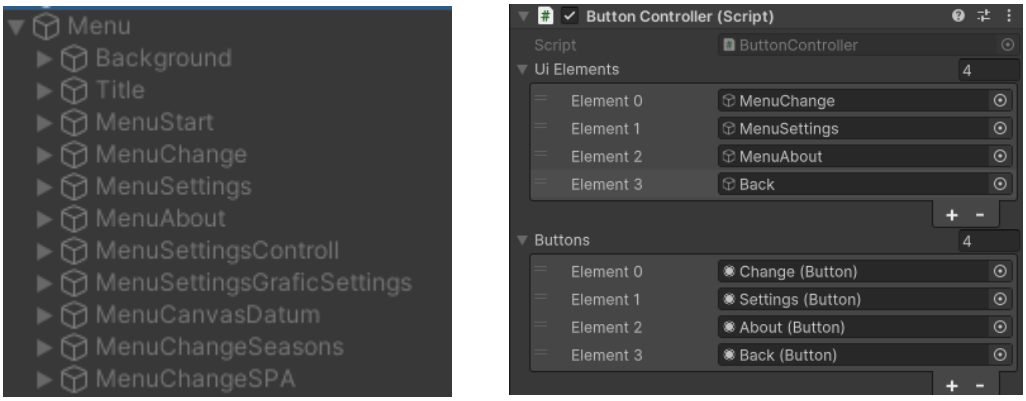


Obr. 4.2.2.1 Vytvoření a připojení shaderu (zdroj: autor)

UI prvek reprezentující statistiky je zde oddělený od objektu Menu, a to z toho důvodu, aby ho měl uživatel stále na očích, zatímco je objekt Menu deaktivovaný. Objekt Menu se aktivuje podle skriptu po stisknutí klávesnice Esc, zatímco objekt reprezentující UI prvky statistik se aktivuje a deaktivuje při stisknutí klávesnice Tab. Zda je objekt Menu aktivní nebo ne, rozhodují metody *Resume()* a *Pause()*. Při aktivaci Menu se aktivuje, zviditelní kurzor myši a naopak. Zároveň při deaktivaci se nastavuje a zamyká kurzor myši na střed obrazovky, při aktivaci se uvolňuje. Celé UI je navrhnuto a naprogramováno tak, aby se uživatel vrátil na poslední pozici. Tedy pomocí stisknutí klávesnice Esc se může z libovolného nastavení ihned vrátit zpět do prostředí a po jeho opětovném zmáčknutí se objeví v Menu na pozici, kde skončil.

```
public void Resume()
{
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
    Menu.SetActive(false);
    Time.timeScale = 1f;
    GameIsPaused = false;
    StartCoroutine(LockCursor());
}
void Pause()
{
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
    Menu.SetActive(true);
    Time.timeScale = 0f;
    GameIsPaused = true;
}
```

Hlavní menu bylo upraveno a rozšířeno o panely pro změnu prostředí, nastavení grafické náročnosti, nastavení ovládání a návrat do předchozí scény. Zůstaly panely o aplikaci a návrat do současné scény. Hlavní panely obsahují potomky, které slouží k aktivaci/deaktivaci obsahu. K tomu byl využit již napsaný skript *ButtonController.cs* pro přepínání mezi jednotlivými obsahy. Pro každý panel bylo vytvořeno rozhraní s objekty a byla využita komponenta *OnClick().* Metoda *OnClick()* funguje na principu toho, že se při kliknutí na UI prvek vykoná metoda připojeného skriptu. Důležité je, aby každý objekt typu *Button* správně odkazoval na část daného skriptu. Podmínkou bylo, aby tyto metody byly veřejné, aby byly viditelné v funkci *OnClick()* (Holub, 2022). Pro zajištění responzivity na různých zařízeních byly pro jednotlivé skupiny použity komponenty *Horizontal Layout Group* nebo *Vertical Layout Group*. Nejvyšší nadřazený prvek poté obsahuje komponentu Graphic Raycaster s nastaveným parametrem na *true - Ignore Reversed Graphics*. V obou scénách nechybí *EventSystem*, který slouží k vyvolání událostí na základě uživatelského vstupu. Aby *EventSystem* správně fungoval, musí být ve scéně právě jednou. *EventSystem* obsahuje komponenty *Touch Input Module* a *Standalone Input Module. Standalone Input Module* slouží k zajištění vstupu z klávesnice, myši a ovladačů (Starý, 2015).



Obr. 4.2.2.2 Hierarchie canvas a nastavení objektů v ButtonController.cs (zdroj: autor)

Nejvýraznější změnou v *prefabu* je vytvoření *UI* pro změnu prostředí, buď na základě historických dat, kdy si uživatel může pomocí kalendáře nastavit rok, měsíc, den a hodinu, kdy chce zobrazit data, nebo podle předvybraných dní / událostí reprezentujících typická roční období pro Českou republiku, nebo změna prostředí odrážející povodňové situace. Funkcionalita překreslování a dotazování historických dat je popsána v kapitole 3.4. Výběr pomocí data je umožněn díky objektu *dropdown*, který je rozdělen na čtyři subskupiny

reprezentující jednotlivé roky, dny, měsíce a hodiny. Výběr SPA a ročních období poté řídí komponenta *Horizontal Layout Group* s hodnotami true u *Child Force Expand*. Objekt tedy pracuje s jednotlivými objekty typu *button* jako s celkem. Vypočítává vzdálenosti v daném panelu pro všechny komponenty dohromady, nikoli pro každou zvlášť (Holub, 2022).

Obr. 4.2.2.3 UI změny scény pomocí data a SPA (zdroj: autor)

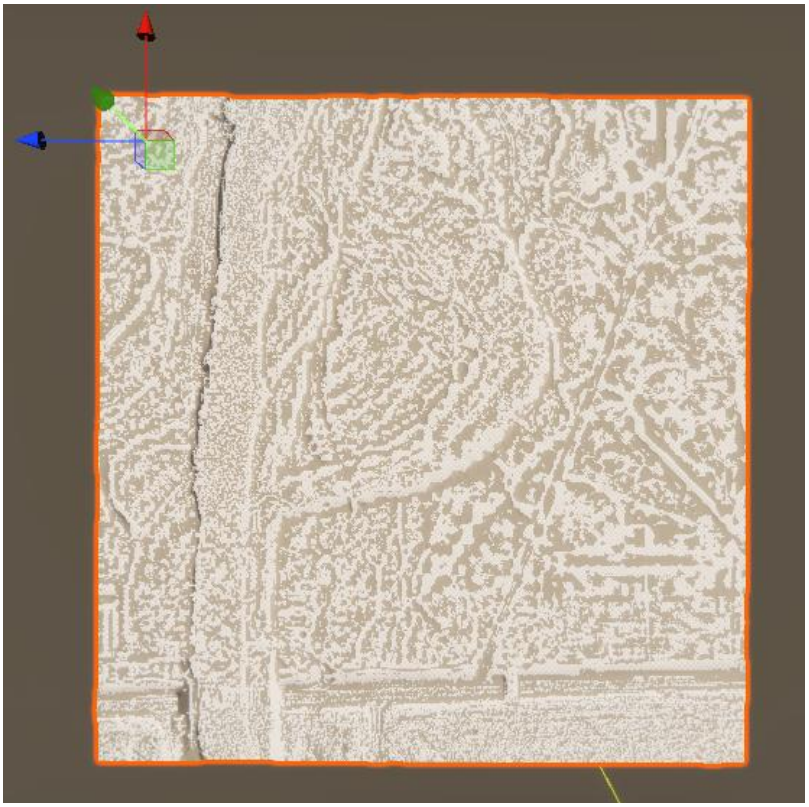
Dále bylo vytvořeno *UI* pro nastavení ovládání a grafické náročnosti. Podrobné nastavení a funkcionalita přepínání nastavení je popsána v kapitole 5.1. V těchto případech jsou aktivní současně jak potomci, tak rodičovský prvek. Pro nastavení ovládání byly vytvořeny čtyři objekty typu *slider*, které reprezentují základní rychlost pohybu, citlivost myši, zrychlení a zpomalení pohybu. Grafickou náročnost reprezentují tři objekty typu *button*, kde si uživatel může vybrat mezi nejnižší, střední a nejvyšší grafickou optimalizací.

Obr. 4.2.2.4 UI ovládání (zdroj: autor)

4.3 Vytváření prostředí

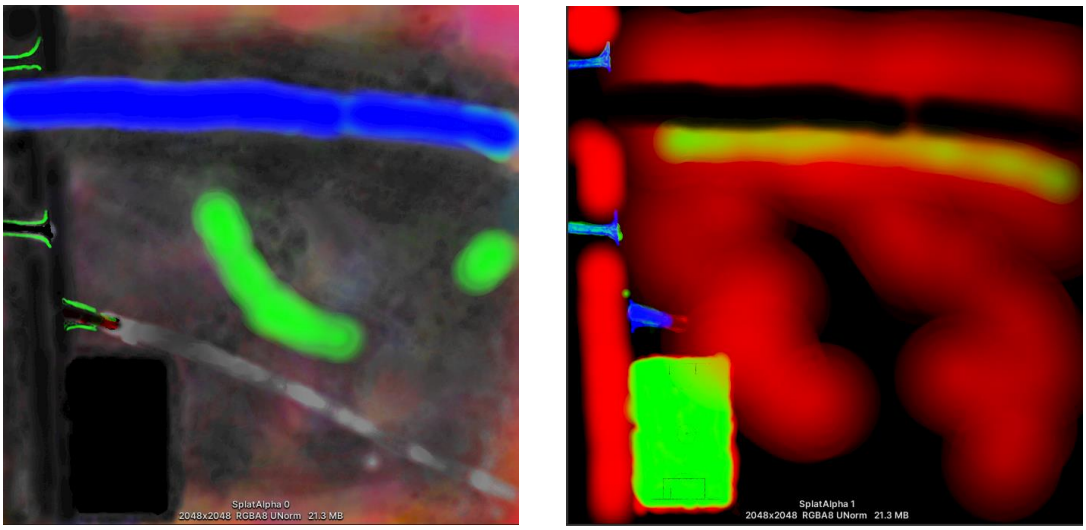
4.3.1 Architektura, modelování a design prostředí

Pro co nejrealističtější napodobení prostředí byla použita data z projektu MOSPREMA, konkrétně DMR, DMP a ortofoto. Prvním krokem bylo vytvořit 3D model DMR a importovat ho do prostředí Unity. K tomu byl použit software QGIS ve verzi 3.28, kde byla nahrána data DMR a bylo zde oříznuto požadované území. Pomocí pluginu DEMto3D byl exportován 3D model území ve formátu .stl. Jelikož Unity nedokáže číst formát .stl, byl použit software Blender ve verzi 3.4, kde byl 3D model území převeden do formátu .fbx a importován do prostředí Unity. Dalším krokem bylo změnit 3D model území na objekt *Terrain*. Objekt typu Terrain v Unity je objekt, který lze dále modelovat, na něj lze aplikovat *assety* stromů či jinou vegetaci a obsahuje větrné zóny (Unity, 2005). K tomu byl použit upravený skript od Shakeela, kde bylo zvoleno rozlišení *heightmap 4097x4097*, *Detail Resolution 1024 px, Control Texture Resolution 512 px a Resolution Per Patch 16 px*.



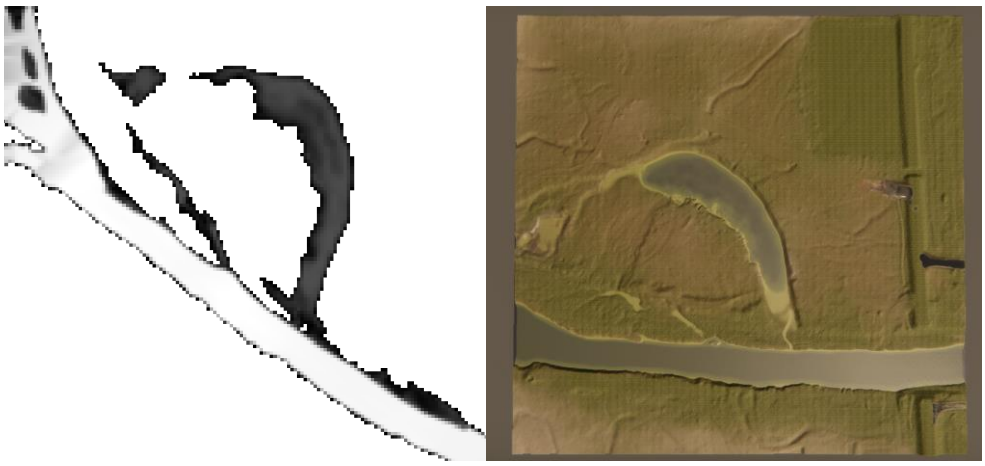
Obr. 4.3.1.1 Vytvoření objektu Terrain (bílý podklad) z dat (hnědý podklad) DMR (zdroj: autor)

Aby terén získal realistický vzhled, bylo potřeba vytvořit vrstvu z jednotlivých, předem vybraných textur, jako jsou písčité půdy, kamenitá zem, listnatý povrch, tráva, mechy a další. Celkem bylo použito devět textur, které byly následně přiřazeny pomocí nástroje *Terrain* na vytvořenou *splat map*. Byly vytvořeny dvě *splat mapy* s označením *Alpha1* a *Alpha2,* které reprezentují barevnou mapu s kódováním a prolínáním jednotlivých textur. Tyto splat mapy nejčastěji obsahují 4 kanály RGBA, přičemž každý kanál odpovídá jinému slotu textury (Nickelcitypixels, 2016). Tyto *splat mapy* sehrály důležitou roli při změně prostředí viz. kapitola 3.4.2.

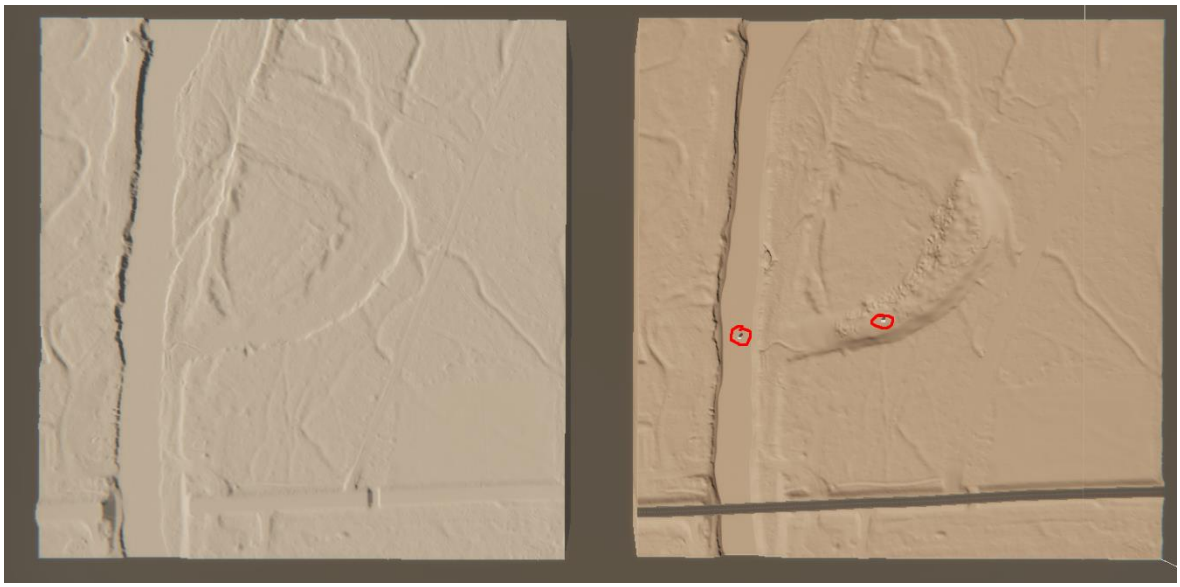


Obr. 4.3.1.2 SplatMaps1 a SplatMaps2 (zdroj: autor)

Jelikož při původním sběru dat pomocí LIDAR technologie byla zaznamenána zkreslená data v důsledku odrazů od vodních ploch, bylo nutné manuálně upravit terén v místech koryt řeky a v okolí tůní. Hloubka koryta řeky byla aproximována na základě dostupných povodňových vrstev vzniklých v rámci řešená projektu MOSPREMA s rozsahem od 150 cm do 435 cm v intervalech po 5 cm. Pro výpočet se rozdíl v poloze Y určil v deseti náhodných intervalech, průměroval se a získala se tak přibližná hodnota Y pro každý 5cm interval. K dosažení správného terénu byly využity nástroje dostupné v objektu Terrain jako je *Set Height*, *Thermal*, *Hydraulic*, *Wind* a Raise or Lower v Unity. Proces zahrnoval vytvoření objektu reprezentujícího řeku v Unity. Vrstvy rozlivu se přilnuly na vytvořený terén v Unity a sloužily k porovnávání jednotlivým stavům. Objekt byl následně zvedán v prostředí Unity tak, aby odpovídal zaplaveným oblastem vzniklým modelováním. Respektive aby zaplavená oblast odpovídala zaplavenému území v software Unity. Tento postup byl proveden minimálně dvacetkrát s náhodnými záplavovými oblastmi, což umožnilo získat průměrný posun v hodnotě Y pro každý 5cm interval. Na základě těchto úprav byla poté vypočtena přibližná hloubka koryta řeky.



Obr. 4.3.1.3 Záplavová oblast a vizualizace výšky záplavy v prostředí Unity (zdroj: autor)

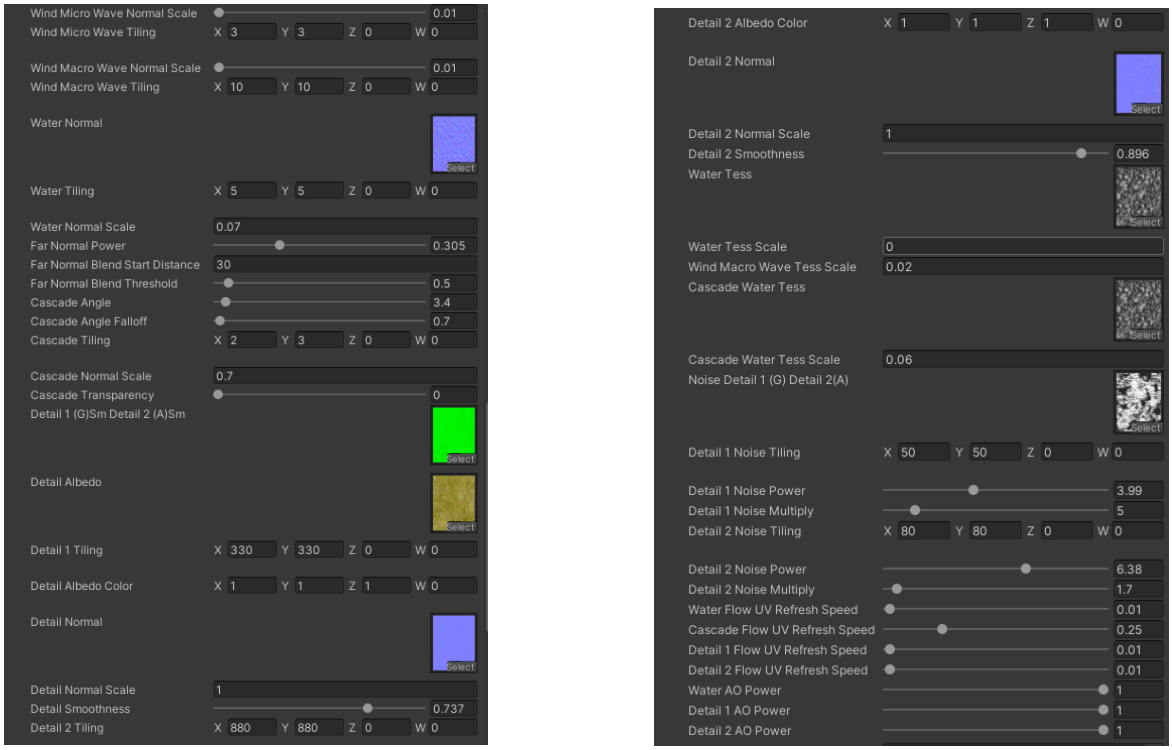


Obr. 4.3.1.4 Raw terén a prohloubený terén o výšku řeky a tůně (zdroj: autor)

Ve vizualizovaném terénu byly vytvořeny tři vodní plochy reprezentující řeku Moravu a stojaté tůně. Velikost objektu řeky Morava se rovnala velikosti modelovaného terénu. Shader a konečný materiál byl nejdříve převzat a upraven z assetů od NatureManufacture, a to konkrétně v jednotlivých částech: *Preserve specular lighting,* v pozicích X, Y u *MWS, WWMS, CMS, D1MS* a *D2MS.* Dále potom u *Edge Falloff Multiply* + *Power*, *Clean Multiply* + *Power, Shallow Color* + *Multiply* + *Power, Wave Translucency Power* + *Hardness* + *Multiply* + *Normal Scale* a *Wave Tiling v* pozicích X, Y. Upraveny ještě o detaily, a to u *Tiling, Power, Multiply, Tiling* a *UV speed.* Problém daného shaderu byla rozpínavost v jednotlivých částech objektu, nepravidelné prolínání textur, problém ve vypočítávání vektoru rychlosti a další manipulace s následnými simulacemi. Proto byl vytvořen nový objekt s novou podporou Water Surface od Unity-Technologies. Nejdříve bylo nutné v HDRP potvrdit render systému pro Water Surface, kdy resolution bylo defaultně nastaveno na 256, současně pro další manipulaci byl systém upraven i o Script Interaction. V HDRP Global Settings v sekci rendering potom bylo povoleno vykreslování shaderu pro Water Surface. Objekt vodní plochy se skládal ze dvou komponent a čtyř segmentů. Pro objekt reprezentující řeku byla sekce *Scattering* nastavena následovně *close* = 0.95, *distant* = 0.85, *fade* = 100 – 500, *max dist* = 0.73, abs *dist* = 1.5, *ambient* a *height term* = 0,35 + 0,2, *displacement* = 0.1, *light tip* = 0.6. V sekci *caustics* bylo nastaveno *simulation band* = 1. *virtual plane* = 4. Pro underwater effect byl připojen *box collider* s true hodnotou pro *trigger*, kdy *volume bounds* reprezentuje velikost *box collider* s tím, že l*ayer overrrides* obsahuje defaultní nastavení. Water System pro tůni se skládá ze dvou částí první část představuje upravený shader od NatureManufacture, kde navíc bylo potřeba změnit shader a nastavit proměnné materiálu v sekci *Exposed,* a to konkrétně u *Tiling*, *Normal Scale, Far Normal Power* + *Blend Start Distance* + *Treshold, Cascade Angle* + *Tiling* a *Albedo Color.* Současně byl použit nativní Water System, kdy sekce *Scattering* byla upravena následovně: *Close* = 0.83, *distant* = 0.616, *fade* = 100 – 500, *max dist* = 0.73, abs *dist* = 1.5, *ambient* a *height term* = 0,35 + 0,2, *displacement* = 0.1, *light tip* = 0.6. V sekci *caustics* bylo nastaveno *simulation band* = 1. *virtual plane* = 4. A stejně jako pro objekt řeku, i objekt reprezentující tůni obsahuje box collider pro detekci hráče, zda se nachází pod vodou – pro aktivaci underwater effect.

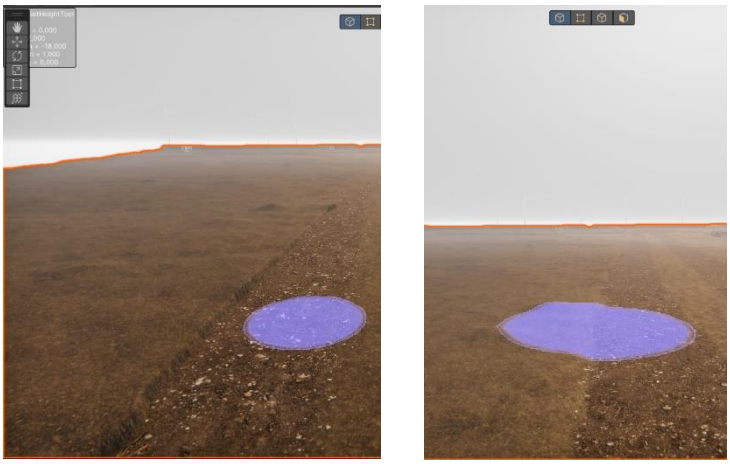
Obr. 4.3.1.3 WaterSystem řeky Morava (zdroj: autor)

Obr. 4.3.1.4 WaterSystem tůně (zdroj: autor)

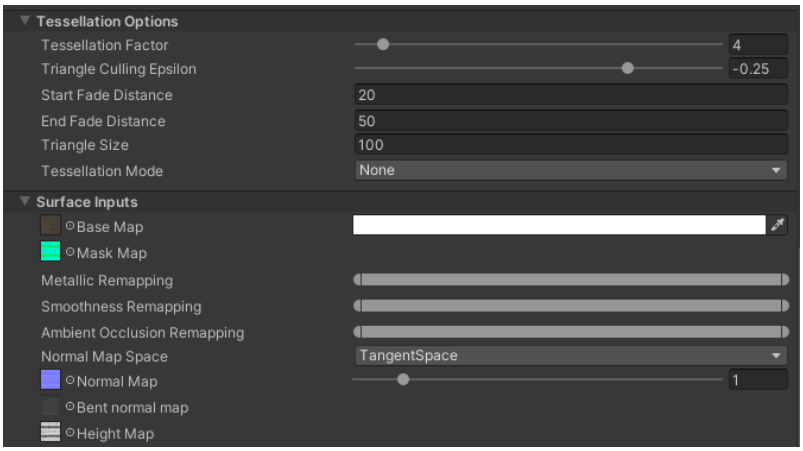


Obr. 4.3.1.5 Základní nastavení materiálu pro tůni (zdroj: autor)

Na vytvoření lesních cest byl použit nástroj z komponenty objektu Terrain - Smooth *Height* a *Set Height*. První nástroj nastavoval globální výšku terénu a druhý nástroj shlazoval terén na základě okolní výšky. Pro rozšíření možností a funkcí jednotlivých nástrojů byla nainstalována knihovna *Terrain Tools v 5.0.4*. Lesní cesta vinoucí se za hřištěm obce Střeň se skládá z 22 objektů *Plane* s nastavením vlastního materiálu viz. obr 4.3.1.7. Pro vytvoření silnice byl použit nástroj od EasyRoad, de byl na upraveném a shlazeném povrchu vytvořen objekt *ER Road*, do kterého byl nastaven směr silnice a její šířka. Vznikl modulární objekt se vztahem k terénu se *Stich Seams* hodnotou true, kdy vazba k terénu byla upravena tak, aby se terén nepřipojil k dané silnici. Na závěr byl vytvořen vlastní HDRP Lit shader pro povrch silnice skládající se z textur od Polyhaven a Ambientcg. V obou případech bylo provedeno spojení meshe do jednoho objektu z důvodu snížení počtu drawcalls.



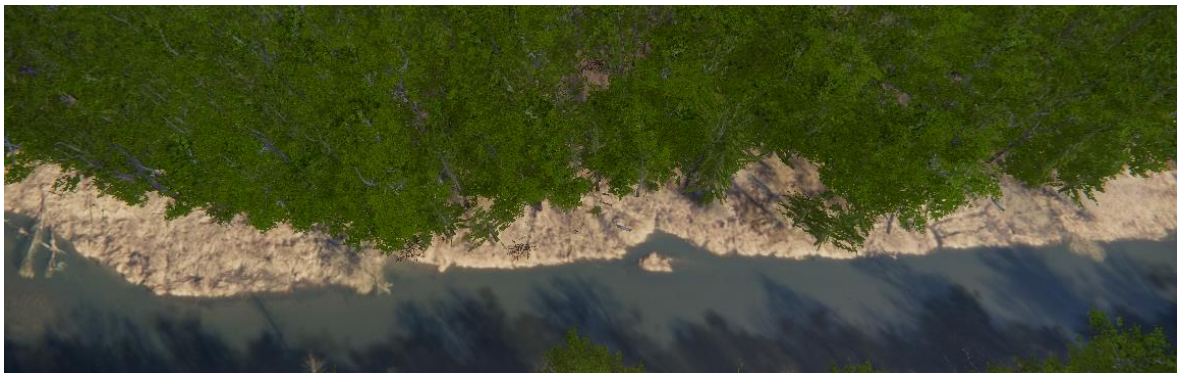
Obr. 4.3.1.6 Úprava terénu (zdroj: autor)



Obr. 4.3.1.7 Základní nastavení materiálu pro tůni (zdroj: autor)

Vysokou vegetaci ve vybraném území představuje devět různých prefabů z assetů od NatureManucafture. Ty byly přiřazeny s vytvořenými *LOD* do objektu *Terrain,* přičemž *NavMeshIndex* vždy bral poslední *LOD*. Rozmístění vysoké vegetace na DMR bylo prováděno dvěma způsoby. Nejdříve byla DMP získána maximální a průměrná výška vysoké vegetace. Poté byly do prostředí Unity importovány RGB tily s přesnější lokalizací jednotlivých druhů vysoké vegetace. Manuálně vybrané druhy vegetace byly umisťovány na přesné pozice, tak aby byla vytvořena reálná hranice lesa. Jakmile kolem cest, řeky a hřiště byly vytvořeny hranice vegetace, následoval druhý způsob přiřazování vysoké vegetace, a to pomocí

automatického přiřazení asi tří tisíc stromů. Pomocí funkce *MassPlaceTrees* byla nastavena jejich výška s offsetem získaným z DMP a zároveň byl uzamknut poměr výšky s šířkou. Parametr *ColorVariation* byl nastavena na hodnotu *0.4* a byla povolena náhodná rotace.



Obr. 4.3.1.8 Usazování stromů na originální ortofoto (zdroj: autor)

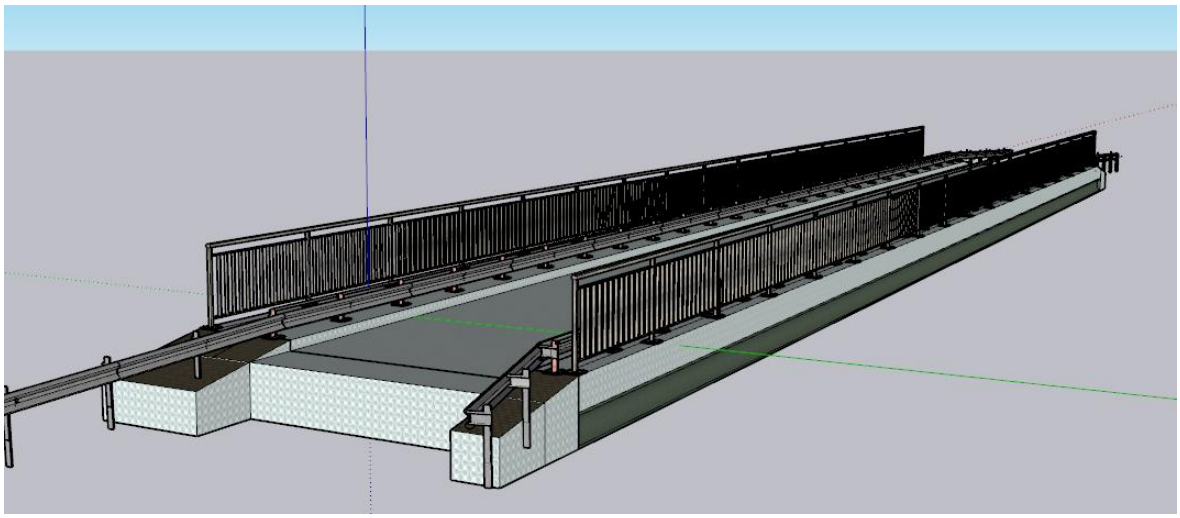
Obr. 4.3.1.9 Usazování stromů na originální ortofoto (zdroj: autor)

Nízká vegetace se skládá z kapradí, travin a dalších nízkých druhů vegetace a je doplněna o detaily, jako jsou pařezy, větve stromů a kameny. U tůní byly ručně upraveny svahy pomocí dostupných funkcí a nástrojů, které prostředí Unity nabízí.

Obr. 4.3.1.10 Výsledná vizualizace tůně (zdroj: autor)

4.3.2 Modelování urbanistických objektů

Jelikož ve vybraném vizualizovaném území se nacházejí urbanistické objekty, bylo nutné je ve 3D softwaru vymodelovat, vytvořit nebo upravit textury, zvolit vhodné shadery a vytvořit jim odpovídající materiály v prostředí Unity. 3D modelování probíhalo v softwaru SketchUp. Pro úpravu textur byl využit software CrazyBump. Veškeré použité textury pocházely od autorů Polyhaven a Ambientcg. Pro urbanistické modelování byla zvolena metoda Hard-Surface, která využívá pro tvorbu 3D modelů tzv. tvrdé hrany. Tedy hrany s přesnou a často uspořádanou geometrií, která se pouze minimálně ohýbá (Holub 2022).



Obr. 4.3.2.1 Model mostu v software Sketchup (zdroj: autor)

Jednotlivé 3D modely byly poté rozmístěny na určená místa v herním enginu Unity, kde jim byl následně přiřazen vytvořený materiál s jedinečnými texturami.

Obr. 4.3.2.2 Hotový model mostu 14.04.2024 v prostředí Unity (zdroj: autor)

4.4 Změna prostředí na základě telemetrických dat

4.4.1 Dotazování na historické data

Jak již bylo zmíněno v kapitole 3.2.1, bylo vytvořeno UI / UX pro dotazování historických dat. Pro správnou funkčnost architektury daného řešení bylo nejprve nutné přidat správné hodnoty k objektům typu *Dropdown*. Následovalo získání informací o časové zóně, kdy se využívá středoevropský standartní čas. Dále se vytvořený objekt *DateTime* převede na UTC pomocí metody *TimeZoneInfo.ConvertTimeToUtc().* Jelikož data jsou průměrovaná za jednotlivou hodinu a ukládána s časem aktualizace, která bývá kolem 30. minuty dané hodiny (může se to měnit), bylo potřeba vytvořit proměnné *startTS* a *endTs*, které se dotazují na daný interval mezi vybranými hodinami. Na závěr byly ošetřeny výjimky, jako neplatná časová zóna nebo neexistující datum.

```
private void Start()
{
    convertButton.onClick.AddListener(ConvertToTimestamp);

public void ConvertToTimestamp()
{
    int year = yearDropdown.value + 2023;
    int month = monthDropdown.value + 1;
    int day = dayDropdown.value + 1;
    int hour = hourDropdown.value;

    try
    {
        TimeZoneInfo timeZone = TimeZoneInfo.FindSystemTimeZoneById("Central Europe Standard
Time");

        DateTime selectedDateTime = new DateTime(year, month, day, hour, 0, 0,
DateTimeKind.Unspecified);
        DateTime utcDateTime = TimeZoneInfo.ConvertTimeToUtc(selectedDateTime, timeZone);
        startTs = new DateTimeOffset(utcDateTime).ToUnixTimeMilliseconds();
        endTs = startTs + 3600000; // 3600000 ms = 1 hodina
    }
    catch (TimeZoneNotFoundException e)
    {
        Debug.Log("Zadaná časová zóna neexistuje: " + e.Message);
    }
    catch (ArgumentOutOfRangeException e)
    {
        Debug.Log("Zadané datum neexistuje: " + e.Message);
    }
    StartCoroutine(PostRequest());
```

Obdobně jako ve scéně Hlavní menu, následovalo asynchronní spouštění HTTP požadavku pomocí vytvořené metody *PostRequest(),* kde se vytvořil *UnityWebRequest* s POST metodou a nastavením hlaviček. Samotný dotaz na platformu ThingsBoard se skládal z jednotlivých jmen dat s proměnnými *startTS* a *endTs,* které vymezovaly časový interval pro správné získání dat.

```
  string url = $"https://mospremasens.upol.cz:443/api/plugins/telemetry/ENTITY_VIEW/---
/values/timeseries?keys=dist_rel,soil_hum,soil_temp,rain,hum_air,temp_air,temp_1,temp_2,temp_3&star
tTs={startTs}&endTs={endTs}&orderBy=ASC";

 string url = $"https://mospremasens.upol.cz:443/api/plugins/telemetry/ENTITY_VIEW/d8fffca0-8b9f-
```

```
11ee-b069-b374078965de/values/timeseries?keys=stav&startTs={startTs}&endTs={endTs}&orderBy=ASC"
```

Aby nedošlo k nepředvídatelným a nesmyslným simulacím v závislosti na datech, uživateli je znemožněno nastavovat datum, které ještě nenastalo, nebo které se stalo dříve než 02.06.2023 ve 4:00 hod.

```
CatchError.SetActive(false);
if (selectedDateTime > currentDateTime)
{
    CatchError.SetActive(true);
    return;
```

Při dotazu na historická data je nejprve ověřeno, zda pro zadané datum existují dostupná data pro každý senzor a webovou službu. To se provádí porovnáním zadaného historického data s aktuálním datem; pokud je historické datum starší než aktuální datum, proces pokračuje. Pokud v zadaném historickém datu chybí nějaká data pro některý ze senzorů nebo webovou službu, nastává situace, kdy je scéna ponechána nezměněna (v určitých případech se může změnit pouze část). Toto chování je způsobeno nekompatibilitou jednotlivých simulací vůči sobě. Jinými slovy, pokud nejsou dostupná data pro všechny potřebné senzory a služby, není možné provést plnohodnotnou dynamickou změnu scény, a uživateli se vypisuje hlášení.

```
try
{
    if (webRequest.result == UnityWebRequest.Result.Success)
    {

catch (ArgumentOutOfRangeException)
{
    Services3Data = false;
```

Zároveň zde byly napsány čtyři metody ekvivalentní ročním obdobím, které se připojily na vybrané objekty typu *button*. Přičemž každé tlačítko ovládá určitý prvek herního prostředí v závislosti na ročním období. Metoda *ButtonSpring()* reprezentuje jarní tlačítko, které nastavuje teplotu viz. postprocessing na 20 stupňů, deaktivuje zimní a podzimní terén a aktivuje kreslení stromů a listí na hlavním terénu. *ButtonSummer()* přiřazuje léto a zvyšuje teplotu na 40 stupňů, opět deaktivuje zimní a podzimní terén a aktivuje kreslení stromů a listí. *ButtonAutumn()* odpovídá podzimnímu tlačítku, které snižuje teplotu na 0 stupňů, aktivuje podzimní terén a deaktivuje kreslení stromů a listí. *ButtonWinter()* reprezentuje zimní tlačítko, snižuje teplotu na -40 stupňů, aktivuje zimní terén a deaktivuje kreslení stromů a listí. Je nutné dodat, že tyhle teploty byly zvolena zpětně, až po vytvoření *Postprocessingu*. Každá metoda zároveň spouští asynchronní požadavek metodou *PostRequest()*, která dále rozvádí a provádí výběr dat a změnu prostředí podle dostupných dat. Na úplně stejném principu funguje přepínání jednotlivých SPA, které byly vybrány na základě statistik dostupných na webu envimonitoring.

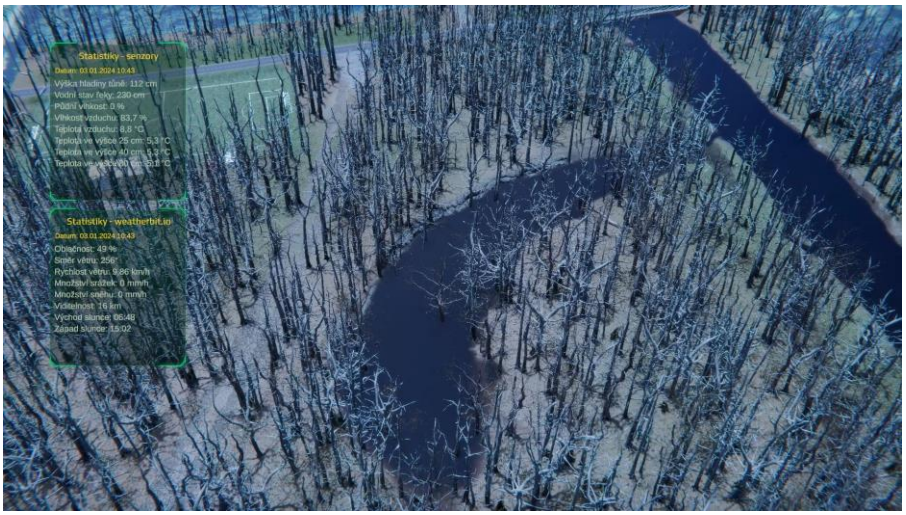
```
public void ButtonAutumn()
{
    WhiteBalance whiteBalance;
    if (volume.profile.TryGet(out whiteBalance))
    {
        whiteBalance.temperature.value = 0f;
    }
    startTs = 1698994800000;
    endTs = 1698998400000;
    WinterTerrain.SetActive(false);
    AutumnTerrain.SetActive(true);
    MainTerrain.drawTreesAndFoliage = false;
```

4.4.2 Změna prostředí

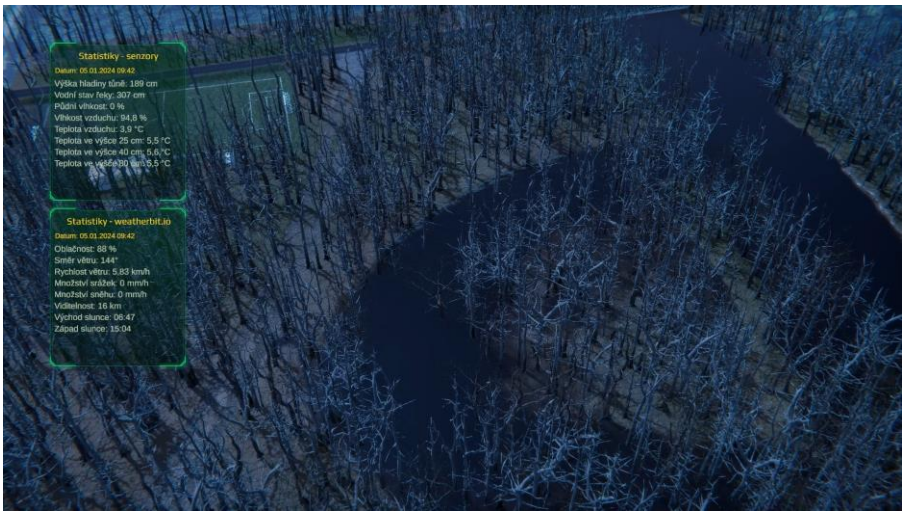
Funkcionalita pro získávání dat byla popsána v minulých kapitolách, nicméně podmínkou a celým jádrem, diplomové práce, bylo realizovat, naprogramovat a správně navrhnout architekturu pro změnu prostředí v závislosti na datech.

Pro změnu výšky řeky hladiny řeky Morava byl napsán skript *Enviroment.cs*, který slouží k ovládání výšky objektu v závislostech na hodnotách dat *stavValue* ze skriptu pro dotazování na historické data *TimeDatumRequest.cs* a z hlavního skriptu hlídající aktuální hodnoty proměnných a jejich hodinové aktualizace *JWT_Main.cs*. Jelikož původní data záplavových území jsou rozdělena do intervalu po 5 cm výšky řeky Morava od 150 cm do 435 cm, byla napsána funkce, která obdobně nastavuje výšku objektu River na základě intervalů, tak aby originální záplavové území odpovídalo záplavovému území na objektu *Terrain*. Metody *UpgradeHeightAfterCall()* a *StartHeight()* získávají aktuální hodnotu *stavValue* z přiřazených objektů. Dále metody zajišťují, že *stavValue* nebude klesat pod nulu pomocí funkce *Mathf.Max*. Následně vypočítávají kolikrát hodnota *stavValue* obsahuje celé intervaly pomocí *Mathf.FloorToInt* a určují kumulativní offset, který se přičte ke každému intervalu. Na závěr vytváří novou Y pozici objektu *River* na základě počáteční hodnoty, počtu intervalů, kroků a kumulativního offsetu. Pokud objekt *River* není null, aktualizuje jeho pozici tak, aby vytvářel plynulý pohyb při zachování stávajících X a Z pozic. Totožná funkcionalita byla napsána pro objekt tůni. Zároveň v momentě, kdy hladina výšky řeky překročila stanovený interval, byl objekt tůně a řeky nahrazen jedním objektem, který pokrýval a reprezentoval celé zaplavené území.

```
IEnumerator StartHeight()
{
    yield return new WaitForSeconds(0.5f);
    stavValue = JWT_data.stavValue;
    stavValue = Mathf.Max(stavValue, 0);
    int intervals = Mathf.FloorToInt(stavValue / interval);
    float cumulativeOffset = (stavValue - intervals * interval) * (step / interval);
    if (River != null)
    {
        Debug.Log(intervals);
        if (intervals >= 37) //Odpovídá 185 cm
        {
            Tune.SetActive(false);
            TuneUnderWater.SetActive(false);
            River1.SetActive(false);
            River.SetActive(true);
            float newY = startY + intervals * step + cumulativeOffset;
            River.transform.position = new Vector3(River.transform.position.x, newY,
River.transform.position.z);
        }
        else
        {
            float newY = startY + intervals * step + cumulativeOffset;
            River1.transform.position = new Vector3(River1.transform.position.x, newY,
River1.transform.position.z);
            dist_rel = JWT_data.dist_relValue;
            float t_newY = T_startY + dist_rel * T_step;
            Tune.transform.position = new Vector3(Tune.transform.position.x, t_newY + 0.02f,
Tune.transform.position.z);
            TuneUnderWater.transform.position = new Vector3(TuneUnderWater.transform.position.x,
t_newY, TuneUnderWater.transform.position.z);
```



Obr. 4.4.1.1 Zaplavené území 03.01.2024 (zdroj: autor)



Obr. 4.4.1.2 Zaplavené území 05.01.2024 (zdroj: autor)

Pro dynamickou a vizuální změnu oblačnosti na základě telemetrických dat byly ve stejném skriptu *Enviroment.cs* napsány metody pro změnu oblačnosti. Pro tento účel byl vytvořen objekt *SkyAndFogVolume* typu *Volume* s komponentou *Volumetric clouds*. Ve *Volumetric clouds* byly upravovány proměnné na základě telemetrických dat. A to konkrétně *Density Multiplier*, *Density Curve*, která řídí hustotu (osa Y) mraků jako funkci její výšky (osa X). A *Shape Factor*, který ovlivňuje tvar a velikost jednotlivých mraků. Vyšší hodnota vytváří menší pokrytí mraků s menšími mraky. Pro správnou funkčnost byla nejdříve napsána metoda *Map*, která provádí lineární mapování hodnoty z jednoho rozsahu na jiný. Je využita k převedení hodnoty v tomto případě *CloudsCoverage* z rozsahu [inMin, inMax] na rozsah [outMin, outMax]. Následovala podmínka *if (volume.profile.TryGet(out volumetricClouds)),* která ověřovala referenci na *Volumetric Clouds* z vytvořeného p*rofile* v objektu *SkyAndFogVolume.* Následně byly získány data pro oblačnost, déšť a sníh ze skriptu *JWT_Main.cs pro* historické dotazování potom *z TimeDatumRequest.cs.*

Poté na základě množství sněhu a deště byla nastavena proměnná *volumetricClouds.densityMultiplier.value.* Druhá podmínka kontroluje oblačnost a provádí lineární mapování hodnoty *CloudsCoverage* na *volumetricClouds.shapeFactor.value* v určeném rozsahu. Zároveň vypíná vytvořený *VFX* efekt a *participle system* demonstrující padající sníh a déšť.

```
public float Map(float value, float inMin, float inMax, float outMin, float outMax)
{
    return (value - inMin) * (outMax - outMin) / (inMax - inMin) + outMin;
}

if (volume.profile.TryGet(out volumetricClouds))
{
    //****CLOUDS COVERAGE***//
    CloudsCoverage = JWT_data.CloudsCoverage;
    Rain = JWT_data.Rain;
    Snow = JWT_data.Snow;
    if ((CloudsCoverage == 0) & (Rain == 0))
    {
        volumetricClouds.densityMultiplier.value = 0;
    }
    else if ((CloudsCoverage > 0) & ((Rain == 0) & (Snow == 0)))
    {
        Rain_Effect.SetActive(false);
        Snow_Effect.SetActive(false);
        volumetricClouds.densityMultiplier.value = 0.08f;
        // Lineární mapování pro shapeFactor přímo při vytváření ClampedFloatParameter
        float minCloudCoverage = 1;  // 1% cloud coverage
        float maxCloudCoverage = 100; // 100% cloud coverage
        float minShapeFactor = 1f;   // Pro 1% cloud coverage
        float maxShapeFactor = 0.5f;    // Pro 100% cloud coverage
        float new_shapeFactor;
        new_shapeFactor = Map(CloudsCoverage, minCloudCoverage, maxCloudCoverage, minShapeFactor,
maxShapeFactor);
        volumetricClouds.shapeFactor.value = new_shapeFactor;
    }
```

Obr. 4.4.1.3 Oblačnost 42% 11.04.2024 a Oblačnost 84% 14.01.2024 (zdroj: autor)

Pro rychlost oblačnosti byl použit odkaz na proměnnou *volumetricClouds.orientation*, ovlivňující pohyb směru oblačnosti. Nejprve metoda získává hodnotu směru větru *Wind_Direction* ze zdroje dat. Následně se provádí normalizaci této hodnoty do rozsahu 0 až 359 stupňů pomocí vzorce *(Wind_Direction - 60 + 360) % 360*, což zajišťuje, že hodnota bude vždy v tomto rozsahu. Jelikož v prostředí Unity je terén přesně otočený o 60° od severu, bylo potřeba normalizovat směr na nové hodnoty. Poté kód využívá normalizovanou hodnotu směru větru k nastavení orientace vizuálních efektů mraků. Parametr *volumetricClouds*.orientation se nastavuje pomocí instance třídy *WindOrientationParameter*, kde jsou specifikovány parametry, včetně normalizované

hodnoty směru větru, indikace vlastní hodnoty a režimu přepsání. Tímto způsobem je dosaženo realistické simulace pohybu mraků v herním prostředí, reagujícího na aktuální směr větru.

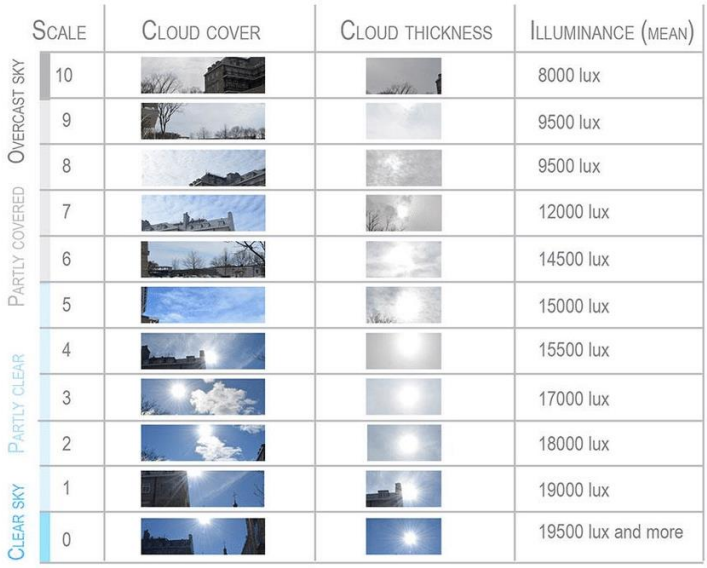
```
//****WIND DIRECTION***//
Wind_Direction = JWT_data.Wind_Direction;
Wind_Direction = (Wind_Direction - 60 + 360) % 360;
volumetricClouds.orientation.value = (WindParameter.WindParamaterValue)new
WindOrientationParameter(Wind_Direction, WindOverrideMode.Custom, true);
```

Rychlost oblačnosti vychází z klasifikace od ČHMÚ, kde kód nejprve definuje rozsah rychlosti větru *minWindSpeed a maxWindSpeed*, kdy hodnota *maxWindSpeed* nepřesahuje nejvyšší naměřenou hodnotu rychlosti větru v České republice a to *33 m/s*. Dále je zde definovaný rozsah hodnot pro vizuální efekty mraků *minVolumetricCloudsValue* a *maxVolumetricCloudsValue*. Poté je získávána hodnota rychlosti větru *Wind_Speed* ze zdroje dat. Následně se provádí lineární mapování této hodnoty do rozsahu vizuálních efektů mraků a přiřazuje ji k parametru *volumetricClouds.globalWindSpeed.value*. V tomto kontextu je opět využita metoda *Map*, která provádí lineární mapování hodnoty z jednoho rozsahu na druhý, a instance třídy *WindSpeedParameter* pro nastavení rychlosti větru s indikací vlastní hodnoty a režimu přepsání. Takto je dosaženo vizuální odezvy na aktuální rychlost větru v herním prostředí.

```
//****WIND SPEED * **//
float minWindSpeed = 0f;
float maxWindSpeed = 30f;
float minVolumetricCloudsValue = 1000f;
float maxVolumetricCloudsValue = 2000f;
Wind_Speed = JWT_data.Wind_Speed;
float new_windspeed;
new_windspeed = Map(Wind_Speed, minWindSpeed, maxWindSpeed, minVolumetricCloudsValue,
maxVolumetricCloudsValue);
volumetricClouds.globalWindSpeed.value = (WindParameter.WindParamaterValue)new
WindSpeedParameter(new_windspeed, WindOverrideMode.Custom, true);
```

Nastavení intenzity zdroje světla objektu *Light* typu *Directional* vychází ze dvou článků od Demers a Balaji zabývající se intenzitou slunečního světla během průběhu dne a podle typu oblačnosti. Kdy hodnota intenzity je ovlivněna množstvím oblačnosti a denní dobou. Nejprve byly definovány konstanty, které představují maximální a minimální hodnoty intenzity slunečního světla v jednotkách (Lux). Dále byla vytvořena proměnná *LuxCloudsCoverage*, do které je uložena normalizovaná hodnota *CloudsCoverage*. Následně dochází k inverzní lineární interpolaci, což je proces, při kterém se hodnota *LuxCloudsCoverage* interpoluje z rozsahu mezi *MinLux* a *MaxLux*. Tato hodnota je pak omezena na interval mezi *MinLux* a *MaxLux* pomocí *Mathf.Clamp* a nakonec přiřazena jako intenzita slunečního světla.

```
//****LUX***//
MaxLux = 45000;
MinLux = 6000;
float LuxCloudsCoverage = CloudsCoverage / 100f;
// Inverzní lineární interpolace
SunIntensity = Mathf.Lerp(MaxLux, MinLux, LuxCloudsCoverage);
SunIntensity = Mathf.Clamp(SunIntensity, MinLux, MaxLux);
DayNight.gameObject.GetComponent<DayNight>().SunParametreSettings();
```



Obr. 4.4.1.4 Intenzita slunečního světla v závislosti na typu oblačnosti (zdroj: https://www.researchgate.net/figure/Cloudiness-scale-according-to-weather-data-in-Quebec-City-March-21-st-to-March-29-th_fig5_313083757)

Následně tyhle hodnoty byly dále využívány ve skriptu *DayNight.*cs, který řídil polohu a nastavení objektu *Light*. Metoda *ChangeSunSettings* má za úkol dynamicky měnit pozici objektu slunce a měsíce v závislosti na aktuálním čase ve hře. Metoda začíná tím, že získává aktuální čas ze skriptu *JWT_Main.cs,* který obsahuje informace o čase v podobě počtu milisekund od epochy (*UNIX timestamp*), východ slunce (*sunriseDateTime*) a západ slunce (*sunsetDateTime*). Na základě získaných časových informací se vypočítávají úhly slunce a měsíce na obloze. Pokud je aktuální čas mezi východem a západem slunce (*RequestTime* < *Sunset a RequestTime* > *Sunrise*), pak se nastaví denní režim. Slunce je aktivní, zatímco měsíc je vypnutý. V opačném případě (aktuální čas je mimo rozsah východu a západu slunce), je nastaven noční režim, kdy objekt měsíc je aktivní a slunce je vypnuté. Pro plynulý pohyb slunce a měsíce se používá lineární interpolace mezi dvěma body (185° a -5°), což odráží úplný pohyb slunce a měsíce na obloze od východu po západ. Tato interpolace se provádí na základě aktuálního času a rozmezí východu a západu slunce. Nakonec se úhly interpolovaného slunce nebo měsíce používají k nastavení rotace odpovídajících herních objektů (*Sun* nebo *Moon*) ve scéně.

```
public void ChangeSunSettings()
{
    // Získání aktuálního času na začátku hry
    TimeStampMillis = JWT_data.TimeStampMillis;
    sunriseTime = JWT_data.sunriseDateTime;
    sunsetTime = JWT_data.sunsetDateTime;
    float Sunrise = (float)sunriseTime.TimeOfDay.TotalSeconds;
    float Sunset = (float)sunsetTime.TimeOfDay.TotalSeconds;
    DateTime dateTime = DateTimeOffset.FromUnixTimeMilliseconds(TimeStampMillis).DateTime;
    float RequestTime = (float)dateTime.TimeOfDay.TotalSeconds;
    if ((RequestTime < Sunset) && (RequestTime > Sunrise))
    {
        night = false;
        Moon.enabled = false;
        Sun.enabled = true;
        float interpolatedValue = Mathf.Lerp(185f, -5f, (RequestTime - Sunrise) / (Sunset -
Sunrise));
        // Nastavení hodnoty interpolovaného úhlu slunce
```

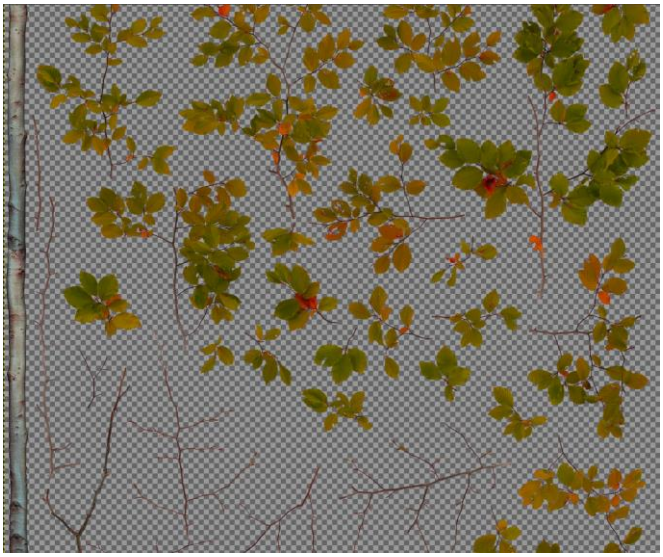
```
        sunAngle = Mathf.Clamp(interpolatedValue, -5f, 185f);
        Sun.transform.rotation = Quaternion.Euler(sunAngle, 234f, 90f);
    }
    else
    {
        night = true;
        Moon.enabled = true;
        Sun.enabled = false;
        float interpolatedValueMoon = Mathf.Lerp(185f, -5f, (RequestTime - Sunset) / (Sunrise -
Sunset));
        moonAngle = Mathf.Clamp(interpolatedValueMoon, -5f, 185f);
        Moon.transform.rotation = Quaternion.Euler(moonAngle, 234f, 90f);
    }
}
```

Pro podrobnější nastavení objektů typu *Light* byla napsána metoda *SunParametreSettings(),* která slouží k nastavení parametrů slunce a měsíce v herní scéně na základě různých faktorů, jako je pokrytí oblohy mraky, intenzita slunce, déšť, sníh nebo ročního období. Metoda nejprve získává hodnoty různých parametrů prostředí, jako je pokrytí mraky *CloudsCoverage*, intenzita slunce *SunIntensity*, déšť *Rain* a sníh *Snow* z objektu *Env*. Pokud je proměnná *night* nastavena na false (což znamená, že je den), metoda pokračuje v nastavování parametrů slunce v závislosti na ročním období (*summerspring, autumn, winter*) a dalších faktorech. Pro každé roční období se nejprve nastaví intenzita slunce podle hodnoty *SunIntensity*. Poté se určuje barva slunce (tzv. barva teploty) na základě aktuálního úhlu slunce (*sunAngle*) a dalších faktorů, jako je pokrytí oblohy mraky, déšť nebo sníh. Tento proces se provádí pomocí lineární interpolace mezi dvěma teplotami (*MaxTempSun* a *MinTempSun*), které se určují podle atmosférických podmínek. Výsledná barva teploty slunce se poté používá k nastavení Sun.*colorTemperature*. Pokud je proměnná *night* nastavena na true (což znamená, že je noc), metoda se pokouší nastavit parametry měsíce. Nejprve se nastavuje intenzita měsíce podle aktuálního úhlu měsíce (moonAngle). Poté se barva teploty měsíce nastavuje na pevnou hodnotu (12647), což odpovídá barvě měsíce v noci.

```
public void SunParametreSettings()
{
    CloudsCoverage = Env.CloudsCoverage;
    SunIntensity = Env.SunIntensity;
    Rain = Env.Rain;
    Snow = Env.Snow;

    if (night == false)
    {
        if ((TimeDatumRequest_data.summerspring == true) || (JWT_data.summerspring == true))
        {
            Sun.intensity = SunIntensity;
            //****ColorTemperature***//
            if ((CloudsCoverage > 70) || (Rain > 0) || (Snow > 0))
            {
                MaxTempSun = 10000;
                MinTempSun = 9000;
            }
            ColorTemperature = Mathf.Lerp(MaxTempSun, MinTempSun, (sunAngle - -5f) / (185f - -5f));
            Sun.colorTemperature = ColorTemperature;
```

Jednou z nejtěžších částí bylo provedení změn olistění stromů podle ročního období na terénu. Jelikož terén obsahuje devět různých prefabů listnatých stromů, musel být

vymyšlen způsob, jak efektivně nahrazovat instance prefabů původních stromů, tak aby byla zachována pozice umístění, šířka, výška, natočení, barevná variace a hlavně jejich typ. Nejdříve byly vytvořeny dvě odlišné varianty každého typu, reprezentující podzimní a zimní období. Pro podzimní typ byla upravena v software Gimp verze 2.10.30 původní textura za barvy - odpovídající barvám stromům na podzim. Tato textura poté byla přiřazena do nově vytvořeného materiálu typu *Opaque* se shaderem *HDRP/Foliage/Foliage*. A následně tento materiál byl upraven a nahrazen u všech devíti nových prefabů, a současně byla upravena i *LOD group*, tak aby *culled = 0 %.* Stejný postup probíhal u prefabů reprezentující zimní typ, s tím rozdílem, že byly odstraněny materiály reprezentující listí a zůstal pouze materiál reprezentující kůru kmenu a větví.



Obr. 4.4.1.5 Změna textur podle ročního období v software Gimp (zdroj: autor)

Pro nahrazování stromů v závislosti na ročním období byl napsán a navržen skript *ReplaceTree*, kde přiřazený seznam objektů reprezentoval nové stromy, které měly být umístěny na terénu. Funkce *void ChangeTrees(GameObject[] newTrees*) nejprve získá odkaz na aktivní terén a pole stromů na tomto terénu. Poté se vytvoří nový seznam stromů, které budou nahrazeny, na základě původního pole stromů. Následuje cyklus *for*, který prochází všechny stromy v seznamu. Pro každý strom se získá jeho index a pokud existuje odpovídající nový strom pro tento index, vytvoří se instance tohoto nového stromu na stejné pozici a s totožnou rotací jako původní strom. Původní strom je pak nahrazen novým stromem v seznamu. Po dokončení cyklu *for*, pokud byly provedeny nějaké změny, funkce *SetTreeInstances* aktualizuje stromy na terénu na základě seznamu. Seznam je poté vyčištěn a aktualizován. Tímto způsobem tento skript umožňuje nahradit stromy na terénu v Unity novými stromy, které jsou definovány jako objekty v poli předaném jako argument funkce *ChangeTrees.*

```
void ChangeTrees(GameObject[] newTrees)
    void ChangeTrees(GameObject[] newTrees)
    {
        Terrain terrain = Terrain.activeTerrain;
        TreeInstance[] treeInstances = terrain.terrainData.treeInstances;
        List<TreeInstance> changedTrees = new List<TreeInstance>(treeInstances);
        for (int i = 0; i < changedTrees.Count; i++)
        {
```

```
            TreeInstance treeInstance = changedTrees[i];
            int treeType = treeInstance.prototypeIndex;
            if (treeType >= 0 && treeType < newTrees.Length)
            {
                GameObject replacementTree = newTrees[treeType];
                Vector3 position = Vector3.Scale(treeInstance.position, terrain.terrainData.size);
                Quaternion rotation = Quaternion.AngleAxis(treeInstance.rotation * 360f,
Vector3.up);

                GameObject newTree = Instantiate(replacementTree, position, rotation);
                newTree.transform.parent = terrain.transform;
                changedTrees[i] = new TreeInstance();
            }
        }

        if (changedTrees.Count > 0)
        {
            terrain.terrainData.SetTreeInstances(changedTrees.ToArray(), false);
            changedTrees.Clear();
            changed = false;
        }
    }
```

Problém tohoto řešení byl, že nově vytvořený seznam vymazal původní stromy a po ukončení *PlayMode*, kde byly nahrazeny jedny stromy za druhé, došlo k úplné ztrátě dat. Proto byl dopsán kód bloku, který vytvářel a ukládal původní kopii stromů na terénu a po ukončení režimu obnovil původní stromy.

```
void Start()
{
    Terrain terrain = Terrain.activeTerrain;
    originalTrees = terrain.terrainData.treeInstances;
}
private void OnApplicationQuit()
{
    Terrain terrain = Terrain.activeTerrain;
    terrain.terrainData.treeInstances = originalTrees;
}
```

Nicméně i když změna stromů fungovala správně, mechanika a optimalizace byla nastavena špatně. Jelikož původní stromy byly vytvořeny přes nástroj *Terrain Tools* viz kapitola 3.3.1. Byly již automaticky optimalizovány pro vykreslování. Unity toho dosahuje tím, že stromy seskupuje do dávek a vykresluje je jako jeden objekt, což zlepšuje výkon. Avšak nově nahrazené stromy byly reprezentovány, jako nové instance prefabů, což znamená, že tyto stromy nebudou automaticky optimalizovány stejným způsobem. Každý strom se stává jednotlivým objektem, což mělo negativní vliv na výkon, i když byly použity *LOD group* v každém nově vytvořeném objektu ve čtyřech stupních po dvaceti procentech až po možnost *culled*, nejednalo se o uspokojující optimalizaci na vykreslování tak velkého množství dat. Aby se zachoval stabilní výkon, byl vymyšlen odlišný postup při změně stromů. Objekt *TerrainMain.asset* se duplikoval a rozdělil na tři části, což umožnilo zachování původních pozic a prefabů stromů a zároveň se vyřešil problém s klonováním terénu, kdy jedna změna provedena na původním terénu se aplikovala na zbylé terény. Potom přes nástroj *Trees* byly nahrazeny stávající stromy v daném terénu v takovém pořadí, v jakém byly přidány v původním terénu. Ve skriptu *TimeDatumRequest.cs* a *JWT_Main.cs*, byl dopsán blok kódu, který slouží k ovládání zobrazení správného typu

stromu na aktuálním nebo vybraném datu. Nejdříve byla vytvořena proměnná *DayMonthCheck*, která kombinuje měsíc a den do jednoho čísla pro jednodušší porovnání. Například, pokud by měsíc byl 4 a den by byl 15, pak by *DayMonthCheck* bylo 415. Následovaly tři podmínky, které aktivovaly jednotlivé vykreslení typů stromů podle datumu a to. Stromy se zelnými texturami listů jsou aktivní od 21. března do 21. září, stromy reprezentující podzimní vzhled jsou aktivní od 22. září do 2. prosince a zimní stromy bez listí jsou aktivní od 3. prosince do 20. března.

```
int DayMonthCheck = month * 100 + day;
if (DayMonthCheck >= 322 && DayMonthCheck <= 921) // 21.3 - 21.9
{
    WinterTerrain.SetActive(false);
    AutumnTerrain.SetActive(false);
    MainTerrain.drawTreesAndFoliage = true;
}
else if (DayMonthCheck >= 922 && DayMonthCheck <= 1202) // 22.9 - 2.12
{
    WinterTerrain.SetActive(false);
    AutumnTerrain.SetActive(true);
    MainTerrain.drawTreesAndFoliage = false;
}
else // 3.12 - 20.3
{
    WinterTerrain.SetActive(true);
    AutumnTerrain.SetActive(false);
    MainTerrain.drawTreesAndFoliage = false;
```

Orientace a deformace vegetace je ovládána pomocí vytvořeného objektu s atributem Wind Zone. Nejprve se získává rychlost větru *Wind_Speed* z *TimeDatumRequest.cs* nebo z hodinové aktualizace *JWT_Main.cs* a přepočítává se na metry za hodinu *Wind_Speed * 3.6f*. Tato rychlost se používá pro deformaci shaderu u meshe listí a větví dané vegetace. Získaná rychlost větru se následně přiřazuje komponentě *NM_Wind* připojené k hernímu objektu *WindTreeObject*. Tato komponenta řídí simulaci větru pro stromy v herní scéně a byla použita a upravena od NatureManufacture. Podobně jako rychlost větru, směr větru Wind_*Direction* se získává a přepočítává tak, aby se nacházel v rozmezí 0 až 360°. Následně se tento směr aplikuje na rotaci herního objektu *WindTreeObject*, což ovlivňuje vizuální efekty pohybu stromů v herní scéně podle směru větru. Zároveň rychlost směru ovlivňuje simulaci surface vodní plochy. Tedy se zvýšenou rychlostí větru vznikají na hladině větší vlny a naopak.

```
//Deformace vegetace
WS_TREE = TimeDatumRequest_data.Wind_Speed;
Wind_Speed_TREE = (float)WS_TREE * 3.6f;
WindTreeObject.gameObject.GetComponent<NM_Wind>().WindSpeed = Wind_Speed_TREE;
Wind_Direction_TREE = TimeDatumRequest_data.Wind_Direction;
Wind_Direction_TREE = (Wind_Direction_TREE - 120 + 360) % 360;
WindTreeObject.transform.Rotate(0, Wind_Direction_TREE, 0);

//Deformace surface vodní plochy
Wind_Speed = TimeDatumRequest_data.Wind_Speed;
river.ripplesWindSpeed = TimeDatumRequest_data.Wind_Speed;
FloodRiver.ripplesWindSpeed = TimeDatumRequest_data.Wind_Speed;
```

5 VLASTNÍ ŘEŠENÍ II

Uživatel v první scéně má zamknutý pohyb a může pouze interagovat s modelem, nicméně v hlavní scéně se uživatel pohybuje volně v předem určeném bounding boxu. Pro volný pohyb byl napsaný skript, který byl připojený na objekt *MainCamera* s *Rigidbody*. Nejprve se získává odkaz na komponentu *Rigidbody* spojenou s entitou kamery. Tam se načítají uložené hodnoty rychlosti pohybu kamery a citlivosti myši z paměti (pomocí *PlayerPrefs*) nebo se používají výchozí hodnoty, pokud nejsou k dispozici. Následuje *Update()* metoda, kde dochází k rotaci kamery na základě polohy kursoru myši. Objekt *MainCamera* sleduje pohyb myši a akumuluje změny rotace rotationX a rotationY na základě pohybu myši a citlivosti. Dále zde omezuje vertikální rotaci mezi -90 a 90 stupni pomocí funkce *Mathf.Clamp().* Metoda dále aplikuje vypočtenou rotaci kamery pomocí metody *Quaternion.AngleAxis()* na lokální transformaci kamery. V závislosti podle stisknuté hodnoty klávesnice následují tři situace:

1. Pokud je stisknuta klávesa Shift:

   Nastavuje rychlost pohybu kamery na vyšší hodnotu (*DefaultSpeed x UpMotor*) pro rychlejší pohyb vpřed a do stran. Aplikuje rychlostní vektor k pohybu kamery pomocí komponenty Rigidbody, která zajišťuje plynulý pohyb s fyzikální simulací.

2. Pokud je stisknuta klávesa Control:

   Nastavuje rychlost pohybu kamery na nižší hodnotu (*DefaultSpeed x DownMotor*) pro pomalejší pohyb vpřed a do stran.

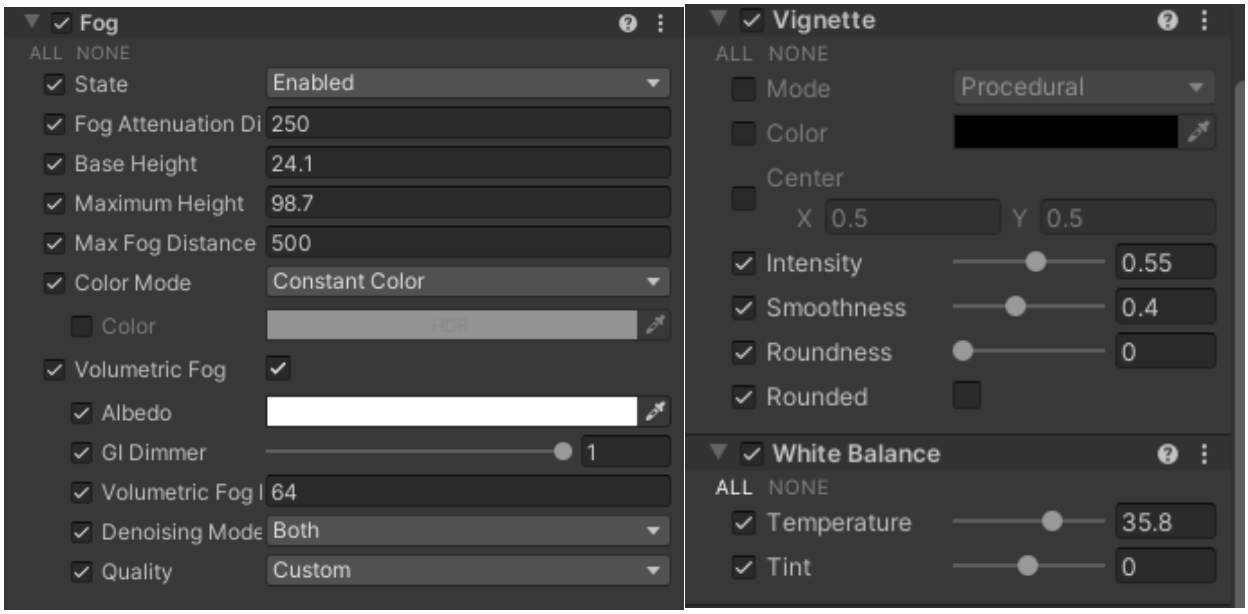
3. Pokud nejsou stisknuty žádné klávesy:

   Pohybuje kameru především vpřed a do stran s konstantní rychlostí (*DefaultSpeed*), což umožňuje plynulý pohyb v herním prostředí.

```
void Start()
{
    rb = GetComponent<Rigidbody>();
    ChangeDefSpeed.value = PlayerPrefs.GetFloat("save_speed", DefaultSpeed);
    ChangeSen.value = PlayerPrefs.GetFloat("save_sen", Sensitivita);

void Update()
{
        {
            rotationX += Input.GetAxis("Mouse X") * Sensitivita * Time.deltaTime;
            rotationY += Input.GetAxis("Mouse Y") * Sensitivita * Time.deltaTime;
            rotationY = Mathf.Clamp(rotationY, -90, 90);
        }

        var mouseDelta = new Vector2(rotationX, rotationY);
        mouseDelta = Vector2.Scale(mouseDelta, new Vector2(sensitivity.x * smooth.x, sensitivity.y
* smooth.y));
        MouseMotor.x = Mathf.Lerp(MouseMotor.x, mouseDelta.x, 1f / smooth.x);
        MouseMotor.y = Mathf.Lerp(MouseMotor.y, mouseDelta.y, 1f / smooth.y);
        transform.localRotation = Quaternion.AngleAxis(rotationX, Vector3.up);
        transform.localRotation *= Quaternion.AngleAxis(rotationY, Vector3.left);
        if (Input.GetKey(KeyCode.LeftShift) || Input.GetKey(KeyCode.RightShift))
        {
            rb.velocity = Vector3.zero;
            rb.velocity += transform.forward * (DefaultSpeed * UpMotor) *
                Input.GetAxis("Vertical") * Time.deltaTime;
            rb.velocity += transform.right * (DefaultSpeed * UpMotor) *
                Input.GetAxis("Horizontal") * Time.deltaTime;
```

5.1 Post Processing & Optimalizace

Jelikož diplomová práce využívá HDPR verze 14.0.10 – jedná se o vykreslovací systém s plnou podporou *Compute Shader* a *Shader Graph*. HDRP taktéž umožňuje realisticky vykreslovat materiál od neprůhledného po průhledný, či od kovového do matného odstínu až po vícevrstvé materiály. HDRP mimo to obsahuje lineární a HDR osvětlení a další metody jako je *SSAO*, *SSGI* nebo *SSR*. Proto je nutné mít správně nastavený projekt pro plné využití výhod, které s sebou HDRP verze 14.0.10 přináší. Nejdříve byly v *Project Settings -> Graphics -> HDRP Global Settings -> Diffusion Profile List* přiřazeny *Diffusion Profile* od NatureManufacture. Dále byly provedeny změny v *Quality* u atributu *Vsync Count* na hodnotu *Don't Sync.* Pro plné využité potenciálu *HDRP* byly vytvořeny dva objekty typu *Volume,* konkrétně *Sky and Fog Volume* a *Global Volume*. V prvním objektu byly připojeny a nastaveny komponenty jako: *Fog, Contact Shadow, Micro Sahdow, Indirect Lighting Controller, Visual Environment, Shadow, Volumetric Clouds, Exposure a Water Rendering*. Ve druhém objektu byly připojeny a nastaveny následující komponenty: *Vignette, White Balance, Chromatic Aberration, Ambient Occlusion, Bloom, Tonemapping a Depth Of Field*.

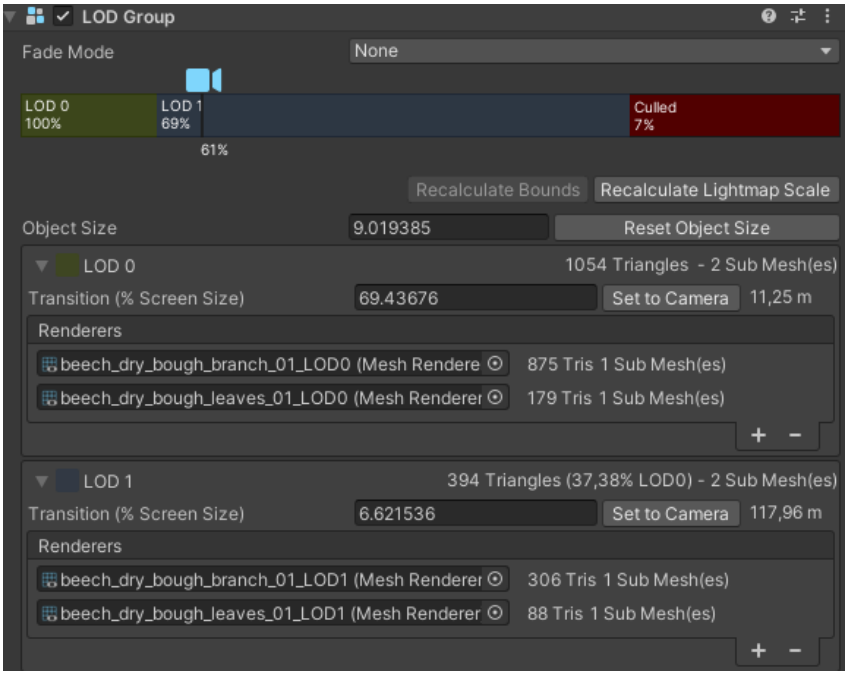


Obr. 5.1.1 Ukázka nastavení Fog a Vignete v objektech Volume (zdroj: autor)

Obr. 5.1.2 Scéna bez Post Processingu (zdroj: autor)

Obr. 5.1.3 Scéna s Post Processingem (zdroj: autor)

Pro optimalizaci aplikace bylo provedeno několik technik. Nejprve byly objekty, které nejsou chápány jako jednotlivé instance, vybaveny různými úrovněmi detailu (LOD) podle jejich velikosti a vzdálenosti od kamery. Tím bylo dosaženo snížení nároků na výpočetní výkon. Dále byla použita metoda CombineMesh ke spojení objektů, které byly blízko u sebe a měly podobnou velikost a četnost. Tato technika umožnila snížit počet drawcalls, což výrazně zlepšilo výkon aplikace. Poté byly objekty rozděleny na statické a dynamické. Statické objekty, které se nemění v průběhu hry, byly podrobeny technice *Occulus Culling*. Tato technika pracuje podle principu 'what you see is what you get' (*WYSIWYP*) a zajistila, že se vykreslovaly pouze objekty, které byly aktuálně viditelné z pozice kamery. Tím se snížil počet objektů, které se vykreslovaly, a zlepšil se celkový výkon aplikace. Celkově těmito kroky byla dosažena efektivní optimalizace aplikace, která umožňuje plynulý a bezproblémový běh s ohledem na zatíženost hardwaru.

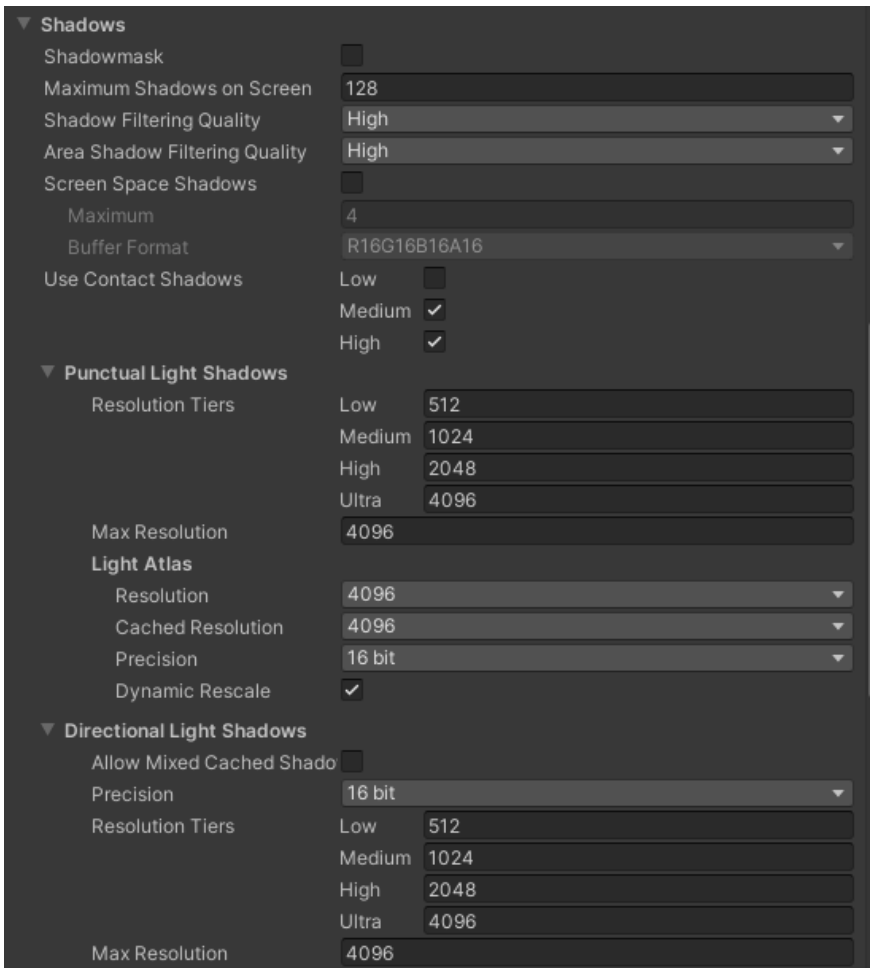


Obr. 5.1.4 Ukázka nastavení LOD group (zdroj: autor)

Dále ještě pro lepší optimalizaci, byly vytvořeny tři úrovně grafické obtížnosti (nízká, střední, vysoká). Nejdříve tedy byly vytvořeny tři *Render Pipeline Assety* reprezentující zmiňující úrovně. U každého *RPS* potom bylo nastaveny komponenty jako *Rendering, Decals, Dynymic resolution, Low res Transparency, Water, Lighting, Volumetric, Cookies, Reflection, Sky, Shadows, Material, Post-processing Quality Settings atd…*

Uživatel má tedy možnost podle výkonu svého počítače si vybrat tři úrovně grafického nastavení. Nízká grafická náročnost byla optimalizována pro zařízení s GPU GTX 1650 a vyšší při minimálních 20 FPS, při 8 GB RAM a více. Střední grafická náročnost byla optimalizována pro zařízení s GPU RTX 3060, při 16 GB RAM a více na minimálních 20 FPS. Nejvyšší grafická náročnost je optimalizována pro zařízení s GPU RTX 4070, při 16 GB RAM a více na minimálních 30 FPS.

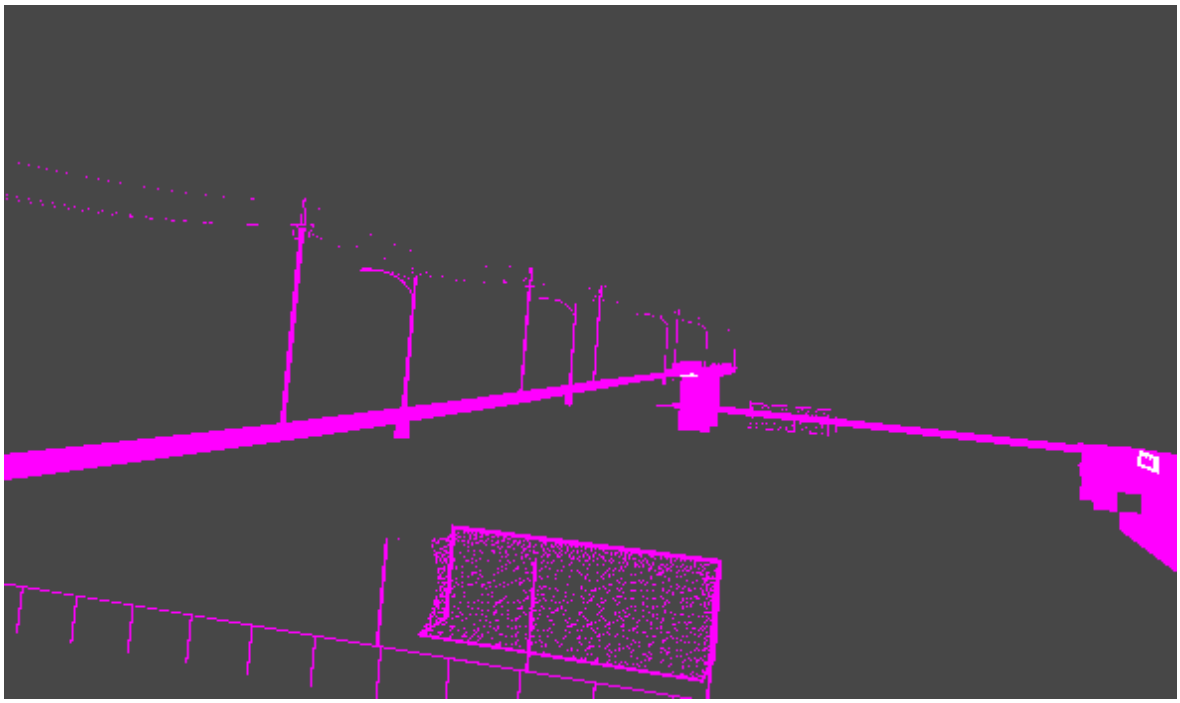
```
QUA = PlayerPrefs.GetInt("save_quality");
if (QUA == 1)
{
    QualityLow();
}

public void QualityLow()
{
    QUA = 1;
    QualitySettings.SetQualityLevel(0);
    PlayerPrefs.SetInt("save_quality", QUA);
```



Obr. 5.1.5 Ukázka nastavení Shadow v Render Pipeline Asset (zdroj: autor)

5.2 Webové řešení

Vytvořené prostředí s funkcionalitou dynamické změny bylo testováno a zkoumáno s ohledem na možnost jeho přenesení na web. Původní aplikace, jak již bylo zmíněno, je vytvářena pomocí HDRP, nicméně pro publikování na web musí být převedena na *Built-in Render Pipeline*. Bylo potřeba veškeré materiály u všech objektů ve scéně změnit na nové shadery odpovídající danému *Render Pipeline*. Dále daný *Render Pipeline* nepodporuje simulaci směru a rychlosti větru, tím pádem vlnitost vodní hladiny, směr a rychlost oblačnosti a samotné generování není možné převést z desktopové aplikace na web. Pro optimalizaci na web navíc byla použita pouze scéna „Hlavní menu“, tak aby uživatel získal v podobě statistiky data ze dvou senzorů a jedné webové služby.

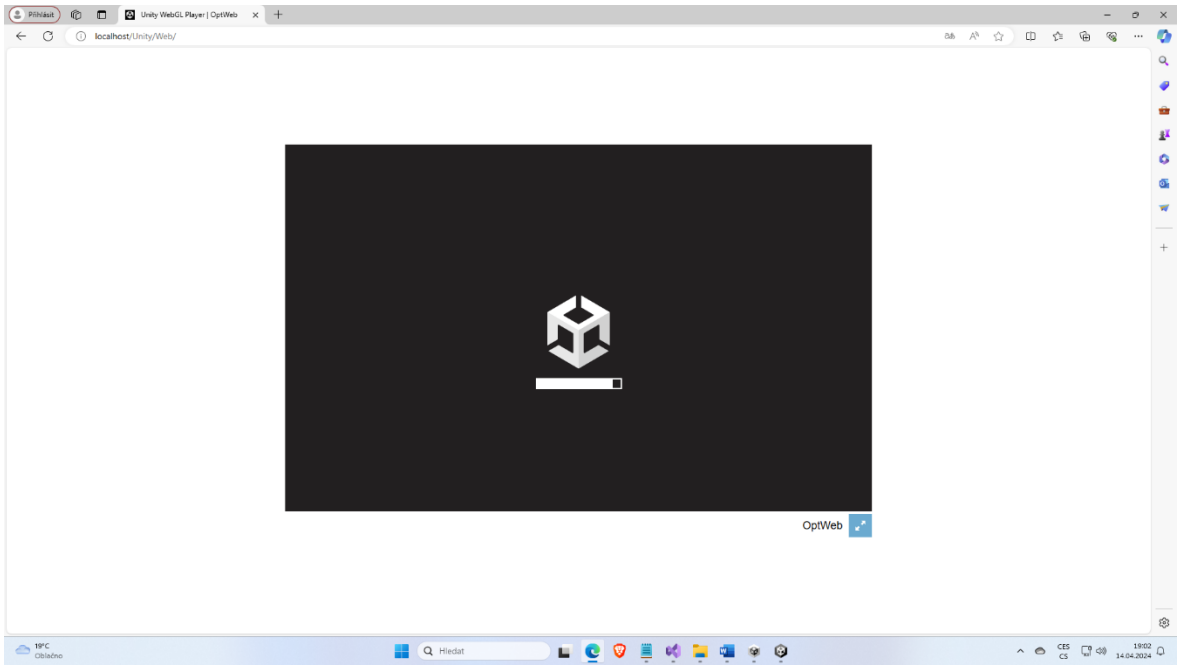


Obr. 5.2.1 Změna HDRP na Built-in Render Pipeline (zdroj: autor)

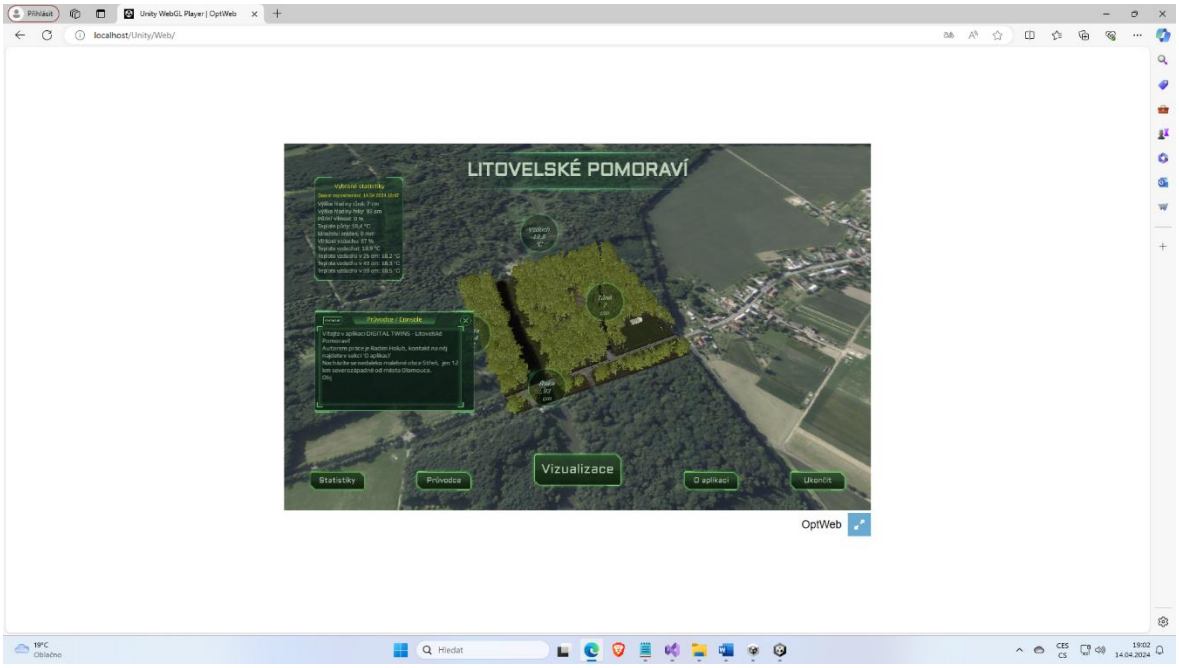
Předtím, než se aplikace převedla podle knihovny webGL, bylo nutné předtím nainstalovat a změnit compilaci z *Mono* na *IL2CPP*. Dále byly nainstalovány knihovny *webGL Build Support* a formát byl zvolen Gzip s *decompression fallback* hodnotou true. Na závěr bylo co nejvíce sníženo grafické nastavení a současně a Color Space byl změněn na Gamma.

Název	Datum změny	Typ	Velikost
Web.data.unityweb	08.04.2024 8:52	Soubor UNITYWEB	589 066 kB
Web.framework.js.unityweb	08.04.2024 8:52	Soubor UNITYWEB	80 kB
Web.loader.js	08.04.2024 8:52	JSFile	44 kB
Web.wasm.unityweb	08.04.2024 8:52	Soubor UNITYWEB	7 921 kB

Obr. 5.1.2 Soubory aplikace určené na web (zdroj: autor)



Obr. 5.2.3 Spuštění aplikace na webu (zdroj: autor)



Obr. 5.2.4 Aplikace ve webovém prostředí (zdroj: autor)

Výsledná aplikace má téměř 600 MB, a to pouze s jednou scénou. Z toho vyplývá, že webové prostředí momentálně není vhodné pro zobrazování složitých 3D aplikací, které využívají materiály, shadery, simulace a metody dostupné pouze při použití specifických *render pipeline*. Příkladem je nedostatečná komprese dat, nebo použité přístupy v diplomové práci nedovolující vytvářet simulace atmosférických podmínek, vodních ploch a dalších ve webovém prostředí. Dále je aplikace nevhodná pro použití na web z důvodu, že aplikace ve webovém prostředí musí být speciálně optimalizovány a momentálně knihovna WebGl plně nepodporuje možnosti, které nabízí herní enginy.

6 VÝSLEDKY

Hlavním výsledkem diplomové práce je vytvoření desktopové aplikace, která má za cíl vizualizovat a simulovat prostředí vybrané lokality v Litovelském Pomoraví. Tato aplikace byla postavena na reálných datech, zaznamenávaných a poskytovaných pomocí dvou senzorů a jedné webové služby. První senzor byl umístěn v tůni za fotbalovým hřištěm u obce Střeň. Tento senzor získává data jako teplotu půdy, výšku hladiny tůně, teplotu vzduchu v 20, 40 a 80 cm nad povrchem a mnoho dalších. Druhý senzor byl umístěn v řece Morava pod mostem spojujícím obce Střeň a Lhota nad Moravu. Slouží pro získávání aktuální výšky hladiny řeky Morava. Posledním zdrojem dat byla webová služba weatherbit.io, která obsahuje data jako je rychlost větru, teplota, pocitová teplota, viditelnost, oblačnost atd. Výsledná aplikace má velikost 2,1 GB, obsahuje více než 60 000 herních objektů a byla navržena tak, aby byla optimalizována pro širokou škálu hardwaru.

Pro dynamickou změnu prostředí na základě telemetrických dat bylo nejprve nutné získat odkazy na jednotlivé data pomocí API. Tato data byla ze senzorů a webové služby posílána do databáze, kde byla zpracovávána a průměrována. Následně byla poskytována z databáze na platformu ThingsBoard. V kapitole 4.1 diplomové práce je podrobně popsán postup získávání a zpracování dat, včetně četnosti aktualizací a způsobu zpracování dat před jejich použitím pro změny prostředí v herním enginu Unity.

Uživateli se po zapnutí aplikace zobrazí loading screen, následně po načtení loading baru je přenesen do scény hlavního menu, jehož postup vytvoření UI / UX je naznačen v kapitole 4.2.1. Scéna Hlavní menu obsahuje základní panely jako je: Průvodce, informace o aplikaci, statistiky dat, ukončení aplikace, nebo možnosti přístupu do další scény. Celá aplikace byla optimalizována pro FULL HD rozlišení, tedy 1920×1080 pixelů. Jelikož hlavní menu slouží jako prvotní kontakt uživatele s aplikací, byl na základě toho vytvořen digitální průvodce. Ten slouží k seznámení uživatele s aplikací a dává mu základní informace o tom, kde se nachází, co ho čeká, co má dělat, možnosti dalšího nastavení apod. Zároveň hlavní menu slouží pouze jako rozcestník. Proto bylo záměrem omezit uživateli zobrazená data a nalákat ho na další pokračování v rámci aplikace. Z tohoto důvodu zde má uživatel zamknutý pohyb a dívá se z výšky, z pohledu dronu na vybrané území jehož vytvoření popisuje kapitola 4.3.1. Kapitola 4.3.2 se poté zabývá modelováním a texturováním urbanistických objektů. Taktéž zde uživatel může vidět aktuální vybrané statistiky ze dvou senzorů a jedné webové služby, a navíc zde může interagovat s modelem. Najede-li uživatel myší na vybrané UI prvky reprezentující výšku řeky a tůně, dané UI prvky zafungují, jako prostředníci pro přemístění do hlavní scény, k místům, kde jsou lokalizovány senzory. Dále si zde uživatel má možnost přečíst si informace o autorovi a aplikaci nebo zde má možnost ukončení aplikace.

Rozhodne-li se uživatel dále pokračovat, je přenesen do hlavní scény, kde se prostředí dynamicky mění v závislosti na získaných telemetrických datech. Nejdříve bylo nutné vytvořit dané prostředí, které popisuje již zmiňovaná kapitola 4.3.1. Pro jeho vytvoření sloužila podkladová data získaná technologií lidar z projektu MOSPREMA, a to konkrétně digitální model reliéfu, digitální model povrchu a ortofoto. Assety v podobě vysoké a nízké vegetace byly získány od NatureManufacture. Vysoká vegetace se skládá z devíti prefabů umístěných manuálně, ale i dávkově na vytvořený terén. Nízkou vegetaci reprezentuje kapradí, traviny a další druhy nízké vegetace. Terén je doplněn o detaily, jako jsou pařezy, větve stromů a kameny. Urbanistické modely, jako fotbalové hřiště, lampy, stožáry elektrického vedení a most byly vymodelovány v softwaru Sketchup a následně pro ně byl vytvořen vhodný materiál se shaderem odpovídající pro daný render pipeline. Textury byly

vytvořeny v programu CrazyBump nebo získány z knihoven od Polyhaven a Ambientcg. Pro prohloubení a další úpravy ať už koryta řeky nebo tůně posloužily záplavové oblasti.

Dynamická změna prostředí tedy probíhá na základě telemetrických dat. Jedná se o komplexní systém, kdy jednotlivé simulace určitým způsobem na sebe navazují a ovlivňují se. Kapitola 4.4.2 poté obecně popisuje nejrůznější metody a přístupy, které řídí a vytváří v závislosti na datech jednotlivé simulace. Základem je interpolace minimálních a maximálních hodnot, které ovlivňují prostředí a slouží jako vstup parametrů simulace v herním enginu. Díky tomu se dynamicky mění, teplota prostředí, která závisí na intenzitě Slunce, délce dne nebo například na ročním období. Rychlost a směr větru poté ovlivňují rychlost a směr oblačnosti, směr a sílu deformace vegetace nebo vlnitost vodních hladin.

Taktéž je zde vytvořený VFX efekt a participle systém reprezentující padání sněhu a deště. Výška hladiny řeky Moravy a tůně se také mění v závislosti na telemetrických datech. Pro reprezentaci vodních hladin byly vytvořeny tři objekty, jejichž obecný způsob vytvoření popisuje kapitola 4.3.1. Pro změnu vegetace v jednotlivých ročních obdobích byly upraveny assety od NatureManufacture tak, aby textura vegetace odpovídala jednotlivým ročním obdobím. Díky vytvořeným splat maps, potom terén může jednoduše měnit texturu v závislosti na tom, zda senzory hlásí přítomnost sněhu nebo ne.

Hlavní dominantou je zde možnost libovolné změny scény na základě historických dat. Kapitola 4.4.1 popisuje způsob mechanismu pro dotazování na historické data pomocí API. Uživatel si zde může zvolit předvybrané události jako jsou čtyři roční období: jaro, léto, podzim, zima. Nebo lze vybrat jednotlivé stupně povodňové aktivity, a sledovat tak na základě reálných dat, jak je postupně vybraná lokalita Litovelského Pomoraví zaplavována řekou Moravou. Samozřejmě si zde uživatel taktéž může zvolit libovolný den, hodinu, měsíc a rok, a jakmile klikne na tlačítko potvrdit (existují-li pro dané datum data), tak se mu automaticky překresluje prostředí na základě jím vybraného data a tudíž i dostupných dat.

Aplikace dále umožňuje individuální nastavení ovládání podle preferencí uživatele, včetně rychlosti pohybu, sensitivitu myši, zrychlení a zpomalení pohybu, což je detailněji popsáno v kapitole 5. Dále uživatelé mohou nastavit grafické parametry podle výkonu svého počítače. Kapitola 5.1 obsahuje doporučené nastavení grafické náročnosti pro různé úrovně výkonu. Uživatelé mohou volit mezi třemi úrovněmi (nízká, střední, vysoká), přičemž každá úroveň je optimalizována pro určitý typ GPU a dostupnou operační paměť. Pro dosažení co nejlepšího výkonu bylo v aplikaci provedeno několik technik. Jednou z nich je použití různých úrovní detailu (LOD) pro objekty podle jejich velikosti a vzdálenosti od kamery. Dále byla aplikována metoda CombineMesh k optimalizaci počtu drawcalls, což výrazně zlepšilo celkový výkon aplikace. A zároveň zde byla i aplikována metoda *Occulus Culling*. Každá úroveň grafické náročnosti je optimalizována pro určitý typ GPU a dostupnou operační paměť, přičemž každá úroveň má stanovený minimální požadavek na počet snímků za sekundu (FPS). Tím je zajištěno, že uživatelé s různými konfiguracemi počítačů mohou plně využít potenciál aplikace a mít tak plynulý a příjemný zážitek. V aplikaci byl také proveden postprocessing, který využívá plnou sílu HDRP verze 14.0.1. Podrobnější jednotlivá nastavení a postup postprocessing je popsán v kapitole 5.1.

Na závěr bylo také provedeno testování webového řešení (kapitola 5.2), které vyžadovalo převod aplikace z HDRP na Built-in Render Pipeline. Tento proces zahrnoval změnu shaderů všech materiálů a odstranění simulace směru a rychlosti větru, vlnitosti vodní hladiny, směru a rychlosti oblačnosti a samotného generování, které nejsou podporovány v daném render pipeline.

7 DISKUZE

Diplomová práce je komplexní aplikace využívající spojení jednotlivých přístupů s moderními technologiemi. I přes veškerou pečlivost, která byla dávána do tvorby aplikace a vývoje, je troufalé si myslet, že aplikace nebude vykazovat žádné chyby, nepřesnosti nebo nebude potřeba ji udržovat či vylepšovat na základě nově vznikajících technik, metod a přístupů. Taktéž není v silách jednoho vývojáře nahradit pozice 3D designer, 3D Environment artist, 3D urbanistic artist, audio designer, backend developer, game architect, game producent, technical artist, UI / UX designer a mnoho dalších.

Prvním nedostatkem je aproximace vytvořených simulací v závislosti na datech. Vstupem do simulací jsou nyní pouze hodnoty za jedno časové období – což ovšem nekoresponduje s reálným světem, kdy dochází k průniku na sobě navazujících událostí. Příkladem může být pokrývka sněhu na terénu. Momentálně se dynamicky mění terén textury v závislosti na jednom časovém období (jsou-li splněné určité podmínky), tedy pokud by předešlé hodiny byl mráz a sněžilo by, na terénu by se objevil sníh. Nicméně v momentě, kdy senzory přestanou hlásit sněžení a záporné hodnoty teploty, sníh z terénu mizí a je nahrazen jinými texturami. V reálném světě by docházelo k postupnému, a ne skokovému úbytku. Na tento problém potom navazují nedokonale vytvořené splat maps, které momentálně nedokážou reflektovat postupnou změnu textur terénu vůči datům, a proto hodinové intervaly mezi určitými obdobími nemusí působit realisticky. Obdobným problémem, kdy se prostředí nemění posupně, ale skokově, je změna vysoké vegetace v závislosti na jednotlivých měsících, generalizovaných do čtyř ročních období. Nedochází například k postupnému opadávání listí v zimních a podzimních měsících, ale jsou vytvořené ostré hranice, kdy dochází ke změně materiálů na vysoké vegetace. Aby bylo dosaženo co nejvěrohodnější napodobení procesu opadávání, popřípadě žloutnutí listí na podzim, muselo by docházet k iteracím u každé instance stromu zvlášť. Tedy nejen, že by se měnily materiály stromů postupně, ale musely by se měnit i samotné textury u jednotlivých children reprezentující větvě daných assetů. Tento způsob změny nebyl zvolen z časových důvodů, ale i z důvodu optimalizace, protože bylo vyhodnoceno, že by nezbyl prostor optimalizovat nejspíše exponenciální nárůst drawcalls, z důvodu změněného přístupu k vykreslování vysoké vegetace.

Jedním z nejviditelnějších rozdílů oproti realitě může být typ assetů vysoké a nízké vegetace a jejich lokalizace. Z důvodu omezených možností nemusí druhy stromů odpovídat realitě. Dále by pro přesné určení lokality každého stromu bylo potřeba udělat automatizovanou klasifikaci nad daty získané pomocí LIDAR, kde by byly zaznamenány jednotlivé pozice a druhy vysoké vegetace, na které by se potom ručně, nebo pomocí nově vymyšleného skriptu nahrazovaly adekvátními assety. Dále by bylo vhodné ukládat ke každému klasifikovanému stromu informace jako je jeho výška vůči povrchu. V diplomové práci byly použity pro umístění vegetace ortofoto mapy a pouze hranice lesů byly přesněji lokalizovány. Zbytek vegetace byl dávkově přidán pomocí mass funkce, proto se může stát, že občas assety vegetace jsou v místech, kde by být neměly, a naopak. Výška stromů potom odpovídala rozptylu mezi průměrnou a maximální výškou získaných z digitálního modelu povrchu, a ne konkrétní výškou u konkrétního stromu.

Na základě rešerše byly testovány a nasazeny různé typy SDK, které vykreslovaly okolní svět pomocí 3D dlaždic. Terén byl potom detailně umístěn a lokalizován do okolního světa. Vzniklo tedy okolní prostředí, které můžeme poznat u Google Earth a jemu podobným desktopovým aplikacím. Nicméně z důvod licenčních podmínek, nakonec nebylo ve výsledné aplikaci použito žádné ze zmiňujících řešení. SDK byly omezeny buď copyrightem, který až moc viditelně zasahoval do výsledné podoby aplikace, nebo množstvím dovolených

requestů pro jeden účet. Proto je okolní prostředí alespoň částečně vykreslováno pomocí dostupného ortofota.

Dalším problémem je aktualizace a průměrování dat, která jsou následně posílány na platformu ThingsBoard. Tedy aplikace nemusí zachytit detailnější jevy, které trvají krátce a aplikace ukazuje přibližnou kopii reality pouze jednou za hodinu. Nicméně reálný svět se mění nepřetržitě. V závislosti na tom nedochází, nebo nemusí dojít k úplnému překreslování scény, chybí-li jakýkoliv typ vstupujících dat ze senzorů.

Z důvodu časové náročnosti v aplikaci chybí detailnější a realističtější přístup k VFX efektům a participle systému simulující padání deště a sněhu, které dynamicky nemění svoji intenzitu v závislosti na datech. Tedy intenzita (síla) jednotlivých efektů je stále konstantní nehledě na vstupujících datech. Taktéž zde chybí zapracování síly a směru větru, které by ovlivňovalo zmiňující efekty v reálném čase. Směr větru poté taktéž neovlivňuje simulaci vln na vodních plochách. I když simulace vlnitosti na základě síly větru je vytvořena, směr větru neovlivňuje vytvořené vlny. Proto vlny postupují pouze jedním směrem ve směru toku řeky Moravy.

V aplikaci taktéž chybí provázanost mezi postavením Slunce, typem oblačnosti a mlhou. Nad ránem a ve večerních hodinách by uživatel očekával zvýšenou koncentraci mlhy nad vodním tokem, tůní a jejich okolím. Nicméně v aplikaci byla opět použita konstantní hodnota, která nemění mlhu v závislosti na intenzitě slunce, která je ovlivňována typem oblačnosti. Samotné simulace oblačnosti, by dále potřebovaly více propracovat v závislosti na teplotě prostředí, postavení Slunce na obloze a dalších atmosférických podmínkách. Dále v aplikaci nejsou zakomponovány jednotlivé fáze Měsíce ani svit hvězd. Tedy noční obloha se mění pouze v závislosti na typu oblačnosti a aktuálním čase (oběh Slunce). Chybí zde objekt Měsíce a simulace jako je intenzita jeho svitu, která by ovlivňovala viditelně vytvořené prostředí. Současně aplikace nedisponuje simulací světla pouličních lamp ve večerních, nočních a ranních hodinách.

Chybám a generalizaci se nevyhnul ani terén v prostředí Unity, který byl manuálně upravován (prohlubován) v závislosti na dostupných záplavových oblastech. Během úprav prohloubení koryta došlo k absolutnímu odstranění sklonu řeky dna. Dále momentální znalosti optimalizace a možnosti prostředí Unity nedovolují vykreslovat s ohledem na vytížení hardwaru dávkové množství nízké vegetace. Proto se terén v určitých místech se může zdát až příliš holý. Břehy kolem řeky Moravy neobsahují dostatek husté nízké vegetace, jako by to bylo v realitě. Taktéž fotbalové hřiště neobsahuje žádnou nízkou vegetaci v podobě assetů. V aplikaci úplně chybí jakákoliv zvuková stopa, která by evokovala šumění listí, tok řeky, padání deště a podobně. Z již zmiňovaných časových důvodů a preferencí, které bylo potřeba doladit v aplikaci nezbyl čas na vytvoření triggerů, odkud by se počítala vzdálenost uživatele, tak aby docházelo k lineárnímu posunu zvukové stopy na základě pozice uživatele.

Pro celistvost aplikace bylo nutné vytvořit systém pro historické dotazování, tak aby se scéna překreslovala na základě zvoleného data podle uživatele a z omezených možností knihovny WebGL, která nedovoluje vykreslovat vytvořené simulace. Proto bylo rozhodnuto, že na webu bude pouze scéna hlavní menu se všemi statistikami. Přesto se nepovedlo vytvořit vhodnou kompresi, aby aplikace pro web měla pod 100 MB, a tedy z uživatelského hlediska, je aplikace pro nasazení do webovém prostředí v případě pomalého připojení k internetu téměř nepoužitelná.

Nicméně je nutné dodat, že pokud kdokoliv z Vás narazí na aplikaci v budoucnosti, jednotlivá zlepšení vyjmenovaná výše budou zakomponovány a upgradovány již mimo diplomovou práci. Proto text diplomové práce v budoucnosti již nemusí korespondovat s budoucím stavem desktopové, popřípadě i webové aplikace.

8 ZÁVĚR

Diplomová práce vznikla za podpory projektu MOSPREMA: Predikce a management kalamitních stavů komárů pro zachování biodiverzity v lužních lesích podpořeného Norskem prostřednictvím Norských fondů.

Cílem diplomové práce bylo dynamicky vizualizovat aktuální i historické fyzickogeografické podmínky s využitím integrace meteorologických a hydrologických dat ve 3D modelu území. V rámci této práce byla vyvinuta desktopová aplikace, která staví na principech herních enginů a využívá nejmodernějších grafických trendů a možností. Pro zajištění celistvosti aplikace bylo nutné vytvořit přívětivé uživatelské rozhraní, provést simulace a dynamicky měnit prostředí v závislosti na telemetrických datech, vytvořit a modelovat vybrané území a optimalizovat aplikaci tak, aby byla spustitelná na co největším spektru zařízení. Jako dílčí úkol bylo provedeno testování architektury aplikace ve webovém prostředí.

Pro splnění stanovených cílů bylo nejprve nezbytné podrobněji studovat řešenou problematiku a porozumět principům digitálních dvojčat, tvorbě simulací a dynamické změně prostředí, a poté tyto principy správně implementovat. Drtivá většina práce probíhala v softwaru Unity s využitím HDRP, kde byla vytvářena kompletní architektura desktopové aplikace. Aplikace využívá reálná data získaná pomocí dvou senzorů umístěných v tůni a v řece Morava, a také z webové služby weatherbit.io.

Po spuštění aplikace se uživateli zobrazí hlavní menu, které slouží jako rozcestník a zahrnuje digitálního průvodce, statistiky dat a možnost přístupu do hlavní scény. V hlavní scéně se prostředí dynamicky mění na základě telemetrických dat, což zahrnuje změny oblačnosti, rychlosti větru, výšku hladiny řeky a tůně, deformace vegetace, měnění povrchu a další. Pro dynamickou změnu prostředí bylo nejprve nutné vytvořit dané prostředí a získat napojení telemetrických dat pomocí API, které byly z databáze ukládány na platformu ThingsBoard. Při vytváření prostředí se vycházelo z dat získaných metodou LIDAR a při úpravách se také používala data obsahující záplavové oblasti. Statické modely byly následně modelovány pomocí metody hard-surface v programu Sketchup. Uživatel má v hlavní scéně možnost prozkoumat prostředí v různých předvybraných ročních obdobích, konkrétně na jaře, v létě, na podzim a v zimě. Dále má možnost sledovat historické události, jako jsou jednotlivé stupně povodňové aktivity, které opět vychází z reálných událostí. Uživatel si také může zvolit vlastní datum - konkrétní rok, měsíc, den a hodinu - a pokud existují pro dané datum data, aplikace se dynamicky přizpůsobí a změní prostředí. Aplikace umožňuje individuální nastavení ovládání a grafických parametrů podle výkonu počítače uživatele. Na závěr byl proveden postprocessing a optimalizace aplikace, která zahrnuje využití různých úrovní detailu pro objekty, optimalizaci počtu drawcalls pomocí metod CombineMesh nebo WYSIWYP.

Vytvořená desktopová aplikace bude volně dostupná veřejnosti prostřednictvím webových stránek Katedry geoinformatiky v Olomouci nebo na osobním webu autora: www.geolabtechnology.cz. Na těchto stránkách budou postupně zveřejňovány nové verze aplikace, které budou obsahovat opravy a vylepšení, jež jsou zmíněny v diskuzi, nebo které vyplynou ze zpětné vazby uživatelů. Cílem vytvořené aplikace je představit využití digitálních dvojčat v oblasti environmentálních otázek a přiblížit problematiku dynamických změn prostředí v herních enginech. Dalším cílem aplikace je seznámit uživatele s problematikou vykreslování environmentálního prostředí a jeho změn na základě telemetrických dat. Aplikace tak může sloužit jako základ pro tvorbu nových aplikací s tematikou digitálních dvojčat. Zároveň je aplikace určena jako propagační a edukační materiál a má za cíl přiblížit problematiku široké veřejnosti. Může být také

využívána k edukaci dětí na základních a mateřských školách díky jednoduchému ovládání aplikace. Zároveň aplikace slouží jako historický deník vybrané lokality a přináší z pohodlí domova plnohodnotnou reálnou kopii části Litovelského Pomoraví, díky níž si lidé mohou uvědomit, jak často dochází k extrémním výkyvům přírody.

POUŽITÁ LITERATURA A INFORMAČNÍ ZDROJE

JUDR. VERONIKA BAUEROVÁ, MSc., 2019. *Dějiny kartografie* [online] [vid. 2024-04-03]. Dostupné z: https://www.ustavprava.cz/blog/2019/01/dejiny-kartografie/

CUZK, [b.r.]. *Fyzickogeografická mapa České republiky 1:500 000* [online] [vid. 2024-04-03]. Dostupné z: https://geoportal.cuzk.cz/(S(rx3smdaby0fhe2cchbnug1gd))/Default.aspx?mode=TextMeta&side=mapy_tiskMUC&metadataID=CZ-CUZK-FGM500-T

THINGSBOARD, [b.r.]. *What is ThingsBoard?* [online] [vid. 2024-04-03]. Dostupné z: https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/

LUKAČOVIČ, Ivo, 2015. *About Windy* [online] [vid. 2024-04-03]. Dostupné z: https://community.windy.com/topic/4/about-windy

Anon., [b.r.]. *What is Google Earth?* [online] [vid. 2024-04-03]. Dostupné z: https://serc.carleton.edu/introgeo/google_earth/what.html

ESRI, [b.r.]. *ArcGIS Maps SDK for Unity* [online] [vid. 2024-04-04]. Dostupné z: https://developers.arcgis.com/unity/

MAPBOX, [b.r.]. *Unity applications* [online] [vid. 2024-04-04]. Dostupné z: https://docs.mapbox.com/help/getting-started/unity/

VGIS, [b.r.]. *vGIS Platform* [online] [vid. 2024-04-03]. Dostupné z: https://www.vgis.io/technical-specification-vgis-high-accuracy-survey-grade-augmented-reality-ar-for-bim-gis-arcgis-esri/

TWI-GLOBAL, [b.r.]. *What is Digital Twin?* [online] [vid. 2024-04-04]. Dostupné z: https://www.twi-global.com/technical-knowledge/faqs/what-is-digital-twin

AKSELOS, [b.r.]. *Digital Twins: A Comprehensive Guide* [online] [vid. 2024-04-04]. Dostupné z: https://akselos.com/digital-twins-a-comprehensive-guide/

GRIEVES, Michael, 2016. *Origins of the Digital Twin Concept* [online] [vid. 2024-04-04]. Dostupné z: https://www.researchgate.net/publication/307509727_Origins_of_the_Digital_Twin_Concept

GRIEVES, Michael, 2005. *Product lifecycle management: the new paradigm for enterprises* [online] [vid. 2024-04-04]. Dostupné z: https://www.researchgate.net/publication/247833967_Product_lifecycle_management_the_new_paradigm_for_enterprises

MISKINIS, Carlos, 2019. *What is the history behind the concept of digital twins and how the idea was turned into reality* [online] [vid. 2024-04-04]. Dostupné z: https://www.challenge.org/insights/digital-twin-history/

UNITY, [b.r.]. *How does a digital twin work?* [online] [vid. 2024-04-04]. Dostupné z: https://unity.com/solutions/digital-twin-definition

GELERNTER, D, 1993. *Mirror Worlds: Or: The Day Software Puts the Universe in a Shoebox. How it Will Happen and What it Will Mean.*

FRÄMLING, Kary K, 2003. *Product agents for handling information about physical objects* [online] [vid. 2024-04-04]. Dostupné z: https://primo.aalto.fi/discovery/openurl?institution=358AALTO_INST&vid=358AALTO_INST:VU1&ctx_enc=info:ofi%2FencUTF-8&rft_val_fmt=info:ofi%2Fkev:fmt:journal&rft.pub=Teknillinen%20korkeakoulu&ctx_tim=2024-03-02T14:31:33EET&rft.place=Espoo&rfr_id=info:sid%2Fpure.atira.dk:pure&ctx_ver=Z39.88-2004&rft.genre=preprint&rft.aufirst=Kary&rft.pages=20&url_ctx_fmt=info:ofi%2Ffmt:kev:mtx:ctx&rft.aulast=Fr%C3%A4mling&url_ver=Z39.88-2004&rft.auinit=K&rft.date=2003&rft.atitle=Product%20agents%20for%20handling%20information%20about%20physical%20objects

GRIEVES, Michael, 2009. *Back to the Future: Product Lifecycle Management and the Virtualization of Product Information* [online] [vid. 2024-04-04]. Dostupné z: https://www.researchgate.net/publication/286475988_Back_to_the_Future_Product_Lifecycle_Management_and_the_Virtualization_of_Product_Information

RANJAN, Sudhansu, 2023. *10 Amazing Examples of Digital Twin Technologies for Industries* [online] [vid. 2024-04-04]. Dostupné z: https://www.toobler.com/blog/digital-twin-examples

OILANDGASADVANCEMENT, [b.r.]. *Akselos and Shell sign framework agreement on digital twin technology* [online] [vid. 2024-04-04]. Dostupné z: https://www.oilandgasadvancement.com/news/akselos-and-shell-sign-framework-agreement-on-digital-twin-technology/

TWI-GLOBAL, [b.r.]. *WHAT IS A SMART CITY? – DEFINITION AND EXAMPLES* [online] [vid. 2024-04-04]. Dostupné z: https://www.twi-global.com/technical-knowledge/faqs/what-is-a-smart-city

GEOSPATIALWORLD, [b.r.]. *Virtual Singapore - Building a 3D-Empowered Smart Nation* [online] [vid. 2024-04-04]. Dostupné z: https://www.geospatialworld.net/prime/case-study/national-mapping/virtual-singapore-building-a-3d-empowered-smart-nation/

WALKER, Andy, 2023. *Singapore's digital twin – from science fiction to hi-tech reality* [online] [vid. 2024-04-04]. Dostupné z: https://infra.global/singapores-digital-twin-from-science-fiction-to-hi-tech-reality/

ESRI, [b.r.]. *A framework to create and integrate digital twins* [online] [vid. 2024-04-04]. Dostupné z: https://esrisingapore.com.sg/digital-twins

BLAIR a kol., 2020. Tackling the challenges of 21st-century open science and beyond: a data science lab approach.

NETZEROCLIMATE, [b.r.]. *WHAT IS NET ZERO?* [online] [vid. 2024-04-04]. Dostupné z: https://netzeroclimate.org/what-is-net-zero-2/

HENRYS, Peter A., 2023. *The role of data science in environmental digital twins: In praise of the arrows* [online] [vid. 2024-04-04]. Dostupné z: https://onlinelibrary.wiley.com/doi/full/10.1002/env.2789

FRESHWATERCOMPETENCECENTRE, [b.r.]. *Smarter river basin management with a digital twin of the river?* [online] [vid. 2024-04-04]. Dostupné z: https://www.freshwatercompetencecentre.com/digital-twin/

LI DEREN, YU WENBO, Shao Zhenfeng, 2021. *Smart city based on digital twins* [online] [vid. 2024-04-04]. Dostupné z: https://link.springer.com/article/10.1007/s43762-021-00005-y

GONZALES-INCA, C., CALLE, M., CROGHAN, D., TORABI HAGHIGHI, A., MARTTILA, H., SILANDER, J., & ALHO, P, 2022. *Geospatial Artificial Intelligence (GeoAI) in the Integrated Hydrological and Fluvial Systems Modeling: Review of Current Applications and Trends* [online]. Dostupné z: https://www.mdpi.com/2073-4441/14/14/2211

GEOMATICS, [b.r.]. *Creating components for river catchment digital twins that improve design, management and forecasting for flood alleviation and habitat management projects.* [online] [vid. 2024-04-04]. Dostupné z: https://www.storm-geomatics.com/service/river-digital-twin/

FORESTTWIN, [b.r.]. *Digital Twin Earth Precursors* [online] [vid. 2024-04-04]. Dostupné z: https://www.foresttwin.org/

ECMWF, [b.r.]. *ECMWF digital twins aid sustainable forestry operations under new Destination Earth use case* [online] [vid. 2024-04-04]. Dostupné z: https://stories.ecmwf.int/ecmwf-digital-twins-aid-sustainable-forestry-operations-under-new-destination-earth-use-case/index.html

THINGSBOARD, [b.r.]. *ThingsBoard REST API* [online] [vid. 2024-04-09]. Dostupné z: https://demo.thingsboard.io/swagger-ui/#/

GLITCHENZO, *No Title* [online] [vid. 2024-04-09]. Dostupné z: https://github.com/GlitchEnzo/NuGetForUnity

RADIM, Holub, 2022. *GAMIFIKOVANÝ VIRTUÁLNÍ PRŮVODCE HISTORICKÝM OBJEKTEM* [online] [vid. 2024-04-09]. Dostupné z: https://geolabtechnology.cz/app/BP_Text_Holub_Radim.pdf

STUDIO, D.F.Y, [b.r.]. *No Title* [online] [vid. 2024-04-09]. Dostupné z: https://assetstore.unity.com/publishers/39517

CUZK, [b.r.]. *Geoportál* [online] [vid. 2024-04-09]. Dostupné z: https://geoportal.cuzk.cz/

DOMLYSZ, [b.r.]. *BlenderGIS* [online] [vid. 2024-04-09]. Dostupné z: https://github.com/domlysz/BlenderGIS

Starý, Jiří, 2015. Interaktivní kniha pro mobilní zařízení v herním enginu Unity [online] [vid. 2022-04-09]. Dostupné z: https://is.muni.cz/th/yqgxy/stary-text_prace.pdf

UNITY, 2005. Unity [online] [vid. 2024-04-09]. Dostupné z: https://unity.com/

SHAKEEL, Abdullah, 2017. Object2Terrain [online] [vid. 2024-04-09]. Dostupné z: http://www.mediafire.com/file/3oyvdjcjix2d3iu/Object2Terrain.cs/file

NICKELCITYPIXELS, 2016. *WHAT'S A SPLAT MAP?* [online] [vid. 2024-04-09]. Dostupné z: https://nickelcitypixels.com/blog/2016/01/19/whats-a-splat-map/

NATUREMANUFACTURE, [b.r.]. *No Title* [online] [vid. 2024-04-09]. Dostupné z: https://assetstore.unity.com/publishers/6887

UNITY-TECHNOLOGIES, [b.r.]. *No Title* [online] [vid. 2024-04-09]. Dostupné z: https://github.com/Unity-Technologies/WaterScenes

EASYROAD, [b.r.]. *No Title* [online] [vid. 2024-04-09]. Dostupné z: https://www.easyroads3d.com/

Ambientcg, [b.r.]. *No Title* [online] [vid. 2024-04-09]. Dostupné z: https://ambientcg.com/

Polyhaven, [b.r.]. *No Title* [online] [vid. 2024-04-09]. Dostupné z: https://polyhaven.com/

CHMI, [b.r.]. *No Title* [online] [vid. 2024-04-13]. Dostupné z: https://www.chmi.cz/files/portal/docs/meteo/om/sivs/vitr.html

CLAUDE DEMERS, [b.r.]. *No Title* [online] [vid. 2024-04-13]. Dostupné z: https://www.researchgate.net/figure/Cloudiness-scale-according-to-weather-data-in-Quebec-City-March-21-st-to-March-29-th_fig5_313083757

SRINIVASAN, Balaji, [b.r.]. *No Title* [online] [vid. 2024-04-13]. Dostupné z: https://www.researchgate.net/figure/The-light-intensity-measured-at-different-times-of-a-typical-sunny-and-a-cloudy-day_fig2_241501654

PŘÍLOHY

SEZNAM PŘÍLOH

Volné přílohy

Příloha 1 Aplikace

Příloha 2 Poster

Příloha 3 Videa

- DigitalTwin_Flood.mp4
- DigitalTwin_Full.mp4
- DigitalTwin_Full_16x.mp4

Příloha 4 Skripty

- ActiveCamera.cs
- ButtonController.cs
- ButtonStatistics.cs
- CamerController.cs
- Console.cs
- DatumCheck.cs
- DayNight.cs
- Enviroment.cs
- Floatingtext.cs
- GlobalManager.cs
- HyperLinks.cs
- JWT.cs
- JWT_Main.cs
- MainMenu.cs
- NextPrevious.cs
- Object2Terrain.cs
- Quaility.cs
- ReplaceTree.cs
- RotationCamera.cs
- TerrainChange.cs
- TimestampConverte.cs
- TypeWriteEffect_Scroll.cs