ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra Kybernetiky

Bakalářská práce

Výuková aplikace s rozšířenou realitou ve vnitřním prostředí

Adam Velemínský
velemada@fel.cvut.cz

Duben 2018
Vedoucí práce: Ing. Ivo Malý, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Velemínský** Jméno: **Adam** Osobní číslo: **457243**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Výuková aplikace s rozšířenou realitou ve vnitřním prostředí

Název bakalářské práce anglicky:

Augmented Reality Application for Education in Indoor Environment

Pokyny pro vypracování:

Analyzujte možnosti tvorby aplikací pro výuku v rozšířené realitě (AR) ve vnitřním prostředí. Zaměřte se na porovnání technik určení pozice pomocí markerů nebo bez jen z obrazu. Dále analyzujte strukturu jednoho výukového místa (úkolu), jeho zapojení do procesu řešení - nalezení úkolu/markeru, ověření realizace úkolu, provedení úkolu a ukončení úkolu. Na základě analýzy navrhněte a implementujte výslednou aplikaci na platformě Unity3D pro mobilní zařízení se systémem Android. Úkoly a jejich definice bude aplikace získávat ze serveru přes definované rozhraní, implementace serveru není součástí této práce. Výslednou aplikace otestujte na alespoň 2 testovacích výukových místech. Při vývoji provádějte průběžné vyhodnocování s uživateli.

Seznam doporučené literatury:

[1] J. Linowes, K. Babilinski, Augmented Reality for Developers: Build Practical Augmented Reality Applications with Unity, Arcore, Arkit, and Vuforia. 2017. Packt Publishing.
[2] J. Hocking, Unity in Action: Multiplatform Game Development in C# with Unity 5. 2015, Manning Publications.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ivo Malý, Ph.D., katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.01.2018** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Ivo Malý, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování / Prohlášení

Rád bych zde poděkoval vedoucímu mé bakalářské práce Ing. Ivo Malému Ph.D za cenné rady, připomínky a poskytnuté konzultace při tvoření této bakalářské práce.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 24.5.2018

.....

Abstrakt / Abstract

Tato bakalářská práce se zabývá vývojem Výukové aplikace s rozšířenou realitou pro mobilní systém Android. Aplikace je vyvíjena v herním enginu Unity3D s rozšířením Vuforia, poskytující rozšířenou realitu. V práci jsem nejdříve analyzoval technologii rozšířené reality a porovnal Markery pro určování polohy. Následně jsem vytvořil konkrétní návrh výukové aplikace a jeho implementaci. V závěru jsem tuto aplikaci otestoval s pomocí uživatelů.

Klíčová slova:

rozšířená realita, Unity3D, Vuforia, Android aplikace, marker

This Bachelor thesis deals with developing Learning aplication with Augmented reality for Android. Aplication is developed in game engine Unity3D with extension Vuforia, which provides Augmented reality. At first I analyzed Augmented reality technology and compared Markers for positioning. After that, I created a specific design and implementation of learning aplication. In the end I tested this aplication with users.

Keywords:

Bachelor thesis, Augmented reality, Unity3D, Vuforia, Android aplication, Marker

Obsah /

Kapitola 1
Úvod

V dnešním světě, kdy digitální technologie přinášejí stále nové možnosti, se staly upravené reality velice populárními. V těchto upravených realitách vznikají stále nové a komplexnější aplikace pro všechna možná zařízení. Jednou z těchto technologií pro úpravu reality je tzv. „rozšířená realita“ (Augmented Reality), která bude předmětem této práce. Tato technologie se za posledních pár let ohromně vyvinula a aplikace využívající tuto technologii se začínají využívat ve spoustě pracovních sektorech jako je marketing, reklama, různé hry, ale také vzdělávání.

Vzdělávací aplikace využívající rozšířenou realitu je schopna poskytnout uživateli spoustu informací a to zajímavou formou. Místo čtení spostu informací z učebnic, o například nějaké budově, jsme si schopni s pomocí aplikace s rozšířenou realitou prohlédnou přímo model této budovy a prozkoumat ho ze všech stran a úhlů. Učení je tak daleko záživnější a zábavnější.

1.1 Cíl práce

Cílem této bakalářské práce je analyzovat možnosti tvorby aplikace v rozšířené realitě. Porovnat různé druhy technik zobrazování v rozšířené realitě. Na základě analýzy následně navrhnout a implementovat výukovou mobilní aplikaci na platformě Android, využívající tuto technologii, která bude schopna zobrazit model i s informacemi o něm. Aplikace by měla obsahovat nějaké interaktivní prvky, které by uživateli poskytly různé možnosti s modelem pracovat a získávat nové informace.

Kapitola 2
Analýza

V této kapitole budeme analyzovat práci s rozšířenou realitou, řekneme si, jaké druhy aplikací s rozšířenou realitou existují. Dále zanalyzujeme nástroje a technologie, které je možné pro vývoj aplikací v rozšířené realitě použít.

2.1 Rozšířená realita

Rozšířena realita, anglicky Augmented Reality (AR), jak už název napovídá, nám obohacuje reálný obraz světa o různé digitální, počítačem vytvořené prvky a to vše probíhá v reálném čase a za použití pouze chytrého telefonu či tabletu. Jedná se o kombinaci počítačové grafiky s reálným světem. Pod pojmem počítačová grafika, se může skývat pouze jednoduchý text, ale také různé počítačové modely [1].

Mobilní aplikace s rozšířenou realitou nám kombinují počítačovou grafiku s videem, získaným z kamery mobilního telefonu. Aplikace rozšířené reality nám mohou například poskytnout informace o objektech, které se nachází v zorném poli kamery, či zobrazit digitální prvek dle nějaké předlohy se kterým můžeme dále nějakým způsobem pracovat.

Rozšířená realita se stala velice populární v roce 2016, kdy vyšla hra Pokémon GO, která tuto technologii využívá. Ukázku hry si můžeme prohlédnout na obrázku **2.1**.

Obrázek 2.1. Aplikace Pokemon GO využívající rozšířenou realitu [1]

2.1.1 Rozdíl mezi rozšířenou realitou a virtuální realitou

Lidé si často pletou pojmy rozšířená a virtuální realita, nyní si zde objasníme jaký je mezi nimi rozdíl.

Virtuální realita je počítačově vytvořená simulace prostoru, která vytváří iluzi skutečného, či fiktivního světa. V této realitě se uživatel může pohybovat a interagovat v ní

pomocí přístrojů. Rozšířená realita, jak jsme se již v předchozí podkapitole dozvěděli, nám kombinuje realitu s virtuálními prvky a uživatelé jsou stále schopni vnímat reálný svět, což u virtuální reality není možné [2].

2.1.2 Způsob určování pozice u aplikací s rozšířenou realitou

Aby byla aplikace s rozšířenou realitou funkční, musí mít přístup k informacím o pozici zařízení, na kterém funguje. Kdyby aplikaci data o pozici nebyla poskytnuta, nevěděla by, jaké digitální prvky má zrovna vykreslovat.

Aplikace rozšířené reality nabízejí dva základní způsoby určování pozic pro vykreslení digitálních prvků. První způsob funguje pomocí určování polohy a druhý pomocí předlohy.

První druh aplikací vykresluje digitální prvky pomocí údajů získaných z GPS, kompasů a akcelerometrů mobilního zařízení. Aplikace je si tak schopna zjistit, kde se nachází a na jaký objekt se uživatel pomocí kamery zařízení zrovna dívá. Následně je například schopna stáhnout informace o daném objektu z internetu a zobrazit je v aplikaci, uživatel tak může zjistit mnoho informací, bez nutnosti hledat je na internetu. Na obrázku **2.2** se můžeme podívat na aplikaci, která nám na mobilním zařízení vykresluje informace o objektech nacházejícími se před námi.

Další druh aplikací využívá rozpoznávání objektů v obraze získaném z kamery zařízení. Po rozpoznání předlohy nám aplikace vykreslí digitální prvek. Jako předlohu je možné použít jakýkoli obrázek či model. Na obrázku **2.3** je vidět vykreslení modelu na základě předlohy.

Později si o těchto předlohách povíme více, jelikož naše výuková aplikace na nich bude založená.

Obrázek 2.2. Aplikace fungující pomocí GPS

Obrázek 2.3. Aplikace fungující pomocí předlohy

2.2 Vývojové prostředí Unity3D

Unity3D je multiplatformní herní engine primárně vytvořený pro tvorbu 3D a 2D videoher, který má celkem čtyři verze: Unity Personal, Unity plus, Unity Pro a Unity Enterprise. V našem projektu budeme používat verzi Unity Personal, sice ostatní verze nabízejí více funkcí, ale také jsou placené a verze zdarma nám pro naše účely bude stačit.

Unity dále umožňuje export na spoustu platforem kromě PC/MAC/LINUX a to na Android či IOS, čehož v této práci využijeme, protože budeme vytvářet aplikaci na mobilní systém Android. Ačkoli Unity3D slouží především pro tvorbu her, je v ní možné tvořit i jiné aplikace. V této práci budeme pracovat s verzí Unity3D 2017.2.0f3, což byla při vytváření aplikace nejnovější dostupná verze tohoto vývojového prostředí [3] [4].

2.2.1 Programování v Unity3D

V Unity se dá programovat více programovacími jazyky, jelikož při programování pracujeme s různými objekty, interaktivními prvky, tlačítky, animacemi a různými efekty, jsou používány objektově orientované jazyky. K dispozici jsou C#, JavaScript a Boo. Jednu aplikaci je dokonce možné tvořit více jazyky a skripty z různých jazyků pak kombinovat. V naší aplikaci budeme používat pouze programovací jazyk C# a skripty budeme psát pomocí programu Visual Studio, což je vývojové prostředí od Microsoftu [4].

C# je vysokoúrovňový objektové orientovaný programovací jazyk vyvinutý firmou Microsoft. Je založen na jazycích C++ a Java, má tedy podobnou syntaxi. Kromě tvorby her v prostředí Unity, lze C# využít k vytváření databázových programů, webových aplikací, stránek a služeb. Definice C# získána z [5].

2.3 Rozšíření Vuforia

Vuforia je rozšíření do vývojového prostředí Unity3D, vytvořené firmou Qualcomm, se kterým budeme při tvorbě naší aplikace pracovat. Jedná se o rozšíření, které umožňuje

práci s rozšířenou realitou. Tento Framework využívá kvalitní, stabilní a efektivní mechanismy pro počítačové rozpoznávání a sledování charakteristik (například objektů) v obraze, čili je schopen objekt v obraze identifikovat a rozpoznat. V předchozích verzích bylo nutné si stáhnout Vuforia SDK (Software Development Kit), neboli balíček vývojářských nástrojů, a naimportovat ho do našeho projektu v Unity. Nyní už není třeba nic externě importovat, jelikož nová verze Unity má tento Framework přímo v sobě. Vuforia umožňuje vytvářet aplikace s rozšířenou realitou pro operační systémy Android i IOS, naši aplikaci budeme vyvíjet a testovat pouze pro Android zařízení [6].

Jednou z nejdůležitějších komponent Vuforie, je Target Managment System. Tato komponenta zpracovává obrazy zachycené kamerou a porovnává je s těmi, co máme v databázi (naše předlohy), která se vytváří přímo na stránkách Vuforii. Pokud komponenta najde shodu, nastaví kameru do pozice ve scéně tak, aby odpovídala pozici telefonu vůči předloze v reálném světě. Dále nám spustí skript, který například může do scény vykreslit nějaký model [6].

2.3.1 Marker (předloha)

Jak již bylo zmíněno výše, naše aplikace bude fungovat na základě rozpoznávání objektů v zorném poli kamery zařízení. Tyto objekty se v anglickém jazyce nazývají „Markers“, v této práci je budeme nazývat předlohy. Jsou to ve většině případů 2D obrázky s texturou, která musí odpovídat pár základním pravidlům (nesmí se opakovat vzor, být jednobarevná). Obrázek, který slouží jako předloha, je nazýván „Image Target“ (dále v této práci jako obrazová předloha), by měl být různorodý, aby zde bylo hodně specifických bodů, dle kterých naše aplikace pozná natočení obrázku, popřípadě nebude vadit částečné zakrytí. Vhodný obrázek pro využití jako předlohu můžeme vidět na obrázku **2.4** [6].

Obrázek 2.4. Vhodný obrázek pro Image Target

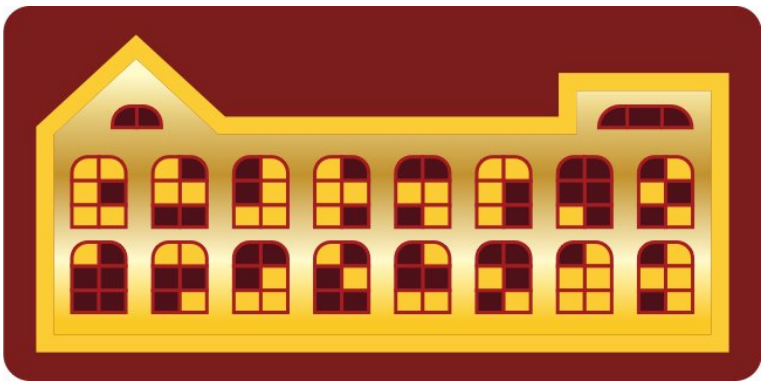
Kromě obrazových předloh lze jako objekt pro rozpoznání použít libovolný model (Model Target), objekt s válcovým a kuželovitým tvarem (Cylinder Target) nebo tzv. „VuMark“.

2.3.2 Tvorba databáze předloh

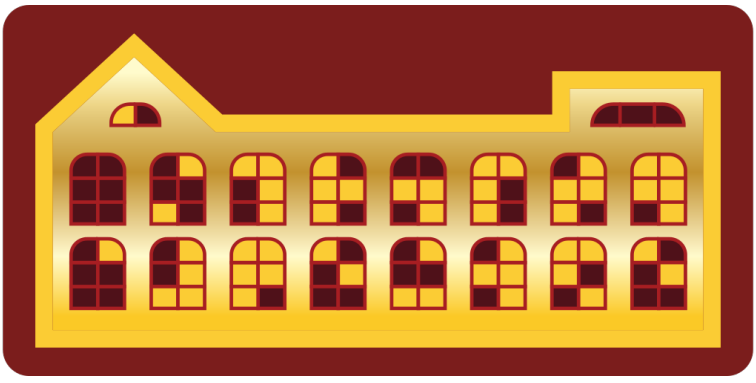
Vytváření předloh probíhá přímo na stránce Vuforii. Po zaregistrování a přihlášení do vývojářské části webu přejdeme do části Target Manager, kde probíhá vytváření databází různých obrazových, či speciálních předloh, které lze vyexportovat v jednom souboru ve formátu vhodném pro importování do Unity. Obrázek, či jiný objekt, zde stačí nahrát a následně vyexportovat ve formátu **unitypackage**, který lze jednoduše naimportovat do našeho projektu [6].

2.3.3 VuMark

VuMark je speciální druh markeru, vytvořený přímo pro užívání v rozšířené realitě. Jedná se o speaciální „bar code“, který je možné navrhnout a přizpůsobit tak, aby vypadal například jako logo aplikace a byl vhodný pro její využití. Další výhodou, kterou VuMark poskytuje, je velice dobré rozpoznávání kamerou zařízení, na kterém aplikace běží. Vlastní VuMark si lze vytvořit pomocí programu, jež je volně ke stažení na strákách Vuforie. V rámci této práce budeme využívat VuMark, který je volně ke stažení. Vytváření VuMark databáze probíhá stejným způsobem jako u klasických předloh. Při stahování VuMark v .png formátu pro vytištění nám Vuforia dává možnost zadat ID, náš VuMark nabízí možnost až deset znaků dlouhého, pomocí kterého se VuMark vygeneruje specifický, se stejným designem, obsahující zakódované ID. Na obrázcích **2.5** a **2.6** lze vidět. že se VuMark s odlišným ID se liší umístěním a množstvím specifických bodů. Tento VuMark má specifické body v podobě žlutých/hnědých čtverečků [6].



Obrázek 2.5. Výchozí VuMark



Obrázek 2.6. VuMark s ID „tower“

VuMark nám tedy dává možnost vytvořit si vlastní předlohu, která je nadesignovaná dle naší představy, pomocí které si jsme schopni vytvořit obrovské množství podobně vypadajících předloh s různým ID [6].

Kapitola 3
Návrh výukové aplikace

V této kapitole se budeme zabývat návrhem naší výukové aplikace. Představíme si, jak by měla naše aplikace vypadat, ukážeme si jednotlivé funkce, které máme k dispozici, rozebereme všechny možnosti a následně se rozhodneme, která bude pro naši aplikaci nejlepší. V další kapitole si zde uvedené věci naimplementujeme.

3.1 Image Target a VuMark

V předchozí kapitole jsme si představili různě druhy předloh, popsali jsme si jak fungují a jaké mají výhody. V náší aplikaci budeme pracovat s obrazovými předlohami a zároveň s předlohou VuMark, speciálním druhem, přímo od Vuforii. V aplikaci budeme pracovat s těmito dvěma druhy předloh, jelikož nejlépe splňují její požadavky a zároveň tak ukážeme, že naše aplikace dokáže pracovat s více druhy předloh najednou.

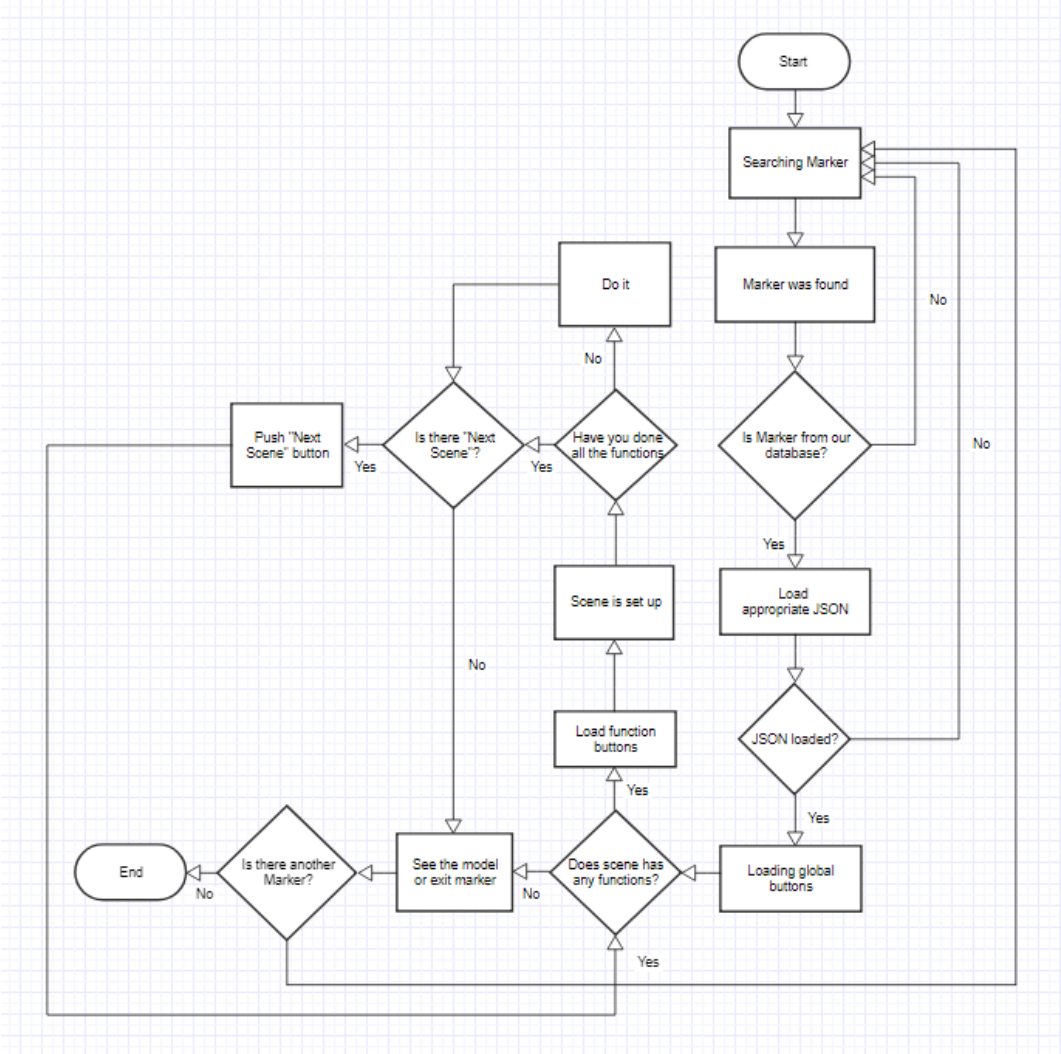
3.2 Základní charakteristika aplikace

V předchozí kapitole jsme si řekli, že aplikace bude fungovat na základě předlohy, ale zatím jsem si nepopsali, jak bude aplikace vypadat a co bude provádět po rozpoznání předlohy.

Po rozpoznání předlohy, aplikace v jeho okolí na mobilním zařízení zobrazí jeden, čí více modelů s různými funkcemi. Tyto funkce budou představovat tlačítka, která budou spouštět různé animace, sekvenci různých úkolů jako je třeba hledání různým částí modelu pomocí přibližování kamery. Dále zde může být tlačítko pro přepínání mezi modely s dalšími funkcemi. Po splnění všech úkolů (přehrání veškerých animací, nalezení nějakého počtu částí modelu), které určitá předloha bude obsahovat, nás aplikace upozorní na jeho splnění a pogratuluje nám. Na obrázku **3.1** můžeme vidět Flow Chart aplikace.

3.3 Načítání scénáře

Načítání scénáře po rozpoznání předlohy naším mobilním zařízením může fungovat více způsoby. Buď si můžeme v pragramu Unity3D vytvořit více scén a modely do každé vkládat ručně a po rozpoznání předlohy načíst danou scénu, či nám bude stačit pouze jedna scéna a danné modely budeme načítat až po rozpoznání předlohy.



Obrázek 3.1. Flow Chart aplikace

3.3.1 Aplikace s více scénami

Tato varianta načítání scénáře by fungovala tak, že po nalezení a rozpoznání předlohy, by aplikace v prostoru načetla jemu odpovídající scénu (scéna by měla stejné jméno, jako VuMark ID, či ID předlohy obrazové).

Načítání scénářů touto metodou má hned několik nevýhod. První nevýhodou je, že každá by se musela ručně vytvářet, což by s rozšiřováním aplikace o nové funkce a zvětšováním scény mohlo být velice pracné. Druhou nevýhodou je, že načítání nové scény a přepínání mezi nimi je náročnější, než pouhé načítání modelů v rámci jedné scény. V zadání této práce je, že aplikace má v budoucnu pracovat se serverem, čili musí data načítat až při běhu aplikace, z tohoto důvodu, je pro nás tato varianta nevyhovující.

3.3.2 JSON soubor

Dříve než si popíšeme variantu načítání scénáře s jednou scénou, popíšeme si tzv. JSON soubor, který při tomto druhu načítání budeme potřebovat.

JSON (JavaScript Object Notation), neboli JavaScriptový zápis objektů, je otevřený standardizovaný textový formát pro přenos dat, která mohou být organizována v polích nebo objektech. Jeho výhodou je přehledná struktura, takže se v něm lidé ihned vyznají

a je pro ně lehké ho tvořit, pokud znají základní pravidla. Zároveň je dobře čitelný i pro programy, takže se s ním jednodušše pracuje. Zápis dat je velice podobný zápisu dat v jazyce JavaScript, jak už název napovídá. Dále je JSON nezávislý na platformě a prostředí. Charakteristika JSON souboru přeložena z [7].

3.3.3 Aplikace s jednou scénou

Jak jsme si v předchozí kapitole zmínili, v této variantě načítání scénáře budeme potřebovat JSON soubor (nebo nějaký jiný soubor ve kterém lze dobře pracovat s daty), který bude obsahovat informace o tom, jak se mají objekty ve scéně načíst.

Tedy po nalezení a rozpoznání předlohy se nám načte JSON soubor, který bude obsahovat všechny potřebné informace o objektech majících se nacházet ve scéně. Na základě těchto informací se nám do scény vloží modely, tlačítka a přiřadí se všem objektům spravné metody. Výhodami tohoto řešení je rychlost načítání, vše probíhá téměř okamžitě a pro uživatele je tak práce s aplikací přívětivější, další výhodou je jednodušší tvoření scén, nemusejí se tvořit ručně, ale stačí vytvořit pouze JSON soubor s potřebnými informacemi a skripty programu se už postarají o jeho zrealizování. Program se tak stává více autonomní.

Z popisu obou variant je jasné, že v rámci této aplikace budeme používat variantu s jednou scénou, která splněje všechny naše požadavky, narozdíl od varianty druhé.

3.4 Stahování dat ze serveru

V zadání této práce je napsáno, že informace a potřebný obsah pro aplikaci má být stahován ze serveru. Avšak při prozkoumávání způsobů stahování dat ze serveru při běhu aplikace, se ukázalo, že možnosti nevyhovují naší aplikaci. Jediný způsob jak stáhnout data ze serveru je takový, že si vytvoříme tzv. „Asset Bundle“.

Asset Bundle je archivovaná kolekce všech možných objektů, dokonce i scén, které mohou být načteny za běhu aplikace. Tyto kolekce nejsou součástí buildu aplikace a většinou jsou uloženy na webovém serveru, odkud je aplikace stahuje. Avšak tvoření těchto kolekcí probíhá přímo v Unity, pomocí skriptu, který si musíme vytvořit, což je pro naše účely poměrně nepraktické. Po konzultacích tohoto problému s vedoucím práce, jsme se dohodli, že načítání dat ze serveru do této práce nebudeme zahrnovat a data budeme načítat z lokálního uložiště. Dalším zásadním důvodem k tomuto rozhodnutí bylo, že server, ze kterého se měly data stahovat, nebyl v době implementace aplikace hotov a implementace serveru není součástí této práce. Informace o archivovaných kolekcích a jejich stahování byly získany z dokumentace Unity3D [8] [9].

3.5 Načítání dat pomocí „Resource Folder“

Resource Folders jsou kolekce všech možných objektů, které mohou, stejně jako kolekce Asset Bundle, být načteny za běhu aplikace. Zásadní rozdíl oproti Asset Bundle je ten, že jeho kolekce jsou součástí buildu aplikace a jeho nesprávné užití může velice zvýšit velikost aplikace, což může mít za následek její špatný chod. Načítání touto metodou je poměrně snadné a s ohledem na možností, ji v naší aplikaci budeme používat [9].

3.6 Akce a události aplikace

V této kapitole se bude zabývat akcemi, které bude aplikace poskytovat. Většina spustitelných akcí budou navázáné na jim odpovídající tlačítka, která se nám ve scéně načtou při rozpoznání předlohy.

3.6.1 Animace

Jednou z akcí naší aplikace bude samozřejmě spuštění animace, pokud tedy model nějakou animací disponuje. Po načtení, bude vždy animace modelu vypnuta a pro její zapnutí bude muset uživatel kliknout na příslušné tlačítko s názvem „Play Animation“.

3.6.2 Zobrazení informací o modelu

Další akcí bude zobrazení základních informací o scéně. Po kliknutí na tlačítko se objeví okno uprostřed obrazovky, ve kterém v krátkem textu popíšeme důležité aspekty scény, což nejspíše budou její modely.

3.6.3 Hledání částí modelu

Nejzajímavější akcí této aplikace bude akce hledání. Tlačítko jež ji bude spoštět bude mít název „Let's Search“ a po jeho stisknutí se nám nějaká část modelu zvýrazní. Naším úkolem bude tuto zvárazněnou čast lokalizovat a přiblížit k ní kameru našeho mobilního zařízení pro její prozkoumání. Po přiblížení je nutné kameru neoddalovat po dobu jedné sekundy, pokud tuto dobu vydržíme, objeví se nám okno, které nám pogratuluje k nalezení hledané části a zaroveň v něm budou zobrazeny nějaké informace o této části modelu. Po nalezení určitého množství částí, nám aplikace pogratuluje ke splnění tohotu úkolu.

3.6.4 Ověření realizace úkolu

Tato funkce se sama spustí splní-li uživatel všechny úkoly aplikace (spustí všechny animace, najde dostatečný počet částí modelů a přečte si informace o modelu). Funkce nám vytvoří okno, ve kterém nám aplikace opět pogratuluje ke splnění scénáře a vyzve nás k vyzkoušení dalšího.

3.7 Tvorba scény

V předchozích kapitolách jsme si řekli, že načítání objektů do scény bude probíhat ve skriptu pomocí JSON souboru. Nyní si detailněji popíšeme strukturu a data tohoto JSON souboru.

Následující kód je návrh jeho struktury:

```
{
  "targetName":"nameOfScenario",
  "modelPath":"path/to/models",
  "scenes": [
     { "name":"first scene",
       "information":"Information about this scene"
       "modelsName":[ "model1" ],
       "models": [
        { "name":"model1",
          "parts":[ {"name":"part1", "info": "Information about part1"},
             {"name":"part2", "info": "Information about part2"} ],
          "scale":"int - scalingNumberOfModel",
          "rotation":[ "0", "90", "0"],
          "position":[ "0", "30", "0"],
          "animator":"animatorOfModel" }
         ],
       "taskToDoBoolean":false ,
```

```
        "sceneButtons": [
           { "name":"buttonToPlayAnimation", "functionID":"1",
             "position":[ "1", "1", "1"], "buttonPref":"prefab" },
           { "name":"buttonToShowInfo", "functionID":"2",
             "position":[ "2", "2", "2"], "buttonPref":"prefab" }
         ]
      }
  ],
  "buttonPrefPath":"path/to/prefab",
  "allScenesButtons": [
     { "name":"NextSceneButton", "functionID":"3",
       "position":[ "5", "5", "5"], "buttonPref":"prefab" }
  ]
}
```

Jelikož se všechny objekty co ve scéně budou, budou načítat z adresáře „Resource Folder“, tak musí náš JSON soubor obsahovat cestu k těmto objektům. Základní proměnné budou (zároveň si je můžeme prohlédnout v kódu): název scénáře (nejspíše shodný s ID předlohy), cestu k modelům, cestu k přednastavenému vzhledu tlačítek a pole proměnných popisující jednotlivé scény, kterých může scénář obsahovat více a pole proměnných popisující jednotlivá tlačítka. Scéna bude mít proměnné jako je jméno, informace o scéně, jména modelů ve scéně a potom pole popisující jednotlivé modely (polohu, rotaci, měřítko pro správnou velikost modelu) a pole popisující tlačítka (jméno, pozici, název objektu s přednastavenými vlastnostmi, atributy a vzhledem tzv. „Prefab“ (základní Prefab, který budeme v aplikaci používat, si můžeme prohlédnout na obrázku **3.2**) a ID funkce, ktera bude tlačítku přidělena). Pouze s těmito informacemi, bychom měli být schopni vytvořit jednoduchý scénář pro jednu předlohu.

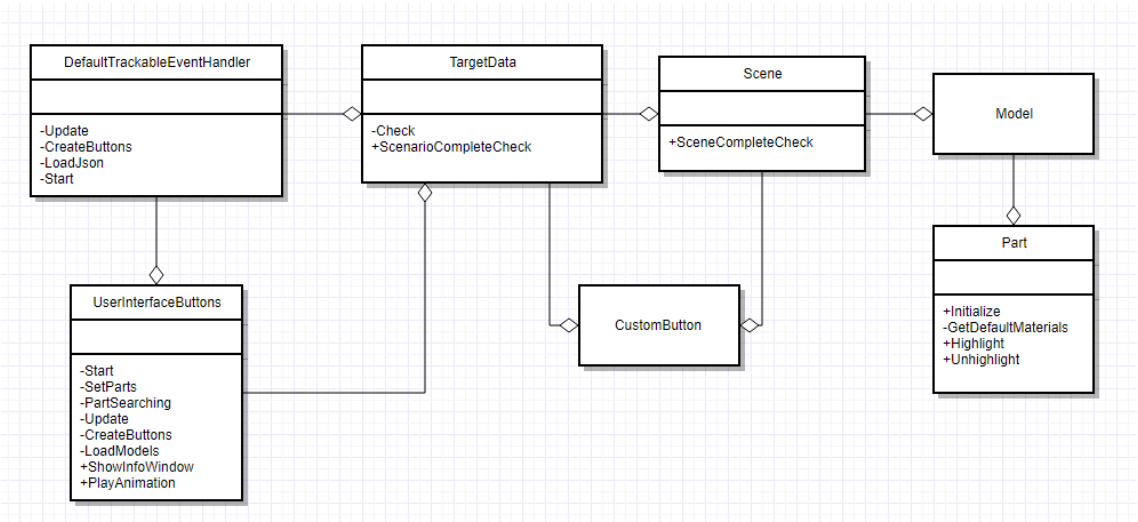
Obrázek 3.2. Button Prefab

Kapitola 4
Implementace

V této kapitole si popíšeme strukturu naší aplikace, různé třídy, které program obsahuje, využité technologie a důležité funkce, které obstarávají chod programu.

4.1 Struktura aplikace

Nyní si popíšeme jakou má naše aplikace strukturu. Vyjmenujeme si veškeré důležité skripty a třídy, které aplikace obsahuje a popíšeme si jejich účel. Struktura tříd je zobrazena na obrázku **4.1**.



Obrázek 4.1. Class Diagram aplikace

4.1.1 Default Trackable Event Handler

Default Trackable Event Handler je nejdůležitější skript v naší aplikaci. Každý nově vytvořený projekt rozšířené reality ho obsahuje, avšak můžeme si ho i upravovat. Všechny předlohy, které jsou ve scéně v Unity, tento skript obsahují a se spuštěním aplikace se spustí i tento skript. Jeho hlavní funkcí je rozpoznat všechny předlohy, které máme v databázi a zobrazit všechny objekty, které jsou ve scéně nastavené jako děti rozpoznané předlohy. Informace získány z [10].

V naší aplikaci však nebudeme mít ve scéně žádné objekty, které by se po rozpoznaní předlohy mohly rovnou zobrazit, jelikož v tu chvíli ještě nebudou ve scéně vůbec načtené. Tento skript tedy bude obstarávat rozpoznání ID předlohy, na základě které pak aplikace načte správný JSON soubor, pomocí kterého dále načte a zobrazí do scény správné globální objekty (objekty, které jsou ve všech scénách daného scénáře), kterým i přiřadí správnou funkci. Skript samozřejmě funguje pro oba druhy předloh, jež v naší aplikaci budeme používat - předloha obrazová a VuMark. V kapitole týkající se popisu našich vytvořených funkcí si podrobně popíšeme, jak v tomto skriptu vše funguje.

4.1.2 Třídy potřebné při načítání JSONu

Nejlepší způsob jak ukládat data z JSON soboru, je takový, že si vytvoříme třídy, které svou strukturou budou přesně odpovídat struktuře JSON souboru. Vytvořil jsem si tedy třídy TargetData, Scene, MyButton a Model. Každá třída má proměnné odpovídající struktuře JSON souboru.

Například proměnné ve třídě TargetData a Scene vypadají takto

```
public class TargetData {
    public string targetName = "";

    public string modelPath = "";
    public List<Scene> scenes = new List<Scene>();

    public string buttonPrefPath = "";
    public List<MyButton> allScenesButtons = new List<MyButton>();

    public bool scenarioComplete = false;
}
```

```
public class Scene {
    public string name = "";
    public string information;

    public string[] modelsNames = { "" };
    public List<Model> models = new List<Model>();
    public List<MyButton> sceneButtons = new List<MyButton>();

    public bool sceneComplete = false;
    public bool animationBool = true;
    public bool searchBool = true;
    public bool infoBool = true;
}
```

Z kódu je vidět, že třídy mají identické proměnné a pole proměnných jako má vzorový JSON soubor, předvedený v předchozí kapitole. Proměnné scenarioComplete a sceneComplete v JSON souboru nenajdeme, jsou to proměnné, které slouží k ověření realizace scény a scénáře

4.1.3 Třída obsarávající funkce aplikace

Nyní si něco povíme o třídě, která obsarává všechny uživatelské funkce aplikace. Třída se jmenuje UserInterfaceButtons, jak již název napovídá, funkce v této třídě slouží ke správnému fungování tlačítek ve scéně, obstarává i načítání a zobrazování všech objektů scény (většinou se bude jednat o modely a tlačítka).

Instanci této třídy bude obsahovat prázdný GameObject v naší scéně, díky tomu se nám skript spustí ihned po startu aplikace, dále tuto instanci ve scéně předáme i DefaultTrackableEventHandleru, který bude jeho funkce využívat a předávat mu informace z JSON souboru. Třída si ve svých privátních proměných uchovává informace o všech objektech v rámci scény, díky tomu vždy ví, které objekty jsou v dannou chvíli aktivní a zda na nich může zavolat funkce, jako je například animace.

Funkce, které tento skript obsahuje, si podrobně popíšeme v následujících kapitolách.

4.1.4 Canvas

Velice zásadní objektem v naší scéně, kromě předlohy a objektu obsahující naši třídu s funkcemi, je Canvas. Canvas je oblast, ve které by se měly nacházet všechny UI (User Interface - Uživatelské rozhraní) elementy, tedy všechny tlačítka, obrázky a jiné UI prvky, které chceme do scény vložit, musí být dětmi Canvasu [11].

V naší aplikaci, si Canvas vytvoříme přímo ve scéně, nastavíme mu velikost, jméno a tag, pomocí kterého ho následně ve skriptu jednodušše nalezneme a budeme do něj moci vkládat tlačítka a další prvky.

Cavas nám také nabízí tři různé způsoby renderování. První způsob renderování se nazývá „Screen Space - Overlay“ a renderuje nám UI prvky nad všemi ostatními objekty z pohledu naší kamery. Další způsob se nazývá „Screen Space - Camera“, který nám UI prvky renderu stejně jako Overlay, ale zde mohou UI prvky navíc mírně rotovat.

Poslední způsob se nazývá „World Space“, zde se UI prvky renderují stejně jako ostatní objekty ve scéně, nejsou vázané na kameru a ostatní objekty je mohou zakrývat [3].

V naší aplikaci budeme používat převážně World Space renderování, ale pro různá vyskakovací okna s informacemi, použijeme i Screen Space - Overlay.

4.2 Funkce pro načtení scény

V této kapitole si podrobně popíšeme naimplementované funkce, které obstarávají načítání scény a přípravu proměnných pro ostatní funkce aplikace.

4.2.1 Určení ID předlohy VuMark a předlohy obrazové

Jako první si popíšeme funkci, která rozpoznává ID našich markerů a na základě toho následně načítá JSON soubory. Kód této funkce vypadá následovně:

```
void Update() {
    StateManager sm = TrackerManager.Instance.GetStateManager();
    IEnumerable<TrackableBehaviour> activeTrackables = sm
        .GetActiveTrackableBehaviours();
    foreach (TrackableBehaviour tb in activeTrackables) {
        if (tb.TrackableName == "ImageTargetID") {
            if (SceneManager.GetActiveScene() != SceneManager
                    .GetSceneByName("SceneWeWant")) {
                SceneManager.LoadScene("SceneWeWant");
            }
            \\ Load JSON and Scene here
        }
    }

    vumark = null;
    foreach (var bhvr in mVuMarkManager.GetActiveBehaviours()) {
        vumark = bhvr.VuMarkTarget;
    }
    if (vumark != null) {
        String vumarkID = vumark.InstanceId.StringValue;
        vumarkID = vumarkID.Replace("\u0000", String.Empty).Trim();
        if (vumarkID.Equals("VuMarkID")) {
            if (SceneManager.GetActiveScene() != SceneManager
                    .GetSceneByName("SceneWeWant")) {
```

```
                SceneManager.LoadScene("SceneWeWant");
            }
            \\ Load JSON and Scene here
        }
    }
}
```

Veškeré rozpoznávání probíhá v metodě Update(), která je volána několikrát za vteřinu (záleží na FPS aplikace, klasicky tedy 60-krát).

V první části kódu se snažíme získat ID obrazové předlohy. Aplikace pomocí kamery skenuje objekty ve svém zorném poli. V případě, že se některý objekt bude shodovat s tím, který má aplikace v databázi našich obrazových předloh, v poli „activeTrackables“ se tento objekt objeví. Následně toto pole projdeme a porovnáme ID těchto objektů s ID našich obrazových předloh, pro které máme připravený scénář. Pokud se bude toto ID shodovat, načteme námi předpřipravenou scénu (tato scéna je nastavena jako výchozí, ale ve vyšších verzích aplikace, může být těchto scén více) a začneme načítat JSON soubor.

Druhá část kódu se věnuje získávání ID VuMarku. Tato část funguje na stejném principu jako část první, pouze používáme jiné jiné druhy proměnných pro získání VuMark ID.

4.2.2 Načítání JSON souboru

Funkce pro načítání JSON souboru je, jak jsme si již v předchozích kapitolách zmínili, ve skriptu „DefaultTrackableEventHandler.cs“. Kód funkce je následující:

```
private TargetData LoadJson(string path) {
    try {
        TextAsset targetFile = Resources.Load(path) as TextAsset;
        if (targetFile != null) {
            string contents = targetFile.text;
            TargetData targetData = JsonUtility
                .FromJson<TargetData>(contents);
            return targetData;
        } else {
            Debug.Log("File does not exist");
            return null;
        }
    } catch (System.Exception ex) {
        Debug.Log(ex.Message);
        return null;
    }
}
```

Tato funkce nám vrací TargetData, což je třída kterou jsme si vytvořili pro účel uchovávání dat z JSON souboru. Jako parametr funkce požaduje string s cestou k JSON souboru v našem adresáři „Resource Folder“. Funkce se nejprve pokusí načíst JSON soubor, pomocí metody „Resources.Load()“, jako čistý text, pokud bylo načtení úspěšné, tak pomocí metody „JsonUtility.FromJson“ a načteného JSON souboru vytvoříme náš objekt TargetData. V případě, že se načtení nepovede, nám funkce vrátí hodnotu null, a v případě, že nastane nějaká chyba, vyloguje se nám do konzole a opět nám funkce vrátí null hodnotu.

4.2.3 Načítání globálních tlačítek

Tato funkce nám načte a zobrazí do scény všechna tlačítka, která se mají zobrazovat dokud je nynější scénář aktivní. Kód pro načítání je:

```
void createButtons(List<MyButton> buttons, string buttonPrefsPath) {
   foreach (MyButton button in buttons) {
       GameObject buttonPrefab = Resources
           .Load<GameObject>(buttonPrefsPath + button.buttonPref);
       if (buttonPrefab != null && myCanvas != null) {
           RectTransform transform = myCanvas
               .GetComponent<RectTransform>();
           GameObject currentButton = Instantiate(buttonPrefab)
               as GameObject;
           currentButton.transform.SetParent(transform);
           currentButton.transform.eulerAngles = new Vector3(90, 0, 0);
           currentButton.transform.position = new Vector3(
               (float)button.position[0], (float)button.position[1],
               (float)button.position[2]);
           currentButton.GetComponentInChildren<Text>().text = button
               .name;
           if (button.functionID == 1) {
               currentButton.GetComponent<Button>().onClick
                 .AddListener(() => interfaceClass.someButtonFunction());
           }
           interfaceClass.globalButtons.Add(currentButton);
       }
   }
}
```

Ačkoli se tato funkce může zdát na první pohled nepřehlednou, není tomu tak. V této funkci poprvé využíváme data získaná z JSON souboru, konkrétně cestu k přednastaveným vzhledům tlačítek (tzv. „Prefab“) a parametry pro nastavení tlačítek, jako je jejich název, funkce a pozice. V metodě pouze projdeme všechny tlačítka, která se v JSON souboru nachází, vytvoříme kopii přednastaveného vzhledu, kterou umístíme do scény pod náš Canvas, aby se tlačítka mohla renderovat. Dále tlačítku nastavujeme rotaci, pozici, jméno a na závěr jejich funkci.

Pomocí velice podobné funkce se budou tvořit i tlačítka v rámci jednotlivých scén, pouze se bude volat za jiných okolností, ne hned při načtení JSON souboru.

4.2.4 Načítání modelů

Tato funkce se nachází ve třídě UserInterfaceButtons. Její funkcí je načíst a zobrazit všechny modely do danné scény.

```
private void LoadModels(List<Model> currentModels, string modelPath) {
   foreach (Model model in currentModels) {
      GameObject modelPref = Resources
         .Load<GameObject>(modelPath + model.name);
      if (modelPref != null) {
         GameObject loadedModel = Instantiate(modelPref) as GameObject;
         loadedModel.transform.localScale = new Vector3(
            (float)model.scale, (float)model.scale, (float)model.scale);
         loadedModel.transform.eulerAngles = new Vector3(
            (float)model.rotation[0], (float)model.rotation[1],
```

```
            (float)model.rotation[2]);
         loadedModel.transform.position = new Vector3(
            (float)model.position[0], (float)model.position[1], \
            (float)model.position[2]);
         // if model has animation
         if (model.animator != "none") {
            Animator animator = loadedModel.gameObject
               .GetComponent<Animator>();
            animator.runtimeAnimatorController = Resources.Load(modelPath
               + model.animator) as RuntimeAnimatorController;
         }
         loadedModel.SetActive(true);
         loadedModel.name = (loadedModel.name).Replace("(Clone)", "");
         this.models.Add(loadedModel);
      }
   }
   if (models != null && models.Count != 0) {
      SetParts(targetData.scenes[currentSceneInx].models);
   }
}
```

Tato metoda fuguje velice podobně jako ta pro načítání a zobrazování tlačítek, procházíme pole modelů z JSON souboru, načteme model z adresáře „Resource Folder“ a vytvoříme si jeho kopii a dále nastavujeme jeho atributy. Změnou oproti načítání tlačítek je, že zde modelu ještě přidáváme i tzv Animator, pokud má model animace, který spravuje animace modelu. Po načtení a zobrazení model přidáme do pole všech aktivních modelů, které budeme využívat v dalších funkcích. Na posledním řádku voláme funkci SetParts kterou si popíšeme v další kapitole.

4.3 Funkce pro akce a události

V návrhu aplikace jsme si popsali, jaké akce bude naše aplikace poskytovat. Nyní si ukážeme a popíšeme realizaci návrhů jež jsem si představili.

4.3.1 Animace

Tato funkce spustí animace všech modelů nacházejících se ve scéně. Kód pro jejich spuštění je následující:

```
public void playAnimation(string[] animation) {
    targetData.scenes[currentSceneInx].animationBool = true;
    if (currentAnimator != null) {
        if (!animationPlaying) {
            currentAnimator.Play(animation[rand.Next(animation.Length)]);
            animationPlaying = true;
        } else {
            currentAnimator.Play("default_state");
            animationPlaying = false;
        }
    }
}
```

V této verzi aplikace umí aplikaci pouštět animace pouze na jednom modelu. Funkce se zavolá vždy při stisknutí odpovídajícího tlačítka (Play Animation), dostane při jejích

zavolání pole textových retězců, obsahující názvy animací, kterou se mohou spustit. Následně zjistíme zda má model Animator (v podstatě zjišťujeme zda mají animaci) a pustíme náhodně jednu z dostupných animací. V případě, že je animace již puštěná, stisknutím tlačítka jí vypneme.

4.3.2 Informace o modelu

Jak jsme si v návrhu řekli, aplikace bude mít tlačítko pro zobrazení informací o modelu ve scéně. Tyto informace získámě z JSON souboru a kód pro jejich zobrazení je následující:

```
Rect infoWindow = new Rect();

w = 0.4F;
h = 0.3F;

infoWindow.x = (Screen.width * (1 - w)) / 2;
infoWindow.y = (Screen.height * (1 - h)) / 2;
infoWindow.width = Screen.width * w;
infoWindow.height = Screen.height * h;
```

Nejdříve si musíme nastavit vyskakovací okno, zde si ho inicializujeme, nastavíme si jeho pozici (v našem případě na střed) a jeho velikost vůči displeji zařízení na kterém aplikace spuštěná.

```
public void ShowInfoWindow() {
    showWindow = !showWindow;
    targetData.scenes[currentSceneInx].infoBool = true;
}
```

Tuto metodu voláme vždy při stisknutí tlačítka pro zobrazení informací (Info Window) a jediné co se zde děje, je změna booleanu na hodnotu true čí false. Dále si pro zobrazení GUI musí vytvořit speciální metodu nazývanou OnGUI, které nám kontroluje zda má výše zmíněný boolean hodnotu true a v případě že ano, změníme velikost pismen (na mobilním zařízení je třeba mít větší písmena vzhledem k malému displeji) a vykreslí pomocí metody WindowProc informace o scéně.

```
void OnGUI() {
   GUI.skin.label.fontSize = GUI.skin.box.fontSize
     = GUI.skin.button.fontSize = 20;
   if (showWindow) {
      infoWindow = GUILayout.Window(0, infoWindow, WindowProc,
         "Information about " + targetData.scenes[currentSceneInx].name);
   }
}

void WindowProc(int windowID) {
    GUILayout.Label(targetData.scenes[currentSceneInx].information);
}
```

4.3.3 Hledání částí modelu

Nyní si vysvětlíme dříve zmíněnou funkci SetParts. Tato funkce nám vytváří pole vlastních objektů nazývajících se Part, popisující části modelu. V tomto objektu si uchováváme proměnné jako je název části, její GameObject a hlavně její původní materiál,

ale zároveň zde je metoda pro zvýrazňování, které využíváme při akci „Hledání částí modelu“. Následující kód představuje třídu Part se všemi jejími funkcem.

```
public class Part{
   public GameObject model;
   public Material[] defaultMaterials;
   public string name;

   public Part(string partName) {
      name = partName;
      model = GameObject.Find(name);
      GetDefaultMaterials();
   }

   private void GetDefaultMaterials() {
      Renderer[] tempRenders = model.GetComponentsInChildren<Renderer>();
      defaultMaterials = new Material[tempRenders.Length];
      for (int i = 0; i < tempRenders.Length; i++) {
         defaultMaterials[i] = Materia
            l.Instantiate(tempRenders[i].material);
      }
   }

   public void Highlight() {
      foreach (Renderer r in model.GetComponentsInChildren<Renderer>()) {
         Material m = r.material;
         m.color = Color.red;
         r.material = m;
      }
   }

   public void Unhighlight() {
      for (int i = 0; i < defaultMaterials.Length; i++) {
         model.GetComponentsInChildren<Renderer>()[i].material
            = defaultMaterials[i];
      }
   }
}
```

Zvýrazňování částí obstarává metoda Highlight. Tato metoda projde všechny podčásti, které se renderují zvlášť a všem jim změní barvu materiálu na červenou a tím vytvoří efekt zvýraznění. Je velice důležité, abychom zde měli proměnou defaultMaterials, pomocí které jsem schopni vrátit části původní vzhled. Vracení původního materiálu probíha v metodě Unhighlight.

Zde je kód funkce SetParts.

```
public void SetParts(List<Model> models) {
    parts = new List<Part>();
    foreach (Model model in models) {
        List<Part> partsNames = model.parts;
        foreach (Part currentPart in partsNames) {
            currentPart.Initialize();
            parts.Add(currentPart);
        }
```

```
    }
}
```

Metoda pouze projde všechny aktivní modely ve scéně a vytvoří pole zinicializovaných objektů Part.

Nyní, když jsem si představili zvýrazňování částí, můžeme si představit další navazující metodu a tou je PartsSearching.

```
private void PartsSearching() {
   if (foundObjects == 3) {
      foundEnough = true;
      searching = false;
      StartCoroutine(FoundEnough());
      foundObjects = 0;
   }
   if (searching) {
      if (pickedPartToSearch) {
         if (!partFound) {
            float distance = CalcDistance(partToSearch);
            if (distance < 50) {
               distanceCount++;
               if (distanceCount > 60) {
                  partFound = true;
                  foundObjects += 1;
                  StartCoroutine(partWasFound());
               }
            } else {
               distanceCount = 0;
            }
         }
      } else {
         partToSearch = parts[rand.Next(parts.Count)];
         partToSearch.Highlight();
         pickedPartToSearch = true;
      }
   } else {
      if (pickedPartToSearch) {
         partToSearch.Unhighlight();
         pickedPartToSearch = false;
      }
   }
}
```

Tato metoda se volá několikrát za vteřinu, jelikož je umístěna v již zmíněné funkci Update, ale funkce hledání probíhá pouze v případě, když má boolean searching hodnotu true (hodnotu true, dostane v případě zmáčknutí tlačítka „Lets Search“). V případě že hledáme, si nejdříve náhodně vybereme část, kterou zvárazníme, pomocí random integer generátoru, tím se nám změní hodnota booleanu pickedPartToSearch na true. Následně počítáme vzdálenost mezi kamerou a hledanou částí a v případě, že je hodnota vzdálenosti větší než 50 (experimentálně určená hodnota) začne se zvětšovat hodnota čísla distanceCount zhruba o 60 za vteřinu. Ve chvíli kdy distanceCount překročí číslo 60, uznáme část za nalezenou. proměnná distanceCount nám zajišťuje, že kamera musí být v blízkosti hledané části alespoň zhruba sekundu.

V tuto chvíli se nám spustí další funkce, která nám resetuje nastavení pro hledání další části.

```
private IEnumerator partWasFound() {
    yield return new WaitForSeconds(5.0f);
    distanceCount = 0;
    partFound = false;
    partToSearch.Unhighlight();
    pickedPartToSearch = false;
}
```

Tuto funkci spouštíme přes StartCoroutine, která nám dovoluje funkci pozastavit na námi zvolenou dobu, pomocí deklarace yield. Máme tedy mezi zvolením další části k hledání prodlevu a můžeme v tuto chvíli si můžeme zobrazit informace o nalezené části, které se začnou zobrazovat ve chvíli, kdy boolean partFound začne mít hodnotu true.

4.3.4 Ověření realizace úkolu

V kapitole **4.1.2** jsme si ve třídách mohli všimnou booleanu sceneComplete, animationBool, searchBool, infoBool a scenarioComplete pomocí kterých kontrolujeme realizaci. Při nalezení dostatečného množství částí se nám změní hodnota searchBool na true, stejně tak při spuštění animace a zobrazení informací se nám hodnota odpovídajícího booleanu změní na true. Hodnota sceneComplete se nám změní v případě, že všechny jednotlivé booleany akcí jsou pravdivé. To kontrolujeme pomocí funkce:

```
public void SceneCompleteCheck() {
   if(animationBool && searchBool && infoBool) {
      sceneComplete = true;
   }
}
```

Tato funkce se spouští několikrát za vteřinu v již zmíněné funkci Update. V případě, že jsou všechny scény kompletní, chceme změnit hodnotu booleanu scenarioComplete, což uděláme pomocí následující funkce, která projde všechny scény a v případě, že jsou všechny splněné změní hodnotu booleanu.

```
public void ScenarioCompleteCheck() {
   scenarioComplete = check();
}

private bool check() {
   foreach (Scene currScene in scenes) {
      if (!currScene.sceneComplete) {
         return false;
      }
   }
    return true;
}
```

Když je boolean scenarioComplete pravdivý, objeví se nám na obrazovce pogratulování ke splnění scénáře a aplikace nás dále vyzve k nalezení a zkompletování dalšího markeru (okno s textem se zobrazuje stejným způsobem jako v kapitole **4.3.2**).

Kapitola 5
Ukázka aplikace

Tato kapitola demonstruje použití naší aplikace ve dvou testovacích scénářích. Scénáře a jeho scény byly vytvořeny tak, aby demonstrovaly schopnosti aplikace a ověřili její korektní chování.

5.1 Scénář využívající VuMark

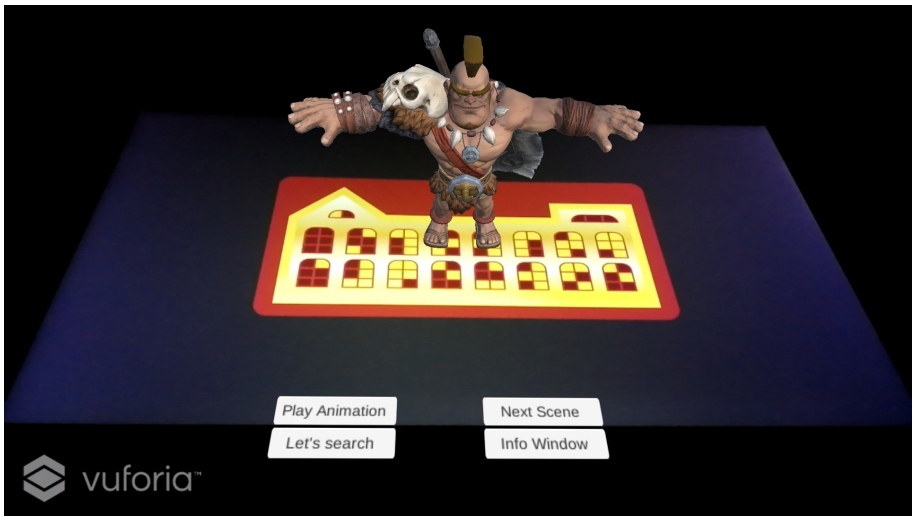
Následujících pět obrázků zachycuje scénář používající VuMark jako předlohu.

Obrázek **5.1** nám představuje vzhled scény ihned po načtení předlohy. Dále lze na obrázku vidět, že tato scéna obsahuje čtyři tlačítka. Jelikož je zde tlačítko „Next Scene“, je jasné, že bude mít tento scénář více scén, konrétně dvě (druhou scénu si můžeme prohlédnout na obrázku **5.5**). Scéna také obsahuje tři tlačítka poskytující uživateli interagovat s modelem, čili scéna obsahuje všechny akce, jež naše aplikace může v tuto chvíli nabídnout. Můžeme zde tedy spustit tři animace, hledat na modelu jeho různé části a nebo se o modelu něco dozvědět ve vyskakovacím okně. Všechny akce si je možné prohlédnout na obrázcích **5.2** , **5.3** a **5.4** .

Již zmíněná scéna **5.5** obsahuje pouze tlačítko pro zobrazení informací.

5.2 Scéna využívající obrazovou předlohu

Obrázek **5.6** nám představuje vzhled scény ihned po načtení naší obrazové předlohy. Tento scénář obsahuje dvě scény, stejně jako scénář VuMark předlohy. První scéna (**5.6**) nám představuje satelit Galileo, scéna obsahuje akci pro hledání částí modelu a pro zobrazení informací o scéně. Druhá scéna (**5.7**) nám představuje satelit Sentinel a stejně jako scéna s Barbarem v předchozím scénáři obsajuje všechny akce, jež naše aplikace v tuto chvíli poskytuje.

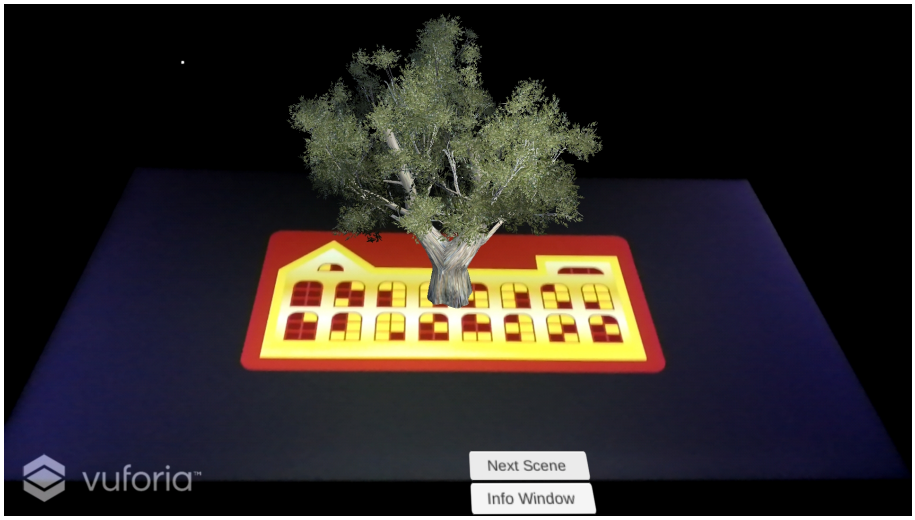


Obrázek 5.1. Scéna s barbarem

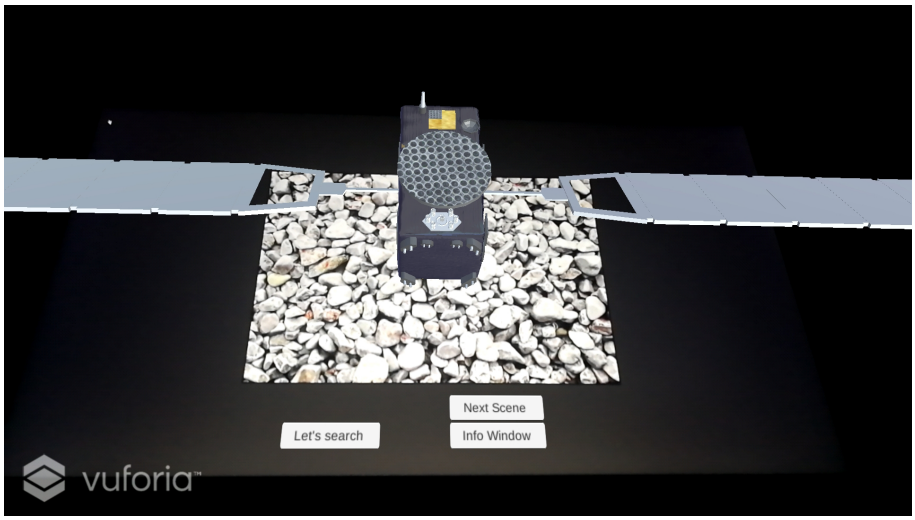
Obrázek 5.2. Zobrazení informací o barbarovi

Obrázek 5.3. Zvýraznění části modelu barbara a zobrazení informací po nalezení

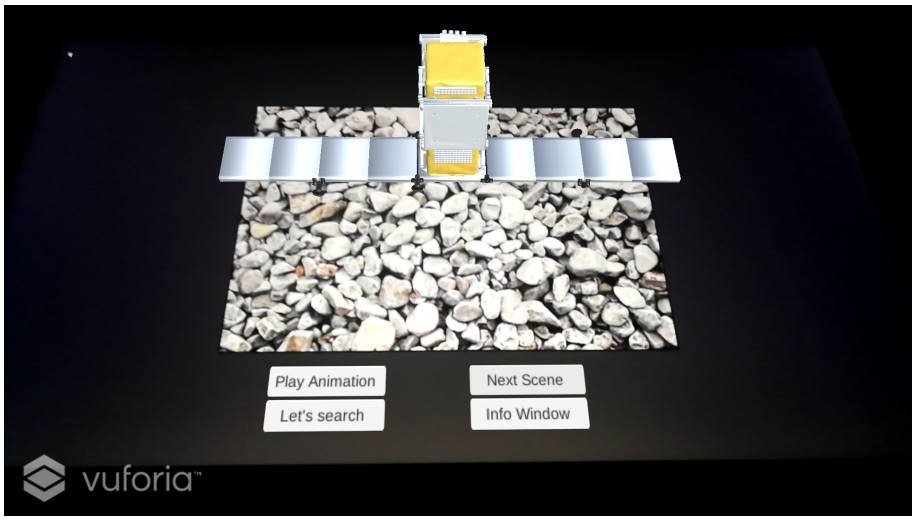
Obrázek 5.4. Barbar při animaci



Obrázek 5.5. Další scéna této předlohy



Obrázek 5.6. První scéna obrázkové předlohy



Obrázek 5.7. Druhá scéna obrázkové předlohy

Kapitola 6
Testování

Tato kapitola se věnuje testování implementované aplikace na dvou testostovacích scénářích s pomocí uživatelů. Cílem je otestovat přehlednost a intuitivnost uživatelského rozhraní. Zajímá nás dojem uživatelů z aplikace a jejich názory jak aplikaci vylepšit.

6.1 Výběr testovacích uživatelů

Testovací uživatel mohl být naprosto kdokoli, kdo umí trochu zacházet s chytrým telefonem a umí trochu anglický jazyk, ale i přes to jsem zvolil pro testování své spolužáky, kteří dokážou uživatelské rozhraní lépe posoudit, jelikož vím, že s různými mobilními aplikacemi mají zkušenosti. Dalším důvodem ke zvolení spolužáku jako testovacích uživatelů byl ten, že cílovou skupinou uživatelů aplikace jsou studenti, čili je logické aby ji i studenti testovali.

6.2 Otázky před testováním

Dříve než jsme se pustili do testování aplikace, zeptal jsem se na pár zásádních otázek:

- **Víte co je to rozšířená realita?** - Všichni účastníci věděli, co to rozšířená realita je.
- **Už jste někdy pracovali s aplikací s rozšířenou realitou?** - Účastnící hráli hru Pokemon GO, která je na rozšířené realitě postavena, ale jinou zkušenost s ní nemají.
- **Mohu vaše hodnocení a návrhy použít při vypracování bakalářské práce?** - Všichni účastníci souhlasili s použitím hodnocení a nárvhů v bakalářské práci.

6.3 Úkoly pro uživatele

Před započetím testování jsem uživatelům vysvětlil jaké všechny funkce aplikace obsahuje a jak s ní zacházet. Jejich úkolem bylo splnit oba testovací scénáře (tedy splnit všechny akce, jež jejich scény obsahují), aby si vyzkoušeli veškeré dosavadní funkce aplikace. Jelikož jsou scénáře pouze testovací, netrvalo uživatelům vyhotovení příliš dlouho. Jejich dojmy z aplikace si popíšeme v následující kapitole.

6.4 Hodnocení a návrhy na zlepšení aplikace

V této kapitole si shrneme zpětnou vazbu uživatelů naší aplikace.

6.4.1 Uživatel č. 1

Prvnímu uživateli se Výuková aplikace líbila. Jelikož s rozšířenou realitou neměl téměř žádné zkušenosti, byl tímto jejím využití nadšený. Možnost prohlédnout si modely zblízka se mu velice líbila, pochválil zpracování a jednoduchou ovladatelnost.

Co aplikaci vytkl, bylo nepříjemné, téměř okamžité, vyskakování oken s informacemi, při nalezení části modelu. Navrhl umístit okno s informacemi někam jinam než doprostřed obrazovky a taky možnost okno zavřít (nyní musel čekat určitý čas, než se okno zavřelo samo). Dále navrhl upravit tlačítka aby pasovaly k daným modelům, tedy u scénáře s družicemi, by měli mít nějaky „vesmírný nádech“. Na žádnou chybu v aplikaci při jejím zkoušení uživatel nenarazil.

6.4.2 Uživatel č. 2

Tomuto uživateli se aplikace také líbila, ale z rozšířené reality nebyl zdaleka tak nadšený jako uživatel první.

K uživatelskému rozhraní měl podobné výtky jako uživatel první, nelíbily se mu informační okna uprostřed obrazovky, nezajímavá tlačítka. Dále vytkl malý počet akcí a událostí, které aplikace poskytuje, navrhl možnost puštění nějakého zvuku, například mluvených informací o scéně. Také narvhl, že dobré zakončení scénáře by byl nějaký druh testu, který by vyzkoušel a zopakoval získané informace a znalosti. Při užívání aplikace také na žádnou chybu tento uživatel nenarazill.

6.5 Shrnutí

Při testování se neobjevily žádné chyby v implementaci aplikace. Z testování plyne:

- Uživatelské rozhraní jako jsou tlačítka by měly pasovat k modelům u kterých jsou zobrazována.
- Vyskakovací okna by neměla být umístěna uprostřed.
- V další verzi aplikace by se měly přidat další akce a události se kterými by mohl uživatel pracovat.
- Při dokončení scénáře přidat test, který uživatele vyzkouší z nově nabytých znalostí.

Kapitola 7
Závěr

Zadáním této bakalářské práce bylo vytvořit výukovou mobilní aplikaci s pomocí technologie rozšířené reality. V práci jsem nejdříve tuto technologii představil, zanalyzoval a následně popsal jaké druhy aplikací s rozšířenou realitou existují. Dále byly vysvětleny základní druhy předloh pro určování polohy. V prací jsem také popsal konkrétní návrh aplikace, včetně veškerých funkcí, které bude nabízet. Po kapitole týkající se návrhu aplikace se všemi jejími funkcemi jsem podrobně popsal hlavní naimplementované funkce aplikace. Pro jejich lepší chápání, jsem uvedl i jejich kód.

Aplikaci byla vyvinuta spolu s návrhem dvou jednoduchých ukázkových a testovacích scénářů, přičemž každý je založen na jiném druhu předlohy. Veškeré testování proběhlo bez větších problémů a uživatelé byli s aplikací z větší části spokojeni. Vyvinutá aplikace i bakalářská práce budou dále sloužit jako základ k rozšíření a vylepšení stávající výukové aplikace, či k vytvoření podobné mobilní aplikace s rozšířenou realitou.

Poslední verze aplikace tedy umí identifikovat a rozpoznat předlohu VuMark i obrazovou předlohu, na základě zvolené předlohy načíst scény (tlačítka, modely, text) pomocí JSON souboru. Aplikace dále poskytuje tři akce, kterými jsou přehrání animací, funkce hledání částí modelů a zobrazení informací o scéně, pomocí níchž může uživatel interagovat s modely. Po dokončení všech akcí, nás aplikace upozorní, že je vše splněné, pogratuluje nám a vyzve nás k nalezení další předlohy.

Jelikož cílem bakalářské práce bylo vyvinout prvotní verzi aplikace, bude její dokončení vyžadovat ještě velké množství práce. Finální aplikace by měla poskytovat více různorodých či složitějších akcí, pomocí kterých by uživatel mohl se scénami dokonaleji pracovat, zároveň by měly být scény rozsáhlejší a komplexnější.

Literatura

[1] J. Linowes, K.Babilinski. *Augmented Reality for Developers: Build Practical Augmented Reality Applications with Unity, Arcore, Arkit and Vuforia*. Packt Publishing, 2017.

[2] Augment. *Virtual Reality vs. Augmented Reality* [http://www.augment.com/blog/virtual-reality-vs-augmented-reality/].

[3] Hocking, J. *Unity in Action: Multiplatform Game Development in C# with Unity 5*. Manning Publications, 2015.

[4] Cichý, L. *Hra vytvořená v platformě Unity 3D* [Dostupné z: http://www1.fs.cvut.cz/stretech/2014/].

[5] International, Ecma. *C# Language Specification* [http://www.ecma-international.org/publications/standards/Ecma-334.htm].

[6] Qualcomm. *Vuforia* [https://library.vuforia.com/].

[7] JSON. *JSON.org* [http://www.json.org/].

[8] Technologies, Unity. *AssetBundles* [https://docs.unity3d.com/Manual/AssetBundlesIntro.html].

[9] Technologies, Unity. *Loading Resources at Runtime* [https://docs.unity3d.com/Manual/LoadingResourcesatRuntime.html].

[10] Technologies, Unity. *Vuforia SDK overview* [https://docs.unity3d.com/2017.2/Documentation/Manual/vuforia-sdk-overview.html].

[11] Technologies, Unity. *Canvas* [https://docs.unity3d.com/Manual/UICanvas.html].