FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Diplomová práce

Digitální pískoviště

Bc. Vojtěch Váchal

PLZEŇ

2023

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Diplomová práce

Digitální pískoviště

Bc. Vojtěch Váchal

Vedoucí práce
Ing. Petr Vaněček, Ph.D.

PLZEŇ **2023**

© Bc. Vojtěch Váchal, 2023.

Všechna práva vyhrazena. Žádná část tohoto dokumentu nesmí být reprodukována ani rozšiřována jakoukoli formou, elektronicky či mechanicky, fotokopírováním, nahráváním nebo jiným způsobem, nebo uložena v systému pro ukládání a vyhledávání informací bez písemného souhlasu držitelů autorských práv.

Citace v seznamu literatury:
VÁCHAL, Bc. Vojtěch. *Digitální pískoviště*. Plzeň, 2023. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ing. Petr Vaněček, Ph.D.

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

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Vojtěch VÁCHAL**
Osobní číslo:	**A21N0074P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Téma práce:	**Digitální pískoviště**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se se stávajícím projektem digitálního pískoviště.
2. Proveďte důkladnou analýzu požadavků na kalibraci, filtraci a zpracování obrazových dat, a metod vhodných pro tyto účely.
3. Navrhněte modulární systém, který umožní jednoduchým způsobem přidávat či měnit stávající funkcionalitu.
4. Implementaci ověřte na reimplementaci stávající funkcionality do nového systému a vytvořením VR aplikace, která umožní využívat výstupy digitálního pískoviště v prostředí virtuální reality.
5. Navržené řešení důkladně zdokumentujte s ohledem na budoucí rozšiřování projektu.

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

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Petr Vaněček, Ph.D.**
Katedra informatiky a výpočetní techniky

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

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

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

V Plzni dne 11. října 2022

Prohlášení

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

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

V Plzni dne 15. května 2023

. . . . . .
Bc. Vojtěch Váchal

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

Abstrakt

Diplomová práce si klade za cíl provést důkladnou analýzu projektu SandyStation, který se nachází na katedře informatiky a výpočetní techniky a vytvoření modulární architektury, která dovolí přidávat či měnit funkčnost celého pískoviště. V teoretické části je důkladně představen projekt SandyStation a jeho funkcionalita. Poté bude představeno zařízení Kinect, které je využito v systému. Po důsledné analýze bude navržena modulární architektura, která bude sloužit jako jádro pro reimplementaci aktuální aplikace. Pro ověření této architektury bude vytvořena i VR aplikace, která bude komunikovat s nově vytvořenou aplikací. Nakonec bude vytvořena dokumentace, která bude sloužit pro případné rozšiřování projektu následujícími generacemi.

Abstract

The thesis aims to perform a deep analysis of the SandyStation project located in the Department of Computer Science and Engineering and to create a modular architecture that allows adding or changing the functionality of the entire sandbox system. In the theoretical part, the SandyStation project and its functionality is thoroughly described. Then, the Kinect device that is used in the system will be introduced. After a detailed analysis, a modular architecture will be designed. Architecture will be used as the core for reimplementing the current application. To validate this architecture, a VR application will also be created to interact with the newly created application. Finally, a user documentation will be created that will be used for possible extensions of the project by the following generations.

Klíčová slova

MS Kinect • SandyStation • .NET • VR • virtuální realita • architektura

Poděkování

Na tomto místě bych rád poděkoval panu *Ing. Petru Vaněčkovi, Ph.D.*, který mi dal příležitost prozkoumat možnosti zařízení Kinect a virtuální reality. Dále bych chtěl poděkovat mé přítelkyni, rodině a přátelům, kteří mě při tvorbě práce podporovali.

Obsah

Úvod 1

Žijeme v době, kdy se různé aplikace a hry snaží držet krok s nejmodernějšími technologiemi. Mezi ně se řadí i virtuální realita (anglicky Virtual Reality, zkráceně VR). Ta se v posledních letech stává stále více populární díky pokrokům v moderních technologiích. Dříve byly technologie a zařízení pro virtuální realitu velice nedostupné a náročné na používání, ale dnes se stávají stále více dostupnějšími [1]. Existují různé druhy VR zařízení, od mobilních headsetů po stolní VR systémy. Moderní VR headsety mají vysoké rozlišení obrazovky, velké pole pohledu a vysoce citlivé senzory, které umožňují uživatelům plně se ponořit do virtuální reality a umocnit tak jejich zážitky. K dispozici je i řada aplikací a her, které využívají VR technologii a tyto aplikace stále přibývají. Díky tomu je svět virtuální reality stále více populární. Využít se dá nejen v počítačových hrách, ale také v průmyslu, vzdělávání nebo ve zdravotnictví.

Cílem diplomové práce je provést důkladnou restrukturalizaci stávajícího projektu **SandyStation**, který se nachází na katedře informatiky a výpočetní techniky (zkráceně KIV). Po pečlivé analýze dojde k reimplementaci na základě nově vytvořené architektury, která bude zaměřena na modulárnost a lepší použitelnost z hlediska komunikace s jinými aplikacemi. Tento postup bude ověřen vytvořením aplikace, která s pomocí virtuální reality dovolí uživateli, prožít funkce systému SandyStation v prostředí virtuální reality.

Diplomová práce je rozdělena do dvou hlavních částí - teoretická a praktická. V teoretické části bude vytvořena důkladná analýza projektu SandyStation, který byl vytvořen v rámci zájmové studie několika studentů. Následně bude představeno zařízení Kinect, které je v tomto systému aktuálně používáno a zároveň představím několik pojmů a základních informací z oblasti virtuální reality.

Praktická část se zaměřuje na vytvoření a popis modulární architektury, která dostatečným způsobem umožní systém nadále rozšiřovat. V této části bude pečlivě popsán proces reimplementace a kroky, které byly provedeny. Nakonec je uveden postup, kterým byla vytvářena VR aplikace a zhodnocení, jestli navržená architektura splňuje požadavky, které na ni byly kladeny.

Stávající projekt 2

V této kapitole se zaměřím na popsání projektu **SandyStation**. Budou zde uvedené jeho základní vlastnosti, funkční hry, použité technologie a další potřebné informace, které by mohli být užitečné pro návrh modulární architektury a reimplementaci projektu.

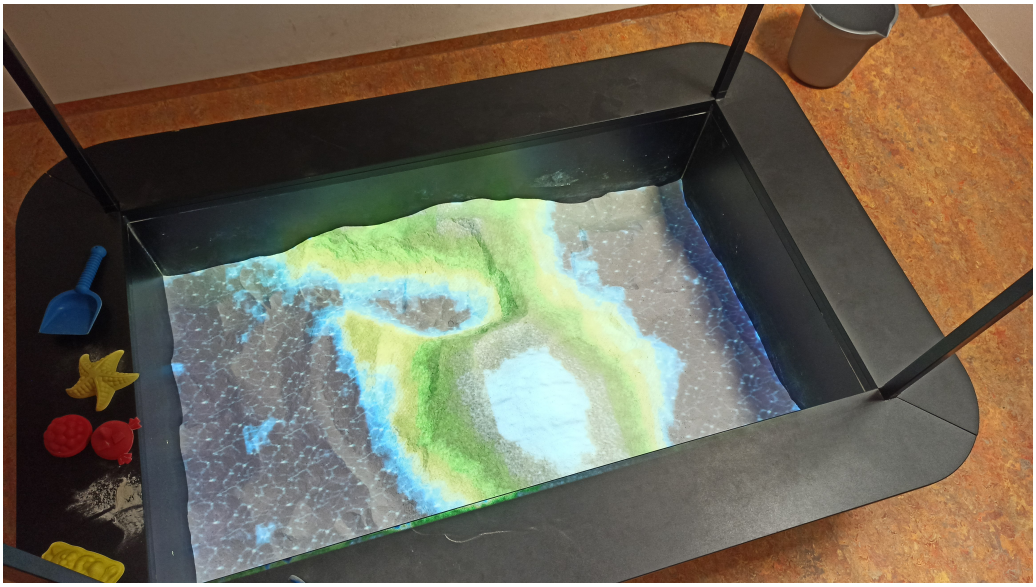
Jak jsem již zmínil v úvodu diplomové práce, prvním krokem bylo seznámit se zařízením a systémem nazvaným SandyStation. Toto zařízení je v aktuální chvíli vystavené v grafické laboratoři na katedře informatiky a výpočetní techniky. Interaktivní pískoviště bylo před několika lety vytvořeno studenty v rámci jejich zájmové činnosti. Následně se dostalo i k dalšímu využití a rozšíření v rámci různých předmětů, které jsou na fakultě aplikovaných věd (zkráceně FAV) vyučovány. Pro představu, jak tento systém vypadá, se lze podívat na obrázek 2.1 a 2.2, kde jsou zobrazeny ukázky přímo z grafické učebny na KIVu.

2.1 Historie projektu

Nejdříve se zaměříme na historii projektu SandyStation, abychom pochopili, jak se projekt rozvíjel a rozšířil mezi komunitu. Jelikož neexistuje moc zdrojů, ze kterých jsem mohl čerpat, obrátil jsem se přímo na zakladatele projektu, kteří byli velice ochotní a poskytli mi klíčové informace o historii projektu, jeho cílech a vizi.

Projekt vznikl jako zájmový projekt pár studentů *Petra Altmana* a *Roberta Ecksteina*, kteří nejspíše chtěli rozšířit své znalosti a zároveň vytvořit zajímavý systém, který bude sloužit pro (převážně) dětské publikum. Proto vznikl projekt SandyStation, který je již na KIVu několik let. První verze projektu byla funkční již v roce 2011. Do projektu byli zapojeni i někteří studenti z fakulty elektrotechnické (zkráceně FEL), kteří pomáhali s tvorbou celé konstrukce a umístěním jednotlivých komponentů. Vývojový tým se zaměřoval převážně na výzkum hloubkových senzorů, které dokážou měřit vzdálenost objektů před senzorem. Projekt SandyStation se zaměřoval hlavně na senzor Kinect od společnosti Microsoft, který se stal populárním díky konzoli Xbox 360 (více informací o senzoru Kinect lze nalézt v kapitole 3 na straně 15). [2]

Obrázek 2.1: Systém SandyStation v prostorách KIVu



Obrázek 2.2: Interaktivní krajina vytvořená systémem SandyStation

Kinect dokáže snímat hloubkové informace o prostoru za pomoci infračerveného světla a specifických kamer. Tento přístup umožňuje interakci uživatele s okol-

ním prostředím za pomoci gest a pohybů. Senzor Kinect se stal základem pro vývoj nového programu, který by umožnil interaktivní zážitek s virtuální krajinou. Software SandyStation umožňuje snímat hloubková data díky senzoru Kinect a na základě těchto dat vykreslovat (nejen) krajinu, která bude promítána pomocí projektoru na pískoviště.

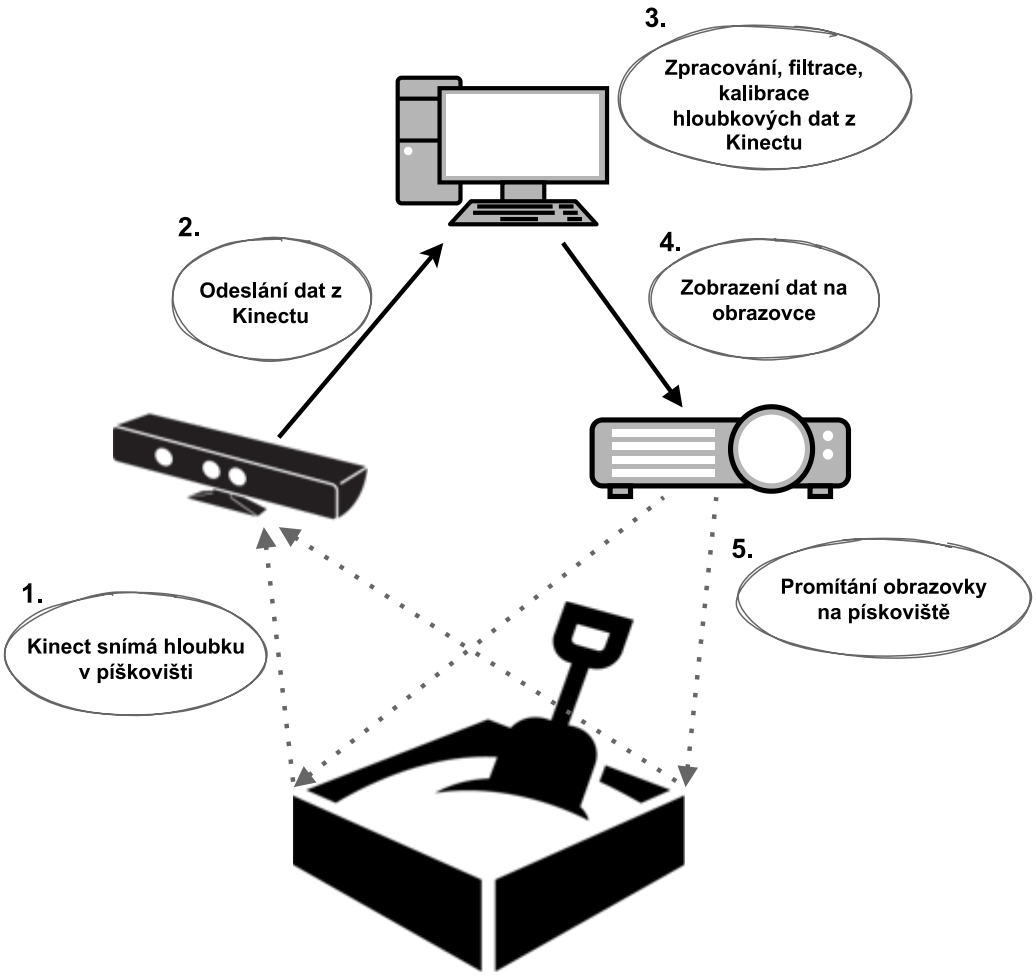
Výsledný projekt byl, po úspěšné implementaci, prezentován na *Dnech vědy a techniky* v Plzni. Prezentace systému SandyStation získala velkou pozornost a projekt se stal populárním nejen mezi návštěvníky, ale také lidmi z oboru. Zakladatelé projektu také zmiňují, že postupem času se projekt rozšířil a byl prezentován na dalších akcích po celé České republice. Mezi tyto akce patřil například technologický festival zvaný *Microsoft Fest* nebo akce s názvem *BackToSchool*. Projekt se těšil velké pozornosti a začali o něm psát různá média, což pomohlo k jeho další popularizaci. [2]

Projekt SandyStation se stal úspěšným příkladem využití nových technologií k vytvoření interaktivního zážitku. Tento projekt také ukázal, jak se dá využít nová technologie (senzor Kinect) k vytvoření nových a neobvyklých zážitků, které by bez této technologie nebyly možné. V aktuální době jsou zakladatelé projektu vlastníky společnosti *SandyStation s.r.o.*, která se stará o vytváření zajímavých her s podobnou tématikou. Kontakt na zakladatele a více informací o tom, co společnost vytváří, lze nalézt na internetové stránce `https://www.sandystation.com`. [3]

2.2 Popis funkcionality

Funkcionalita celé aplikace je jednoduchá. Celé zařízení je zkonstruováno z celkem tří částí - box s obyčejným pískem, senzor Kinect a obyčejný projektor. Senzor Kinect a projektor jsou připojeny do počítače s operačním systémem `Windows 10`. V počítači běží unikatní aplikace, která dokáže extrahovat data ze senzoru Kinect, které následně zpracovává a analyzuje. Mezi aktivity zpracování patří například kalibrace, či filtrace hodnot, které přesahují dynamicky specifikovaný rozsah. Nakonec se provede aktualizace herní scény, která se vytváří z těchto dat. Herní scéna je nakonec pouze zobrazena na ploše počítače konkrétního serveru.

Pro přenos výstupu programu na plochu pískoviště se používá připojený projektor, který promítá plochu serveru právě na oblast pod ním (tj. pískoviště). Tímto způsobem se na pískovišti objeví konkrétní zabravení podle toho, který režim hry je aktuálně zvolen. Velice zjednodušený princip fungování systému je znázorněn na obrázku 2.3 na straně 10.



Obrázek 2.3: Princip fungování aplikace SandyStation

2.2.1 Použité technologie

Celá aplikace využívá řadu technologií, přičemž jednou z nejdůležitějších je moderní objektově orientovaný jazyk C# a platforma **.NET Framework**. Všechny vytvořené projekty využívají verzi `4.8`, což je v současné době druhá nejnovější verze z řady .NET Framework. Microsoft se aktuálně zaměřuje spíše na framework zvaný **.NET Core**. [4]

Pro uživatelsky přívětivější ovládání aplikace bylo vytvořeno i základní grafické uživatelské rozhraní (anglicky Graphic User Interface, zkráceně GUI) využívající technologii **WinForms** nebo **Windows Presentation Foundation** (zkráceně WPF).

2.2.1.1 Knihovny třetích stran

Následně je v aplikaci použito několik dalších externích knihoven pro zpracování dat a jejich vykreslování do výsledné scény. Mezi ně lze zařadit knihovnu **OpenCV, SlimDX** a **Emgu.CV**.

2.3 Projekty

Jelikož už je program vyvíjen přes 10 let, není divu, že je velice rozšířený a obsahuje mnoho různých funkcí, projektů a her. Aplikace disponuje rozsáhlou strukturou projektů a každý projekt má svoji vlastní funkci. Seznam všech existujících projektů je zobrazen na obrázku 2.4. Většina projektů slouží pro definici konkrétních her. Některé další projekty jsou zde spíše jako testovací či nedokončené, nepotřebné nebo nefunkční.

2.3.1 Popis funkčních her

V rámci celého projektu bylo vytvořeno několik her, které jsou aktuálně funkční a lze si je vyzkoušet na pískovišti. Projekt zahrnuje hry v různých fázích vývoje a zároveň i různé experimenty původních vývojářů. V této sekci budou popsány hry, které v době tvorby diplomové práce lze hrát na katedře informatiky a výpočetní techniky.

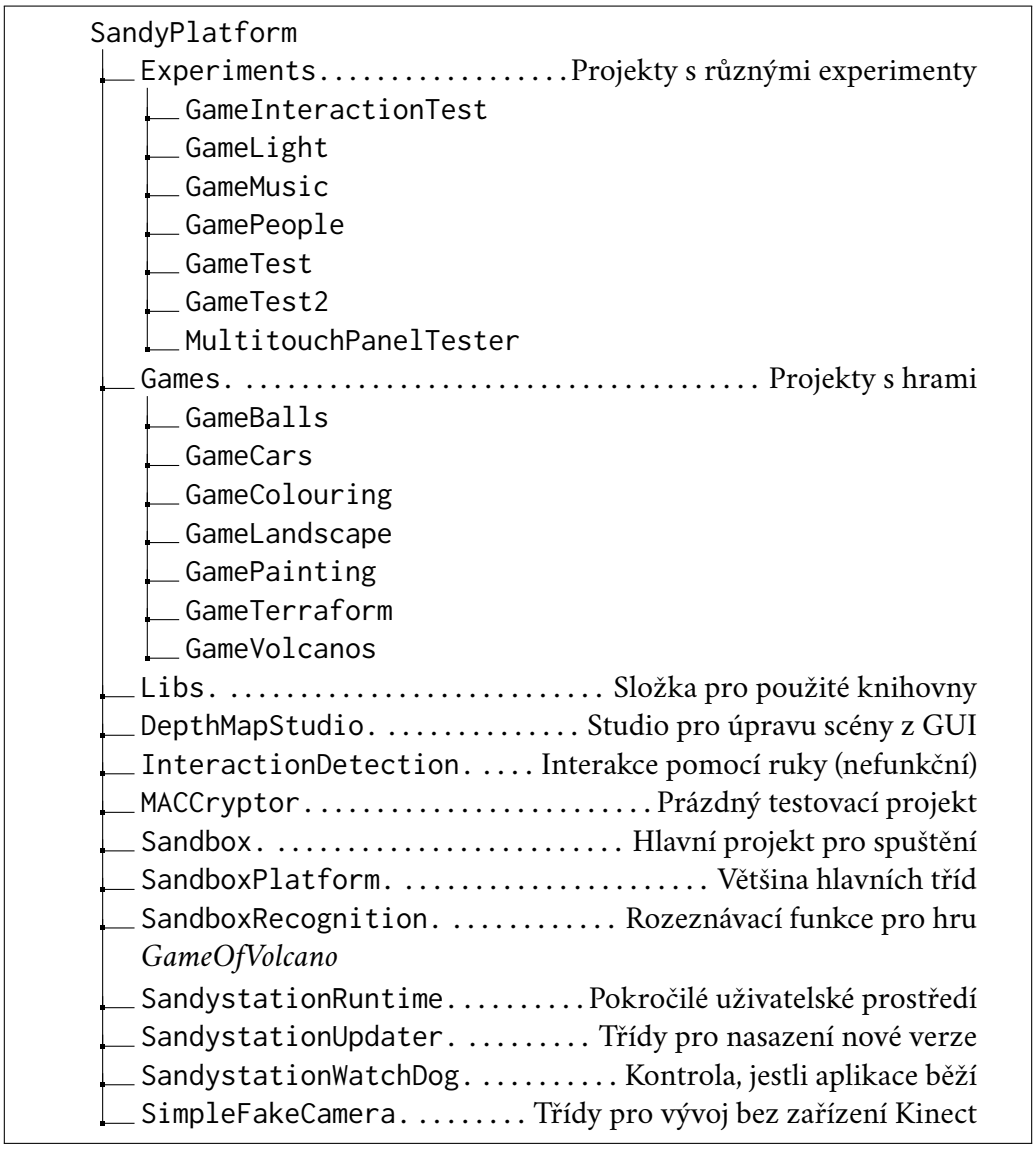
2.3.1.1 GameLandscape

Hra *GameLandscape* je velmi jednoduchá. Uživateli se na pískovišti zobrazí přírodní prvky podle konkrétní výšky, kterou snímá senzor Kinect. Vysoká místa představují kopce či hory a místa, kde je výška písku nízká, reprezentují louky, řeky, moře, atd. Do této hry byla přidána animace vln ve vodě, která přidává do pískoviště větší autentičnosti živého prostředí.

2.3.1.2 GameCars

Hra *GameCars* využívá základů předchozí hry (tj. *GameLandscape*) a rozšiřuje ji o další funkcionalitu. Do hry bylo přidáno jedno autíčko, které se pohybuje po krajině vytvořené na pískovišti. Na základě pohybu autíčka lze usuzovat, že se udržuje pouze na vodě a snaží se vyhýbat pevnině.

S velkou pravděpodobností se jedná o nedodělanou hru, ve které bylo potřeba zaměnit textury, které se zobrazují na pískovišti. Větší smysl by dávalo zobrazovat texturu se silnicí (místo vody), po které by jezdilo autíčko.



Obrázek 2.4: Struktura a popis jednotlivých projektů

2.3.1.3 GameColouring

V této hře se povrch pískoviště obarví jednoduchými barvami - zelená, modrá a červená. Zelená barva udává, že výška v daném místě je „správně“ vysoká. Červená barva znázorňuje, že je terén příliš vysoký a naopak modrá barva značí, že je terén příliš nízko. Hráč se snaží, aby byl celý povrch pískoviště zabarven zelenou barvou.

Po vytvoření krajiny v pískovišti může uživatel přepnout na jinou hru a prohlédnout si, jak by daná krajina vypadala s vykreslenou vodou a horami, nebo v ní vytvořit autíčko a nechat ho jezdit po krajině.

2.3.1.4 GameVolcanos

Hra *GameVolcanos* je nejimpozantnější a zároveň i nejkomplexnější hra, kterou lze v pískovišti hrát. Uživatel si v této hře může nejen vytvářet krajinu, ale také vytvářet sopky z kopců, které si sám vymodeluje z písku.

Pokud uživatel vytvoří kopec a na jeho vrcholku udělá prohlubeň, aplikace tento tvar rozpozná a začne v tomto místě kreslit aktivní sopku, z níž vytéká láva a zaplavuje okolní krajinu.

2.3.1.5 GameTerraform

V této hře se v krajině objeví kruh, ze kterého po krátké době začne proudit voda. Voda slouží jako vláha okolnímu prostředí. Tedy na okolním povrchu začne růst tráva. Po nějaké době (asi 30 vteřin) se změní místo, odkud vychází proud vody. Tímto způsobem může hráč postupně nechat rozkvést krajinu pískoviště. Když je krajina správně a dlouhodobě zavlažována zobrazí se na různých místech farmy. Nakonec i farmáři, kteří danou půdu obdělávají.

2.3.1.6 GameBalls

Poslední hra je velice podobná hře *GameCars*. Avšak tentokrát, je namísto autíčka, vytvořeno několik červených balónků nebo míčků, které se kutálí z kopců. Uživatel, tak může přesouvat míčky pomocí manipulace s pískem a tvorbou krajiny.

Pohybový sensor Kinect

3

Kinect je periferní kamera, vytvořena společností *Microsoft* pro videoherní konzoli *Xbox 360*. Jedná se o systém pohybového ovládání, který snímá pohyby uživatelů před senzorem a převádí je na ovládací akce pro herní konzoli Xbox 360 bez potřeby ovladače prostřednictvím přirozeného uživatelského rozhraní (anglicky Natural User Interface, zkráceně NUI) a to pouze s pomocí gest a mluvených příkazů. [5] Ukázka toho, jak takové zařízení vypadá, je na obrázku 3.1.

Obrázek 3.1: Senzor Kinect v1 [6]

3.1 Historie

Jak již bylo zmíněno, zařízení Kinect bylo vyvinuto společností Microsoft a patentováno v roce 2006 pod názvem **Projekt Natal**. Záměr vytvořit revoluční herní ovladač pro Xbox 360 byl iniciován představením konkurenční konzole Wii na konferenci *Tokyo Game Show 2005*. Společnost *Nintendo* představila nové herní zařízení nazvané **Wii Remote**, které dokáže detekovat pohyb ve třech dimenzích a obsahuje optický senzor, který zjišťuje, kam zařízení ukazuje. To donutilo divizi Xbox, společnosti Microsoft, aby začala pracovat na konkurenčním zařízení, které by překonalo herní ovladač Wii Remote. Proto Microsoft vytvořil dva konkurenční týmy, které měly vytvořit požadované zařízení. První tým pracoval s technologií **PrimeSense** a druhý s technologií vyvinutou společností **3DV**. [7]

Vydání senzoru Kinect se setkalo se smíšenými ohlasy. Herní weby obecně uznaly, že technologie je skvělá, ale domnívaly se, že hráče rychle omrzí. To však nezpomalilo prodej Kinectu. Za prvních 60 dní se prodalo přibližně 133 tisíc kusů denně. Tímto překonal prodejní rekordy iPhonu nebo iPadu a stanovil nový Guinnessův světový rekord. Nebylo to tím, že by se herní recenzentské weby mýlily, ale tím, že lidé chtěli Kinect tak jako tak, ať už si s ním hráli každý den, nebo jen pár hodin. Byl to kus budoucnosti, který si každý mohl pořídit do své domácnosti. [5, 7]

Dne 17. června 2011 společnost Microsoft konečně uvolnila beta verzi vývojářské sady Kinect Software Development Kit (SDK) pro veřejnost pod nekomerční licencí, poté, co ji několik týdnů předváděla na akcích jako například *MIX*. Jak bylo slíbeno, obsahovala algoritmy pro rozpoznávání kostry, díky nimž není nutné vytvářet počáteční pózu, a také několik dalších užitečných funkcí. Každý vývojář měl nyní přístup ke stejným nástrojům, které společnost Microsoft používala interně pro vývoj aplikací pro Xbox 360. [7]

Společnost Microsoft nabízí dvě verze zařízení Kinect. První z nich, *Kinect pro Xbox 360*, je určena přímo pro konzoli Xbox 360 a byla uvedena na trh v listopadu 2010. Poté, co byl Kinect prolomen a internetem se rozšířilo mnoho různých aplikací, si společnost Microsoft všimla existence zcela nového trhu. Na základě tohoto zjištění společnost Microsoft navrhla druhou verzi senzoru, *Kinect pro Windows*, zaměřenou na vývoj komerčních aplikací pro osobní počítače. Mězi těmito dvěma verzemi existují pouze nepatrné rozdíly. Nejdůležitější rozdíl mezi Kinectem pro Xbox 360 a Kinectem pro Windows spočívá zejména v dodatečné podpoře snímání hloubky v blízkém dosahu, která umožňuje senzoru vidět ze vzdálenosti 40 centimetrů namísto 80 centimetrů. [8]

3.2 Software Development Kit

Pro vývoj vlastních aplikací pro zařízení Kinect je k dispozici několik sad pro vývoj softwaru. Jednou z nich je knihovna `libfreenect`. Vznikla jako výsledek hackerského úsilí v době, kdy ještě společnost Microsoft neuvolnila veřejné ovladače a vývojové sady pro počítač. Další knihovnou je `OpenNI`. Byla vydána společností *PrimeSense* a kromě standardních vstupů zahrnuje i sledování kostry. V dnešní době již existuje oficiální sada (SDK) společnosti Microsoft pro Kinect, která je zároveň použita i v systému SandyStation. [8, 9, 10]

V rámci diplomové práce bude použit právě oficiální SDK od společnosti Microsoft a proto bude nadále uvažována pouze tato vývojová sada. Konkrétně se bude jednat o sadu `Kinect for Windows SDK v1.8` , která je dostupná na adrese `https://www.microsoft.com/en-us/download/details.aspx?id=40278`.

3.2.1 Kinect for Windows SDK

Kinect for Windows SDK je sada nástrojů a knihoven, které mohou vývojáři používat k vytváření aplikací využívajících senzor Kinect v systému Windows. Sada SDK poskytuje řadu funkcí, jako je sledování zvuku, kostry, rozpoznávání řeči nebo 3D skenování, které vývojářům umožní vytvářet inovativní a interaktivní aplikace, jež mohou reagovat na gesta a hlasové příkazy uživatelů. [11, 12]

Sada byla poprvé zveřejněna v roce 2011 po úspěšném vydání Kinectu na herní scénu. Od té doby prošla několika verzemi, přičemž každá nová verze přidávala nové funkce a zlepšovala výkon a funkčnost SDK. [11, 9]

Jednou z hlavních výhod sady Kinect for Windows SDK je, že poskytuje snadno použitelné rozhraní, které mohou vývojáři používat k přístupu k datům a funkcím senzoru. Vývojáři se tak mohou soustředit na tvorbu inovativních aplikací, a netrávit čas nízkoúrovňovými programovými úlohami. Pomocí této knihovny lze vytvářet WPF aplikace, aplikace WinForms, XNA a mnoho dalších. Kupodivu není možné vytvářet aplikace pro Xbox s SDK pro Windows. [7]

3.3 Hardware

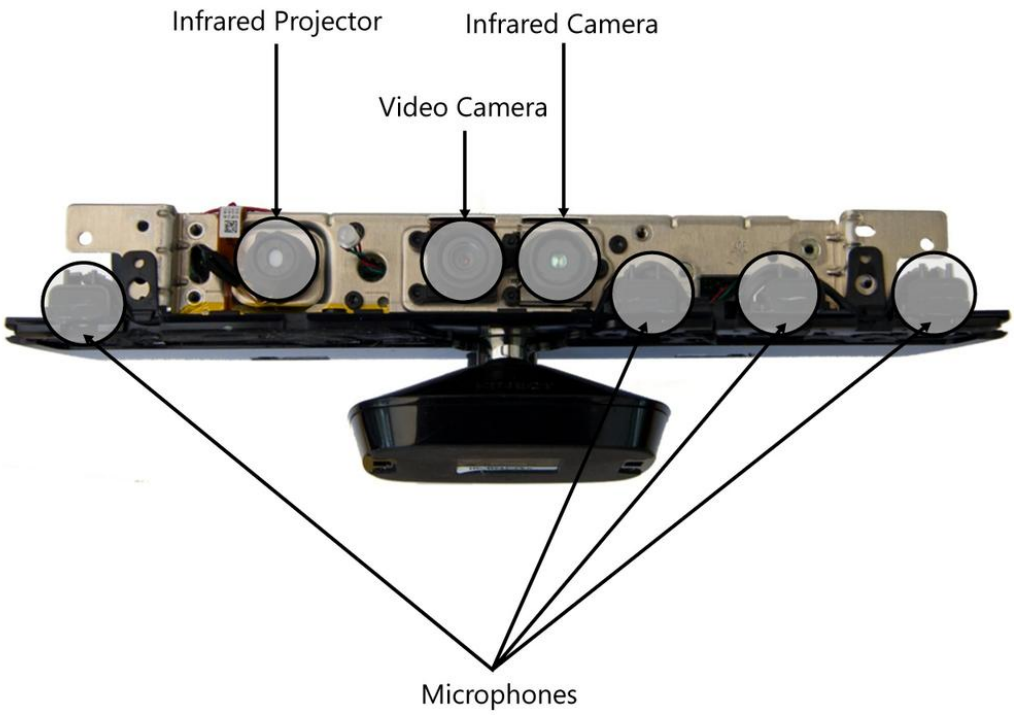
Senzor Kinect je sofistikované zařízení, které způsobilo revoluci v interakci počítačů s okolím. Obsahuje řadu čidel a zpracovávajícího hardwaru. Tyto nástroje umožňují snímat a interpretovat data z okolí v reálném čase. [5]

Obrázek 3.2 ukazuje senzor Kinect se sejmutým krytem. Uprostřed jsou vidět dvě kamery. Na levé straně se nachází speciální zdroj světla, který produkuje infračervené záření. Čtyři mikrofony jsou rozmístěny podél spodní části snímače. Všechny tyto komponenty společně zajišťují „pohled“, který má Kinect na svět před sebou. [6]

3.3.1 Popis jednotlivých částí

3.3.1.1 Hloubková kamera

Zařízení Kinect je založeno především na technologii snímání hloubky, která se skládá z infračervené kamery a infračerveného zářiče umístěného v určité vzdálenosti mezi nimi. Principem snímání hloubky je vysílání předem definovaného obrazce infračerveným zářičem a snímání jeho odraženého obrazu, který je deformován fyzickými objekty, pomocí infračervené kamery. Procesor uvnitř senzoru následně porovnává původní vzor a jeho deformovaný (odražený) obraz a určuje hloubku na základě rozdílů mezi oběma vzory. Výsledný hloubkový obraz má horizontální rozlišení 640 pixelů a vertikální rozlišení 480 pixelů. Lze snímat objekty,



Obrázek 3.2: Rozložený senzor Kinect [6]

které jsou ve vzdálenosti až osmi metrů a naměřené hodnoty jsou udávány v milimetrech. [8, 5]

3.3.1.2 Barevná kamera

Zařízení je navíc vybaveno barevnou (RGB) kamerou s rozlišením až 1280 × 960 pixelů, kterou lze použít jako další zdroj dat pro rozpoznávání. [8, 5]

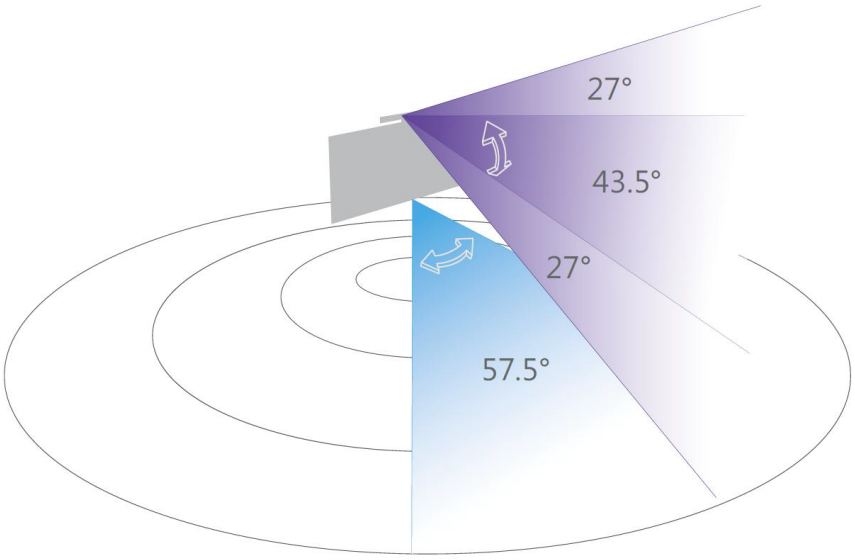
3.3.1.3 Mikrofony

Zařízení Kinect má čtyři mikrofony umístěné ve spodní části lišty, které jsou vidět na obrázku 3.2. Dva z těchto mikrofonů jsou umístěny na každém konci lišty, zatímco další dva jsou umístěny na pravé straně zařízení. Tyto mikrofony využívá Kinect k identifikaci zdroje hlasu v místnosti a to s ohledem na dobu, kterou zvuk potřebuje k šíření vzduchem. [5, 13]

3.3.1.4 Zorné pole

Senzor pracuje podobně jako kamera, a proto má omezený pohled na prostor před sebou. Tato pozorovatelná oblast se v souvislosti s kamerami a snímači označuje jako zorné pole (anglicky Field of View, zkráceně FOV) [14].

Senzor zpracovává 57.5 stupňů v horizontální ose a 43.5 stupňů v ose vertikální. Disponuje ale motorizovanou hlavicí, kterou lze posouvat kamerou o 27 stupňů na každou stranu a tím docílit toho, že snímač bude „koukat“ na správné místo v prostoru. Na obrázku 3.3 je předvedeno celkové zorné pole senzoru. [8, 5]



Obrázek 3.3: Zorné pole senzoru Kinect [15]

3.3.2 Detailní popis hloubkového senzoru

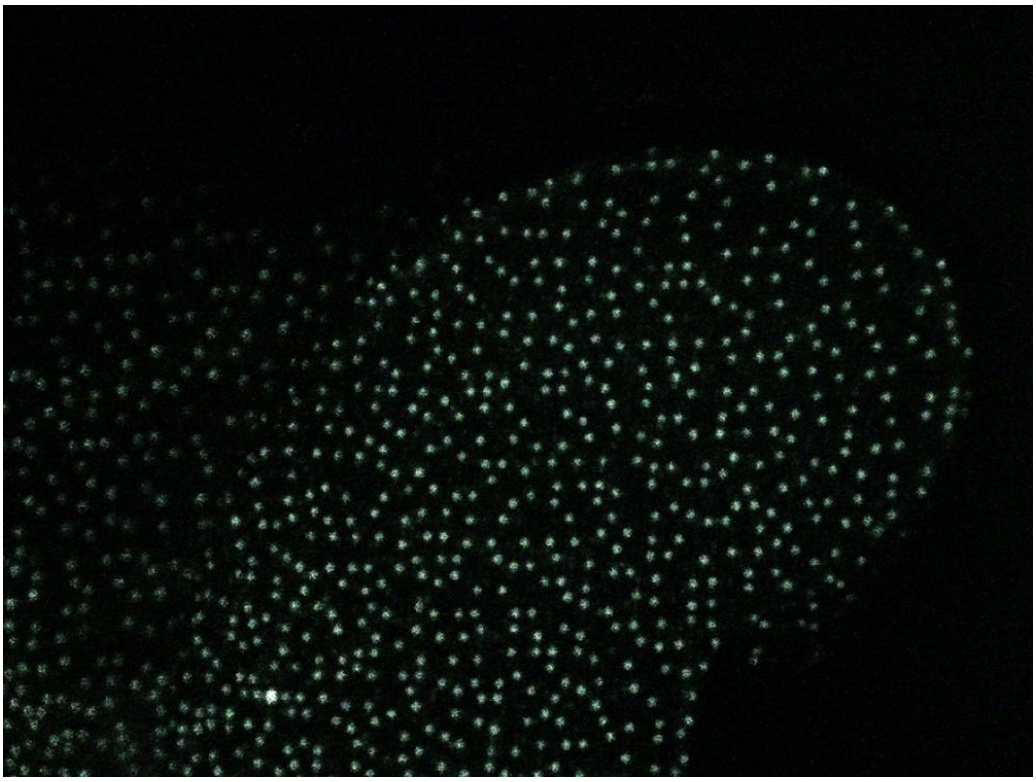
Kinect má jedinečnou schopnost vnímat a sledovat objekty ve třech rozměrech. Na rozdíl od většiny ostatních systémů počítačového vidění je systém Kinect schopen vytvářet hloubkovou mapu okolního prostoru. Tato mapa je vytvořena výhradně v liště snímače a poté je odeslána po kabelu typu USB (anglicky Universal Serial Hub) do hostitelského počítače stejně jako typický obraz z kamery. Namísto přenosu barevných informací pro každý pixel však snímač odesílá hodnoty vzdálenosti. Senzor využívá důmyslnou techniku, která využívá infračervený projektor a kameru. Projektor vyzařuje vhodně navržený vzor infračerveného světla. Vyzařované světlo dokáže zachytit infračervená kamera a ze získaných dat vypočítat vzdálenost od daného bodu v prostoru. [6]

3.3.2.1 Tvorba hloubkové mapy

Tato sekce popisuje podrobnější popis toho, jak senzor Kinect dokáže měřit vzdálenost od objektů, které se před ním nacházejí. Na obrázku 3.4 lze vidět scénu, ve které se nachází senzor Kinect. Ve scéně se nachází pouze jeden gauč, na který je namířena kamera (tj. Kinect). Pokud by se v místnosti kompletně zatemnilo a byla by

využita skoro jakákoliv kamera, která má funkci *Noční vidění*, zjistilo by se, že Kinect opravdu generuje pseudonáhodný vzor bodů, které následně dokáže rozpoznávat. [16] Tento vzor je detailněji ukázán na obrázku 3.5.

Obrázek 3.4: Ukázky projekce infračerveného vzoru v pokoji [6]

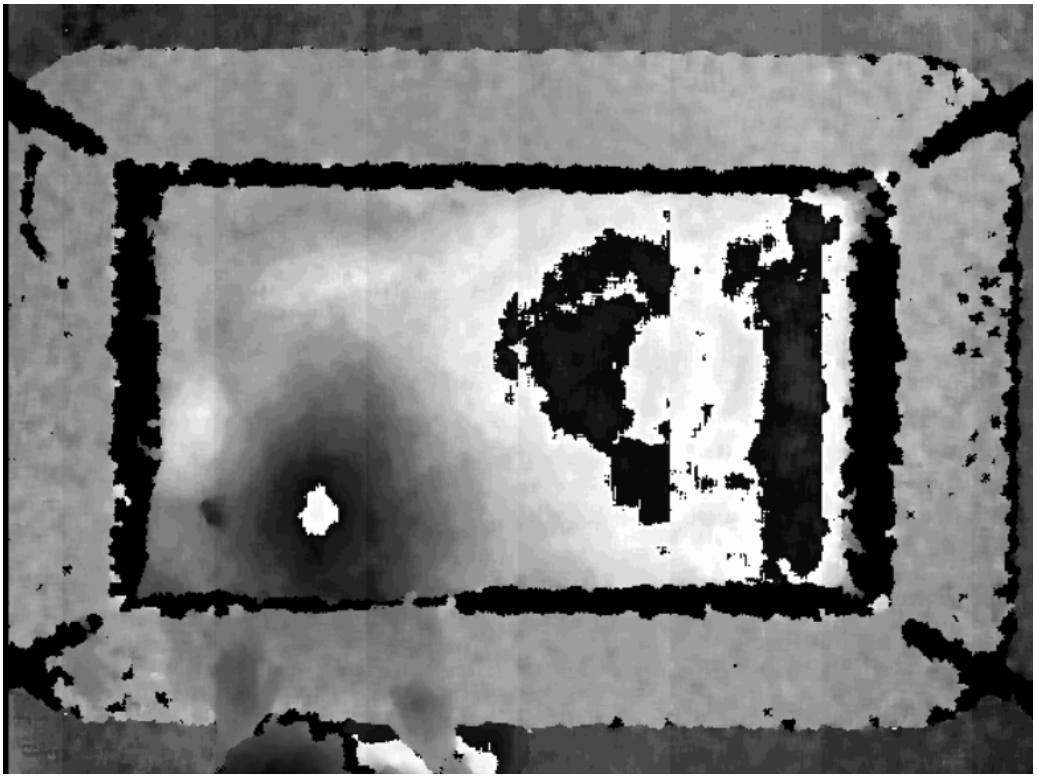


Obrázek 3.5: Ukázka pseudonáhodného vzoru infračerveného světla [6]

Jelikož Kinect zná strukturu promítaného vzoru, dokáže následně zjistit, jak daleko se od snímače nachází daný bod. Díky tomu, že snímač obsahuje filtr proti obyčejnému světlu, není problém se zachycením infračerveného světla i ve velice světlé místnosti. [6]

3.3.2.2 Problémy naměřených dat

Hloubková mapa s naměřenými hodnotami podléhá několika různým problémům. Mezi ně lze zařadit velký šum, vertikální čáry nebo například problém s lesklými plochami. Na obrázku 3.4 lze vidět ukázku toho, jak vypadá hloubková mapa, ve které je možné pozorovat popsané problémy. Zachycená hloubková mapa pochází přímo ze systému SandyStation.



Obrázek 3.6: Hloubková mapa

Pro odstranění či redukci těchto problemů bude nutné použít některé metody pro filtrování obrazových dat. Zároveň by bylo vhodné, aby bylo možné filtry různě kombinovat a zaměňovat.

Virtuální realita 4

V této kapitole představím pojem virtuální realita, nástroje používané pro vývoj aplikací pro virtuální realitu a zařízení, které se pro zobrazování virtuální reality používají.

4.1 Co je virtuální realita?

Virtuální realita je termín, který popisuje trojrozměrný prostor generovaný, nebo vytvořený počítačovým programem. Virtuální prostředí může člověk zkoumat a interagovat s ním prostřednictvím různých zařízení a počítačových technologií, jako je například náhlavní souprava, všesměrové běžecké pásy nebo speciální rukavice.

Virtuální realita má potenciální zábavní hodnotu a mnoho vytvořených aplikací pro různé oblasti (např. architektura, sport, medicína, herní průmysl, atd.). Mezi základní vlastnosti virtuální reality patří trojrozměrné objekty v životní velikosti, které se mění podle toho, jak se osoba v prostoru pohybuje. Jedná se o jednu z přirozených forem interakce člověka a k tomu odpovídající reakce na základě jeho chování. [17, 18]

4.2 Typy virtuální reality

Aplikace, které vytvářejí virtuální realitu, by se daly rozdělit do tří základních skupin, z pohledu úrovně ponoření (tzv. imerze). Těchto skupin existuje více, zde budou uvedeny pouze tři nejčastěji zmiňované.

4.2.1 Pohlcující (Immersive)

Pohlcující zážitek ve VR vyžaduje tři základní komponenty. Zaprvé počítačem vytvořené prostředí, které je komplexní a propracované. Zadruhé technické zařízení, které je schopno sledovat naše pohyby a podle toho přizpůsobovat zážitek a reagovat na pohyby uživatele v programu. A zatřetí hardwarové příslušenství jako je integrovaný displej v brýlích nasazených na hlavě, stereofonní zvuk a senzorické

rukavice, které společně vytvářejí plně pohlcující virtuální svět a prohlubují uživatelský zážitek. [19, 20]

4.2.2 Rozšiřující (Augmented)

Rozšiřující realita (anglicky Augmented Reality, zkráceně AR) kombinuje prvky reálného a virtuálního světa. Pro tento typ je důležitá kamera, která snímá obraz ze skutečného světa. Následně tento obraz vykreslí a doplní ho o virtuální prvky a objekty, které se vyskytují pouze ve virtuálním světě. [20, 21]

4.2.3 Nepohlcující (Non-immersive)

Nepohlcující (někdy též zvaná - *jednoduchá*) realita je zjednodušenou verzí pohlcující reality. Zde se namísto komplexních zařízení používá například obyčejný monitor připojený k počítači. Pro pohyb ve virtuálním prostoru se používají klasické periferní zařízení (například klávesnice a myš). V dnešní době se jedná o většinu vytvořených počítačových her. [20, 19]

4.3 Nástroje pro tvorbu aplikací pro VR

Pro tvorbu VR aplikací je zapotřebí několik nástrojů a zařízení. Mezi ně řadíme herní engine, který napomůže s celkovým vývojem aplikace a zároveň zařízení, které dokáže zobrazovat prostředí virtuální reality.

4.3.1 Herní engine

Základní koncepce herních enginů spočívá v tom, že poskytují výkonnou sadu nástrojů, které se starají o většinu základních funkcionalit při vývoji her a umožňují vývojářům soustředit se převážně na estetiku a hratelnost. Jedná se především o zpracování vstupu od uživatele, vykreslování scény či kolize mezi jednotlivými objekty. V počátcích vývoje počítačových her se každá hra vytvářela od nuly (bez knihoven) a možnosti využití herního enginu. V důsledku toho, že se pro každou hru muselo začít kompletně od začátku, začaly vznikat herní enginy, které usnadnily tvorbu her herním studiím. Herní enginy nejsou univerzální softwarové sady, které by dokázaly vytvořit jakoukoli myslitelnou hru. Jsou vysoce specializované, ale zároveň jsou velmi flexibilní. [22, 23]

V dnešní době existuje nespočet herních enginů, které lze využít pro vývoj VR aplikace. Mezi ty nejznámější a nejpoužívanější se řadí herní engine **Unity** a **Unreal Engine**.

4.3.1.1 Unity

Unity je herní engine vytvořený společností *Unity Technologies*. Je dostupný na velkém množství platforem a umožňuje vytváření her nejen pro osobní počítače, ale také konzole a mobilní zařízení. Jelikož se jedná o jeden z nejpoužívanějších herních enginů, existuje velká komunitní základna a detailní programátorská dokumentace na oficiální stránkách produktu. Pro vytváření konkrétní aplikace lze využít programovací jazyk `C#`. [24, 25, 26]

4.3.1.2 Unreal Engine

Herní engine Unreal od společnosti *Epic Games* vznikl před téměř 20 lety jako střílečka z pohledu první osoby (anglicky First Person Shooter, zkráceně FPS). Od té doby se herní engine Unreal výrazně vyvinul a nyní je volně dostupný. V herním enginu Unreal můžete vyvíjet pomocí jazyka `C++` nebo vizuálního skriptovacího systému `Blueprints`, který usnadňuje vytváření her pro začínající vývojáře. [22]

4.3.1.3 Zhodnocení a volba nástroje

Pro vytváření VR aplikace a reimplementaci projektu bude nadále uvažován herní engine `Unity` (konkrétně se bude jednat o verzi `2021.3.9f1` a `2022.2.11f1`) a objektově orientovaný jazyk `C#`.

Důvodem je, že herní enginy Unity používá objektový jazyk `C#`, který bude použit i pro implementaci ostatních částí aplikace. Zároveň je tento jazyk použit i ve vývojové sadě Kinect SDK. Bude tedy možné použít pouze 1 programovací jazyk pro všechny vytvořené aplikace a zároveň přesouvat různé funkce mezi jednotlivými aplikacemi bez větších potíží. Pokud by byl použit Unreal Engine bylo by nutné převést danou metodu z jazyka `C++` do jazyka `C#`. Dalším důvodem může být předchozí zkušenost a znalost obou technologií a zároveň větší komunita.

4.3.2 Pohyb ve VR

Lokomoce ve VR umožňuje pohyb ve virtuálním prostředí, kterého lze dosáhnout různými metodami. Mezi příklady lokomoce ve VR patří umělý pohyb, teleportace nebo všesměrové běhání. [27]

Umělá lokomoce zahrnuje použití ovladačů k navigaci v prostředí. Nevýhodou tohoto přístupu je, že může způsobit tzv. *VR sickness* (nemoc z VR). Jedná se o rozdíl mezi tím, co uživatel vidí, a tím, co detekují systémy související s pohybem ve vnitřním uchu. [27]

Oproti tomu může být použita teleportace. Teleportace je technika používaná v některých VR programech, která umožňuje uživatelům pohybovat se ve virtuálním prostředí, aniž by museli fyzicky chodit nebo používat joystick simulující chůzi.

Místo toho se uživatelé mohou teleportovat na dané místo výběrem cílového bodu ve VR prostředí a stisknutím tlačítka na ovladači. [27]

Nejpropracovanější technologií jsou všesměrové běžecké pásy, které vytvářejí iluzi přirozeného pohybu v omezeném prostoru a přesměrovaná chůze umožňuje volný pohyb v rámci vyčištěných hranic prostoru. [27]

4.3.2.1 Výběr řešení

Pro vývoj VR aplikace bude využita lokomoce typu **teleportace**. Hlavním důvodem je, že všesměrové pásy jsou aktuálně velice nedostupné a zároveň nechceme uživateli navozovat pocit nemoci, kterou způsobuje umělý pohyb.

Zpracování obrazových dat

5

Kapitola slouží pro porozumění toho, které operace bude nutné implementovat z důvodu přesného promítání výsledné scény na plochu pískoviště. Zároveň bude potřeba prověřit možné alternativy daných řešení, které by se ke konkrétnímu účelu daly použít a případně odůvodnit výběr řešení. Pro představu lze vycházet z kapitoly 2 (viz. strana 7) a z obrázku 2.1 a 2.2 na straně 8.

5.1 Kalibrace

Z výše uvedených informací je zřejmé, že je nutné kalibrovat okraje promítané scény, a to z důvodu zajištění správného zobrazení na pískovišti. Tato operace je nutná, protože samotný senzor Kinect má velké zorné pole (viz. sekce 3.3.1.4 na straně 18) a tudíž může zachytit i okolí samotného pískoviště. To by mohlo způsobovat zobrazování zbytečných informací na pískovišti a zároveň by zobrazované prvky nebyly zarovnané na správných místech v pískovišti (např. na kopci v pískovišti by se zobrazovala voda).

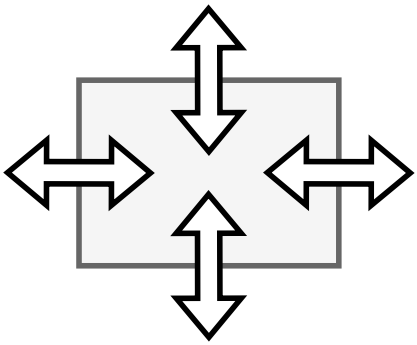
5.1.1 Možná řešení

5.1.1.1 Posouvání okrajů scény

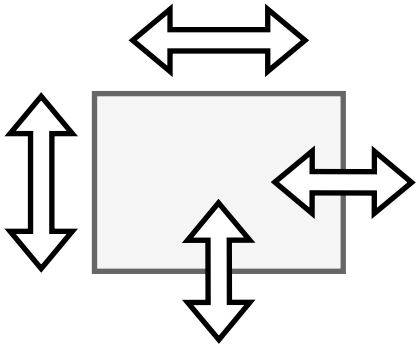
Jedním z řešení, které lze použít pro kalibraci je povolit uživateli pohyb všech 4 okrajů herní scény. Tímto způsobem bychom mohli docílit i posunu scény bez potřeby roztahování či stlačování výsledného obrázku. Stačilo by posunout protilehlé okraje o stejnou hodnotu. Názorná ukázka je zobrazena na obrázku 5.1.

5.1.1.2 Posouvání scény a okrajů

Druhé řešení vychází z předchozí varianty. Namísto manipulace se všemi čtyřmi okraji, by bylo povoleno operovat pouze se dvěma (sousedními) okraji. Druhá operace, kterou by uživatel mohl využít, by bylo posouvání celé scény v dvojrozměrném prostoru. Tento přístup je ukázán na obrázku 5.2.



Obrázek 5.1: Kalibrace všech okrajů



Obrázek 5.2: Kalibrace pomocí okrajů a posunu scény

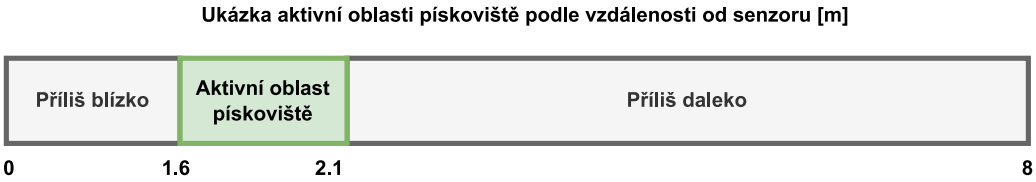
5.1.2 Výběr řešení

Jelikož jsou si obě řešení velice podobná, bude záležet na konkrétní implementaci, které řešení bude zvoleno.

5.2 Filtrace

Jelikož má senzor Kinect schopnost snímat objekty ve velkém rozsahu, bude potřeba využít i filtraci, která bude filtrovat objekty, které jsou příliš blízko senzoru (např. ruce člověka, který upravuje pískoviště) nebo naopak dále. Je tedy nutné definovat spodní a horní hranici, pomocí které budou filtrována získaná data ze snímače.

Pro filtraci bylo provedeno měření přímo na KIVu a to pomocí programu, který dokáže měřit hloubku konkrétního bodu. Výsledek měření a zároveň aktivní plocha celého pískoviště je nastíněna na obrázku 5.3.



Obrázek 5.3: Aktivní plocha pískoviště na KIVu

Naměřené hodnoty budou definovat rozmezí hodnot, které se budou vizualizovat. Tudíž spodní/horní hranice určuje nejnižší/nejvyšší místo v pískovišti. Vhodným řešením by bylo dovolit uživateli, aby tento rozsah hodnot mohl dynamicky měnit a výsledné nastavení se ukládalo (např. do souboru), aby se nemuselo pískoviště nastavovat po každém spuštění.

Architektura 6

Tato kapitola bude sloužit pro popis navržené architektury. Nejdříve budou uvedeny problémy stávající architektury. Poté zde budou nastíněny výhody a nevýhody navržené architektury a soupis některých potíží, které by mohla obsahovat.

6.1 Problémy projektu SandyStation

Architektura systému SandyStation se potýká s několika problémy, které znemožňují další rozvíjení a použitelnost aplikace.

Prvním problémem, který trápí architekturu systému SandyStation, je vysoká složitost přidávání nových funkcí a aplikací, které využívají stejná data ze snímače. V aktuální architektuře probíhají všechny operace (tj. získávání dat ze senzoru Kinect, jejich zpracování, kalibrace, filtrace a vykreslování) v rámci jedné jediné aplikace. Tento přístup není vhodný, protože následné rozšiřování systému je složité a není jednoduché přidat jinou aplikaci, která by mohla se systémem nějakým způsobem komunikovat. Jelikož chceme vytvořit VR aplikaci, která uživateli dovoluje zkoumat okolí pískoviště ve VR, je třeba tento problém odstranit a sdílet získaná data z Kinectu.

Dalším potencionálním problém je, že pokud by došlo ke ztrátě zdrojových souborů aplikace, bylo by obtížné (velmi pravděpodobně nemožné) přidávat další hry, bez větší znalosti systému. Zde je důvodem naprostá absence jakékoliv dokumentace.

Všechny tyto nedostatky je potřeba opravit, abychom mohli vytvořit VR aplikaci, která bude s novým systémem komunikovat. Základem nové architektury bude architektonický model zvaný **klient-server**, který je pro tyto účely vhodný.

6.2 Co je architektura klient-server?

Pojem **klient-server** je definovaný jako architektonický model, který se využívá, existuje-li větší množství klientů, kteří žádají centralizovaný server o služby a přijímají od něj odpovídající odpovědi. Klientské počítače poskytují rozhraní pro vyžá-

dání služeb a zobrazení výsledků, zatímco servery čekají na požadavky a odpovídají na ně. Tento model je zvláště efektivní, pokud mají klienti a servery odlišné úkoly. K informacím na serveru může přistupovat více klientů současně. Zároveň klientské počítače mohou simultánně vykonávat jiné úlohy. [28, 29]

6.3 Popis navržené architektury

Z výše uvedené definice lze navrhnout architekturu, která opraví nedostatky původní architektury. Cílem navržené architektury je nejen zlepšit modulárnost celé aplikace, ale celkově zjednodušit rozšiřitelnost a použitelnost celého programu. Proto byla navržena vhodnější architektura, která by tyto vlastnosti měla splňovat a byla dále rozšiřitelná. Celá aplikace bude využívat koncept architektury zvaný *klient-server* (více informací viz. sekce 6.2 na straně 29).

Původní program bude rozdělen na dvě oddělené aplikace - server a klient. Hlavní činností serveru bude starat se o zpracování dat ze senzoru Kinect a správu klientů. Klientské aplikace budou přijímat již zpracovaná data ze serveru. Tato data následně pouze zobrazí či provede jiné operace dle specifikace příslušné aplikace. Navržená architektura z pohledu propojení jednotlivých aplikací je zobrazena na obrázku 6.1, který se nachází na straně 32.

6.3.1 Server

Server bude mít na starosti dvě hlavní funkce. První činností serveru bude získávání dat ze senzoru Kinect. Server nejdříve získá informace o vzdálenostech ze snímače a hloubková data zpracuje podle požadovaných vlastností. Zde je důležité umožnit uživatelům (správcům) serveru rychlou a jednoduchou záměnu/úpravu funkcionality, případně přidání nových modulů, které budou chtít zpracovávat, a nebo jiným způsobem zacházet s naměřenými hodnotami. K tomu bude sloužit tzv. *pluginový systém* (více informací viz. níže).

Druhým úkolem je umožnit připojení klientským aplikacím, kterým budou následně odesílána zpracovaná data. Velmi častým řešením tohoto problému je přenos dat pomocí internetových protokolů. Zároveň by mohlo být vhodné, aby se klientská aplikace nejdříve přihlásila k příjmu dat a to z důvodu, když by některé aplikace pouze komunikovali se serverem. Tedy by nedostávali informace o vzhledu pískoviště.

6.3.1.1 Pluginový systém

Architektura serveru bude v podstatě jeden velký pluginový systém, který výrazně zlepší použitelnost a rozšiřitelnost aplikace.

Pojem **plugin** (někdy též *zásuvný modul*) je druh programu, který nedokáže fungovat samostatně. Ovšem může být přidán do „základní“ aplikace, která s ním umí pracovat a může ho začít používat. Rozšířená verze programu s pluginem dokáže provádět více funkcí než verze nerozšířená [30]. Výhodou je, že se dá měnit funkcionalita serveru, a to i bez potřeby rekompilace celého systému. Stačí pouze vytvořit nový plugin (resp. třídu, metodu, knihovnu), se kterým bude server schopen pracovat. Detailní informace o implementaci pluginového systému, jsou popsány v sekci 7.3, která se vyskytuje na straně 41. Příkladem pluginového systému mohou být různé internetové prohlížeče nebo vývojová prostředí (anglicky Integrated Development Environment, zkráceně IDE).

Pluginový systém výrazně zpřístupní rozšiřování celé aplikace, protože se bude skládat právě z několika pluginů, které jdou velice jednoduše zaměnit, protože deklarace jednotlivých metod jsou stejné.

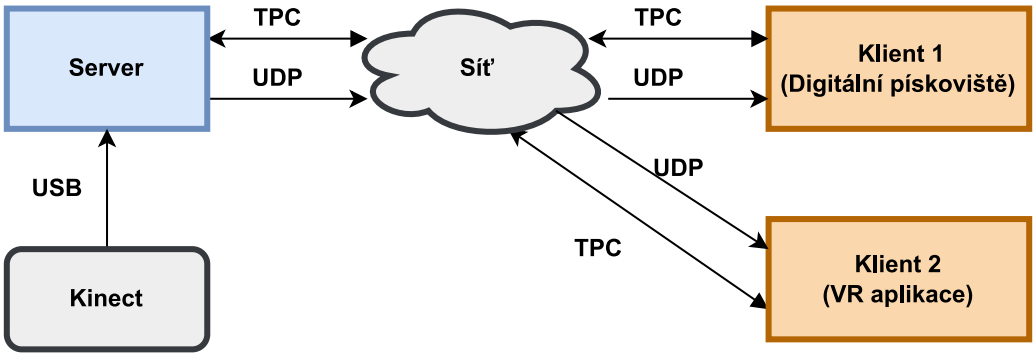
6.3.2 Klient

Po spuštění se klient pokusí připojit přes internetovou síť k serveru a zaregistruje se k odběru hloubkových dat. Poté, co proběhne úspěšné připojení, klient začne přijímat zprávy, které obsahují zpracovaná a vyfiltrovaná data. Z pohledu klientské aplikace bude zdrojem dat server.

Klient následně může tato data využít k dalšímu zpracování či vizualizaci. Může se jednat právě o instanci původní aplikace či VR aplikaci, která zobrazuje pískoviště v prostředí virtuální reality. Zároveň to přidává možnost do budoucna dále rozšiřovat tento systém, k čemuž už nebude nutné upravovat zásadním způsobem aplikaci, která zde reprezentuje server.

6.4 Propojení zařízení a programů

Jelikož je aplikace rozdělena do několika programů a zařízení, které spolu potřebují komunikovat, je nutné k tomu použít příslušné technologie a standardy. Jak již bylo zmíněno, celý systém bude sestaven z centrálního serveru, který bude spuštěn na počítači, který je umístěný v grafické učebně na KIVu (tj. na místě, kde je v době tvorby diplomové práce nainstalovaný program SandyStation). K tomuto počítači je připojený senzor Kinect, který sleduje pískoviště a jeho okolí. Snímá vzdálenost jednotlivých bodů, které následně odesílá do připojeného PC. Následně se k serveru budou moci připojovat i další aplikace a to pomocí internetových protokolů. Propojení jednotlivých aplikací lze pozorovat na obrázku 6.1.



Obrázek 6.1: Architektura z hlediska propojení jednotlivých aplikací

6.4.1 Propojení počítače a senzoru Kinect

Pro připojení senzoru Kinect k počítači, je potřeba využít speciální adaptér, který slouží pro napájení Kinectu a zároveň pro přenos z výpočetní jednotky snímače do hostitelského počítače. Tento způsob je využit i ve stávajícím projektu, který je aktuálně umístěn na KIVu. Adaptér si lze prohlédnout na obrázku 6.2.

Obrázek 6.2: Kinect adaptér s napájením pro Xbox 360 a PC [31]

6.4.2 Propojení serveru a klientů

Jak již bylo zmíněno výše, k propojení aplikací budou využity internetové protokoly, které jsou pro to určené. Konkrétně se bude jednat o protokoly ze čtvrté (transportní) vrstvy referenčního modelu *ISO/OSI*, tedy protokol **User Datagram Protocol** (zkráceně UDP) a **Transmission Control Protocol** (zkráceně TCP) [32].

6.4.2.1 Transmission Control Protocol

Protokol TCP byl speciálně vytvořen tak, aby poskytoval spolehlivý přenos dat mezi koncovými body přes nespolehlivou internetovou síť. TCP byl navržen tak, aby se dynamicky přizpůsoboval vlastnostem internetové sítě a byl odolný vůči mnoha druhům selhání. Další vlastností je, že při ztrátě paketu dochází k opětovnému odeslání. [32]

Tento protokol bude použit pro spolehlivou komunikaci mezi serverem a ostatními klienty, kteří budou se serverem komunikat. Důvodem je právě spolehlivost protokolu a také fakt, že v podstatě nedochází ke ztrátě paketů.

6.4.2.2 User Datagram Protocol

Sada internetových protokolů podporuje přenosový protokol bez spojení, který se nazývá UDP. Protokol UDP umožňuje aplikacím odesílat zapouzdřené datagramy bez nutnosti navazovat spolehlivé spojení. Toto spojení je nespolehlivé a není zde zaručené, že odeslaný paket bude v pořádku doručen. Pokud dojde ke ztrátě paketu, není znovu odeslán a tím pádem je ztracen navždy [32].

6.4.2.3 Výhody a nevýhody obou protokolů

Výhodou protokolu TCP je, že vytváří spolehlivé spojení, ve kterém dochází k potvrzování přijetí paketů a k jejich opětovnému odeslání. Umožňuje obousměrnou komunikaci. Přesně tyto vlastnosti jsou vhodné pro komunikaci mezi klientem a serverem.

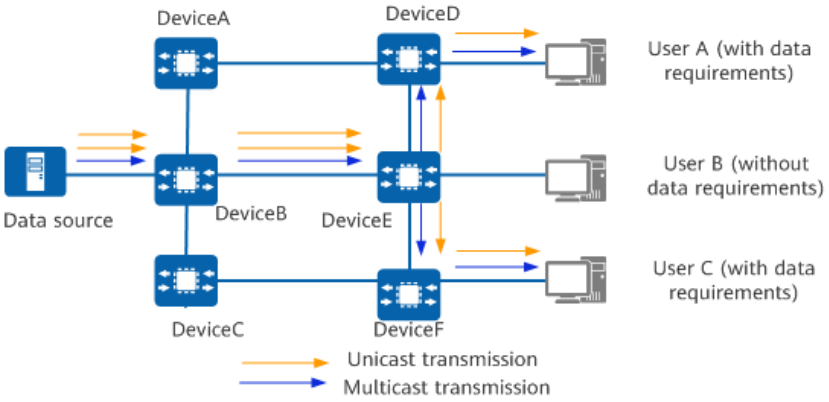
V některých případech jsou tyto vlastnosti nechtěné a proto je použit UDP protokol. Tento protokol nevytváří trvalé spojení a podporuje komunikaci pouze jedním směrem. Zároveň se nekontroluje, zdali byl paket v pořádku doručen. Pokud se tedy nějaký paket ztratí, nebude nikdy doručen. Tento protokol se využívá hlavně v real-time komunikaci, streamování videa či online her. Proto bude tento protokol použit pro odesílání naměřených dat ze serveru ostatním klientům.

6.4.2.4 Multicast vs. Unicast

Jelikož se k serveru bude s velkou pravděpodobností připojovat několik klientů, bylo by vhodné tomu uzpůsobit i práci s komunikační vrstvou aplikace. V rámci protokolu UDP se definují pojmy multicast a unicast. **Unicast** je typ přenosu informací, který je užitečný v případě, že se na něm podílí pouze jeden odesílatel a jeden příjemce. Při použití typu **multicast** se na přenosu dat podílí jeden odesílatel a více příjemců nebo naopak. [33, 34]

Oba přenosy jsou si velice podobné. Lze použít několik přenosů typu unicast a docílit tím stejného výsledku jako s přenosem typu multicast. Rozdíl je v tom,

že multicast využívá sofistikovanější způsob odesílání paketů. Místo toho, aby vytvářel nový paket pro každého klienta, je odeslán pouze jeden paket. Na vhodném přístupovém bodu jsou vytvořeny kopie paketu, které jsou následně odeslány jednotlivým klientům. Z toho vyplývá, že snižuje celkovou zátěž internetové sítě. Srovnání režimů přenosu paketů lze vidět na obrázku 6.3. [34]



Obrázek 6.3: Srovnání režimů přenosu multicast a unicast [34]

Navzdory všem výhodám, které přináší multicast, je zvolen typ přenosu **unicast**. Příčinou je bezpečností riziko, které je zabezpečeno Centrem informatizace a výpočetní techniky (zkráceně CIV), které se nachází na půdě Západočeské univerzity v Plzni.

Server 7

Jak již bylo zmíněno, server zde bude sloužit převážně pro distribuci hloubkových dat a zároveň pro jejich předzpracování a filtraci. V této sekci se pokusím popsat všechny základní informace o tom, jak všechny části serveru fungují, komunikují a jak byly vytvořeny. Zároveň zde podrobně vysvětlím činnost pluginového systému a jednotlivých funkcí, které jsou s tím spojené. Současně budou popsány i použité technologie a různé problémy spojené s vývojem.

7.1 Použité technologie

Pro implementaci serveru bude použit moderní objektově orientovaný programovací jazyk `C#`. Tento jazyk bude použit ve spolupráci s technologií `.NET Framework`. Konkrétně se bude jednat o jednu z nejnovějších verzí - `4.8`. Hlavním důvodem pro volbu této verze je použití sady `Kinect for Windows SDK 1.8`, která způsobuje potíže při kombinaci s novějšími verzemi (tj. s `.NET Core`), které byli původně plánované pro vývoj.

Pro serializaci jednotlivých tříd do formátu `JSON` a zpět, byla využita knihovna zvaná `Newtonsoft.Json`, která je velmi populární a hodně používaná. Konkrétně se jedná o aktuálně nejnovější verzi - `13.0.3`. [35]

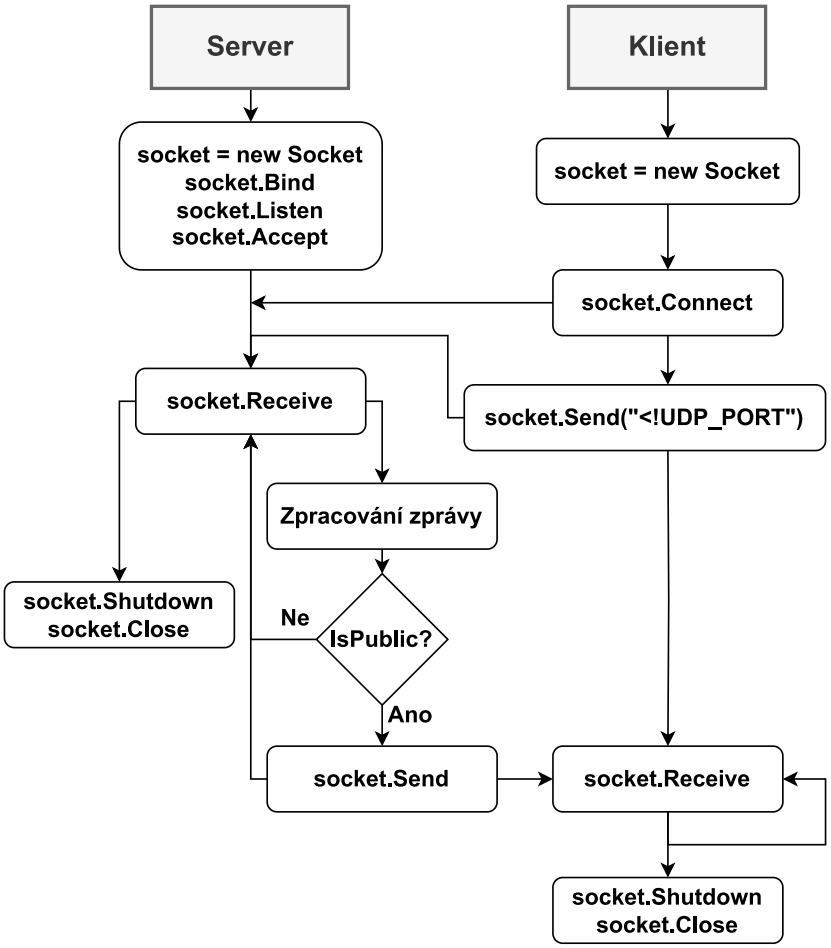
7.2 Komunikace

Protože jsou v navržené architektuře použity oba základní protokoly transportní vrstvy, je potřeba k tomu správně přistoupit i v implementaci dané aplikace.

Jak již bylo zmíněno, protokol TCP bude použit pro spolehlivou komunikaci mezi serverem a klientem. Klient se tímto komunikačním kanálem bude registrovat k odběru hloubkových dat. Celkově se bude jednat o kanál, který nebude natolik vytížen (neočekává se vysoký počet zpráv). Ukázku toho, jak bude vypadat celý proces TCP komunikace mezi serverem a klientem je zobrazen na obrázku 7.1.

Protokol UDP bude následně sloužit pouze pro odesílání zpracovaných dat připojeným a zaregistrovaným klientům. Zátěž toho kanálu bude vysoká, protože bude

potřeba posílat velké množství informací za krátkou jednotku času. Proto bude nutné zvolit i vhodné předzpracování dat před jejich odesláním do internetové sítě.



Obrázek 7.1: Sekvenční diagram komunikace mezi serverem a klientem

7.2.1 Navázání spojení s klienty

Název této kapitoly je trochu zavádějící, protože spojení se serverem je inicializováno ze strany klienta. Ovšem server je nucen zpřístupnit možnost připojení jednotlivým klientům. S tím je spojené vytvoření nového *soketu*, ke kterému bude přiděleno číslo dostupného TCP portu, na kterém bude server naslouchat a přijímat zprávy od klientů.

7.2.1.1 Vytvoření soketu

Sokety slouží (v počítačových sítích) k navázání spojení mezi dvěma nebo více počítači, a k odesílání dat z jednoho počítače do druhého. Každý počítač v síti se nazývá uzel. Sokety využívají IP adresy uzlů a síťový protokol k vytvoření zabezpečeného

komunikačního kanálu a používají jej k přenosu dat. Při komunikaci vystupuje jeden uzel jako *posluchač* a druhý uzel jako *klient*. Posluchač se naváže na předem stanovenou IP adresu a port daného protokolu. Nakonec začne na tomto přístupovém bodu odchytávat příchozí spojení. Klienti, kteří chtějí odesílat zprávy na server, začnou vysílat zprávy na identickou IP adresu a číslo portu, na kterém posluchač naslouchá. [36, 29]

V případě serveru se tedy nejdříve musí vytvořit soket, se kterým bude svázána IP adresa a konkrétní číslo TCP portu. Tyto informace budou sloužit všem klientům, kteří se budou chtít k serveru připojit. Vytvoření a nastavení soketu je velice jednoduché (viz. ukázka 7.1). Nejdříve se vytvoří instance třídy `Socket`, kde jsou použity 3 základní parametry, které definují typ daného soketu.

Výpis kódu 7.1: Vytvoření soketu na straně serveru

```
1 // Create the socket and bind it.
2 _serverSocket = new Socket(AddressFamily.InterNetwork,
      SocketType.Stream, ProtocolType.Tcp);
3 _serverSocket.Bind(new IPEndPoint(IPAddress.Any, port));
```

Použité parametry vypadají následovně:

- `AddressFamily` - určuje schéma adresování, které bude použito. Hodnota `AddressFamily.InterNetwork` značí, že bude využita IP adresa verze 4.
- `SocketType` - parametr nastaven na hodnotu `SocketType.Stream`. Tento typ soketu zajišťuje spolehlivý přenos dat oběma směry prostřednictvím spojení, které zabraňuje duplicitě dat. Je určen pro komunikaci s pouze jedním klientem a před zahájením komunikace vyžaduje připojení vzdáleného hostitele. [37]
- `ProtocolType` - typ protokolu, který bude použit pro komunikaci. Jelikož při komunikaci s klientem vyžadujeme, aby tento přenos byl spolehlivý a nedocházelo ke ztrátám jednotlivých paketů, je zde použit protokol TCP (tj. `ProtocolType.Tcp`).

Následně je zavolána metoda `Bind`, která soket „spojí“ s IP adresou a portem. V ukázce 7.1 se místo konkrétní IP adresy používá konstanta `IPAddress.Any`, která identifikuje, že server musí aktivně naslouchat na všech dostupných síťových rozhraních. Číslo portu je definované v rámci konfiguračního souboru `App.config`.

7.2.1.2 Naslouchání

Jelikož server potřebuje zpracovávat hloubková data a zároveň naslouchat a přijímat zprávy od klientů, je nutné, aby tyto dvě operace probíhali paralelně. Z toho důvodu

je potřeba, aby komunikace a vytváření nových spojení probíhaly v jiném vlákně. K tomu se v objektovém jazyce C# používá třída zvaná `Thread`. Vytvoření nového vlákna je velice jednoduché (viz. ukázka 7.2). Do konstruktoru třídy `Thread` je předán tzv. *delegát*. Delegát představuje referenci na metodu. Když dojde k vytvoření nové instance vlákna, stačí pouze zavolat metodu `Start`. Tím dojde ke spuštění nového vlákna, které začne vykonávat funkční kód přidělené metody.

Výpis kódu 7.2: Vytvoření nového vlákna

```
1 var thread = new Thread(ServerLifeLoop);
2 thread.Start();
```

Nakonec je nutné pouze přesunout stav soketu do režimu naslouchání. To je provedeno pomocí metody `Listen`, kterou poskytuje třída `Socket`.

Pak už je soket připraven schopný přijímat nová spojení a zároveň server pokračuje ve vykonávání dalšího kódu (tj. zpracování hloubkových dat).

7.2.1.3 Připojení nového klienta

Vlákno, které se stará o vytváření nových spojení, používá funkční kód z výpisu 7.3, který se nachází na straně 38. Server nejdříve zavolá metodu `Accept`. Metoda `Accept` čeká, dokud se některý z klientů nepokusí o připojení k serveru. Když se tak stane, dojde k vytvoření klientského soketu, který je následně přidán do seznamu všech klientských soketů (tj. seznam `_clientSockets`). Pro každé nové připojení je následně vytvořené další samostatné vlákno, které se stará o přijímání zpráv z nově vytvořeného klientského soketu.

Výpis kódu 7.3: Připojení nového klienta k serveru

```
1  while (isRunning)
2  {
3      Socket client = _serverSocket.Accept();
4      _clientSockets.Add(client);
5
6      Console.WriteLine($"New client connected from IP ->
           {client.RemoteEndPoint}");
7
8      // start a new thread to handle this client's messages
9      var t = new Thread(() => HandleClientMessages(client));
10     t.Start();
11 }
```

7.2.1.4 Formát TCP zpráv

Pro přeposílání zpráv bylo potřeba zvolit vhodný a univerzální formát jednotlivých zpráv. Formát, který bude univerzální pro všechny možné typy zpráv a zároveň

takový, aby je bylo možné od sebe jednoduše rozeznat. Přesně z toho důvodu byl vytvořen formát zpráv, který je zobrazen v ukázce 7.4.

Výpis kódu 7.4: Třída `TcpMessage`

```
1 public class TcpMessage
2 {
3     public string Code { get; set; }
4     public string Message { get; set; }
5     public string Footer { get; set; }
6     public bool IsPublic { get; set; }
7 }
```

Popis jednotlivých parametrů a záměr jejich vytvoření je následující:

- `Code` - identifikátor zprávy, který může být definovaný jako jakýkoliv řetězec (např. `<!UDP_PORT!>`)
- `Message` - hodnota udává obsah dané zprávy. Pokud by klient potřeboval odeslat složitější strukturu, stačí ji převést na řetězec, který se uloží do tohoto parametru.
- `Footer` - patička musí být vždy ve tvaru `<!EOF!>`. Díky tomu lze snadno odlišit, zdali je zpráva určena serveru.
- `IsPublic` - jestli má být zpráva přeposlána ostatním klientům. Z důvodu lepší rozšiřitelnosti byl vytvořen i tento parametr, který dodává možnost komunikace mezi jednotlivými klienty.

7.2.2 Zpracování přijatých TCP zpráv

Když je inicializováno spojení mezi serverem a klientem, dojde k vytvoření speciálního vlákna, které se stará o přijímání zpráv daného klienta. Pro získání dat, které klient odešle, se používá metoda `Receive` (viz. výpis 7.5). Po jejím zavolání začne vlákno čekat, dokud nepřijde nová zpráva od klienta. Obsah této zprávy je uložen do pole bajtů a návratovou hodnotou je počet přijatých bajtů. Díky tomu dokážeme snadno zjistit, zdali bylo přijetí zprávy úspěšné. Ukázka toho, jak může vypadat validní obsah této zprávy je předveden v ukázce 7.6

Výpis kódu 7.5: Ukázka použití metody `Receive`

```
1 var numReceivedBytes = client.Receive(messageBuffer);
```

Výpis kódu 7.6: TCP zpráva s kódem `UDP_PORT` ve formátu `JSON`

```
1 {
2     "Code":"<!UDP_PORT!>",
```

```
    "Message":"34342",
    "Footer":"<!EOF!>",
    "IsPublic":false
}
```

Pokud je zpráva přijata v pořádku, je převedena z pole bajtů do řetězce. Tento řetězec je pomocí tzv. deserializace transformován do instance třídy `TcpMessage`. Tuto část kódu lze vidět ve výpisu 7.7. Pokud dojde v této části k chybě (např. z důvodu špatného formátu příchozí zprávy), je vypsána chybová hláška, která informuje o této skutečnosti a vypíše základní informace do konzole. Příchozí zpráva je nakonec ignorována.

Výpis kódu 7.7: Deserializace příchozí zprávy na instanci třídy `TcpMessage`

```
try
{
    // Deserialization from JSON string into TcpMessage
       instance.
    message =
       JsonConvert.DeserializeObject<TcpMessage>(jsonMessage);
}
catch (Exception)
{
    Console.WriteLine($"<TCP> Error while parsing received
       message from {client.RemoteEndPoint} - {jsonMessage}");
    break;
}
```

V opačném případě, tj. když je zpráva úspěšně deserializována, se začne zpráva zpracovávat podle nastavených parametrů.

7.2.3 Registrace klienta k odběru hloubkových dat

Každý klient, který chce přijímat hloubková data ze serveru, se musí nejdříve přihlásit k jejich odběru, protože v některých případech by klient nemusel chtít přijímat hloubková data. Zároveň nestačí pouhé připojení klienta k serveru, protože bychom nezjistili, na kterém UDP portu klient očekává hloubková data.

Klient se nejdříve připojí k serveru pomocí TCP připojení. Po úspěšném vytvoření spojení mezi oběma stroji, klient odešle zprávu se speciálním kódem, která identifikuje žádost o odběr hloubkových dat. Obsahem této zprávy je UDP port, na kterém klient čeká a bude přijímat hloubková data. Formát této zprávy byl již představen v ukázce 7.6.

Poté, co server přijme zprávu s kódem `<!UDP_PORT!>`, je zavolána metoda s názvem `AddUDPClient`. Metoda se stará o přidání nového klienta, který se chce

zaregistrovat k odběru hloubkových dat. Z přijatého paketu jsou extrahovány základní informace o klientovi (tj. IP adresa a číslo UDP portu). Na základě těchto informací je vytvořena nová instance třídy `IPEndPoint`, která je následně přidána do struktury, ve které se nachází všichni klienti, kteří požadují příjem výsledných dat ze serveru. Obsah metody `AddUDPClient` je zobrazen v ukázce 7.8.

Výpis kódu 7.8: Obsah metody `AddUDPClient`

```
1 bool AddUDPClient(Socket socket, string port)
2 {
3     var remoteIpEndPoint = (IPEndPoint)socket.RemoteEndPoint;
4     var res = int.TryParse(port, out var udpPort);
5     if(!res) return false;
6
7     _udpSender.AddClient(new
          IPEndPoint(remoteIpEndPoint.Address, udpPort));
8     return true;
9 }
```

7.2.4 Přeposílání TCP zpráv

Pokud má přijatá zpráva nastavený parametr `IsPublic` na hodnotu `true`, znamená to, že klient požaduje, aby tato zpráva byla odeslána i ostatním klientům, kteří jsou k serveru připojeni. Tj. server musí projít všechny připojené klienty, kterým je tato zpráva přeposlána pomocí TCP soketů, které se vytvořili při inicializaci jednotlivých spojení.

7.3 Pluginový systém

Velká část serveru je sestavena z tzv. pluginového systému, aby bylo možné dále rozšiřovat aplikaci nezávisle na dané implementaci serveru. Z toho důvodu bylo nutné vytvořit několik šablon, které definují funkcionalitu daného pluginu.

7.3.1 Co je vlastně šablona?

Šablona (někdy též zvaná *rozhraní*) definuje metody, jejichž definice je známa, ale konkrétní implementace dané metody je definována až jednotlivými potomky, kteří implementují obsah dané metody/funkce. Rozhraní dává možnost záměny konkrétní implementace, a to dokonce i bez ovlivnění zbytku programu. Důvodem je, že deklarace jednotlivých metod zůstává stejná (tj. název, vstupní i výstupní parametry), protože se pracuje s daným typem jako s instancí třídy daného rozhraní. [38]

7.3.2 Typy pluginů

V rámci vytvořeného pluginového systému bylo potřeba vytvořit množinu pluginů. Pro identifikaci, jestli lze plugin využít v serveru je vytvořeno rozhraní `IPlugin`. Tedy každá třída, která reprezentuje plugin, musí implementovat toto rozhraní. Obsah celého rozhraní `IPlugin` lze vidět ve výpisu 7.9. Zároveň díky tomuto rozhraní můžeme získat základní informace o zásuvném modulu.

Výpis kódu 7.9: Rozhraní `IPlugin`

```
1 public interface IPlugin
2 {
3     string Name { get; }
4     string Description { get; }
5 }
```

Na základě předešlé definice, bylo vytvořeno celkem 5 šablon, které definují různé typy pluginů, které server používá převážně pro práci se získanými hloubkovými daty. Jak již bylo před chvíli zmíněno, všechny rozhraní dědí od šablony `IPlugin` a tudíž se jedná o konkrétní typy pluginů, které lze použít v aplikaci. Přesněji se jedná o následující typy.

1. `IDepthSource` - metody pro třídu, která pracuje jako zdroj hloubkových dat
   - musí být vždy použit pouze 1 plugin tohoto typu (lze jich definovat více)
2. `IProcessor` - metody pro analýzu získaných dat
3. `IFilter` - metody pro úpravu a filtraci hloubkových dat
4. `IConvertor` - metody pro převod hodnot
   - musí být vždy použit pouze 1 plugin tohoto typu (lze jich definovat více)
5. `ICompressor` - metody pro kompresi výsledných hodnot
   - vždy použit pouze 1 modul tohoto typu (použit pouze, pokud je definovaný)

7.3.3 Konfigurace pluginů

Aby mohl začít server využívat jednotlivé pluginy byl vytvořen způsob, díky kterému lze definovat, které pluginy budou použity. K tomu slouží konfigurační soubor s názvem `plugin.config.json`, jehož struktura je zobrazena v ukázce 7.10. V tomto souboru jsou uvedeny všechny názvy pluginů, které se po spuštění program musí načíst.

Výpis kódu 7.10: Struktura souboru `plugin.config.json`

```
{
  "IDepthSource": [
    {
      "Assembly": "MSKinect",
      "Name": "Kinect"
    },
  ],
  "IPreprocessing": [],
  "IFilters": [
    {
      "Assembly": "SandboxProcessing",
      "Name": "DepthFilter"
    }
  ],
  "IPostprocessing": [],
  "IDepthDataConvertor": [
    {
      "Assembly": "SandboxProcessing",
      "Name": "DepthConvertor"
    }
  ],
  "ICompressor": [
    {
      "Assembly": "SandboxProcessing",
      "Name": "GzipCompressor"
    }
  ]
}
```

7.3.4 Načítání pluginů

Po spuštění serveru se nejdříve musí načíst všechny dostupné a definované zásuvné moduly. Celé načtení má na starosti třída zvaná `PluginLoader`, která nejdříve načte všechny dostupné pluginy (tj. třídy, které dědí od rozhraní `IPlugin`). Tyto třídy hledáme v souborech zvaných **Assembly**. Jedná se o úspěšně přeložené a zkompilované soubory konkrétní aplikace či knihovny. Způsob, kterým lze z těchto souborů vyčíst, zdali obsahují validní plugin pro program `DigitalSandbox`, je zobrazen v ukázce 7.11. Pokud takový assembly soubor existuje, je uložen do struktury `Dictionary<string, Assembly>`. Jako klíč je zde použit název daného assembly.

Výpis kódu 7.11: Kontrola, zdali assembly obsahuje plugin

```
assembly.GetTypes().Where(t =>
    t.GetInterface("DigitalSandboxCore.IPlugin") != null).Any()
```

Následně jsou načtena data z konfiguračního souboru `plugin.config.json`. Jelikož jsou zde data uložena ve formátu `JSON`, je potřeba použít deserializaci a převést uložená data na konkrétní instanci objektu. K tomu je využita knihovna `Newtonsoft.Json`. Poté, co jsou načtené validní pluginy a konfigurace ze souboru, dochází k hledání a vytváření konkrétních instancí, které jsou definovány v konfiguračním souboru. Tento proces je velice přímočarý. Pro každý typ pluginu je nalezeno specifické assembly v již vytvořeném slovníku. V tomto assembly se následně hledá daná třída a kontroluje se, zdali se jedná o implementaci konkrétního typu pluginu. Pokud ano, je vytvořena nová instance této třídy a uložena do seznamu, který je návratovou hodnotou této metody. V opačném případě (když plugin není načten) se přejde na další definovaný plugin.

7.4 Získávání dat ze snímače Kinect

Pro získávání dat z Kinectu byla vytvořena třída `MSKinect`, která implementuje rozhraní `IDepthSource`, jehož popis můžete vidět v ukázce 7.12. Jelikož se jedná rovněž o implementaci třídy `IPlugin`, je možné tyto třídy libovolně zaměňovat bez potřeby kompilace. Tento přístup byl využit převážně při testování konkrétní implementace, kde jsem většinou neměl po ruce přímo zařízení Kinect a tudíž bylo potřeba vytvořit jiný zdroj hloubkových dat.

Výpis kódu 7.12: Rozhraní `IDepthSource`

```
1  public interface IDepthSource : IDisposable, IPlugin
2  {
3      event EventHandler<DepthFrameEventArgs> DepthFrameAvailable;
4      event EventHandler<EventArgs> OnDisconnected;
5
6      bool IsInitialized { get; }
7      bool Initialize();
8      void Uninitialize();
9      void GetDimensions(out int width, out int height);
10 }
```

Pro implementaci třídy `MSKinect` byl využit návrhový vzor zvaný **Pozorovatel** (anglicky *Observer*). Z definice tohoto návrhového vzoru vyplývá, že existují dva objekty - pozorovatel a pozorovaný. Pozorovatel reaguje na změnu stavu pozorovaného objektu nebo sledované události, kterou poskytuje pozorovaný objekt. Pozorovatel se nejdříve musí přihlásit k dané události u pozorovaného objektu a ten pak upozorní zaregistrované pozorovatele, pokud daná událost nastane. [38]

Při práci se senzorem Kinect, je nutné nejdříve zaregistrovat reakci na událost s názvem `DepthFrameReady`, která je součástí třídy `KinectSensor`. Třída `KinectSensor` je již součástí vývojové sady (SDK). Pokud tedy Kinect zachytí nový

snímek, který obsahuje hloubková data, vyvolá tuto událost a následně se zavolají všechny metody, které jsou k události zaregistrovány. Metoda, která je součástí třídy `MSKinect`, a zároveň je registrována jako reakce na událost `DepthFrameReady`, je zobrazena ve výpisu 7.13.

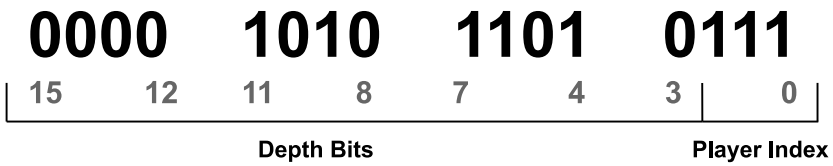
Výpis kódu 7.13: Metoda `OnDepthFrameReady` (třída `MSKinect`)

```
1  public void OnDepthFrameReady(object sender,
       DepthImageFrameReadyEventArgs e)
2  {
3      using (var depthFrame = e.OpenDepthImageFrame())
4      {
5          if (depthFrame == null) return;
6
7          if (_depthImagePixels == null)
8              _depthImagePixels = new
                   short[depthFrame.PixelDataLength];
9
10         // Copy the pixel data from the image to a temporary
               array.
11         depthFrame.CopyPixelDataTo(_depthImagePixels);
12
13         var depthData = new short[_depthImagePixels.Length];
14         Parallel.For(0, _depthImagePixels.Length, i =>
15         {
16             // Extract depth information from depth pixel (3. -
                   16. bit)
17             depthData[i] = (short)(_depthImagePixels[i] >>
                   DepthImageFrame.PlayerIndexBitmaskWidth);
18             if (depthData[i] < 0) depthData[i] = 0;
19         });
20
21         DepthFrameAvailable?.Invoke(this, new
               DepthFrameEventArgs(depthData));
22     }
23 }
```

Nejdůležitější část metody `OnDepthFrameReady` se nachází na řádcích 12-16, kde dochází k extrakci hloubky z jednotlivých pixelů, které jsou k dispozici ve snímku. Pokud se na tento cyklus podíváme detailněji, lze si povšimnout bitového posunu, který je proveden pro všechny dostupné hodnoty. Každý pixel je reprezentován jako datový typ `short` (tj. velikost je 16 bitů). Ovšem pouze některé bity z této hodnoty reprezentují hloubku. Rozdělení jednotlivých bitů je ukázáno na obrázku 7.2. Z obrázku lze vyčíst, že informace o hloubce daného pixelu se vyskytují pouze mezi bity 3-15. Tedy pro získání vzdáleností je potřeba provést posun o 3 bity doprava, protože první 3 bity obsahují informace o indexu hráče [7]. Také si lze

povšimnout, že byl využit paralelní cyklus `Parallel.For`, který by se měl postarat o rychlejší zpracování celého procesu zpracování hloubkových dat.

Když se zpracují všechny body ze získaného snímku, dojde k vyvolání události `DepthFrameAvailable`, která je součástí právě daného rozhraní `IDepthSource`. Tímto způsobem se distribuují data do dalších kroků zpracování a následně i k odeslání jednotlivým klientům.

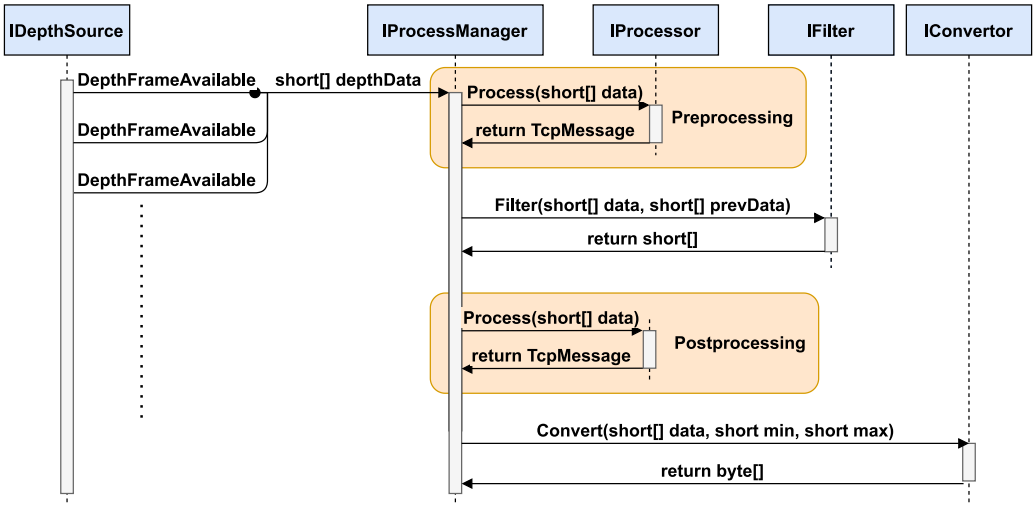


Obrázek 7.2: Rozložení hloubkových bitů [7]

7.5 Proces zpracování hloubkových dat

Po získání vzdáleností z hloubkového senzoru dojde ke zpracování a filtraci těchto dat. Tento proces probíhá v několika krocích. Tyto kroky jsou znázorněny na obrázku 7.3. Na obrázku si můžeme všimnout vytváření událostí s názvem `DepthFrameAvailable`, která je součástí rozhraní `IDepthSource`, které bylo popsáno v předešlé sekci. K této události je přihlášen objekt typu `IProcessManager`, který se stará o zpracování hloubkových dat.

Celkový proces zpracování probíhá celkem ve čtyřech krocích. Mezi ně se řadí - předzpracování, filtrace, zpracování filtrovaných dat a konverze dat. V této posloupnosti operací figurují všechny definované instance jednotlivých rozhraní a každý má svůj vlastní smysl.



Obrázek 7.3: Proces zpracování hloubkových dat

7.5.1 Předzpracování

Předzpracování slouží pro operace, které nijak neupravují celkovou podobu výsledných dat. Může se jednat o ukládání jednotlivých snímků do souboru či rozpoznávání určitých struktur v datech. Všechny třídy, které provádí předzpracování dat musí implementovat rozhraní s názvem `IProcessor`, které je znázorněno ve výpisu 7.14. Toto rozhraní je velice jednoduché a snadno pochopitelné. Někoho by ovšem mohlo zarážet, že návratovou hodnotou metody `Process` je instance třídy `TcpMessage`. Důvodem je, že v některých případech by bylo vhodné zpracované informace distribuovat všem klientům, kteří by je nadále mohli využít. Příkladem by mohl být plugin, který by se staral o rozpoznávání sopek a jejich umístění v terénu. Dané informace by se uložily do specifické třídy, která by byla následně serializována a odeslána jako obsah třídy `TcpMessage`. Nakonec by tato zpráva byla odeslána jednotlivým klientům, kteří zprávu přijmou a zpracují podle svého uvážení.

Výpis kódu 7.14: Rozhraní `IProcessor`

```
1 public interface IProcessor : IPlugin
2 {
3     TcpMessage Process(short[] data);
4 }
```

7.5.2 Filtrace

Po předzpracování dojde k filtrování jednotlivých dat, které slouží pro různé operace. Společným cílem filtrů je upravovat a měnit obsah hloubkových dat a zlepšovat tak jejich kvalitu. Můžeme sem zařadit například omezení dat do nějakého rozsahu hodnot podle nejnižšího a nejvyššího místa v pískovišti. Všechny filtry se musí řídit definicí metod, které jsou obsahem rozhraní `IFilter`, které je ukázáno v ukázce 7.15.

Výpis kódu 7.15: Rozhraní `IFilter`

```
1 public interface IFilter : IPlugin
2 {
3     void Setup(IDepthSource depthSource);
4     short[] Filter(short[] data, short[] previousPrefilterData);
5 }
```

Všechny filtry se spouští podle definovaného pořadí v konfiguračním souboru. To znamená, že výstupní data prvního filtru jsou zároveň vstupními daty pro filtr druhý. V tomto kroku záleží na pořadí provedení, na rozdíl od předzpracování dat, kde je pořadí irelevantní.

7.5.2.1 Redukce šumu

Po analýze hloubkových dat, které jsou naměřené senzorem Kinect bylo zjištěno, že jsou velmi často rozdílné ve dvou po sobě následujích snímcích. Zároveň je nepřijatelné, aby se na pískovišti objevovali velmi rychlé přechody hloubky. Změny musí být postupné, ale zároveň dostatečně rychlé, aby to odpovídalo provedeným změnám. Pro tento problém lze použít tzv. **filtr s nekonečnou impulzní odezvou**.

Filtr funguje tak, že je aktuální hodnota vynásobena definovaným koeficientem. Koeficient je desetinné číslo definováno jako $c \in [0.0, 1.0]$ a lze jej upravovat přes konfigurační soubor. K výsledné hodnotě je připočtena předešlá hodnota snímku, která je přenásobena koeficientem definovaným jako $1 - c$. Jelikož je k aktuální hodnotě vždy přičtena předešlá hodnota (přenásobená koeficientem), je do každého nového snímku zahrnuta i každá předešlá hodnota. Tím dochází ke zmírnění změn provedených na povrchu pískoviště.

Druhým možný řešením, které by bylo možné využít, je **filtr s konečnou odezvou**. Princip filtru je podobný jako u předešlé varianty. Rozdíl je v tom, že není brán v potaz každý předešlý snímek, ale např. posledních 20 snímků. Tento přístup je výpočetně i paměťově náročnější. Přesně z toho důvodu byl zvolen filtr s nekonečnou odezvou. V ukázce 7.16 je tento přístup ukázán přímo v kódu.

Výpis kódu 7.16: Ukázka filtru s nekonečnou odezvou

```
1  public short[] Filter(short[] data, short[]
       previousPrefilterData)
2  {
3      if (storage == null) return data;
4
5      Parallel.For(0, data.Length, (i) =>
6      {
7          storage[i] = (short)((data[i] * ValuePower) +
               (storage[i] * ValuePowerOpposite));
8      });
9
10     Buffer.BlockCopy(storage, 0, resultStorage, 0,
           storage.Length * sizeof(short));
11     return resultStorage;
12 }
```

7.5.3 Zpracování filtrovaných dat

Tento proces je v podstatě totožný jako proces v sekci 7.5.1 na straně 47. Je zde použito stejné rozhraní a funkce jednotlivých tříd je stejná. Rozdílem je, že se toto zpracování provádí na již vyfiltrovaných hodnotách.

7.5.4 Konverze dat

Když jsou všechny filtry a zpracování dokončena je zapotřebí převést výsledné pole do datového typu `byte[]`, které bude následně odesláno pomocí internetové sítě (viz. sekce 7.6). Převedení jednotlivých vzdáleností se provádí vždy pouze jednou. Třída, která je schopna převádět tyto vzdálenosti musí implementovat rozhraní nazvané `IConvertor`, jehož obsah je ukázán ve výpisu 7.17.

Výpis kódu 7.17: Rozhraní `IConvertor`

```
1 public interface IConvertor : IPlugin
2 {
3     byte[] Convert(short[] data, short min, short max);
4 }
```

Výsledné hodnoty jsou nakonec poslány připojeným klientům, kteří je již mohou zpracovávat a prezentovat podle svého vlastního uvážení.

7.6 Odesílání hloubkových dat

Když hloubkový zdroj vytvoří nový snímek, vyvolá událost `DepthFrameAvailable`. K této události je zaregistrovaná požadovaná reakce. V rámci této reakce se provede filtrace a zpracování dat z konkrétního zdroje (více informací viz. sekce 7.5 na straně 46). Po dokončení celého procesu se musí zpracovaná data odeslat všem klientům, kteří se přihlásili k jejich odběru.

7.6.1 Velikost hloubkových dat

Než ale přejdeme přímo na odesílání dat, musíme se zaměřit na senzor Kinect. V kapitole 3 na straně 15 je uvedeno rozlišení hloubkového snímače. Tento senzor disponuje rozlišením o hodnotě `640 x 480` pixelů. Každý pixel má velikost 2 bajty a obsahuje informace o indexu hráče a vzdálenosti daného bodu od senzoru (viz. obrázek 7.2 na straně 46). Tj. výsledné vzdálenosti je potřeba ukládat do datového typu `short`, který má velikost také 2 bajty. Po vynásobení všech hodnot dostáváme, že výsledná velikost jednoho snímku je

$$640 \cdot 480 \cdot 2 = 614.4kB.$$

Pokud budeme uvažovat i snímkovou frekvenci (30 snímků za vteřinu), vychází nám:

$$614.4 \cdot 30 = 18.432MB/s$$

Výsledná hodnota udává objem dat, který bude nutné odesílat klientovi za jednotku času. Hodnota je celkem vysoká, a tudíž bude nutné velikost dat nějakým

způsobem zredukovat. Jednou z možností je přemapovat hodnoty z datového typu `short` na `byte`. Nevýhodou přístupu je, že dojde ke ztrátě přesnosti naměřených dat. Velikost jednoho snímku bude

$$640 \cdot 480 = 307.2kB$$

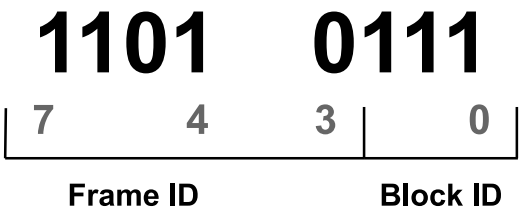
Pro přenos jednotlivých snímků bude zároveň využita komprese, která by měla celkovou velikost přenosu ještě snížit. Bohužel přesné hodnoty vysoce závisí na datech a zároveň na konkrétní implementaci kompresního algoritmu. V rámci diplomové práce bude využit kompresní algoritmus `Gzip`, který je možné využít přímo v rámci programovacího jazyka `C#` a jeho použití je velice přímočaré a snadné.

7.6.2 Rozdělení dat

Z důvodu lepšího přenosu bude celý snímek rozdělen na osm částí, které budou odeslány samostatně a proto bude potřeba, aby obsahem odeslaného paketu byly dodatečné informace. Tyto informace budou sloužit pro identifikaci, o kterou část snímku se jedná.

Jedním řešením může být vytvořit hlavičku, která bude připojena ke každému odeslanému paketu. Obsahem hlavičky bude nejen identifikátor konkrétního bloku, ale bude také informovat o jaké číslo snímku se jedná.

Hlavička bude mít velikost 1 bajt a bude přidávána vždy na začátek paketu. Prvních 5 bitů hlavičky definují hodnotu aktuálního snímku. Poslední 3 bity identifikují, která část snímku je obsahem paketu. Ukázku toho, jak vypadá rozložení těchto bitů, lze vidět na obrázku 7.4.



Obrázek 7.4: Rozložení bitů v hlavičce odesílaného bloku

7.6.2.1 Další operace

Zároveň je nutné si uvědomit, že při použití senzoru Kinect není využit celý rozsah hodnot, které se ve výsledném snímku mohou objevit. Dle nastavení vzdálenosti pískoviště od snímače je kritický pouze „malý“ rozsah hodnot (např. pouze 30 centimetrů od nejnižšího místa v pískovišti). To znamená, že výsledné hodnoty se zároveň dají vyfiltrovat, aby obsahovali pouze tyto hodnoty.

7.7 Jádro

Server pracuje s velkým množství tříd a metod, které by bylo vhodné používat i v ostatních aplikacích (např. třída `TcpMessage`). Proto všechny třídy a rozhraní, které by bylo vhodné sdílet a používat v ostatních projektech, byly přesunuty do oddělené knihovny (jádra). Jádro je tedy knihovna, která se bude používat pro vytváření nových pluginů nebo klientů.

Klient 8

Klient bude v nově navržené architektuře sloužit k obarvení a prezentaci hloubkových dat, které obdrží od serveru. Aktuální kapitola pojednává o technologiích, které byly použity při implementaci klienta. Následně popíši, jakým způsobem probíhá komunikace mezi klientem a serverem. Nakonec se zaměříme na vytváření herních objektů a kalibraci scény.

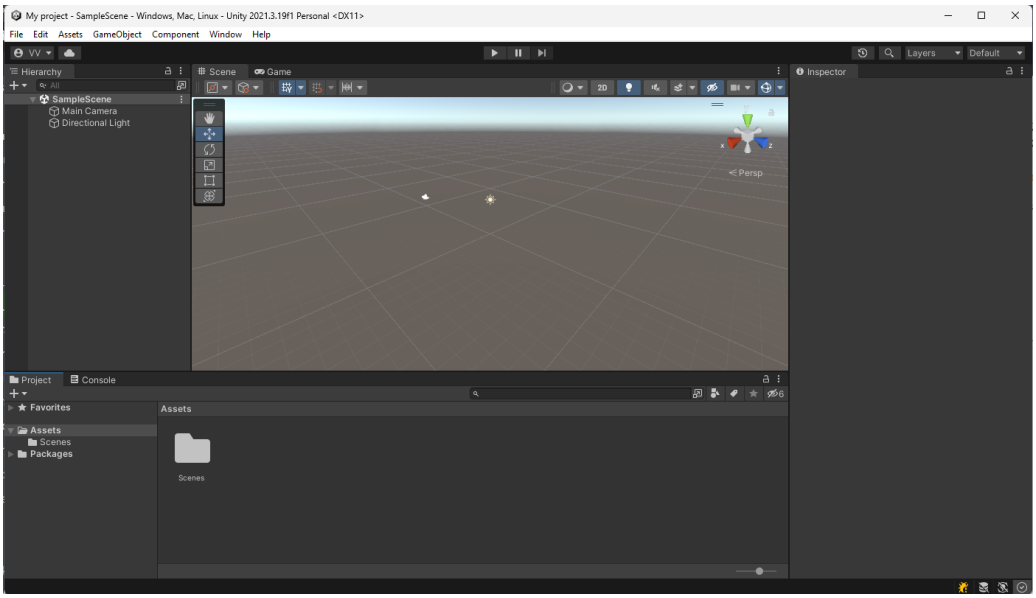
8.1 Použité technologie

Pro vytvoření klientské aplikace by se dala využít řada technologií - herní enginy, různé grafické knihovny (`OpenGL, OpenTK, ...`). Mojí volbou je herní enginy `Unity`, protože práce s tímto nástrojem je intuitivní a zároveň vytváření nových her či herních prvků je podle mého názoru jednodušší. Protože se jedná o herní engine, je zde vyřešeno i mnoho záležitostí, které by se dali v budoucnu využít pro implementaci dalších rozšíření. Pro vývoj byla využita verze `Unity 2021.3.9f1`.

Jak jsem již dříve vysvětlil, Unity je herní engine, který slouží k vývoji 2D i 3D her. Usnadňuje proces vývoje her tím, že umožňuje velké množství již vyřešených problémů. Hry vytvořené v tomto enginu mohou být dále použity na různých zařízeních. Je zde možné definovat prvky herní scény, tlačítka a další objekty, které mohou, ale nemusí interagovat s uživatelem. Ukázka grafického uživatelského prostředí herního enginu Unity je zobrazena na obrázku 8.1.

8.1.1 Herní objekty

Herní objekt je základním stavebním prvkem, ze kterého se skládají hry vytvořené v herním enginu Unity. Herní objekt může představovat libovolný prvek hry, jako jsou postavy, objekty ve scéně, světelné zdroje, zvuky a další prvky, které jsou součástí herního prostředí.



Obrázek 8.1: Grafické prostředí herního enginu Unity

8.1.2 Prefaby

V Unity je **prefab** (zkratka z anglického *"prefabricated object"* neboli *"předvýrobní objekt"*) předem vytvořená šablona herního objektu nebo skupiny objektů s určitými vlastnostmi a komponentami. Prefaby umožňují opakované použití stejného objektu nebo skupiny objektů v různých scénách nebo v různých částech stejné scény.

Když vytvoříte prefab, vytvoří se kopie objektu se všemi jeho komponentami, nastavením a vazbami na jiné objekty. Tuto kopii můžete uložit jako prefab a používat ji v různých částech hry, například pro vytváření stejných nepřátel nebo dekorací. Zároveň se díky tomu mohou sdílet herní objekty v rámci různých projektů.

8.1.3 Skripty

Běh herních objektů je v Unity řízen pomocí metod, které ovlivňují a definují životní cyklus herních objektů. Vytvořené skripty jsou napsány v jazyce `C#`. Těchto metod je definováno velké množství. Lze mezi ně zařadit funkce - `Awake`, `Start` nebo `Update`.

Funkce popsané výše definují akce, které se provedou v různých fázích hry. Metoda s názvem `Awake` se spustí v okamžiku, kdy je herní objekt načten spolu s herní scénou, nebo pokud je objekt převeden z neaktivního do aktivního stavu. Druhou důležitou funkcí je `Start`, která se spustí až poté, co je daný skript povolen. Zde se většinou nachází kód závislý na načtení ostatních objektů, které nejprve musí spustit funkci `Awake`. Tato funkce se spustí pouze jednou v životním cyklu objektu, a to ve snímku předtím, než je vyvolána metoda `Update`. Poslední ze zmíněných

funkcí se opakovaně spouští, dokud není daný objekt zničen nebo dokud hra neskončí. Slouží k aktualizaci herního objektu v závislosti na čase. Nejčastěji se jedná o změnu pozice figurky hráče na základě stisknutých kláves a jeho rychlosti.

8.2 Import jádra

V rámci implementace serveru došlo k vytvoření jádra, které by bylo vhodné využít i pro implementaci klientské aplikace. Jedná se hlavně o třídy, které nějakým způsobem ulehčují práci s komunikační vrstvou obou aplikací. Pro použití jádra je nutné tuto knihovnu nejdříve importovat do vytvořeného projektu v Unity.

Pro import knihovny do prostředí projektu v Unity, je nutné soubor knihovny (v našem případě soubor `DigitalSandboxCore.dll`) přesunout do složky `Assets`, která se nachází ve vytvořeném projektu Unity. Tímto způsobem by měla být knihovna importována a měli bychom ji být schopni používat.

8.3 Konfigurace

Předtím než se klient bude moci připojit k serveru, je nutné vytvořit jednoduchý způsob, pomocí kterého půjde měnit konfigurace klienta. Zároveň je vhodné, aby byla konfigurace uložena (např. v souboru). V souboru bude uložena IP adresa serveru, jeho TCP port a UDP port, na kterém bude klient přijímat hloubková data.

Současně s těmito hodnotami lze do konfiguračního souboru ukládat i další informace, které je nutné ukládat. Příkladem může být kalibrace, která musí být po každém zapnutí stejná, aby uživatel nemusel kalibrovat pískoviště po každém spuštění. V ukázce 8.1, která se nachází na straně 56, je zobrazena část konfiguračního souboru. Pro ukládání a nastavování hodnot z kódu je vytvořena třída `Config` a `ConfigManager`.

8.3.0.1 `Config.cs`

Jedná se o třídu/strukturu, která udržuje všechny potřebné informace, které se budou ukládat do souboru v rámci konfigurace. Může se jednat o již zmíněné TCP/UDP porty, IP adresu serveru nebo kalibrační hodnoty.

8.3.0.2 `ConfigManager.cs`

Třída `ConfigManager` poskytuje metody pro načítání konfigurace ze souboru a zároveň metodu, která převede instanci třídy `Config` na řetězec, který je následně uložen ve formátu `JSON` do konfiguračního souboru. K převodu na řetězec je opět použita knihovna `Newtonsoft`.

Výpis kódu 8.1: Část konfiguračního souboru

```
{
    ...
    "serverAddress": "127.0.0.1",
    "serverTcpPort": 33333,
    "clientUdpPort": 22222,
    ...
}
```

8.4 Komunikace se serverem

Pro implementaci komunikační vrstvy na straně klienta, je potřeba postupovat podobným způsobem jako v případě serveru. Tj. bude potřeba vytvořit dvě samostatná vlákna (jedno vlákno pro TCP komunikaci, druhé vlákno pro UDP).

Pro každé vlákno bude vytvořen samostatný skript/třída, aby jednotlivá implementace byla logicky oddělena. Všechny skripty, které pracují na komunikační vrstvě aplikace se nachází ve složce `Scripts/Network`. Pro TCP komunikaci mezi klientem a serverem byl vytvořen skript s názvem `TcpManager.cs`. Skript `UdpListener.cs` obsahuje stejnojmennou třídu, která naslouchá na specifikovaném portu a přijímá UDP pakety od serveru.

8.4.1 TCP komunikace

Nejprve je nutné vytvořit nové vlákno, které se pokusí připojit k serveru. Vytvoření vlákna probíhá v rámci běhu metody `Start`, která je součástí třídy `TcpManager`. Metoda pouze vytvoří nové vlákno, kterému je přiřazen delegát hlavní metody vlákna. Nakonec je vlákno spuštěno a jeho životní cyklus je zahájen.

8.4.1.1 Připojení k serveru

Pro navázání spojení se serverem je nutné vytvořit nový soket, který bude sloužit jako vstupní/výstupní bod pro odesílání/příjem TCP zpráv. Když je soket vytvořen, klient se pokusí připojit k serveru dle hodnot definovaných v konfiguračním souboru (více informací o konfigraci viz. sekce 8.3 na straně 55). Pokud je server nedostupný, je nutné tento proces provádět opakovaně, dokud nebude klient připojen nebo nebude dosaženo definovaného počtu pokusů. Ukázka toho, jak celý proces připojování vypadá, je zobrazen v ukázce 8.2.

Výpis kódu 8.2: Ukázka připojení k serveru

```
bool ConnectToServer(string ipAddress, int tcpPort)
```

```
2  {
3      socket = new Socket(AddressFamily.InterNetwork,
           SocketType.Stream, ProtocolType.Tcp);
4
5      var attempt = 1;
6      while (!socket.Connected)
7      {
8          if (!IsRunning || attempt > ATTEMPT_LIMIT) return false;
9
10         Debug.Log($"<TCP> Connecting to {ipAddress}:{tcpPort}
               (Attempt: {attempt}) ");
11
12         try
13         {
14             socket.Connect(ipAddress, tcpPort);
15         }
16         catch (SocketException)
17         {
18             attempt++;
19         }
20     }
21
22     Debug.Log("<TCP> Client is connected to the server.");
23     return true;
24 }
```

8.4.1.2 Registrace pro odběr hloubkových dat

Když se klient připojí k serveru, musí se zaregistrovat k odběru hloubkových dat. Proto je třeba odeslat serveru TCP zprávu, ve které bude uveden UDP port, na kterém klient očekává hloubková data.

K vytvoření zprávy je použita jedna z továrních metod, které jsou obsahem třídy `TcpMessageFactory`. Třída `TcpMessageFactory` je součástí jádra, které bylo vytvořeno při implementaci serveru. Obsah a formát zprávy se zde již objevil, konkrétně v ukázce 7.6 na straně 39.

Po odeslání této zprávy začíná klient naslouchat serveru a přijímat zprávy ze serveru, které by mohl potenciálně použít k rozšíření funkcí.

8.4.1.3 Reakce na příchozí TCP zprávy

Reakce na příchozí zprávy ze serveru je velice podobná jako přijímání zpráv na serveru. Po přijetí nové zprávy přes soket dojde k převedení zprávy na instanci třídy `TcpMessage` (pomocí deserializace). Po úspěšném převedení zprávy, klient vyvolá

událost `OnNewTcpMessageReceived`. Na tuto událost může být připojeno několik dalších tříd, které by se starali o zpracování příchozí metody.

8.4.2 UDP komunikace

Stejně jako u TCP komunikace je nutné vytvořit samostatné vlákno, které bude čekat na vytvořeném UDP soketu. Na tento soket bude server odesílat hloubková data.

8.4.2.1 Vytvoření soketu

Celá inicializace UDP spojení je zahájena vytvořením UDP soketu, ke kterému je asociována IP adresa a port, na kterém klient poslouchá a přijímá příchozí UDP pakety. Nejdříve je tedy nutné vytvořit soket, který bude definovaný tak, aby využíval UDP protokol. Stejně jako v případě serveru, jsou zde použity 3 základní parametry.

Zároveň je při vytváření nové instance třídy `Socket` upraven parametr `ReceiveBufferSize`. Parametr určuje velikost vyrovnávací paměti používané pro příjem příchozích dat. Když soket přijímá data, ukládá příchozí data do vyrovnávací paměti, dokud je aplikace nezpracuje. Vlastnost `ReceiveBufferSize` určuje velikost této vyrovnávací paměti v bajtech. Ve výchozím nastavení je parametr nastaven na 8192 bajtů. Protože server (resp. snímač) generuje data, která mají velkou výslednou velikost je nutné zvětšit vyrovnávací paměť, aby nedocházelo k častým ztrátám paketů z důvodu zahlcení vyrovnávací paměti. Popsaná část je ukázána v části kódu 8.3.

Po vytvoření soketu a jeho navázání na konkrétní přístupový bod, dojde k vytvoření daného vlákna a jeho spuštění. Vlákno následně začne pouze poslouchat na přístupovém bodu, dokud neobdrží nový paket od serveru nebo nedojde k nečekanému odpojení či ukončení programu.

Výpis kódu 8.3: Vytvoření UDP soketu a vlákna

```
1  void Start()
2  {
3      ...
4      ipEndPoint = new IPEndPoint(IPAddress.Any,
           ConfigManager.Config.clientUdpPort);
5      socket = new Socket(AddressFamily.InterNetwork,
           SocketType.Dgram, ProtocolType.Udp)
6      {
7          ReceiveBufferSize = RESOLUTION + BLOCK_COUNT
8      };
9      socket.Bind(ipEndPoint);
10     listenerThread = new Thread(new ThreadStart(ThreadMethod));
11     listenerThread.Start();
12 }
```

8.4.2.2 Přijetí nového snímku

Přijímání nových paketů probíhá pomocí metody `Receive`, která je volána nad instancí soketu. Když soket zachytí nový paket, je uložen do definovaného pole bajtů a následně je potřeba extrahovat přijatá data.

Celý paket je nutné nejdříve dekomprimovat, jelikož všechny odesílané pakety jsou na straně serveru komprimovány. K dekompresi dat je použit algoritmus `Gzip`, protože je důležité, aby obě strany používaly stejný kompresní algoritmus. Potom jsou data v původní podobě a je tedy možné s nimi nadále pracovat.

8.4.2.3 Zpracování paketu

První důležitou operací je získat informace o tom, ke kterému snímku, a jeho bloku, se data vztahují. K tomu slouží metoda `GetFrameInfo`, která je zobrazena v ukázce 8.4. Klient využívá základní bitové operace pro extrakci čísla snímku a konkrétního bloku. Více informací o tom, jak jsou tyto data vytvořena a přidána k paketu, jsou obsahem sekce 7.6 na straně 49.

Výpis kódu 8.4: Extrakce informací o paketu

```
1 void GetFrameInfo(byte b, out byte frameId, out byte blockId)
2 {
3     // Shift the source byte right by 3 bits
4     frameId = (byte)(b >> 3);
5
6     // Mask the frameId and blockId bits
7     frameId = (byte)(frameId & Constants.FrameIdMask);
8     blockId = (byte)(b & Constants.BlockIdMask);
9 }
```

Protože UDP protokol nevytváří trvalé spojení mezi oběma stroji, může docházet k situaci, že přijaté pakety nebudou přijaty ve stejném pořadí, ve kterém byly odeslány ze serveru. Proto je nutné kontrolovat, jestli aktuálně přijatý blok není obsahem „starého“ snímku. Z toho důvodu je nezbytné udržovat informaci o tom, jaký snímek je aktuálně zpracovávaný a pak provést kontrolu, zdali patří přijatý blok do starého, aktuálního nebo nového snímku.

Pokud je snímek starší, je asi nejrozumnější přijaté informace zahodit, protože budou mít nevypovídající hodnotu. Pokud blok patří do aktuálního snímku, jsou hodnoty uloženy na řádnou pozici do alokovaného místa v paměti. Jestliže identifikátor přijatého paketu informuje o tom, že se jedná o blok nového snímku, je nejdříve vytvořena kopie aktuálního snímku. Kompletní snímek je potřeba propagovat ostatním třídám, které jej chtějí využít. K tomu opět slouží událost s názvem `OnNewFrameReady`. Tento princip je viditelný v ukázce 8.5. Po odeslání kopie aktuálního snímku je uložen obsah paketu na konkrétní místo v poli.

Výpis kódu 8.5: Distribuce kompletního snímku

```
1  void UpdateActualFrame()
2  {
3      if (OnNewFrameReady == null) return;
4
5      var frame = new float[HEIGHT, WIDTH];
6      // Copy storage into new array.
7      Buffer.BlockCopy(frameStorage, 0, frame, 0, frame.Length *
           sizeof(float));
8
9      OnNewFrameReady(frame);
10 }
```

8.4.3 Herní objekt

Vytvořené skripty bylo nutné připojit k hernímu objektu, aby došlo k jejích spuštění a zahájení veškeré komunikace směrem k serveru.

8.5 Terén

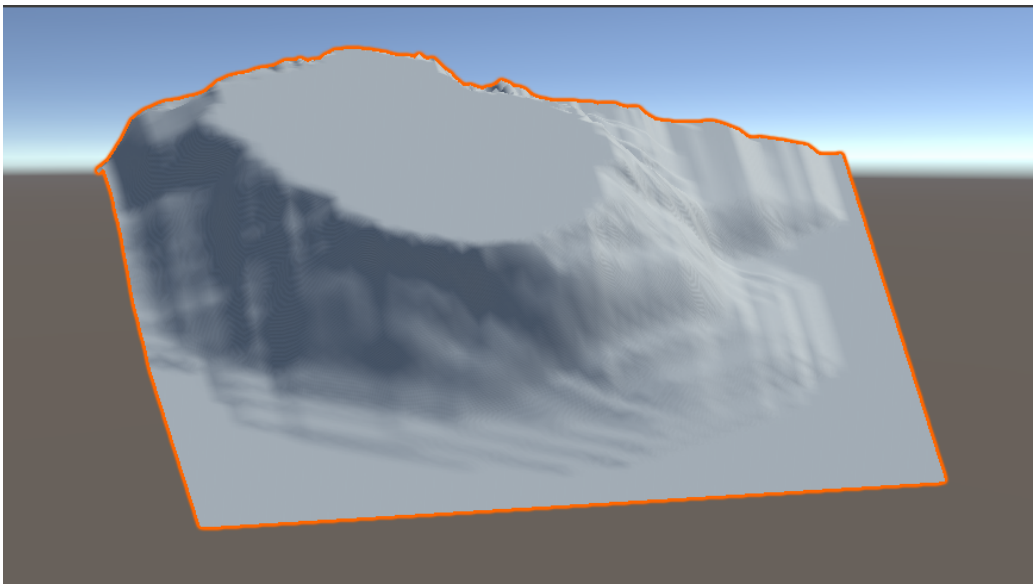
Program tedy dokáže komunikovat se serverem a přijímá hloubková data zachycená senzorem Kinect. Teď je důležité získané informace vizualizovat.

Editor Unity obsahuje vestavěnou sadu nástrojů, které umožňují přidat do hry krajinu. V editoru můžete vytvořit více dlaždic terénu, upravit jejich výšku nebo vzhled krajiny a přidat do ní stromy nebo trávu. Zároveň herní engine Unity optimalizuje vestavěné vykreslování terénu s ohledem na efektivitu [39]. Manuálně vytvořený herní objekt s krajinou je ukázán na obrázku 8.2.

Objekt terénu se bude používat pro vytvoření výšek, které budou přijaté ze serveru. Tím docílíme toho, že terén bude mít stejný hloubkový profil (tvar) jako písek v pískovišti. Příjem hloubkových dat byl již popsán v sekci 8.4.2 na straně 58.

Třída `UdpListener` poskytuje událost `OnNewFrameReady`, která je vyvolána, když je k dispozici nový kompletní snímek s naměřenými daty. Proto je vytvořen nový skript s názvem `TerrainController`, který řídí kompletně všechny operace, které jsou spojené s konkrétním terénem. V interní metodě objektu s názvem `OnEnable` je zaregistrována reakce na událost `OnNewFrameReady`. Naopak v metodě `OnDisable` bude reakce na událost odebrána, aby nedocházelo ke zbytečné aktualizaci herního objektu, který není viditelný. Reakce na událost pouze uloží instanci snímku, která bude použita až v metodě `Update`, která se stará o aktualizaci herní scény, a to znamená i o aktualizaci vzhledu terénu (resp. jeho hloubkového profilu).

Obsahem metody `Update` je nastavení hloubkového profilu podle přijatých hloubkových dat. Pro aktualizaci hloubkové mapy je využíván parametr `terra-`



Obrázek 8.2: Obyčejný terén v editoru Unity

`inData`, který je součástí terénu (viz. výpis kódu 8.6). V ukázce je získán odkaz na konkrétní herní objekt, u kterého je nejdříve upravena velikost celého terénu podle nastavené kalibrace (více informací o kalibraci lze najít v sekci 8.7). Následně jsou pomocí metody `SetHeights` nastaveny výšky jednotlivým bodů v terénu na základě přijatých dat, které jsou uloženy v poli `heights`. Je důležité upozornit, že metoda `SetHeights` přijímá vzorky, které jsou reprezentovány jako plovoucí hodnoty od 0 do 1. Tudíž při příjmu bajtů se provádí zároveň i přemapování z bajtů na čísla s plovoucí řádkou v daném rozsahu.

Výpis kódu 8.6: Aktualizace hloubkové mapy terénu

```
1 Terrain terrain = GetComponent<Terrain>();
2
3 terrain.terrainData.size = new
      Vector3(ConfigManager.Config.calibration.Stretch.x, depth,
      ConfigManager.Config.calibration.Stretch.y);
4 terrain.terrainData.SetHeights(0, 0, heights);
```

8.5.1 Obarvení terénu

Když už máme připravenou komunikaci mezi serverem a klientem a je implementováno správné zacházení s terénem (tj. profil odpovídá vzhledu písku v pískovišti), můžeme začít obarvovat terén podle toho, jak je daný bod vysoko. Tedy je důležité, aby byli kopce pokryté sněhem a v údolích byla voda. K tomuto účelu budou sloužit tzv. `texture array` a technologie zvaná `shader`.

8.5.1.1 Pole textur

`Texture array` (neboli pole textur) je v podstatě množina (kolekce) 2D textur (obrázků) stejné velikosti, které pro grafickou výpočetní jednotku (anglicky Graphics Processing Unit, zkráceně GPU) vypadají jako jeden objekt a mohou být vzorkovány v shaderu pomocí indexu v poli textur. Jsou užitečná pro implementaci vlastního terénu nebo jiných speciálních efektů, kde je potřeba efektivní způsob přístupu k mnoha texturám stejné velikosti a formátu [39]. Ukázku toho, jak takové pole textur vypadá, je možné si prohlédnout na obrázku 8.3. Ukázkové pole textur je zároveň použito i pro reimplementaci klientské aplikace.

Obrázek 8.3: Pole textur

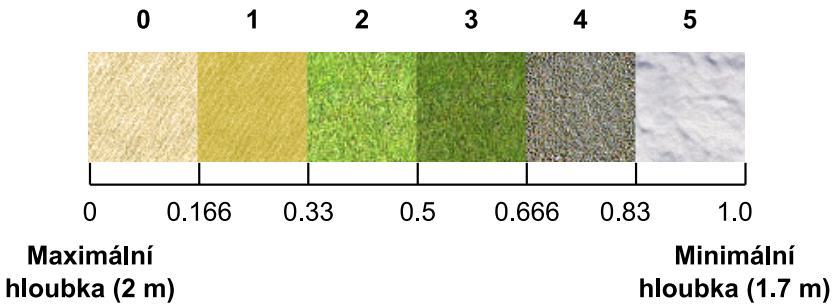
8.5.1.2 Shader graf

Shader je počítačový program nebo část kódu, která je spouštěna na grafické kartě počítače. Řeší vykreslování a obarvování jednotlivých pixelů na obrazovce. Shader je nezbytný pro vytváření vizuálních efektů. Vytváření shaderů je velice komplexní záležitost. Ovšem tento problém řeší přímo herní engine Unity pomocí tzv. *shader grafu*.

Shader graf umožňuje vizuálně vytvářet shadery a sledovat výsledky v reálném čase [24]. Shader graf urychluje a usnadňuje vytváření shaderu uživatelům, kteří s tímto nemají velké zkušenosti. Vytváření shader grafu je uskutečněno pomocí vytváření uzlů, které mají definované určité vlastnosti. Uzly mohou obsahovat množinu vstupů i výstupů, které mohou být opět použity jako vstup či výstup do dalšího uzlu. Díky tomu se mohou definovat i složité shadery, které provádí i několik operací. Herní editor Unity poskytuje velké množství již předpřipravených uzlů, které mají nejrůznější využití - sčítání různých hodnot, přemapování hodnot, atd.

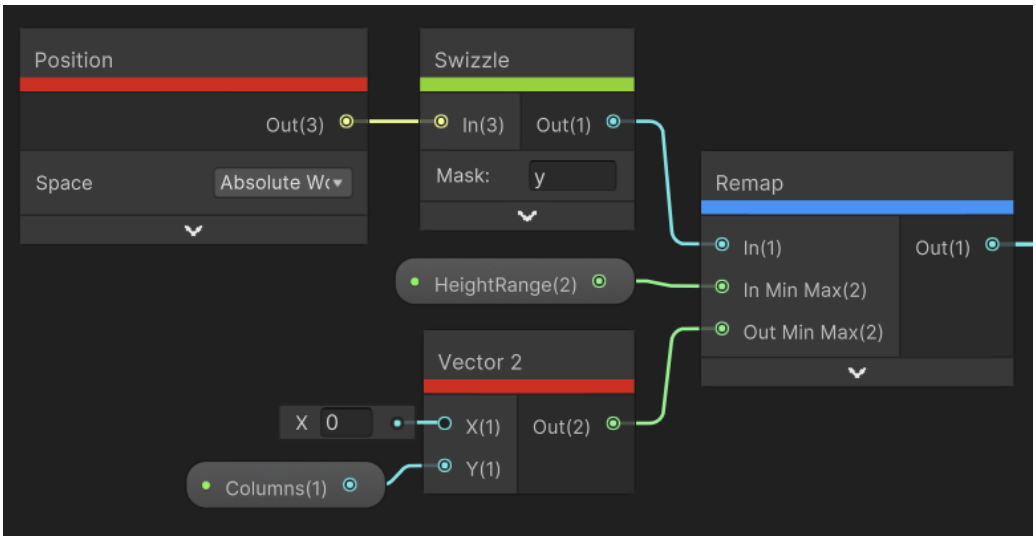
Pomocí shaderu a shader grafu lze vytvořit řešení, které bude rozpoznávat, jak vysoko je daný pixel a podle této informace ho obarví. Na obrázku 8.4 je vidět, které textury jsou použity pro konkrétní rozmezí hodnot. Problémem je, že výška daného bodu v herní scéně není v rozmezí 0 - 1, ale nula až maximální výška terénu.

Pro nalezení správné textury v poli je tedy nutné nejdříve přemapovat výšku na rozsah definovaný jako 0 až počet textur v poli. Pokud tedy bude výška dosahovat hodnoty např. 23 a maximální výška terénu bude 50, pak bude hodnota přemapována na hodnotu 2.76 a pro obarvení budou vybrány textury na indexu 2 a 3. Ukázka toho, jak přemapování hodnot vypadá přímo v shader grafu je ukázáno na obrázku



Obrázek 8.4: Mapování textur podle hloubky

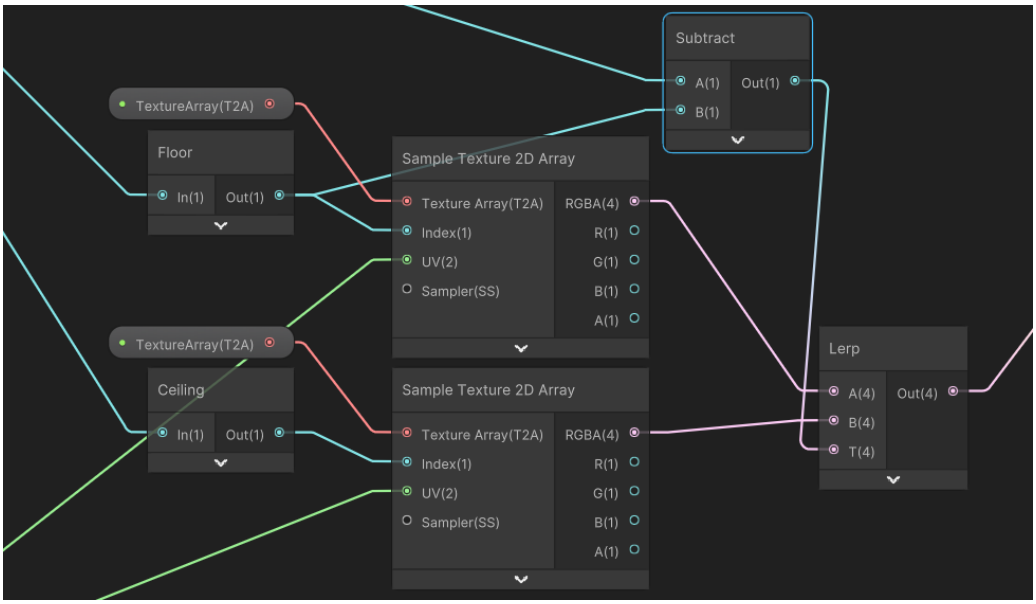
8.5. Nejdříve je pomocí uzlu `Position` a `Swizzle` získána hodnota bodu na ose Y (tj. výška v herní scéně). Výška slouží jako vstup do uzlu `Remap`, kde je vstupem i rozsah výšek terénu a vektor definovaný jako rozsah 0 - počet možných textur v poli. Tím docílíme, že přemapované hodnoty budou v definovaném rozsahu.



Obrázek 8.5: Přemapování hodnot mezi výškou a rozsahem textur

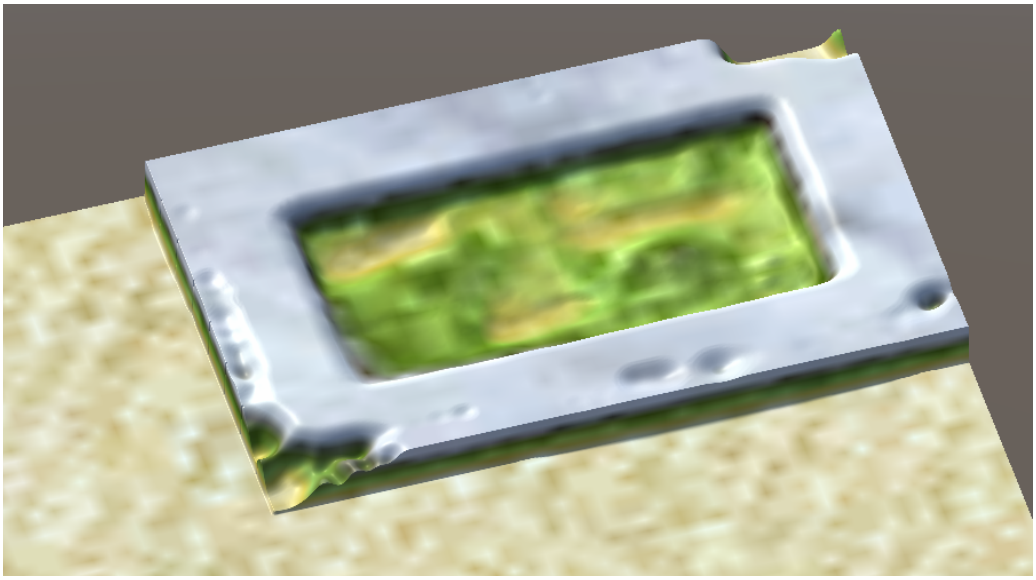
Přemapovaná hodnota je následně zaokrouhlena dolů a nahoru. To nám zaručí, že budeme znát, mezi kterými dvěma texturami se hodnota nachází a budeme moct provést lineární interpolaci (pomocí uzlu `Lerp`) mezi oběma texturami. Výsledná textura bude následně použita pro vykreslení daného bodu. Popsaná část byla vytvořena i pomocí uzlů v shaderu grafu (viz. obrázek 8.6). Vstupem do uzlů `Floor`, `Ceiling` a `Substract` je právě výstup z uzlu `Remap`, který byl již popsán.

Jak jsem již informoval, výsledná textura je použita pro obarvení konkrétního bodu v herní scéně. Po aplikování shaderu na nový materiál, který je následně použit pro obarvení terénu dostáváme výsledek, který je zobrazen na obrázku 8.7. Na obrázku si lze povšimnout, že výsledná textura je velice rozmazaná. Problémem je, že v shaderu chybí definice tzv. *tilingu*.

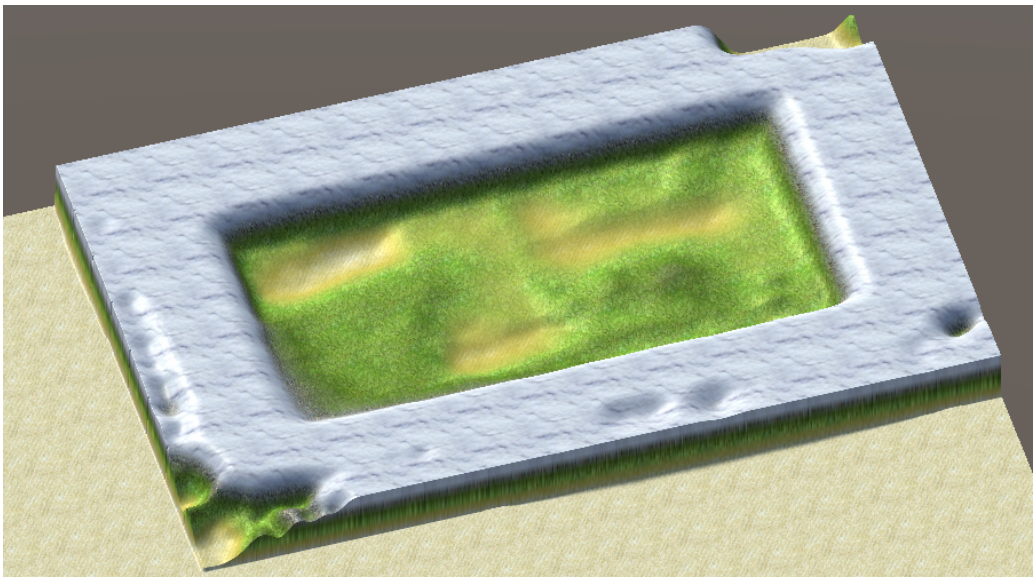


Obrázek 8.6: Lineární interpolace textur

Tiling se stará o to, že textury nebudou roztahovány přes celý terén, ale bude se s nimi pracovat jako s dlaždicemi. Pokud bude tiling nastaven na hodnotu např. 100, bude v každé ose použito 100 dlaždic s touto texturou a výsledný obraz tedy nebude tolik rozmazaný. Na obrázku 8.8 si lze všimnout nejen větší kvality obarvení terénu, ale zároveň i vzoru, který se vytváří na bílých místech. Právě v těchto místech lze zjistit, že se využívá právě popsaná technika. Pro využití tilingu v shader grafu existuje specifický uzel zvaný `Tiling And Offset`.



Obrázek 8.7: Terén bez použití *tilingu*



Obrázek 8.8: Terén s použití *tilingu*

8.6 Voda

Máme připravenou komunikaci se serverem a obarvený terén podle výšky, ale stále chybí v terénu voda, která dodává krajině realističnost. Pro vytvoření vody bude použita stejná technika jako pro obarvování terénu, tedy shader graf. Bude vytvořena rovná plocha, která bude z části průhledná a bude animovat chování vody podle definovaného shaderu. Vytvořený shader graf je zobrazen na obrázku 8.9.

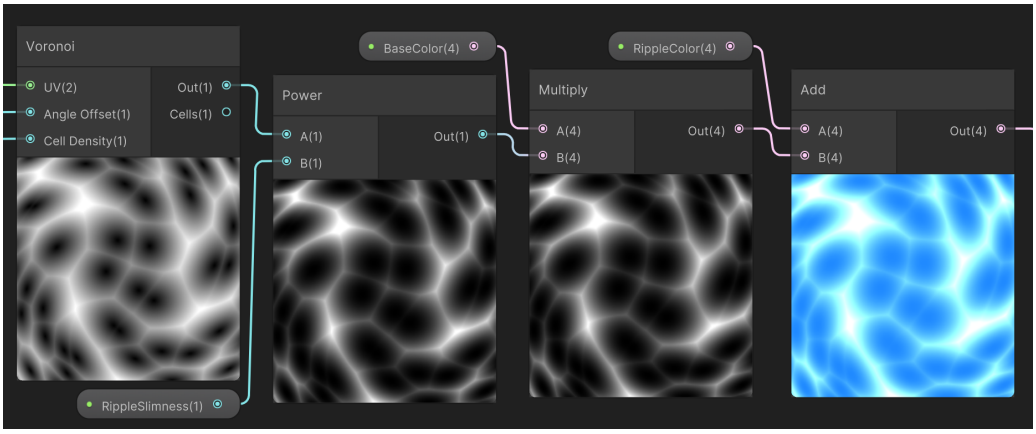
Pro animaci vody bude sloužit uzel s názvem `Voronoi`. Ten generuje Voronoiův nebo Worleyho šum na základě vstupních UV souřadnic. Samotný Voronoiův šum je ovšem nehybný a je tedy nutné jej rozhýbat, a simulovat tak pohyb vody za určitou jednotku času. K tomu slouží uzel `Time` a `RippleSpeed`. Uzel `RippleSpeed` je parametr, který ovlivňuje rychlost jakou se pohybují jednotlivé body v šumu. Rychlost buněk je přenásobena hodnotou `Time`, což zajistí pohyb bodů v čase.

Dalším vstupním parametrem uzlu `Voronoi` je hustota bodů. Jelikož se tento parametr těžko odhaduje pouze z náhledu shaderu, je vytvořen další vstupní parametr, kterým bude hustota bodů snadno upravena, poté co bude shader aplikován na konkrétní herní objekt.

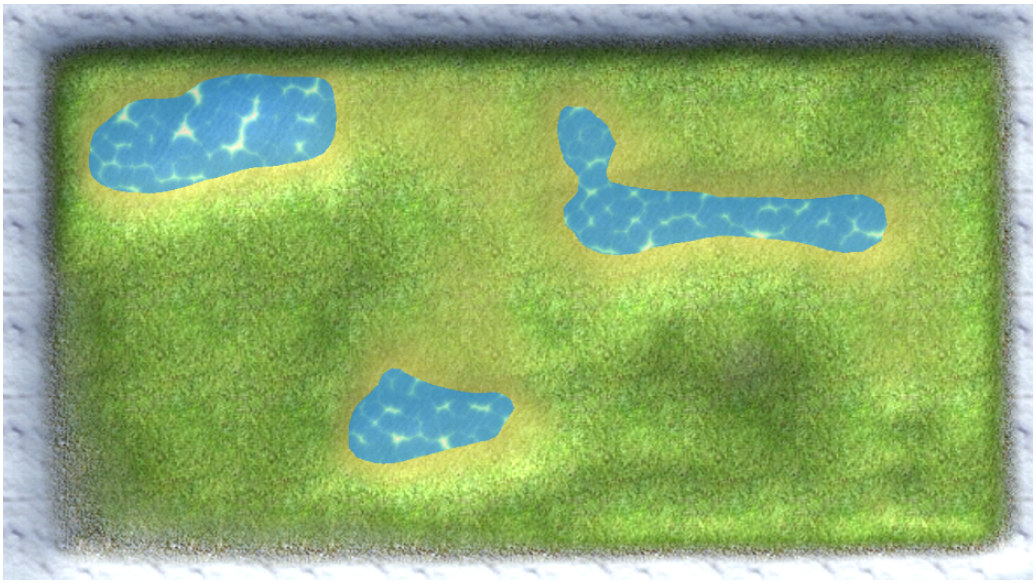
Výstup z uzlu `Voronoi` je umocněn s pomocí uzlu `Power` a vstupní konstanty `RippleSlimness`. Tímto postupem budeme moci měnit tloušťku jednotlivých buněk, které generuje Voronoiův šum.

Následné použití uzlů `Multipy` a `Add` slouží pro obarvení jednotlivých komponent ve výsledném shaderu. Tj. parametr `BaseColor` definuje barvu mezer mezi buňkami a parametr `RippleColor` naopak barvu, kterou budou mít jednotlivé buňky.

Po vytvoření všech popsaných uzlů a aplikování na herní objekt typu `Plane` bylo ovšem zjištěno, že vytvořený shader působí hodně jednotně a uměle. Proto byl do výsledného shader grafu přidán ještě uzel `Radial Shear`, který vytváří vlnovitý efekt a dochází tak k deformaci buněk a zároveň k eliminaci umělého vytvoření.



Obrázek 8.9: Shader graf vody



Obrázek 8.10: Terén s vodou

Po použití shaderu dostáváme výsledek z obrázku 8.10, který ukazuje vykreslený terén spolu s animovanou vodou.

8.7 Kalibrace

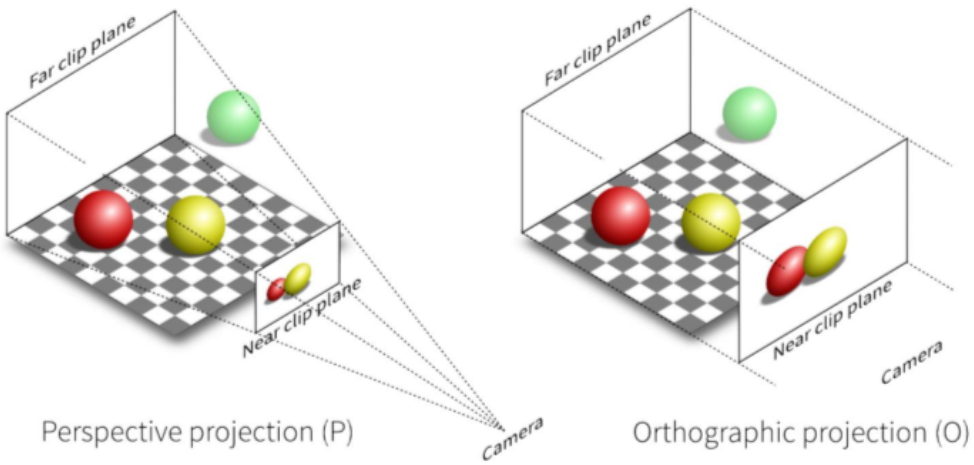
Když jsou vytvořeny herní objekty, je nutné zařídit jejich správné vykreslení a zobrazení v herní scéně.

8.7.1 Kamera

Pro zobrazování herních objektů ve scéně se používá objekt zvaný kamera. Kamera zachycuje prostředí herní scény a zobrazuje ho uživateli. Pozice a nastavení kamery vysoce závisí na tom, jak bude výsledná scéna vykreslena.

Nejprve je důležité nastavit pozici kamery, tak aby směřovala kolmo k vytvořeným objektům a zobrazovala je jako 2D obrázek. Po správném nastavení pozice a kamery, je potřeba změnit způsob, kterým kamera simuluje práci s perspektivou.

Perspektivu lze měnit pomocí parametru `Projection`. Tento parametr přepíná schopnost simulovat perspektivu. Existují dvě možnosti, které lze nastavit - `Perspective` nebo `Ortographic`. Perspektivní kamera vykresluje objekty s nedotčenou perspektivou. Naopak ortografická kamera vykresluje objekty rovnoměrně, bez pocitu perspektivy [39]. Ukázka toho, jak oba přístupy fungují popisuje obrázek 8.11. Pro naše účely bude sloužit ortografická kamera, aby nedocházelo k deformaci hloubky, atd.

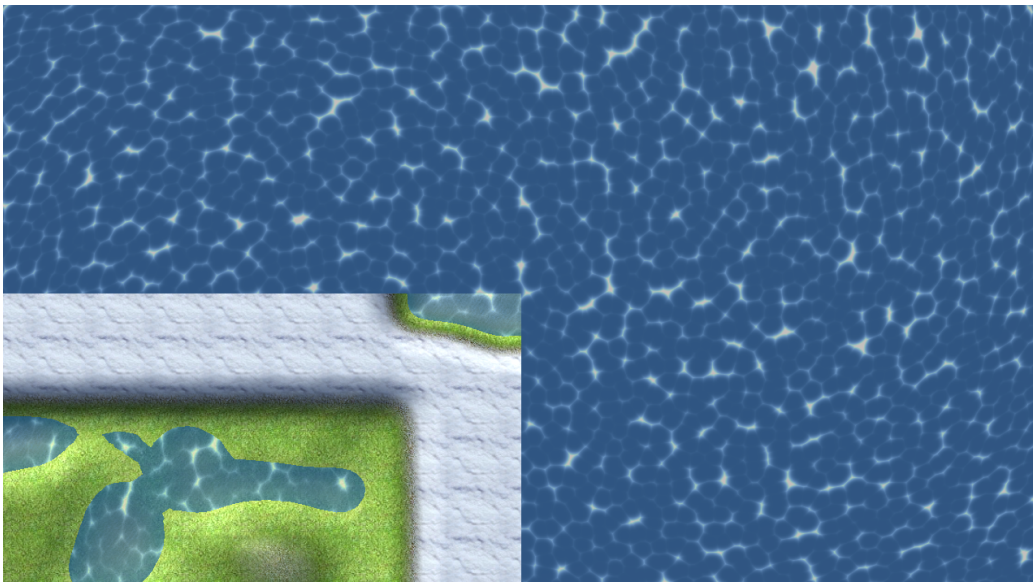


Obrázek 8.11: Rozdíl mezi perspektivní a ortografickou projekcí [40]

8.7.2 Kalibrace terénu

Po nastavení kamery a všech ostatních herních objektů ve scéně, dostaneme výsledek, který je zobrazen na obrázku 8.12. Z obrázku lze zjistit, že je něco v nepořádku. Terén vykreslující pískoviště není skoro vůbec vidět. Proto bude nutné použít kalibraci (viz. sekce 5.1 na straně 27). Ta bude také sloužit pro správnou projekci na fyzické pískoviště.

Kalibrace bude uživateli dodávat možnost srovnat zobrazování pískoviště podle jeho potřeb. V analytické části byly definovány dvě podobné řešení, které můžeme



Obrázek 8.12: Herní scéna bez použití kalibrace

použít. Z důvodu lepší uživatelské přívětivost a celkově i jednodušší implementaci, bude využit způsob, ve kterém se používá posunu terénem a zároveň jeho zmenšování (resp. zvětšování). Protože kalibraci bude provádět uživatel, je potřeba definovat uživatelské ovládání, které bude jednoduché a zároveň bude dostatečně plnit svoji funkci.

8.7.3 Ovládání kalibrace

Uživatelský vstup lze v Unity definovat prostřednictvím dvou různých přístupů - `Input Manager, Input System`.

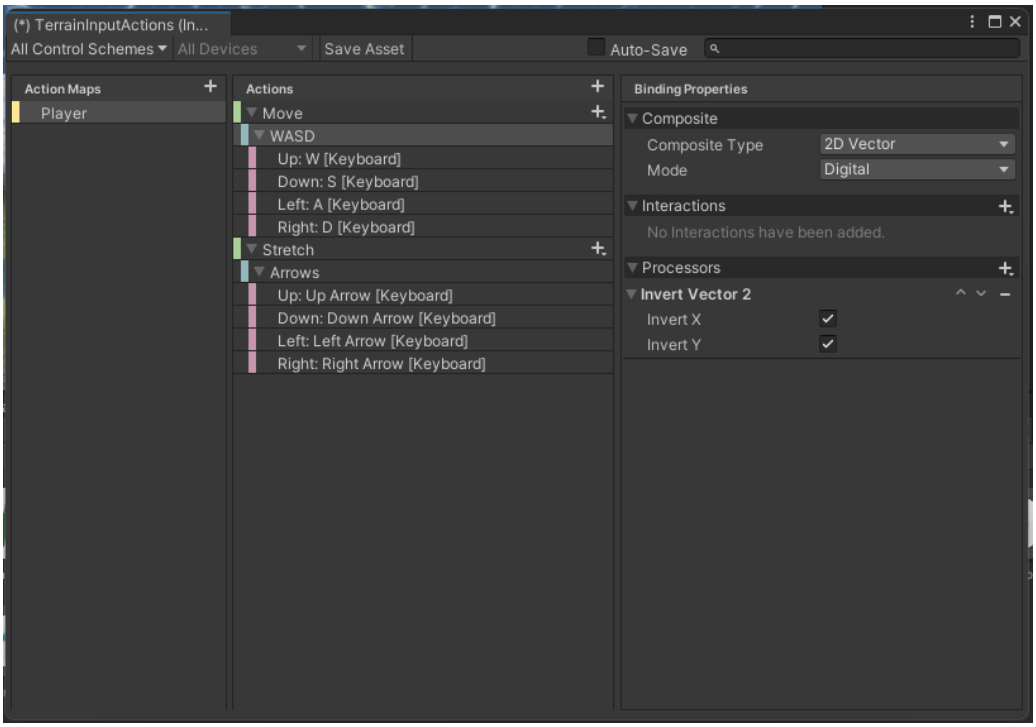
Starší systém, který je integrovaný přímo do editoru, se nazývá `Input Manager`. Je součástí jádra platformy Unity a je výchozí, pokud si nenainstalujete novější systém. [39]

Novější přístup (balíček `Input System`) je flexibilnější systém, který umožňuje používat jakýkoliv druh vstupního zařízení k ovládání obsahu Unity (klávesnice, ovladač, ...). Je určen jako náhrada staršího přístupu. Je označován jako *„The Input System Package“* nebo jen *„Input System“*. Chcete-li jej používat, musíte jej nainstalovat do svého projektu pomocí tzv. `Package Manager` okna. [39]

Pro naše potřeby bude využit novější přístup. Důvodem je hlavně fakt, že je tento systém novější a dodává větší flexibilitu.

Nejprve je nutné vytvořit tzv. `Input Actions` soubor. V Unity editoru se tento soubor otevře ve specifickém okně, které je ukázáno na obrázku 8.13. V tomto okně může vývojář hry definovat jednotlivé akce (např. pohyb hráče, střelba ze

zbraně, atd.). K těmto akcím jsou přidruženy i konkrétní klávesy či jiné způsoby, které vyvolají danou akci.



Obrázek 8.13: Ovládací akce

Na předchozím obrázku (viz. 8.13) je vidět definice obou akcí, na které budeme reagovat a na jejich základě budeme upravovat velikost a pozici terénu. Jako ovládací klávesy byly zvoleny klasické skupiny kláves, tedy `WSAD` pro posun terénu a šipky pro zvětšování. Výběr kláves byl čistě subjektivní, ale při výběru jsem se řídil převážně uživatelskou přívětivostí. Zároveň jsou tyto skupiny kláves velice často použité v různých hrách.

8.7.3.1 Reakce na změnu

Po definování všech akcí je nutné v editoru vybrat volbu `Generate C# class`. Tím dojde k vygenerování nové třídy se stejným názvem (v našem případě `TerrainInputActions`). Tato třída definuje jednotlivé akce, ke kterým můžeme přistupovat přímo z kódu. Pro použití třídy `TerrainInputActions` je nutné vytvořit novou instanci, která bude vytvořena v rámci metody `Awake` třídy `CalibrationController`. Abychom mohli získávat informace o akcích uživatele, je potřeba uložit reference na konkrétní akce a zároveň tyto akce povolit (viz. ukázka 8.7).

Výpis kódu 8.7: Povolení uživatelských akcí

```
1 void OnEnable()
```

```
{
    move = inputActions.Player.Move;
    stretch = inputActions.Player.Stretch;

    move.Enable();
    stretch.Enable();
}
```

Když už máme tohle všechno připraveno, lze reagovat na chování uživatele. Kontrola, zdali uživatel provedl některou z definovaných akcí probíhá v rámci metody `Update`. Konkrétní ukázku funkčního kódu lze vidět ve výpisu 8.8. Nejdříve probíhá získání hodnot z jednotlivých akcích. Hodnota je reprezentována jako vektor o velikosti 2. Pokud uživatel stiskl nějakou z definovaných kláves, pak výsledný vektor může vypadat následovně *(0.0, 1.0)*. Jestliže budou oba získané vektory rovny nulovému vektoru, uživatel nevyvolal danou akci a tedy není potřeba aktualizovat kalibraci.

V opačném případě je potřeba aktualizovat kalibraci, která je uložena v konfiguračním souboru. Nejdříve dojde k sečtení aktuálních kalibračních hodnot a vektoru, získaného pomocí metody `ReadValue` a nakonec jsou nové hodnoty kalibrace uloženy do konfiguračního souboru.

Výpis kódu 8.8: Reakce na uživatelské akce

```
void Update()
{
    var moveVector = move.ReadValue<Vector2>();
    var stretchVector = stretch.ReadValue<Vector2>();

    if (!moveVector.Equals(Vector2.zero) ||
        !stretchVector.Equals(Vector2.zero))
    {
        if (Keyboard.current.ctrlKey.isPressed)
        {
            moveVector *= calibrationScale;
            stretchVector *= calibrationScale;
        }

        // Update calibration and calibration file.
        ConfigManager.Config.calibration.Move += moveVector;
        ConfigManager.Config.calibration.Stretch +=
            stretchVector;
        configManager.SaveConfig();
    }
}
```

Protože byla práce s kalibrací z pohledu uživatele velice pomalá a trvala příliš dlouhou dobu, bylo nutné tento problém nějak vyřešit a dát uživateli možnost

provést kalibraci rychleji. Došlo tedy k definici škálovací konstanty. Pokud uživatel provede některou z možných akcí a současně drží i klávesu `Ctrl` dojde k přenásobení výsledného vektoru a tudíž i k rychlejšímu ovládání kalibrace.

Implementace VR aplikace

9

Pro ověření implementace byl zvolen způsob, ve kterém bude vytvořena VR aplikace. VR aplikace bude uživateli umožňovat využití VR zařízení pro zobrazování pískoviště a člověk se tímto způsobem bude moci procházet po krajině a prozkoumávat krásy vytvořeného prostředí na vlastní oči. Zároveň bude v reálném čase pociťovat provedené změny, které nastanou na povrchu pískoviště.

V této kapitole popíši všechny kroky, které bylo potřeba provést pro implementaci VR aplikace, použité technologie a bude zde uvedena i ukázka finálního vzhledu prostředí.

9.1 Použité technologie

Pro vytvoření VR aplikace budou použity skoro totožné technologie jako pro vývoj klienta. Tedy základem pro vývoj bude herní engine Unity. V tomto případě jsem zvolil verzi `2022.2.11f1`. Unity s každou verzí přidává a zlepšuje funkcionalitu herního enginu. Zmíněná verze byla vydána 15. března 2023 a má lepší podporu nových technologií, jako je virtuální a rozšířená realita. Je to proto, že společnost Unity neustále aktualizuje a zlepšuje herní engine, aby držela krok s nejnovějšími trendy v herním průmyslu. [24]

Dále bylo nutné využít zařízení, které dokáže zobrazovat virtuální realitu, která bude vytvořena v herním enginu. K tomu bude primárně sloužit zařízení `HTC Vive`, ale lze použít i jakékoliv jiné VR zařízení. Zařízení slouží jako brýle pro VR, které jsou umístěny na hlavě hráče a uživatel dokáže skrz brýle pozorovat vykreslenou krajinu.

9.2 Příprava projektu

Předtím než začneme se samotnou implementací, budeme potřebovat připravit projekt tak, abychom mohli využívat VR technologie a zároveň prvky, které byly vytvořeny v rámci implementace klientské aplikace.

9.2.1 Stažení balíčků

Prvním balíčkem, který budeme potřebovat stáhnout, je balíček `OpenXR Plugin`. OpenXR je otevřený, bezplatný standard vyvinutý společností *Khronos*, jehož cílem je zjednodušit vývoj AR/VR a umožnit vývojářům bezproblémově využívat širokou škálu zařízení. [24]

Další užitečný balíček je `XR Interaction Toolkit`. Obsahuje systém pro vývoj interaktivních zážitků ve virtuální a rozšířené realitě, který je založen na komponentách a funguje na vysoké úrovni. Umožňuje snadno vytvářet 3D interakce a interakce uživatelského rozhraní pomocí vstupních událostí z Unity. Základními stavebními kameny tohoto systému jsou komponenty `Interactor` a `Interactable`, které jsou propojeny správcem interakcí. Kromě toho jsou součástí i komponenty pro lokomoci[1] a vizualizaci, které můžeme použít ve svých projektech. [24]

Balíček `XR Interaction Toolkit` obsahuje i řadu příkladů. Mezi poskytované balíčky patří i sada `Starter Assets`. Sada sdružuje prostředky pro zjednodušení chování, včetně výchozí sady vstupních akcí a předvoleb.

9.2.2 Přidání prvků

Jelikož je VR aplikace velice podobná klientské aplikaci, bylo by vhodné využít již vytvořené herní objekty, skripty, atd. Editor Unity dovoluje export konkrétních prvků projektu a následný import do dalších projektů.

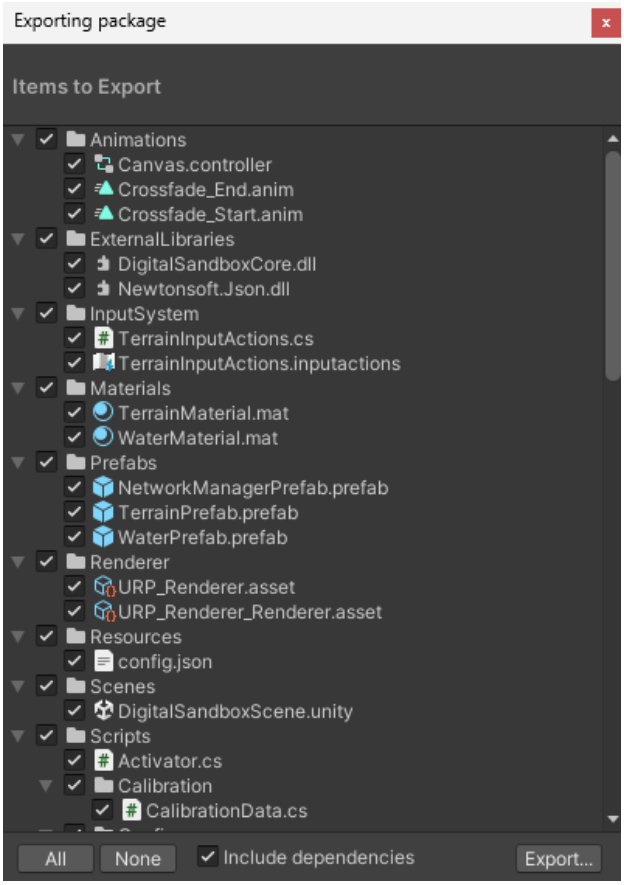
9.2.2.1 Export prvků

Výběr jednotlivých prvků pro export probíhá v okně, které je zobrazeno na obrázku 9.1. Po výběru všech prvků a stisknutí tlačítka `Export`, vytvoří Unity soubor s příponou `.unitypackage`. Tento soubor je možné importovat do dalších Unity projektů.

9.2.2.2 Import prvků

Po inicializaci nového projektu provedeme import souboru, který byl vytvořen v rámci tvorby klientské aplikace (tj. soubor s příponou `.unitypackage`). Po úspěšném importu balíčku je ještě třeba nainstalovat balíčky `Shader Graph` a `Terrain Tools`, které byly použity i v rámci vývoje klienta.

[1] Pohyb člověka, ve smyslu změny místa pomocí svalové činnosti.



Obrázek 9.1: Export prvků

9.3 Příprava scény

Když jsou nainstalovány všechny potřebné balíčky a prvky, je na řadě příprava scény. Ve scéně bude použito několik importovaných prvků a případně dojde i k jejich úpravě nebo vytvoření nových komponent.

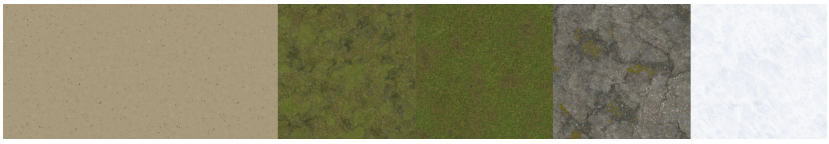
9.3.1 Komunikace se serverem

Pro komunikaci se serverem bude využita stejná množina skriptů jako v klientské aplikaci. Pro použití stačí pouze využít prefab `NetworkManagerPrefab`, který byl importován spolu s potřebnými skripty.

9.3.2 Terén

Herní objekt pro zobrazování terénu je použit stejný jako v klientské aplikaci. Ovšem bylo nutné upravit celkovou vizáž vykreslovaného terénu. Pro úpravu terénu byly zvoleny jiné textury, které se zobrazují na terénu (viz. obrázek 9.2 na straně 76).

Důvodem byla hlavně vizuální stránka terénu, protože uživatel vidí terén více detailněji a je tedy potřeba využít kvalitnější textury. Výměna textur je velice jednoduchá, protože dojde pouze k záměně dvou parametrů v shader graphu, který se stará o obarvování terénu.

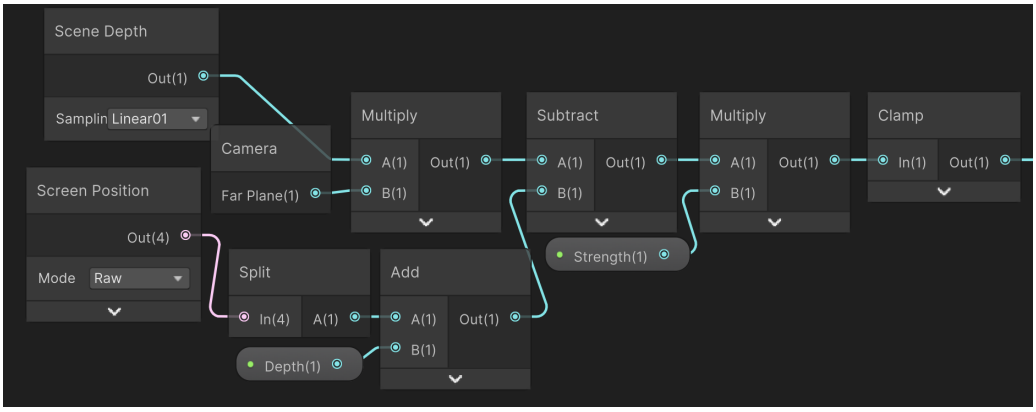


Obrázek 9.2: Textury terénu ve VR aplikaci

9.3.3 Voda

Spolu s terénem byla upravena i voda. Byl vytvořen úplně nový shader graph, který se stará o realističtější zobrazování vody.

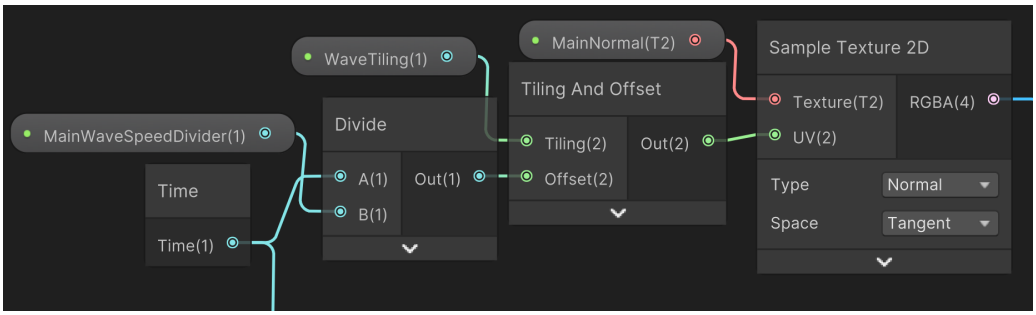
Voda se složena ze dvou částí. První část určuje barvu vody v závislosti na její hloubce. U břehu by měla mít bílou barvu a ve větších hloubkách by měla být tmavě modrá barva. K tomu slouží uzly `Scene Depht`, `Camera` a `Screen Position`. Jejich výstupní hodnoty jsou dále použity a upraveny na základě vstupních hodnot uživatelsky definovatelných uzlů - `Depht`, `Strength`. Před lineární interpolací mezi definovanými barvami je použit uzel `Clamp`, který omezuje hodnoty na určitý rozsah. Nakonec je použit uzel `Lerp` k provedení lineární interpolace definovaných barev na základě intenzity v určitém bodě. Popsaný shader graph lze vidět na obrázku 9.3.



Obrázek 9.3: Část shader grafu obarvujícího vodu

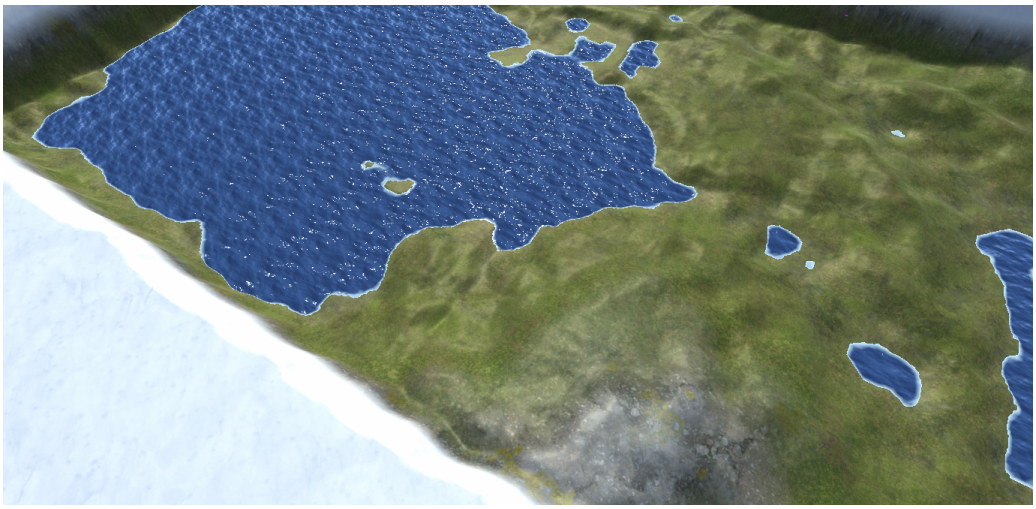
Druhá část shaderu se stará o přidání struktury (vln) a jejich rozpohybování. K tomuto účelu jsou použity dvě rozdílné normálové mapy. Pro načtení těchto map slouží uzly `Sample Texture 2D`. Po přidání normálových map je nutné přidat pohyb, který bude připomínat vlnění. Z toho důvodu je opět využit uzel `Time` a

`Tiling And Offset`, stejně jako v předešlé implementaci vody. Po několika experimentech bylo zjištěno, že je čas ještě potřeba vydělit definovanými konstantami. Nakonec je použit uzel `Add`, který spojí výsledné hodnoty z obou textur. Před použitím hodnoty je ještě použit uzel `Normal Strength`, který upraví sílu normálové mapy. Část tohoto shader grafu lze vidět na obrázku 9.4. Tento graf obsahuje ještě jednu velmi podobnou větev. Jediný rozdíl je v použitých konstantách.



Obrázek 9.4: Část shader grafu, který přidává vodě strukturu

Výsledek toho, jak vypadá realisticky vypadající voda je zobrazen na obrázku 9.5. Na obrázku si lze i povšimnout rozdílných textur, které přidává terénu větší realističnost.



Obrázek 9.5: Upravený terén s realistickou vodou

9.3.4 Kamera

V herní scéně je zapotřebí upravit hlavně jednu základní věc - kameru. Kamera je hlavní rozdílem mezi klientskou a VR aplikací. VR aplikace využívá tzv. `XR Origin`, což je herní objekt, který reprezentuje střed herní scény v rozšířené realitě

(anglicky Extended Reality, zkráceně XR). Zároveň je jeho obsahem i kamera, která bude sloužit pro prezentaci herní scény uživateli (tzn. že bude odstraněna výchozí kamera).

9.3.5 Ovladače

Obsahem objektu `XR Origin` jsou i další dva objekty, které se nazývají `LeftHand Controller` a `RightHand Controller`. Objekty reprezentují ovladače, které má uživatel při hraní v ruce a slouží pro různé interakce s prostředím - pohyb, interakce s jinými objekty, atd.

K ovladačům byly přiřazeny jednotlivé akce, které reagují na ovládání skrz ovladače. Tyto akce jsou přímo součástí balíčku `Starter Assets` a tudíž nastavení ovladačů je velice jednoduché.

Velkou nevýhodu herních objektů vidím v tom, že není vytvořen žádný výchozí vzhled ovladače. Proto bylo využito prefabu z balíčku `Starter Assets`, aby uživatel ve VR viděl jednotlivé ovladače. Ukázku toho, jak vypadá vzhled ovladačů ve VR, je možné vidět na obrázku 9.6.

Obrázek 9.6: Vzhled ovladače ve VR

9.4 Pohyb ve VR

Když je připravená celá scéna můžeme začít pozorovat svět pískoviště v prostředí virtuální reality, ale nelze se v něm nijak pohybovat a prozkoumávat různá zákoutí terénu. Pro pohyb po krajině bude využita teleportace (viz. sekce 4.3.2 na straně

25). Zároveň se zaměříme i na implementaci gravitace, protože je nechtěné, aby se uživatel pohyboval několik „metrů“ nad zemí.

9.4.1 Gravitace

Před nastavením teleportace je zapotřebí vytvořit způsob, kterým lze simulovat gravitaci a ovládat pozici hráče ve vertikální ose. Pro implementaci gravitace je možné využít komponentu zvanou `Continuous Move Provider`, která po nastavení automaticky simuluje gravitaci a táhne uživatele směrem k zemi. V původním projektu byl použit i tento způsob, ale po delším zkoumání a testování byla objevena zásadní chyba. Když se uživatel postavil na náhodné místo v terénu a jiný uživatel změnil povrch pískoviště na stejném místě, na kterém stojí hráč, dostal se pod vytvořený kopec, ale nevěděl o tom. Pro vyřešení problému byl použit tzv. *raycast*.

Raycast vypadá v herním enginu jako neviditelný laserový paprsek, který vychází ze svého zdroje a šíří se předem určeným směrem. Tento paprsek dokáže detekovat, zdali protíná konkrétní objekt, jaká je vzdálenost od zdroje paprsku, atd.

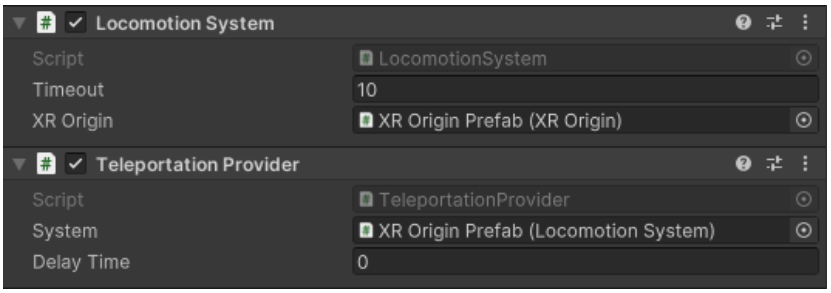
Pro simulaci gravitace a zároveň vyřešení problému s „propadáváním“ do kopců bylo zásadní zjistit, zdali se uživatel nachází nad/pod terénem a následně ho přesunout požadovaným směrem. Přesně k tomu bude použita třída `Raycast`. Byla vytvořena třída `PlayerController`, která v metodě `Update` ověřuje, jestli se pod hráčem vyskytuje terén a posune ho daným směrem. Nicméně pokud se pod uživatelem nenachází terén (tj. je pod terénem), je posunut směrem nahoru a postupně se tedy dostává nad terén. Obsah metody `Update` je zobrazen v ukázce 9.1.

Výpis kódu 9.1: Metoda `Update` třídy `PlayerController`

```
1  void Update()
2  {
3      ray = new Ray(transform.position, -Vector3.up);
4      var isHit = Physics.Raycast(ray, out RaycastHit hit);
5
6      // If ground is hit and player is still to high -> move
           player down (gravity).
7      if (isHit && hit.collider.CompareTag("ground") &&
           hit.distance > minDistance)
8          transform.position += Vector3.down * speed;
9
10     // If nothing is hit -> move player up.
11     if (!isHit)
12         transform.position += Vector3.up * speed;
13 }
```

9.4.2 Teleportace

Pro nastavení teleportace musíme přidat několik komponent, které se budou nastavovat převážně nad objektem `XR Origin`. Jedná se o komponenty `Locomotion System` a `Teleportation Provider`. Nastavení jednotlivých komponent lze vidět na obrázku 9.7.



Obrázek 9.7: Nastavení teleportace

Po nastavení objektu `XR Origin` bylo nutné ještě upravit herní objekt terénu. Abychom mohli začít používat teleportaci musíme hernímu objektu přidat komponentu `Teleportation Area`, která bude sloužit pro definici, kam se uživatel může teleportovat. Tímto je vše nastaveno a uživatel již může používat ovladače pro pohyb (teleportaci) po terénu.

9.4.3 Ukazatel

V sekci výše byly již popsány funkce herních objektů `LeftHand Controller` a `RightHand Controller`. Oba objekty zároveň ve výchozím nastavení disponují několika komponentami, které se starají o vykreslování čáry, která udává směr, kam uživatel ukazuje daným ovladačem. Slouží k tomu komponenty `XR Ray Interactor`, `XR Interactor Line Visual` a `Line Renderer`. Ukázka toho, jak vypadá výchozí ukazatel je zobrazena na obrázku 9.8.

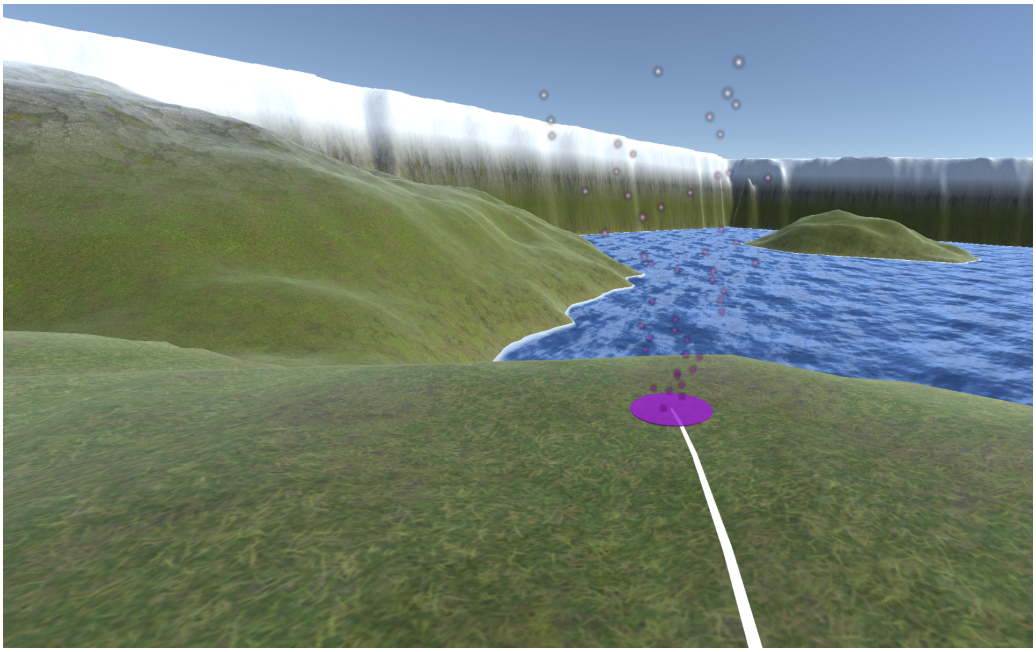
Jelikož ukazatel není příliš hezký ani uživatelsky přívětivý pro pohyb po celé krajině, byly upraveny parametry vykreslované čáry a přidán zaměřovač (tzv. reticle) na konec ukazovátka.

Ze začátku byl upraven celkový tvar ukazovátka. Namísto rovné čáry je čára vykreslována pomocí Bézierovi křivky. Zároveň byla zvětšena délka samotného ukazovátka, aby mohl hráč rychle urazit dlouhé vzdálenosti.

Dále byl na konec ukazovátka přidán zaměřovač, který uživateli dává lepší zpětnou vazbu o tom, kam přesně bude hráč teleportován. Výsledný zaměřovač byl vytvořen z jednoduchého válce, který byl upraven do požadovaného tvaru. Současně byl použit i jednoduchý částicový systém, který z místa pro přesunutí generuje čás-

Obrázek 9.8: Výchozí vzhled ukazatele

tice, které se vznáší nad daným místem. Výsledný vzhled zaměřovače, a zároveň pohled z vytvořené VR aplikace, lze vidět na obrázku 9.9.



Obrázek 9.9: Pohled z VR aplikace

Budoucí možná vylepšení a rozšíření

10

Vzhledem k tomu, že je nová aplikace postavena na architektuře klient-server, nabízí se nepřeberné množství integrace různých aplikací a funkcí do stávajícího systému. To zahrnuje rozšíření stávajících aplikací, vytvoření nových her nebo zásuvných modulů, které lze snadno integrovat s funkcemi serveru.

10.1 Nové hry v klientské aplikaci

Klientská aplikace poskytuje dostatek příležitostí k zahrnutí více her, jako tomu bylo v předchozím systému. Spolu s hrami uvedenými v kapitole 2 na straně 7 skrývá klientská aplikace nekonečné možnosti integrace dalších her. Potencionální novou hrou může být vykreslování vrstevnic namísto zobrazované krajiny.

10.2 Integrace klientských aplikací

Další možností, jak rozšířit vytvořený systém, je propojení více klientských aplikací, které využívají hloubková data serveru. Zajímavá možnost integrace zahrnuje zobrazení pozice VR hráče na povrchu pískoviště. V důsledku toho by mohli kamarádi hráče upravovat terén podle toho, kde se hráč nachází ve VR. Tento integrační mechanismus lze vytvořit prostřednictvím funkce, která dovoluje přeposílat TCP zprávy.

10.3 Nové aplikace

Další varianta zahrnuje výrobu nových aplikací a her. Výhodou je, že hry mohou využívat stejná data jako ostatní aplikace, což je vlastně základním cílem této práce. Příkladem nové hry může být software, který zobrazuje uživatele jako superhrdinu schopného létat nad terénem zobrazeným na obrazovce, přičemž je nepředvídatelně napadán různými protivníky.

Zhodnocení dosažených výsledků

11

V průběhu diplomové práce byla provedena důkladná restrukturalizace původního projektu SandyStation. Byla vytvořena modulární architektura, která umožňuje snadné rozšíření celkové funkčnosti programu. Navíc byla přidána možnost vytvářet různé aplikace, které spolu mohou komunikovat prostřednictvím TCP zpráv.

V rámci diplomové práce vznikli tři aplikace, které spolu mohou komunikovat a zobrazovat pískoviště prostřednictvím počítačové grafiky, nejen ve VR. Vzniká tak zcela nový zážitek, kdy uživatelé mohou pocítit změny provedené v pískovišti. Tímto způsobem byla rozšířena funkcionalita interaktivního pískoviště.

Architektura serveru je navržena tak, aby byla vysoce rozšiřitelná pomocí pluginového systému. Vzhledem k tomu, že server používá k distribuci dat internetové protokoly, je možné používat další aplikace, které se mohou připojit k serveru.

Jednou z nevýhod vyvinutého řešení je, že nebyly znovu implementovány všechny hry vytvořené v původním systému a že neexistují žádné automatizované testy serverových ani klientských funkcí. Bylo by vhodné vytvořit sadu testů, které by kontrolovaly alespoň klíčové funkce serveru.

Celkově lze říci, že práce úspěšně plní svůj cíl a efektivním a příjemným způsobem rozšiřuje funkčnost a modularitu stávajícího řešení.

Závěr 12

V diplomové práci jsem se důkladně zabýval systémem SandyStation, který je nainstalován na KIVu. V rámci analýzy byl poskytnut krátký souhrn historie projektu. Projekt byl detailně prozkoumán a byly popsány jeho základní funkcionality a poskytované hry. V analýze jsem se zaměřoval na možnosti rozšíření aplikace a celkovou funkčnost programu.

Po důkladném prozkoumání aplikace SandyStation jsem popsal historii vývoje senzoru Kinect a základní popis jednotlivých částí, které mají na starosti zaznamenávání prostoru před senzorem. Zároveň se jedná o klíčovou část stávajícího projektu.

Dále se moje pozornost zaměřila na problémy kalibrace a filtrace, které bylo nutné implementovat v rámci nových aplikací.

Klíčovou částí diplomové práce bylo navržení architektury, která bude modulární a současně bude poskytovat jednoduchý mechanismus pro výměnu či přidávání funkcionality. Proto byla navrhnuta architektura na principu architektonického modelu *klient-server*.

Po analytické části přišla na řadu implementace jednotlivých aplikací. Po dokončení diplomové práce vznikly celkem 3 hlavní aplikace - server, klient a VR aplikace. Server zpracovává hloubková data ze senzoru Kinect a odesílá je klientským aplikacím. Architektura serveru je postavena na pluginovém systému, který dodává další možnosti rozšiřitelnosti aplikace. Klientské aplikace získávají data ze serveru, které následně vizualizují.

V aktuální chvíli jsou již všechny aplikace přístupné ke spuštění a otestování v grafické laboratoři na KIVu. Diplomová práce zároveň vytváří nespočet dalších využití, díky kterým lze vytvořit a rozšířit funkcionalitu celého systému. V rámci kapitoly 10 byly představeny různé návrhy, které by se dali použít pro další rozšiřování systému budoucími generacemi.

Jedním z výsledků diplomové práce je i uživatelská a programátorská dokumentace, která podrobně popisuje způsoby pro vytváření nových her, přidávání pluginů k serveru či integraci jednotlivých aplikací.

Uživatelská dokumentace

A.1 Postup pro použití Kinectu

Tato sekce obsahuje návod pro nastavení počítače tak, aby bylo možné použít MS Kinect.

1. Stáhněte a nainstalujte aplikaci `Kinect for Windows SDK 1.8`, která je dostupná na adrese `https://www.microsoft.com/en-us/download/details.aspx?id=40278`.
2. Po úspěšném dokončení instalace SDK se ujistěte, že je senzor Kinect připojen k externímu zdroji napájení, a poté připojte senzor Kinect do USB portu počítače. Ovladače se načtou automaticky.
3. Snímač Kinect by nyní měl fungovat dle očekávání.

A.2 Kompilace a spuštění

A.2.1 Server

Tato sekce obsahuje návod na sestavení serveru.

1. Stáhněte a nainstalujte aplikaci `Visual Studio`, která je dostupná na stránkách `https://visualstudio.microsoft.com/cs/`.
   - Při instalaci zvolte možnost instalovat nástroje pro `Vývoj desktopových aplikací pomocí .NET`.
2. Stáhněte a nainstaluje `.NET Framework 4.8` ze stránky `https://dotnet.microsoft.com/en-us/download/dotnet-framework/net48`.
3. Po nainstalování obou nástrojů, otevřete soubor `Server/DigitalSandboxServer.sln`. Soubor by se měl otevřít v aplikaci `Visual Studio`. Spolu s ostatními projekty.

4. Následně v okně `Build/ConfigurationManager` změňte typ konfigurace z `Debug` na hodnotu `Release`. Pokud je tato volba již zvolena, tento krok přeskočte.
5. Zavřete okno a stiskněte možnost `Build/Rebuild Solution`.
6. Nakonec spusťte skript s názvem `buildServer.bat`, který musí být umístěný ve stejném adresáři jako složka s projektem `DigitalSandboxServer`.
7. Tím vznikne adresář `SERVER`, který obsahuje spustitelnou verzi aplikace.
8. Pro spuštění stačí otevřít soubor `DigitalSandboxServer.exe`.

A.2.2 Klient

Tato sekce obsahuje návod na sestavení klientské aplikace.

1. Stáhněte a nainstalujte aplikaci `Visual Studio`, která je dostupná na stránkách `https://visualstudio.microsoft.com/cs/`.
    - Při instalaci zvolte možnost instalovat nástroje pro `Vývoj her pomocí Unity`.
2. Stáhněte si aplikaci `Unity Hub`, která je dostupná na stránkách `https://unity.com/download`.
3. Po stažení si aplikaci nainstalujte a spusťte.
4. Až se aplikace spustí, zvolte možnost `Installs` a následně volbu `Install Editor`.
5. Vyberte verzi `Unity 2021.3.9f1`.
6. Klikněte na tlačítko `Continue`. Pak stiskněte tlačítko `Install` a počkejte, až se vybraná verze nainstaluje.
7. Po dokončení instalace přejděte do záložky `Projects` a klikněte na tlačítko `Open` nebo `Add project from disk`.
8. Najděte cestu k projektu s vytvořenou hrou. Poté by se měl projekt s příslušným názvem zobrazit mezi ostatními projekty.
9. Pokud se daný projekt neotevře automaticky, otevřete ho.
10. Po spuštění aplikace otevřete scénu ze souboru `Scenes/DigitalSandboxScene.unity`.

11. Pro kompilaci klikněte na možnost `File/Build Settings`.
12. Zvolte platformu `Windows, Mac, Linux` a klikněte na tlačítko `Build`.
13. Vyberte cílový adresář a vyčkejte na vytvoření.
14. Sestavenou verzi hry následně naleznete ve vybraném adresáři.
15. Pro spuštění hry stačí otevřít soubor `Digital_Sandbox.exe`, který se nachází ve vytvořeném adresáři.

A.2.3 VR aplikace

Tato sekce obsahuje návod na sestavení VR aplikace. Pro kompilaci VR aplikace postupujte podle návodu A.2.2. U této aplikace se postup liší a to následovně:

1. Místo verze `Unity 2021.3.9f1` vyberte verzi `Unity 2022.2.11f1.`
2. Otevřete okno `Edit/Project Settings` a zkontrolujte, zdali
   - je v sekci `XR Plug-in Management` vybrána možnost `OpenXR`.
   - je v sekci `OpenXR` přidaný profil `HTC Vive Controller Profile`.
3. Následně postupujte jako v klientské aplikace. (tj. bod 11.)
4. Pro spuštění zkontrolujte nastavení připojeného headsetu.

Poznámka - VR aplikace byla otestována pouze se zařízením `HTC Vive`.

A.3 Nasazení

Nasazení aplikace se provádí v grafické laboratoři na KIVu. Postup je následující:

1. Zkompilujte server a klientskou aplikaci pomocí návodů v sekci A.2.
2. Po zkompilování obou aplikací spusťte skript s názvem `buildServer.bat` a následně `buildClient.bat`.
   - Skript `buildServer.bat` musí být spuštěn v adresáři, kde se nachází projekt `DigitalSandboxServer` (obsahem této složky je soubor `DigitalSandboxServer.sln`).
   - Při spuštění skriptu `buildClient.bat` budete požádání o zadání cesty k vytvořenému projektu. Spuštění musí probíhat ve stejném adresáři jako spuštění skriptu `buildServer.bat`.

3. Po spuštění obou skriptů by mělo dojít k vytvoření složek - `SERVER` a `CLIENT`.
4. Zkontrolujte, že jsou obě složky ve stejném adresáři. Pokud ne, přesuňte je do stejného adresáře.
5. Obě složky nahrajte na USB flash disk a přesuňte na pracovní plochu stroje *SandyStation*.
6. Zkuste restartovat počítač *SandyStation* a aplikace by se měla zapnout po opětovném spuštění počítače.

Pokud nedojde ke spuštění aplikace, postupujte následovně:

1. Vytvořte zástupce skriptu `run.bat`, který by se měl nacházet na pracovní ploše.
2. Stiskněte klávesouvou zkratku `Windows` + `R` a zadejte `shell:startup`
3. Přesuňte vytvořený soubor zástupce do okna, které se otevře.
4. Restartuje počítač, a aplikace by se měla spustit po zapnutí počítače.

A.4 Vytvoření nového zásuvného modulu

Sekce popisuje způsob, kterým lze vytvořit nový zásuvný modul a přidat ho k serveru.

1. Vytvořte nový projekt v aplikaci `Visual Studio`.
2. Po inicializaci projektu, klikněte pravý tlačítkem na položku projektu.
3. Zvolte možnost `Add Reference` a najděte knihovnu `DigitalSandboxCore.dll`, která by měla být obsahem aplikace `DigitalSandboxServer`.
4. Po přidání závislosti, vytvořte novou třídu, která bude reprezentovat zásuvný modul.
5. Pro implementaci zásuvného modulu je nutné, aby třída implementovala dané rozhraní. Jednotlivá rozhraní jsou obsahem knihovny `DigitalSandboxCore.dll`.
6. Implementujte potřebné metody daného rozhraní.
7. Po vytvoření a zkompilování zásuvného modulu, je potřeba přesunout vytvořenou knihovnu do složky serveru. Ta se nachází na pracovní ploše počítače *SandyStation*. Konkrétně složka s názvem `SERVER`.

8. Nakonec je potřeba přidání identifikačních informací do konfiguračního souboru `plugin.config.json` (tj. název knihovny a název modulu), který se nachází ve složce `SERVER`.

Tímto způsobem je aplikace serveru připravena využívat vytvoření plugin.

A.5 Vytvoření nového klienta

Pro vytvoření nového klienta lze postupovat následovně:

1. Vytvořte nový projekt v herním enginu Unity.
2. Po vytvoření projektu zvolte možnost `Assets/Import Package/Custom Package`.
3. Najděte soubor s názvem `DigitalSandbox.unitypackage`, který obsahuje vytvořené prefaby, textury, atd.
4. Pro vytváření nové aplikace lze použít vytvořené prefaby z daného balíčku.
5. Následně je již implementace plně v režii vývojáře.

A.6 Konfigurace

Sekce popisuje, jaké jsou možnosti konfigurace jednotlivých aplikací.

A.6.1 Server

Pro konfiguraci serveru existují dva konfigurační soubory - `plugin.config.json` a `App.config`. Pokud je již aplikace zkompilována, název souboru `App.config` je upraven na `DigitalSandboxServer.exe.config`.

1. `plugin.config.json` - slouží pro konfiguraci modulů, které server používá. Pro přidání nového pluginu je potřeba uvést jméno knihovny a následně jméno třídy, která reprezentuje zásuvný modul. Zároveň je možné měnit pořadí jednotlivých modulů, které ovlivňuje pořadí, ve kterém se jednotlivé pluginy používají.
2. `App.config` - zde se nastavují ostatní hodnoty. Lze měnit číslo TCP portu, na kterém bude server naslouchat. Minimální a maximální vzdálenost, ve které budou snímány hodnoty pískoviště. Nakonec ještě konstanta, která upravuje filtr s nekonečnou odezvou.

A.6.2 Klientská aplikace

Pro konfiguraci vytvořených klientské aplikace slouží jeden konfigurační soubor `config.json`, který se nachází ve složce `StreamingAssets/`. Pokud je projekt otevřen v herním enginu Unity, nachází se složka přímo ve složce `Assets/` daného projektu. Pokud je již hra zkompilována, je složka uložena ve složce s příponou `Digital_Sandbox_Data`.

V souboru lze upravovat hodnoty kalibrace. Dále pak i IP adresu serveru a číslo TCP portu, na kterém server naslouchá. Pak se zde vyskytuje nastavení UDP portu, na kterém bude klient přijímat hloubková data. Nakonec lze měnit i výšku vody, aby se docílilo lepšího uživatelského zážitku.

A.7 Ovládání klienta

Sekce popisuje, jak ovládat a kalibrovat klientskou aplikaci.

1. Pro posouvání terénu použijte klávesy `WSAD`.
2. Pro zvětšování a změnšování terénu použijte šipky.
3. Pokud při kalibraci budete současně držet i tlačítko `Ctrl`, kalibrace bude probíhat rychleji.

Bibliografie

1. EPP, Rain; LIN, Dayi; BEZEMER, Cor-Paul. An Empirical Study of Trends of Popular Virtual Reality Games and Their Complaints. *IEEE Transactions on Games*. 2021. Dostupné z DOI: 10.1109/TG.2021.3057288.
2. HRMA, Jiří. *Sandystation: Interaktivní pískoviště s využitím Kinectu*. 2011-11. Dostupné také z: https://smartmania.cz/sandystation-interaktivni-piskoviste-s-vyuzitim-kinectu-video-1672/.
3. *Sandy Station*. Dostupné také z: https://www.sandystation.com/.
4. MICROSOFT. *What is .NET framework? A software development framework.* Dostupné také z: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework.
5. NITESCU, Daria. *Evaluation of pointing strategies for Microsoft Kinect Sensor device: Final project report*. 2012. Dis. pr.
6. MILES, Rob S. *Start here!: Learn the kinect API*. Microsoft, 2012. ISBN 978-0-7356-6396-1.
7. WEBB, Jarrett; ASHLEY, James. *Beginning Kinect programming with the Microsoft Kinect SDK*. APress, 2011. ISBN 978-1-4302-4104-1.
8. ALTMAN, Petr. *Using MS Kinect Device for Natural User Interface*. 2013. Dis. pr. Západočeská univerzita v Plzni.
9. JEAN, Jared St. *Kinect hacks*. O'Reilly, 2013. ISBN 978-1-4493-1520-7.
10. KEAN, Seanand; HALL, Jonathan; PERRY, Phoenix. *Meet the Kinect: An Introduction to Programming Natural User Interfaces*. Apress, 2011. ISBN 978-1-4302-3888-1.
11. *Kinect for Windows SDK 1.8*. Dostupné také z: https://learn.microsoft.com/en-us/previous-versions/windows/kinect-1.8/hh855347(v=ieb.10).
12. *Kinect for windows SDK*. Dostupné také z: https://www.microsoft.com/en-us/download/details.aspx?id=40278.

13. CRUZ, Leandro; LUCIO, Djalma; VELHO, Luiz. *Kinect and RGBD images: Challenges and applications*. IEEE, 2012.
14. AWATI, Rahul. *What is field of view (FOV)?* TechTarget, 2022. Dostupné také z: https://www.techtarget.com/whatis/definition/field-of-view-FOV.
15. ABREGO, por Maleny. *Sabías que Kinect para windows puede escuchar y puede verte?* 2014. Dostupné také z: https://malenyabrego.wordpress.com/2012/10/22/sabias-que-kinect-para-windows-puede-escuchar-y-puede-verte/.
16. *Kinect in infrared*. 2019. Dostupné také z: https://bbzippo.wordpress.com/2010/11/28/kinect-in-infrared/.
17. *Virtual reality - latest virtual reality news*. 2017. Dostupné také z: https://www.vrs.org.uk/.
18. BOAS, Yuri Antonio Gonçalves Vilas. Overview of Virtual Reality Technologies. *Computer Science*. 2012.
19. ISAR, Cosmina. A glance into virtual reality development using Unity. *Informatica Economica*. 2018, roč. 22, č. 3/2018, s. 14–22. Dostupné z DOI: 10.12948/issn14531305/22.3.2018.02.
20. ŽÁRA, Jiří; FELKEL, Petr; BENEŠ, Bedřich. *Moderní počítačová grafika*. Computer Press, 1998.
21. *Augmented reality (AR) vs. Virtual Reality (VR): What's the difference?* Dostupné také z: https://www.pcmag.com/news/augmented-reality-ar-vs-virtual-reality-vr-whats-the-difference.
22. LAVIERI, Edward. *Getting started with unity 2018 a beginner's guide to 2D and 3D game development with unity*. Packt, 2018.
23. ARM. *What is a gaming engine?* Dostupné také z: https://www.arm.com/glossary/gaming-engines.
24. *Unity*. Dostupné také z: https://unity.com/.
25. CHMELÍK, Tomáš. *Videomapping v Unity*. 2016. Dis. pr. Západočeská univerzita v Plzni.
26. HOCKING, Joseph. *Unity in action*. Manning, 2017.
27. WIGMORE, Ivy. *What is VR locomotion (virtual reality locomotion)?: Definition from TechTarget*. TechTarget, 2018-05. Dostupné také z: https://www.techtarget.com/whatis/definition/VR-locomotion-virtual-reality-locomotion.
28. *Client-server architecture*. Encyclopædia Britannica, inc. Dostupné také z: https://www.britannica.com/technology/client-server-architecture.

29. KALITA, Limi. Socket Programming. *International Journal of Computer Science and Information Technologies*. 2014, roč. 5, č. 3, s. 4802–4807.

30. *Slovník pojmů*. Dostupné také z: https://www.vodafone.cz/uzitecne-odkazy/slovnik-pojmu/plugin/.

31. PROKONZOLE.CZ. *X360 AC Adaptér pro Kinect*. Dostupné také z: https://www.prokonzole.cz/x360p/x360-ac-adapter-pro-kinect-nove/.

32. TANENBAUM, Andrew S.; WETHERALL, David. *Computer Networks, Fifth Edition*. Prentice Hall, 2010.

33. *Difference between unicast, broadcast and Multicast in computer network*. GeeksforGeeks, 2023-03. Dostupné také z: https://www.geeksforgeeks.org/difference-between-unicast-broadcast-and-multicast-in-computer-network/.

34. GUIXIANG, Chen. *What is multicast? multicast vs unicast?* 2021-09. Dostupné také z: https://info.support.huawei.com/info-finder/encyclopedia/en/Multicast.html.

35. *Newtonsoft.JSON*. Dostupné také z: https://www.newtonsoft.com/json.

36. *Socket programming in C#*. Dostupné také z: https://www.c-sharpcorner.com/article/socket-programming-in-C-Sharp/.

37. *Socket Třída (system.net.sockets)*. Dostupné také z: https://learn.microsoft.com/cs-cz/dotnet/api/system.net.sockets.socket?view=netframework-4.8.

38. PECINOVSKÝ, Rudolf. *Návrhové vzory: 33 vzorových postupů pro objektové programování*. Computer Press, 2007. ISBN 978-80-251-1582-4.

39. UNITY. *Unity user manual 2021.3 (LTS)*. Dostupné také z: https://docs.unity3d.com/2021.3/Documentation/Manual/index.html.

40. *Computer Graphics - 3D projection and visualization*. Dostupné také z: https://edirlei.com/aulas/cg-2021/CG_Lecture_04_Projection_Visualization_2021.html.

Seznam použitých zkratek

AR Augmented Reality

CIV Centrum informatizace a výpočetní techniky

FAV Fakulta aplikovaných věd

FEL Fakulta elektrotechnická

FOV Field of View

FPS First Person Shooter

GPU Graphics Processing Unit

GUI Graphic User Interface

IDE Integrated Development Environment

KIV Katedra informatiky a výpočetní techniky

NUI Natural User Interface

RGB Red Green Blue

SDK Software Development Kit

TCP Transmission Control Protocol

UDP User Datagram Protocol

USB Universal Serial Bus

VR Virtual Reality

WPF Windows Presentation Foundation

XR Extended Reality

Seznam obrázků

Seznam výpisů

Elektronické přílohy

Obsahem adresáře je několik složek. V tomto dokumentu je popsán obsah jednotlivých obsah a jednotlivých souborů. Více informací lze nalézt v textu diplomové práce (viz. složka `Text_prace`).

- `Aplikace_a_knihovny` - obsahuje zdrojové kódy všech vytvořených aplikací, skripty, atd.
  - `DigitalSandboxClient` - adresář s projektem klientské aplikace
  - `DigitalSandboxClient_Compiled` - adresář se spustitelnou klientskou aplikací
  - `DigitalSandboxServer` - adresář s projektem serveru
  - `DigitalSandboxVR` - adresář s projektem VR aplikace
  - `DigitalSandboxVR_Compiled` - adresář se spustitelnou VR aplikací
  - `Libraries` - Složka s použitými knihovnami
    * `Newtonjson` - knihovna `Newtonsoft.JSON`, která je použita v rámci klientské a VR aplikace
  - `DigitalSandboxClient.unitypackage` - Unity balíček prvků, které jsou exportovány z vytvořené klientské aplikace. Lze je použít pro vývoj dalších aplikací.
  - `buildClient.bat` - skript, který slouží pro přípravu složky CLIENT, kterou lze nasadit na počítač SandyStation
  - `buildServer.bat` - skript, který slouží pro přípravu složky SERVER, kterou lze nasadit na počítat SandyStation
  - `run.bat` - soubor pro spuštění serveru a klientské aplikace ze složek SERVER a CLIENT. Slouží pro spuštění aplikace na počítači SandyStation.
- `Poster` - obsahuje plakát v různých formátech
  - `Vachal_Vojtech_2023.pub` - plakát ve formátu .PUB

  - `Vachal_Vojtech_2023.pdf` - plakát ve formátu .PDF

- `Text_prace` - obsahuje zdrojové LaTeXsoubory a výsledný text diplomové práce ve formátu PDF
  - `Vachal_Vojtech_DP_Text.PDF` - výsledný text práce
  - `Latex` - obsahuje LaTeXprojekt
    * `img` - složka s použitými obrázky
    * `chapters` - zdrojové kódy jednotlivých kapitol
    * a další