VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HRA PRO MOBILNÍ TELEFON S VYUŽITÍM ROZPOZNÁNÍ RYSŮ TVÁŘE

SMARTPHONE GAME USING RECOGNITION OF FACE FEATURES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JIŘÍ SKOTÁK**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Skoták Jiří, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Hra pro mobilní telefon s využitím rozpoznání rysů tváře**
Smartphone Game Using Recognition of Face Features
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s problematikou vývoje pro iOS. Zaměřte se na pořizování obrazu/videa z kamery a jeho zpracování v reálném čase.
2. Seznamte se s problematikou detekce lidské tváře a jejích dílčích rysů v obraze; zaměřte se na algoritmy provozovatelné na mobilních zařízeních v reálném čase.
3. Navrhněte hru pro smartphone založenou na detekci rysů tváře a rozpoznání výrazu člověka a dalších charakteristik, jež je možné vyčíst z obrazu tváře. Zvažte použití i dalších informací, které je možné získat z kamery či dalších senzorů zařízení.
4. Prototypujte dílčí prvky navrhované hry. Testujte prototypy na uživatelích a iterativně je vylepšujte.
5. Implementujte navrhovanou hru, testujte ji na uživatelích a iterativně vylepšujte.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011
- Jesse Schell: The Art of Game Design, A Book of Lenses, Morgan Kaufmann, ISBN 978-0-12-369496-6
- Asit K Datta, Pradipta Kumar Banerjee: Face Detection and Recognition Theory and Practice, CRC Press, 2015, DOI: 10.1201/b19349

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3, značné rozpracování bodů 4 a 5.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Práce popisuje tvorbu hry pro mobilní telefon platformy iOS, která využívá rozpoznání rysů tváře a další informace, které je možné získat z kamery či dalších senzorů zařízení. V práci je uvedeno několik přístupů vhodných pro detekci lidské tváře a jejích rysů v reálném čase. Dále jsou popsány a porovnány možnosti takové detekce v prostředí iOS. Následně je uveden návrh výsledné hry a popis úrovní, které ji tvoří. Úrovně mimo detekci dílčích rysů využívají i detekci objektů v obraze, zpracování barevnosti vstupního obrazu a další. Na závěr je představena výsledná podoba hry, která je vydaná a dostupná na App Store.

Abstract

This master's thesis focuses on smartphone game for iOS, which uses recognition of face features and other information, which can be obtained from a smartphone's camera and sensors. This work describes a few approaches for real-time face detection and then introduces and compares possibilities for such task on iOS. Moreover, the thesis contains a draft of the final game and its levels. The game showcases various technologies in its levels such as object detection, processing an image color and others. Finally, the thesis introduces the final form of the game that is released and available on the App Store.

Klíčová slova

iOS, hra, chytrý telefon, obličej, rozpoznávání, detekce, Swift, Viola-Jones, HoG, CNN, Dlib, OpenCV, Keras, Custom Vision, Core ML

Keywords

iOS, game, smartphone, face, recognition, detection, Swift, Viola-Jones, HoG, CNN, Dlib, OpenCV, Keras, Custom Vision, Core ML

Citace

SKOTÁK, Jiří. *Hra pro mobilní telefon s využitím rozpoznání rysů tváře*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Hra pro mobilní telefon s využitím rozpoznání rysů tváře

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Skoták
19. května 2019

Poděkování

Děkuji prof. Ing. Adamu Heroutovi, Ph.D. za odborné vedení práce a cenné rady při návrhu a vývoji aplikace.

Obsah

Kapitola 1

Úvod

Každým rokem jsou představovány nové verze chytrých telefonů, které jsou výkonnější a poskytují více funkcionality než jejich předchozí verze. Možnosti využití těchto telefonů už dnes dalece přesahují jejich původní účel. Pro příklad mohou díky zabudovaným kamerám snímat fotky a natáčet videa ve vysokém rozlišení, pomocí mikrofonů nahrávat zvuk a s přesností na několik metrů detekovat geografickou polohu. Se získanými daty pak umožňují dále pracovat přímo na zařízení. Nikoho dnes již nepřekvapí, že naše telefony dokáží ve fotce detekovat lidské tváře nebo v reálném čase rozpoznat řeč.

Cílem mojí diplomové práce je vytvořit hru pro chytrý telefon s operačním systémem iOS, která využívá pokročilých funkcí těchto telefonů. Hra bude místo tradičního dotykového ovládání využívat zpracování dat dostupných z kamery a dalších senzorů zařízení. Primárně se zaměřím na detekci lidské tváře a jejich dílčích rysů v reálném čase, které budou sloužit jako jeden z ovládacích prvků. Hra cílí na užší skupinu uživatelů, kteří zkoumají možnosti telefonů a baví je přemýšlet nad netradičním ovládáním her. Na App Store je řada podobných úspěšných her, které pracují právě s ovládáním pomocí hlasu, očí apod. Věřím, že si tedy své fanoušky najde a já se při jejím vývoji blíže seznámím s programovým ovládáním telefonu a zároveň i se zpracováním obrazu v reálném čase.

Práce je členěna do několika kapitol. Kapitola 2 uvádí teoretické znalosti týkající se platformy iOS. Kapitola 3 představí několik postupů, které lze využít pro detekci lidské tváře a jejich rysů v reálném čase. V kapitole 4 jsou probrány konkrétní řešení, které lze využít pro detekci lidské tváře a jejich dílčích rysů v reálném čase na iOS. V kapitole 5 jsou představeny hry, které mi sloužily jako inspirace, a návrh výsledné hry. Kapitola 6 obsahuje popis jednotlivých úrovní a v kapitole 7 jsou uvedeny postupy využité při implementaci. Na závěr je v kapitole 8 uvedeno vyhodnocení a testování výsledné hry.

Kapitola 2

Vývoj pro iOS

2.1 Operační systém

Ve stručnosti lze za operační systém v kontextu chytrých telefonů považovat mezivrstvu mezi uživatelem a fyzickým zařízením. Operační systém interpretuje příkazy softwarových aplikací a umožňuje jim přístup k funkcím, které zařízení nabízí.

2.2 iOS

Jedná se o operační systém of firmy Apple, který běží na zařízeních iPhone, iPad a iPod touch. Původně se tento systém nazýval iPhone OS, jméno však bylo změněno na iOS s představením iPadu v dubnu 2010 [31]. První verze nesoucí název iPhone OS 1 nepodporovala nativní aplikace a tlačila tak vývojáře do vývoje webových aplikací, které se měly chovat jako aplikace nativní. Od tohoto přístupu však bylo upuštěno s verzí iPhone OS 2 a představením iPhone SDK. Operační systém iOS je aktualizován firmou Apple každý rok. Tyto aktualizace jsou zdarma a obvykle jsou podporovány i zařízeními, které Apple vydal několik let zpět. V době psaní této práce je aktualní verze systému iOS 12, která mimo novější zařízení podporuje i chytrý telefon iPhone 5S, vydaný v roce 2013 [24].

2.3 Nástroje pro vývoj na iOS

2.3.1 Xcode

Xcode je vývojové prostředí IDE[1], které slouží pro vývoj nativních iOS aplikací. První verze byla vydaná v roce 2003, verze v době psaní této práce je 10.1. Toto IDE lze na operačních systémech Mac OS High Sierra a Mac OS Mojave zdarma stáhnout z Mac App Store.

2.3.2 Objective-C a Swift

Za vznikem jazyku Objective-C stojí Stephstone company. Ta jej vytvořila okolo roku 1980. Jazyk byl veřejnosti představen v roce 1988 s vydáním knihy od autorů Brad Cox a Tom Love, která nesla název „Object-Oriented Programming: An Evolutionary Approach“. Ke konci 80. let byl jazyk uveden pod licenci firmy NeXT Computer, Inc.,kde byl využit pro

[1] Integrated Development Environemnt

vývoj NeXTStep frameworků a eventuálně byl získán firmou Apple. Zde se stal standardem na mnoho let pro vývoj iOS aplikací. [39]

Jazyk Objective-C byl ovlivněn jazyky C a Smalltalk. Právě kombinaci těchto jazyků vděčí Objective-C za svou komplikovanou syntaxi. Syntaxe spojená s objekty je totiž založena na syntaxi jazyku Smalltalk, zatímco neobjektově orientovaná syntaxe je téměř stejná jako v jazyce C.

Na jazyku Swift začala firma Apple pracovat již v roce 2010, pro veřejnost je však dostupný až od roku 2014. Jedná se o moderní jazyk, který již není založen na jazyku C nebo Smalltalk. Lidé se znalostí jazyku Objective-C zpravidla nemají potíže s přechodem na jazyk Swift, v opačném směru se jedná o přechod komplikovanější. Obecně lze říci, že je Swift díky svým moderním praktikám odolnější proti chybám. Jazyk Swift je firmou Apple neustále vyvíjen a zdokonalován [39]. Zajímavým faktem je, že v průzkumu oblíbenosti programovacích jazyků, který byl proveden portálem Stackoverflow v roce 2017, je Swift jedním z nejoblíbenějších jazyků, zatímco Objective-C je jedním z těch nejméně oblíbených [44].

2.3.3 iOS SDK

iOS SDK[2] obsahuje nástroje a zdroje, které jsou potřebné pro vývoj nativních iOS aplikací. Toto SDK tedy umožňuje vývojářům vyvíjet, instalovat a testovat jejich aplikace v iOS simulátorech a na fyzických zařízeních. Za hlavní stavební kameny SDK lze považovat jazyky Objective-C a Swift a iOS systémové frameworky.

[2]Software Developmnet Kit

Kapitola 3

Přístupy využité pro detekce lidské tváře v reálném čase

V této kapitole jsou představeny některé z technik, díky kterým je možné detekovat lidské tváře v reálném čase. Tato kapitola poskytuje teoretický základ pro kapitolu další, kde jsou uvedeny již konkrétní možnosti detekce lidské tváře a jejích rysů na platformě iOS, které jsou na těchto technikách založeny.

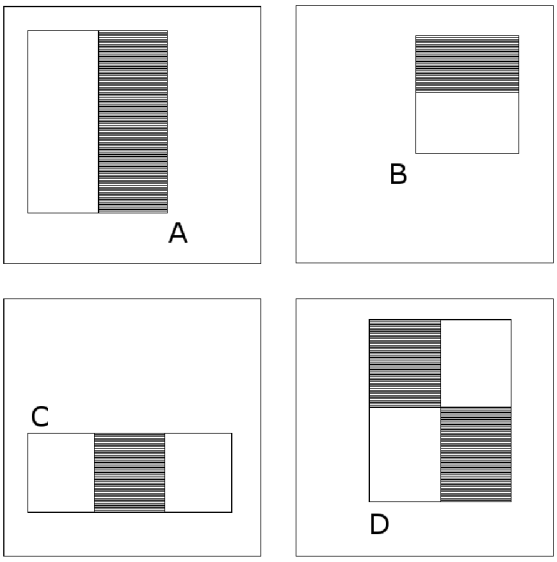
Jelikož se v následujících sekcích využívá termín strojové učení, pro jasnost ho ve zkratce představím. Za strojové učení lze považovat obor informatiky, který se zabývá konstrukcí a studií algoritmů, které se mohou učit z dat a dělat nad nimi predikce [29]. Díky strojovému učení můžeme vytvářet modely, které lze využít k získání informací z dat bez nutnosti psaní kódu pro specifický problém. V případě, že jsou pro trénování modelu poskytnuta data označená očekávanou výstupní hodnotou, jedná se o tzv. učení s učitelem. V případě absence očekávané výstupní hodnoty se jedná o tzv. učení bez učitele.

3.1 Framework Viola-Jones

Jedná se o framework z roku 2001, který je schopný s vysokou úspěšností detekovat lidské tváře v reálném čase. Této rychlé detekce framework dociluje primárně pomocí tří technik. Těmi jsou integrální obraz, využití efektivních klasifikátorů získaných pomocí algoritmu AdaBoost a využití metody pro kombinování těchto klasifikátorů tzv. „kaskádovým“ způsobem. Pro detekci tváře mohou být ve frameworku využity příznaky typy Haar, není na ně ale vázaný a lze využít i příznaky komplexnější [47].

3.1.1 Příznaky

Detekce tváře je vyhodnocena na základě hodnot jednoduchých příznaků (z angl. features). V rámci tohoto frameworku je vhodnější pracovat s hodnotami příznaků než s hodnotami pixelů. Mezi hlavní důvody patří rychlost zpracování a možnost využití příznaků pro uchování znalostí získaných z omezeného množství trénovacích dat. V rámci frameworku jsou využity příznaky, které jsou vytvořeny na základě Haarovy funkce. Typy příznaků se liší v počtu obdélníků, ze kterých jsou složeny a využity k získání hodnoty. Jejich ukázku lze vidět na obrázku 3.1.



Obrázek 3.1: Ukázka využitých příznaků typu Haar (Zdroj obrázku: wikimedia.org). Výsledná hodnota příznaku je vypočtena jako suma hodnot pixelů uvnitř tmavé části příznaku, od které je odečtena suma hodnot pixelů ve světlé části příznaku. Příznaky A a B lze nazývat jako hranové příznaky, příznak s označením C pak jako čárový příznak a příznak typu D, složený ze 4 obdélníků, se pak nejčastěji používá k detekci nakloněných čar [48].

3.1.2 Integrální obraz

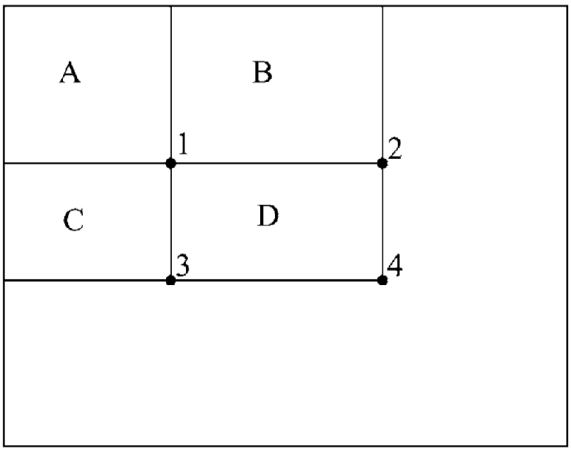
Výpočet hodnot jednotlivých příznaků může být velice rychlý za použití reprezentace obrazu ve formě tzv. integrálního obrazu. Integrální obraz na pozici x, y obsahuje hodnotu rovnou součtu všech hodnot pixelů, které se od dané pozice nacházejí ve směru doleva a nahoru. V integrálním obraze tedy platí následující vztah:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

kde *ii(x,y)* představuje pozici v integrálním obrazu a *i(x',y')* jednotlivé pozice v obraze původním. Za využití následujících vztahů:

$$s(x, y) = s(x, y - 1) + i(x, y)$$
$$ii(x, y) = ii(x - 1, y) + s(x, y),$$

kde *s(x,y)* je kumulativní součet řádku a kde *s(x, -1) = 0* a *ii(-1,y) = 0)*, může být integrální obraz vypočten pouze pomocí jednoho průchodu původním obrazem. Za použití integrálního obrazu lze pro výpočet sumy libovolného obdélníku použít pouze čtyři reference pro získání hodnot (viz 3.2).



Obrázek 3.2: Grafická reprezentace principu výpočtu sumy hodnot pixelů za použití integrálního obrazu [47]. Sumu hodnot pixelů v obdélníku D lze spočítat pomocí čtyř referencí do integrálního obrazu. Hodnota v bodě 1 je suma pixelů uvnitř obdélníku A. Hodnota v bodě 2 je rovna $A + B$, v bodě 3 pak $A + C$ a v bodě 4 to je $A + B + C + D$. Suma v obdélníku I je tedy rovna 4 + 1 - (2 + 3) [47].

3.1.3 AdaBoost

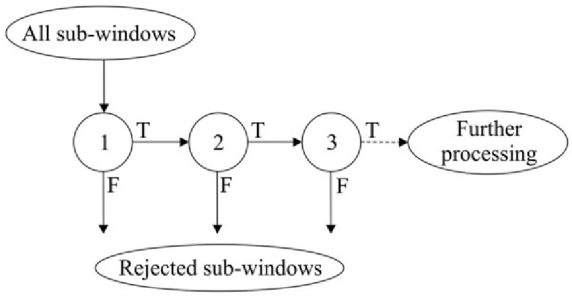
Při poskytnutí sady příznaků a trénovací sady pozitivních (obsahující hledaný objekt) a negativních obrázků (neobsahující hledaný objekt) je možné pro získání klasifikační funkce pro vyhledání požadovaného objektu využít libovolný přístup strojového učení. Pro urychlení celého trénovacího procesu a následné detekce však není potřeba velká sada příznaků, stačí pouze několik z nich. Algoritmus AdaBoost je schopný efektivně sestavit silný klasifikátor z lineární kombinace ním vhodně zvolených a významných klasifikátorů slabých. Algoritmus AdaBoost je tedy ve frameworku Viola-Jones využit jak pro trénovaní, tak pro výběr vhodných příznaků. Jelikož je slabý klasifikátor vždy spjat s jedním vhodným příznakem, lze říct, že algoritmus AdaBoost volí obdélníkové příznaky, které mají největší úspěšnost při oddělování pozitivních a negativních trénovacích obrázků a nastavuje u nich prahovou hodnotu, která tyto obrázky nejlépe rozlišuje. Následně jejich kombinací vytváří klasifikátor silný.

3.1.4 Kaskáda klasifikátorů

Pro urychlení procesu detekce není nutné pro každou část obrázu použít klasifikátory všechny. Lze nejprve použít rychlé jednodušší klasifikátory, které dokáží odhalit nezajímavé části obrázku a zároveň s velkou přesností odhalit části pozitivní (s možným výskytem hledaného objektu). Tyto jednodušší klasifikátory v případě negativního výsledku zamezí dalšímu zkoumání nezajímavé oblasti složitějšími klasifikátory. Pro natrénování jednotlivých fází, které využívají silné klasifikátory s rostoucí složitostí, je použit již zmíněný algoritmus AdaBoost.

Nejprve je tedy při detekci aplikován silný klasifikátor spjatý s dvěma příznaky. Tento klasifikátor má nastavený práh tak, aby detekoval co nejvíce pozitivních výsledků ve snaze nalezení co nejméně chybných pozitiv (chybně nalezený neexistující objekt). Pro samotnou detekci hledaného objektu (např. lidské tváře) je pouze tento klasifikátor nedostačující, svým rozhodováním však ušetří výpočetní čas v dalších fázích. Celkový proces detekce lze tedy

přirovnat k rozhodovacímu stromu s ukončující podmínkou, který nazýváme „kaskáda“ 3.3. Pozitivní výsledek z prvního klasifikátoru je zpracován klasifikátorem druhým. Pozitivní výstup z druhého klasifikátoru je zpracován klasifikátorem třetím a tak dále. Negativní výstup v libovolné fázi vede k okamžitému odmítnutí zkoumané části obrazu. Postup zpracování v kaskádě klasifikátorů je ilustrován na obrázku 3.3.



Obrázek 3.3: Schématické zobrazení zpracování obrazů v kaskádě klasifikátorů [47]. Klasifikátory jsou aplikovány na každé „podookno“ obrazu. První klasifikátor odstraní velké množství negativních vstupů a zamezí tak jejich dalšímu zpracování. První klasifikátor nevyžaduje mnoho výpočetního času. Další klasifikátory v pořadí dále odhalují negativní vstupy za potřeby dodatečných výpočtů. Negativní výstup z jakéhokoliv klasifikátoru znamená ukončení výpočtu ve zkoumaném „podokně“.

Struktura kaskády je sestavena tímto způsobem vycházejícího z faktu, že většina „podoken“ v obrazu neobsahuje hledaný objekt. Je tedy dáván důraz na to, aby se těchto negativních „podoken“ odstranilo co nejvíce již v prvních fázích zpracování a ušetřil se tak výpočetní čas [47].

3.2 HOG a SVM

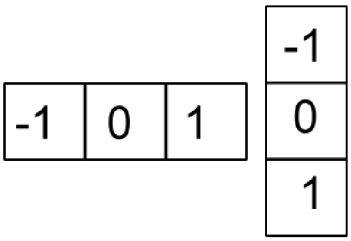
Kombinací reprezentace obrazu pomocí histogramu orientovaných gradientů (HOG) s metodou strojového učení *support vector machine* (SVM) lze detekovat tváře v reálném čase [46]. V této sekci jsou tyto pojmy blíže představeny.

3.2.1 Histogram orientovaných gradientů

Histogram orientovaných gradientů (zkráceně HOG) slouží k reprezentaci obrazu popisem jeho vlastností. HOG reprezentuje obraz pomocí orientace a velikosti gradientů, které určují směr a velikost růstu intenzity barev v obraze. Tuto reprezentaci lze využít pro hledání objektů uvnitř obrazu. Dále je uveden postup, který vede k získaní reprezentace obrazu pomocí HOG. Pro konkrétnost jsou při samotném popisu převodu uvedeny poznatky, které získali autoři Dalal a Triggs [17] při vytváření algoritmu pro detekci osob v obraze pomocí HOG.

Výpočet gradientu

Pro převod obrazu není nutné v rámci předzpracování normalizovat jeho barevné složky. Autoři [17] uvádí, že tento proces nevedl k lepším výsledkům. Prvním krokem tedy může být rovnou výpočet hodnot gradientů pomocí aplikace filtrů na obraz. Filtry lze vidět na obrázku 3.4



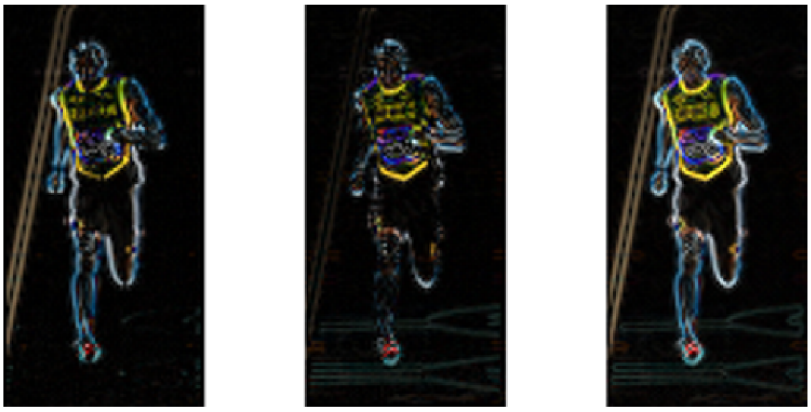
Obrázek 3.4: Ukázka možných použitých filtrů použitých pro výpočet gradientu jak pro detekci změn ve vertikálním, tak v horizontálním směru [28]. Autoři [17] experimentovali i s jinými složitějšími filtry, ale ty v jejich snaze detekovat osoby v obraze nevedly k lepším výsledkům.

Gradient je vypočten pro každý pixel a určuje pro něj směr a velikost změny, tyto hodnoty lze určit pomocí následujících vztahů:

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

kde g značí velikost a θ směr gradientu. Dále g_x a g_y reprezentují změnu intenzity pixelu postupně ve směru x a y. Výsledek této operace lze vidět na obrázku 3.5.



Obrázek 3.5: Vlevo je znázorněna změna ve směru x, uprostřed změna ve směru y v horizontálním a vpravo celková velikost výsledného gradientu g [28]. Lze vidět, že gradient ve směru x nabírá nejvyšších hodnot u vertikálních hran a ve směru y u hran horizontálních. Celková velikost jednotlivých gradientů je pak nejvyšší v místech s nejvyšší změnou intenzity nezávisle na směru. V oblastech bez velkých změn intenzity má gradient hodnotu malou, tímto způsobem jsou pak z původního obrazu odstraněny informace, které nejsou pro určení hledaného objektu nezbytné.

Rozdělení obrazu do buněk

Po aplikování filtrů je pak každý pixel reprezentován velkostí a směrem gradientu. Není však potřeba uchovávat informace o každém pixelu. Obraz je vhodné rozdělit do buněk o vhodných velikostech, pro příklad do velikosti 8x8 pixelů. Velikost buněk není náhodná a volí se dle vstupního obrazu a hledaného objektu. Tyto buňky pak lze znázornit histogramem

(vektorem), kde společně gradienty přispívají svými hodnotami, znázorňující převážnou hodnotu a směr gradientu buňky.

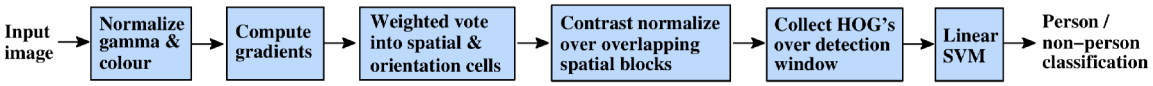
Normalizace bloků

Hodnoty gradientů jsou závislé na světlosti a kontrastu obrazu. V případě ztmavení obrazu se automaticky zmenšuje i velikost gradientů, což zmenšuje velikost histogramů. Je tedy vhodné jednotlivé velikosti gradientu normalizovat. Normalizaci v případě experimentu popisuje Dalal a Triggs [17]. Pro ně bylo při hledání osob v obraze nejefektivnější normalizaci provést v rámci bloků o velikosti 16x16, velikost bloků lze volit libovolně dle vhodnosti. Bloky se překrývají a buňka tak může do výsledné hodnoty bloku přispět více než jednou. Výsledný blok je pak reprezentován vektorem, který je tvořen konkatenací vektorů reprezentujících histogramy jednotlivých buněk, které blok zahrnuje. Nad tímto vektorem je pak provedena normalizace, která udává hodnotu pro výsledný deskriptor vlastností HOG. Takto získané informace z obrazových vstupních dat lze pak efektivně využít pro trénování modelu pro vyhledání objektů v obraze [17].

3.2.2 Support Vector Machine

SVM, nebo-li support vector machine, je metoda strojového učení s učitelem. Při poskytnutí trénovacích vstupů, u kterých je jednotlivě uvedena příslušná kategorie, je SVM schopný vytvořit model, který po přijmutí nového vstupu určí jeho příslušnost do jedné ze dvou natrénovaných tříd. Model SVM reprezentuje vstupy jako body v prostoru a mapuje je tak, aby body spadající do odlišných kategorií byly prostorově co nejvíce odděleny jeho nalezenou nadrovinou [38].

Jako trénovací vstupy a následné vstupy k rozpoznání pro model SVM lze využít právě deskriptory vlastností HOG, které mohou vhodně reprezentovat hledaný objekt. Celý proces od získání deskriptoru vlastností až po vyhledání objektu pomocí modelu SVM lze vidět na obrázku 3.6



Obrázek 3.6: Přehled postupu od extrakce vlastností až po detekci objektu [17]. Detekční okno postupně prochází obraz, ve kterém hledá požadovaný objekt. Toto okno postupně získává vektory vlastností HOG. Kombinace těchto získaných vektorů jsou pak vloženy jako vstup do modelu SVM který rozhoduje, zda-li daná část obrazu obsahuje hledaný objekt. Toto detekční okno prochází vstupní obraz přes všechny pozice. Pro nalezení hledaných objektů různých velikostí je vstupní obraz poskytnut v několika velikostech získaných změnou velikostí obrazu originálního.

3.3 Konvoluční neuronová síť

Neuronová síť tvoří klíčovou část mnoha algoritmů pro strojové učení. Snaží se uvnitř počítače napodobit chování a myšlení biologických struktur. Neuronová síť je složena z neuronů. Ty lze interpretovat jako funkce, které z libovolného počtu vstupů generují výstup, který je dále upraven hodnotou tzv. *bias*. Neurony jsou řazeny do vrstev a mezi vrstvami pro-

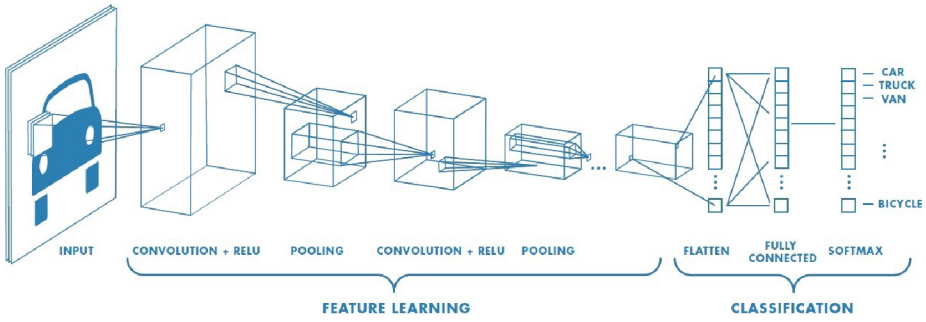
pojeny ohodnocenými přechody. Hodnota přechodu udává, jak moc je přenášená informace propagována. Síť propojených neuronů tvoří neuronovou síť [15].

Konvoluční neuronová síť (dále CNN) je kategorie neuronových sítí, která se prokázala jako extrémně efektivní pro klasifikaci a rozpoznání obrazu. Na rozdíl od tradičních neuronových sítí, kde je vstupem vektor, může být vstupem pro CNN i obraz. [20].

Pro detekci lidské tváře dosahují detektory využívající CNN přesnějších výsledků než ty, které využívají framework Viola-Jones. Jak již bylo zmíněno v 3.1, framework Viola-Jones využívá předem vybraných klasifikátorů, které typicky nestačí pro všechny varianty prostředí (světelné podmínky, orientace obličeje, měřítko, rozmazání, atd.) a okluzím[1], kterým se v reálném prostředí často nelze vyhnout. Detektory s CNN na tyto změny můžou reagovat a upravit svoje chování tak, aby dosahovaly lepších výsledků. Toho je však docíleno za cenu vyššího nároku na výpočet a paměť, což vrhá značné omezení na snahu využití tohoto přesnějšího přístupu ke zpracovávání obrazu v reálném čase [33]. Z důvodu vyšší přesnosti je nyní snaha o vytvoření přístupu, který by využíval pro detekci lidské tváře CNN za co nejmenších nároků na výpočetní čas a paměť.

3.3.1 Modely využívající CNN

Pro vytrénování a otestování modelu využívajícího CNN, jsou vstupní obrazy postupně zpracovány sérií konvolučních vrstev, pooling vrstev, plně propojených vrstev (voz angl. fully connected layers, dále FC) a na závěr je na ně aplikována aktivační funkce (např. funkce softmax). Grafickou ukázku takové CNN lze vidět na obrázku 3.7.



Obrázek 3.7: Grafické znázornění CNN s více vrstvami. (Zdroj obrázku: mathworks.com)

Konvoluční vrstva

Konvoluční vrstva tvoří hlavní stavební blok CNN. Slouží k získání vhodných příznaků ze vstupního obrazu. Obsahuje filtry, které se aktivují v případě, že na určitém místě ve vstupním obraze detekují specifické vlastnosti. Filtry jsou postupně aplikovány na části vstupního obrazu. CNN hledá správné ohodnocení použitých filtrů tak, aby mohly co nejlépe řešit požadovaný klasifikační problém.

V rámci konvoluční vstvy lze pomocí střídy nastavit počet pixelů, o které se filtr na vstupních datech postupně posouvá. Hodnotou *padding*, lze vyřešit problém v případě, že použitý filtr nesedí na velikost vstupních dat. Vstupní data je možné rozšířit nulovými hodnotami (*zero-padding*), nebo filtr použít jen na místech s dostatkem obrazových informací (*valid-padding*).

[1] Částečné zakrytí obličeje v obraze.

Rectified Linear Unit

Po konvoluční vrstvě je zpravidla použita jednotka ReLU (Rectified Linear Unit). Ta se používá z důvodu zavedení nelinearity do systému, který v konvoluční vrstvě prováděl pouze lineární operace (násobení a sčítání). ReLU aplikuje funkci $f(x) = max(0, x)$ na všechny vstupní hodnoty. Tímto způsobem jsou dál předávány pouze aktivní příznaky, které označuje kladná hodnota. Této vrstvě se proto někdy říká „aktivační funkce“.

Pooling layer

Další vrstvou po vrstvě konvoluční je zpravidla tzv. pooling vrstva. Ta je použita z důvodu redukce velikosti dále předávaných dat. Redukce probíhá pomocí různých typů filtrů. Jeden z takových typů je například filtr *max pooling* – ten prochází obraz se střídou rovnou vlastní velikosti a v každé části vybírá a ukládá maximální hodnotu. Na podobném principu fungují další filtry jako *sum pooling* nebo *average pooling*. Pomocí těchto filtrů je možné zachovat pro CNN podstatné obrazové informace na méně datech [20].

Dropout layer

Tato vrstva v době trénování modelu náhodně vybere neurony, jejichž hodnoty budou při daném průchodu ignorovány. Tato vrstva se používá z důvodu omezení vzniku jevu *overfitting*. Jedná se o jev, při kterém model používá pro detekci příznaky, které fungují velice dobře na trénovacích datech, ale špatně určují data nová. Zjednodušeně tedy model rozhoduje na více parametrech, než bylo možné na základě trénovacích dat dostatečně obecně pro danou třídu definovat [43].

Fully Connected Layer

Po pooling vrstvě jsou data převedeny do vektoru (tzv. *flatenning*) a předány pro zpracování do vrstvy FC. V této vrstvě jsou příznaky získané průchodem jednotlivými vrstvami CNN propojeny a vytváří model. Na závěr je aplikovaná aktivační funkce. Tou může být například funkce softmax, která rozdělí pravděpodobnost náležitosti vstupních dat do jednotlivých tříd. Součet pravděpodobností všech tříd je jedna [20].

Kapitola 4

Možnosti detekce lidské tváře a jejích rysů na iOS

V této kapitole jsou uvedeny možnosti pro detekci lidské tváře a jejich rysů na chytrém telefonu s operačním systémem iOS.

4.1 Nativní iOS přístup

Apple už od iOS 10 používá pro detekci lidských tváří mimo tradiční přístup založený na frameworku Viola-Jones i tzv. „deep learning“. Jedná se o metodu strojového učení využívající CNN, které obsahují více než jednu vrstvu. Firmu Apple pro využití tohoto přístupu motivoval fakt, že poskytoval mnohem přesnější výsledky než dosavadně používané algoritmy.

Přes vysokou výkonnost moderních chytrých telefonů tyto zařízení stále nepředstavují vhodnou platformu pro všechny modely založené na CNN skrze jejich vysoké nároky na výpočetní sílu a paměť. Většinou je tento problém možné řešit pomocí vzdálených API dostupných přes internet. Toto řešení umožňuje odeslat obrazy k zpracování na server, kde je dostupná dostatečná výpočetní síla a paměť pro běh složitých CNN modelů.

Apple však klade vysoký důraz na soukromí uživatelů využívající jejich produkty, a tak pro toto zpracování nebylo možné využít například iCloud Photo Library, což je firmou Apple nabízené cloudové řešení pro uložení fotek a video souborů. Zde jsou totiž tyto fotky a video soubory ukládány zašifrované a dešifrovat je dokáže pouze zařízení, které je zaregistrováno pod účtem spojeným s daným úložištěm. Proto bylo nutno vytvořit takový přístup, který by byl schopný běžet přímo na zařízení. Navíc by tento přístup nevyžadoval neustále přípojení k internetu a umožňoval by rychlejší zpracování.

Na tomto řešení firma pracovala již od roku 2014. Velkým krokem vpřed byl přístup nazvaný „OverFeat“, který ukázal, že CNN může být účinně a efektivně využita pro detekci objektů v obraze. Na poznatcích z OverFeat firma Apple začala vytvářet nové řešení. Po provedení mnoha optimalizací se firmě Apple nakonec podařilo vytvořit přístup využívající CNN, který umožňuje provádět detekci objektů, tedy i lidských tváří přímo na zařízení. Tento přístup je programátorům dostupný přes framework Vision [13].

Framework Vision

Jedná se o framework představený na Apple WWDC 2017[1] [14]. Tento framework umožňuje od iOS 11 vývojářům řešit problémy spojené se zpracováním obrazu přímo na zařízení za využití CNN. Framework je mimo jiné schopný detekovat tváře i její rysy. Také umožňuje spolupráci s frameworkem Core ML pro vytrénování a zakomponování modelů pro detekci téměř libovolného objektu v obraze [37].

Framework Core Image a AV Capture

Frameworky Core Image a AV Capture umožňovaly detekci lidských obličejů v obraze již před uvedením frameworku Vision. Framework Core Image navíc dokáže detekovat i pozici očí a úst. Srovnání rychlosti a kvality nativních přístupů lze vidět v tabulce 4.1.

	Vision	**Core Image**	**AV Capture**
Accuracy	Best	Better	Good
Processing time	Fast	Faster	Fastest
Power usage	Good	Better	Best
Availability	iOS, macOS, tvOS	iOS, macOS, tvOS	iOS capture only

Tabulka 4.1: Srovnání frameworků, které umožňují detekci lidského obličeje v obraze na iOS, na základě přednášky WWDC 2017. Framework Vision dává dle společnosti Apple nejpřesnější výsledky, ale je nejnáročnější na výpočet a baterii. Tento nedostatek je nejvíce znatelný na starších zařízeních [14].

Framework ARKit

Framework ARKit s příchodem iPhone X poskytuje robustní a velice přesnou detekci lidského obličeje založenou na CNN. Tento přístup detekuje pozici a orientaci tváře s rychlostí 60 FPS[2]. Zároveň dokáže rozpoznat a rozlišit přes 50 dílčích rysů ve tváři [11]. Jeden takto rozpoznaný dílčí rys lze vidět na obrázku 4.1. Pro nalezení těchto dílčích rysů je využíta přední kamera typu *TrueDepth*. Tento způsob detekce je tedy omezen na zařízení iPhone X a novější, starší verze tento typ kamery totiž nemají [5].

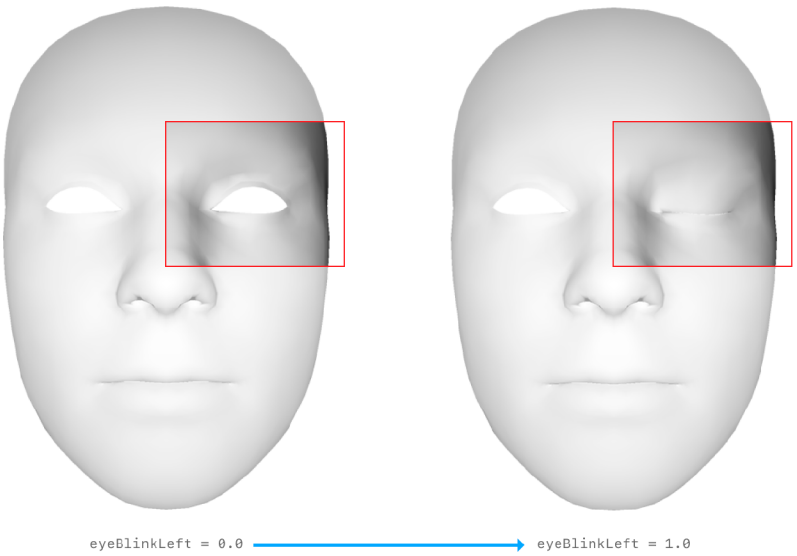
4.2 Detekce na zařízení s využitím OpenCV

OpenCV (Open Source Computer Vision Library) je knihovna vydaná pod BSD licencí. Je napsaná v C/C++ a má rozhraní pro jazyky Python a Java a podporuje Windows, Linux, Mac OS, iOS a Android. Tato knihovna obsahuje přes 2500 optimalizovaných algoritmů, mezi které se řadí i mnoho algoritmů pro zpracování obrazu a algoritmy pro strojové učení. Tyto algoritmy mohou být použity i pro detekci lidské tváře [35].

Využití přístupu Viola-Jones

Pro detekci lidského obličeje nabízí OpenCV modely založené na frameworku Viola-Jones, které pro detekci využívají příznaky typu Haar. Výhoda tohoto přístupu je ta, že vstupy

[1] Worldwide Developer Conference je každoroční konference firmy Apple určená pro vývojáře.
[2] Frames Per Second, nebo-li počet snímků za sekundu.



Obrázek 4.1: Ukázka jednoho ze sledovaných dílčích rysů pomoci frameworku ARKit. Vidíme, že hodnota rysu je v rozmezí od 0.0 do 1.0. Tento dílčí rys nese název eyeBlinkLeft a značí zavření levého oka. Je dobré myslet na to, že je rys nazván správně jako levé oko a je v přední kameře, kde zpracování probíhá, zobrazen zrcadlově. (Zdroj obrázku: developer.apple.com)

dokáže zpracovávat v reálném čase na CPU[3], využívá jednoduché architektury a detekuje tváře o různých velikostech. Nevýhodou je, že tento přístup nedává příliš přesné výsledky, rozpozná obličeje pouze z frontálního směru a nerozpozná lidské obličeje s okluzí [22].

Využití CNN

OpenCV od verze 3.3 z roku 2017 nabízí i detektor lidských obličejů založený na vícevrstvých CNN. Tento model byl vytrénován na obrázcích volně dostupných na internetu, ale trénovací sada nebyla zveřejněna. Tento model dává velice přesné výsledky (detekuje i pootočené lidské tváře a rozpozná i lidské tváře s okluzí) a dokáže běžet v reálném čase na CPU [22].

4.3 Dlib

Dlib je open-source[4] multiplatformní softwarová knihovna napsaná v C++. Obsahuje algoritmy strojového učení a nástroje pro vytváření komplexního softwaru, který umožňuje řešit problémy i v oblasti zpracování obrazu na mobilních zařízeních [18].

HOG a SVM

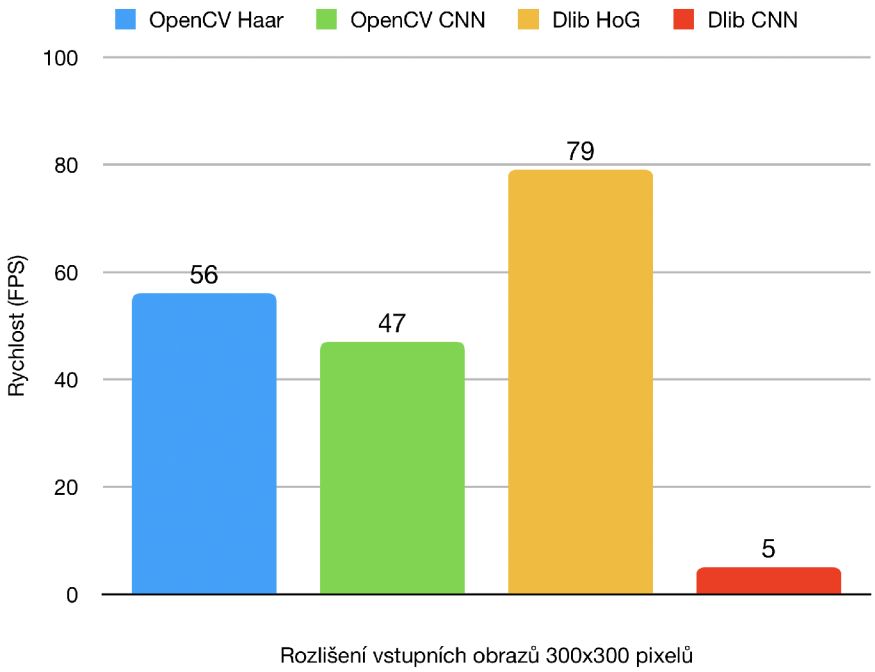
Model pro detekci lidké tváře dostupný v knihovně Dlib je založený na již zmíněném přístupu HOG v kombinaci s SVM. Trénovací data, která sloužila k natrénování tohoto modelu, obsahují 2825 obrázků, které byly získány z LFW (Labeled Faces in the Wild) databáze

[3]Centrální procesorová jednotka.
[4]Kód volně dostupný pro další vývojáře, kteří ho mohou upravovat.

a manuálně anotována autorem Dlibu, kterým je Davis King. Tento přístup umožňuje zpracování v reálném čase na CPU a detekuje i mírně pootočené obličeje a obličeje s mírnou okluzí. Tento model není vhodný pro detekci lidských obličejů o malé velikosti [22].

CNN

Dlib obsahuje také model pro detekci lidských obličejů založený na CNN. Model byl vytrénován na datasetu čítajícího 7220 obrázků, které opět anotoval autor Dlibu Davis King. Tento model je velmi robustní a detekuje tváře natočené do různých směrů i tváře s okluzí. Tento model běží velmi rychle na GPU[5], ale je velmi pomalý na CPU. Srovnání rychlosti uvedených přístupů, které poskytuje jak OpenCV, tak Dlib lze vidět na obrázku 4.2.



Obrázek 4.2: Srovnání rychlosti detekce lidských obličejů metodami, které poskytuje OpenCV (Haar, CNN) a Dlib (HOG, CNN). Pro získání těchto výsledku byla každá metoda spuštěna 1000x nad vstupním obrazem a to v deseti iteracích na procesoru Intel Core i7 6850K – 6 Core. Výsledná rychlost byla získaná průměrem naměřených hodnot. Největší rychlosti na CPU dosahovala metoda Dlib HOG. (Zdroj naměřených dat: [22]).

Detekce dílčích rysů tváře

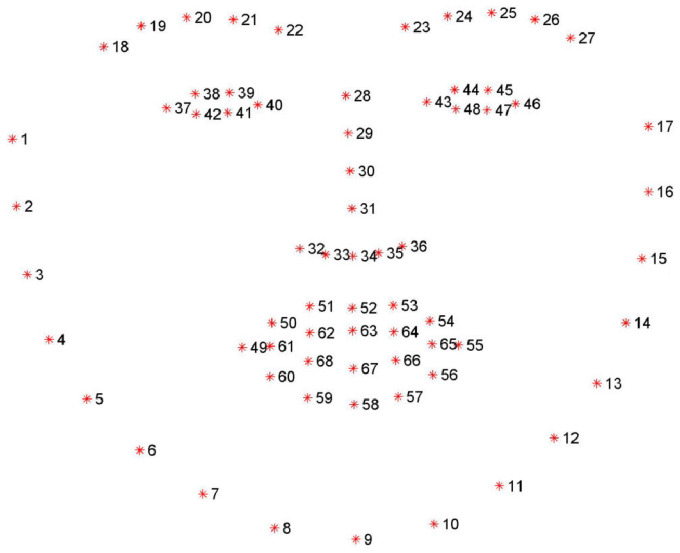
Dlib mimo jiné také obsahuje model, pomocí kterého je možné detekovat dílčí rysy tváře. Nejprve je však potřeba získat pro nalezenou tvář tzv. bounding box (ohraničující obdelník). Toho lze na iOS docílit libovolnou metodou představenou v této kapitole.

Model sloužící pro detekci dílčích rysů tváře, je vytvořen autorem knihovny Dlib na základě algoritmu [23]. Tento algoritmus využívá tzv. kaskády regresorů. Jednotlivé „regresory“ studují a popisují závislost mezi příznaky v obraze a tvarem tváře (příznaky mohou

[5]Grafická procesorová jednotka.

být Haar, HOG, rozdíly intenzit pixelů a další). Tyto regresory jsou následně umístěny do kaskády. V kaskádě pak jednotlivé regresory využívají příznaky získané v závislosti na odhadovaném tvaru obličeje a postupně tento odhadovaný tvar obličeje upřesňují. V případě [23] jsou jednotlivé regresory tvořeny rozhodovacími stromy a pracují s příznaky získanými pomocí rozdílů intenzit pixelů [29].

Tento přístup dokáže pracovat s vysokou přesností v reálném čase na CPU [3]. Konkrétně tento detektor označuje pozici 68 bodů, které značí významné rysy 4.3.



Obrázek 4.3: Vizualizace 68 bodů, které udávají pozici významných rysů v obličeji. Takto vyznačit tyto rysy umožňuje i model, který byl vytrénovat na datech s označením iBUG 300-W dataset a je dostupný v knihovně Dlib. Tento detektor je schopný tyto body vyznačit na detekovaném obličeji v reálném čase na CPU. (Zdroj obrázku: pyimagesearch.com)

Kapitola 5

Inspirace a doporučení pro tvorbu hry a její návrh

V kapitole nejdříve představím dvě úspěšné hry, které pro svůj průběh využívají detekci význačných rysů lidské tváře. Následně uvedu hru, která mi sloužila jako hlavní inspirace. V závěru pak uvedu návrh vlastní hry a několik zásad, kterých jsem se při návrhu držel.

5.1 Hry na mobilní telefon využívající detekci tváře

5.1.1 FaceConnect

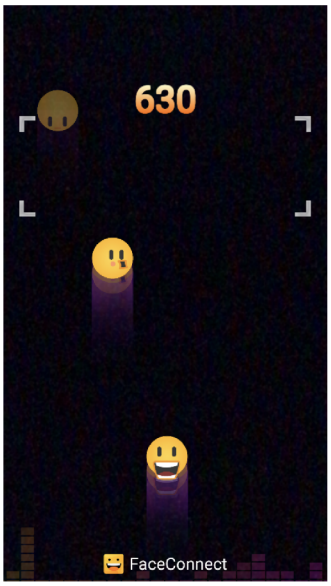
„Face Connect“ je hra, která využívá detekci dílčích rysů obličeje. Principem hry je co nejrychleji a nejpřesněji napodobovat hrou představené výrazy tváře. V rámci hry je pak například představen obrázek znázorňující zavřené oči s otevřenou pusu a k získání bodů je potřeba tento obličej napodobit. Ukázku lze vidět na obrázku 5.1. Tato hra byla v době jejího vydání velmi populární a to hlavně v Asii [45]. Nyní má aplikace na Google Play přes jeden milion stáhnutí, údaj o počtu stáhnutí na App Store není zveřejněn.

Přesto, že mě hra svou náplní příliš neoslovila, zaujala mě svou hlavní myšlenkou a její úspěch jí skrze dosažené počty stažení nelze odepřít. Z mého pohledu stála za hlavním úspěchem této hry možnost lidí se při jejím hraní nahrávat a tato videa pak zveřejnit pro pobavení ostatních, což vytvořilo velice silný nástroj pro marketing.

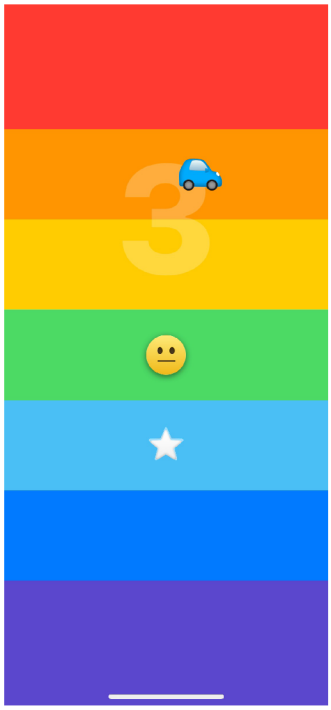
5.1.2 Rainbrow

Jednoduchá hra, jejíž náplní je měnit pozici figurky v rámci obrazovky tak, aby nedošlo ke kolizi s jinými herními objekty. Hráč může figurkou pohybovat nahoru a dolů pomocí pohybu obočí. Hra je dostupná pouze pro iPhone X a novější, jelikož pro svůj průběh využívá využívá TrueDepth kameru. Detekce dílčích rysů v obličeji je zde využita jako náhražka standardního ovládání podobných her pomocí dotyku obrazovky, který je uživatelsky přívětivější. Po chvíli se tedy z pohybu obočí stává pouze méně pohodlný způsob ovládání. Hra ale svou inovativní myšlenkou zaujala a na App Store má nyní přes 2 tisíce hodnocení s průměrem 4.6 / 5.

Hra mě svým originálním nápadem oslovila, ale po chvíli nenabízela nic nového. Dokážu si představit hru na podobném principu s využitím více dílčích rysů, která by mohla sloužit pro cvičení obličejových svalů.



Obrázek 5.1: Ukázka z průběhu hry Face Dance.



Obrázek 5.2: Ukázka z hry Rainbrow, kde je pro ovládání použita detekce pozice obočí.

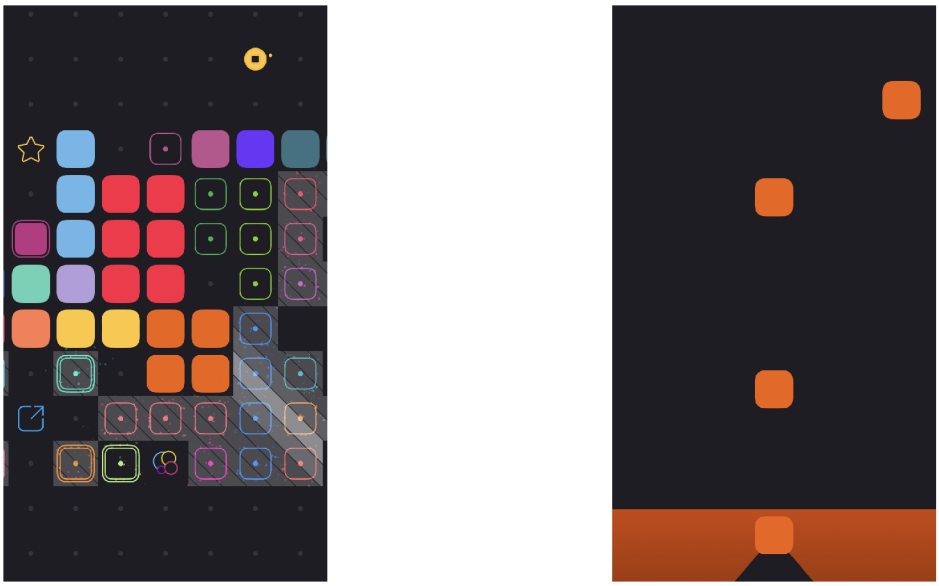
5.1.3 Blackbox

Hra „Blackbox“ nepředstavuje hru, která by využívala zpracování obrazu, nebo detekci dílčích rysů obličeje. Přesto se mi stala jednou z hlavních inspirací. Jedná se o hru typu puzzle. Hráči poskytuje několik úrovní, které jsou znázorněny barevnými čtverci viz 5.3. V každé úrovni je velmi abstraktně znázorněn úkon, který musí hráč pro její splnění provést. Jedná se o netradiční úkony, mezi které může patřit třeba nabití baterie na 100 %, nebo spuštění dané úrovně každou hodinu po celý den. Tato hra je velice populární a na App Store má přeš 21 000 recenzí s průměrným hodnocením 4.8/5.

Tato hra mě svou náplní nadchla a rozhodl jsem se na jejím základě vytvořit hru vlastní, jejíž návrh popíšu v následující sekci.

5.2 Návrh výsledné podoby hry

Na základě inspirace získané ze hry „Blackbox“ v této práci vytvářím hru typu puzzle. Typem puzzle lze chápat, že hra bude testovat vynalézavost jejího řešitele. Hra se bude skládat z několika úrovní obsahující hádanky. Tyto hádanky budou pro své řešení vyžadovat netradiční úkon, při němž může být mimo jiné využita i detekce tváře a jejich dílčích rysů a další informace, které lze získat ze senzorů zařízení. Pro hru typu puzzle existuje několik praktik, na které je dobré myslet [41].



Obrázek 5.3: Ukázka hry Blackbox. Vlevo je zobrazen přehled hráči dostupných úrovní, které jsou znázorněny čtverci. Nevyplněné čtverce značí doposud nesplněnou úroveň. Vpravo pak lze vidět detail jedné z úrovní, kde je hráči velice abstraktně naznačen úkon pro její splnění.

Pocit postupu

Na rozdíl od osamocené hádanky, musí hra typu puzzle hráči poskytovat určitý pocit postupu. Hráč se tak necítí, že stagnuje na místě a je mu poskytnuta naděje, že se eventuálně dostane k cíli.

Pocit vyřešitelnosti

Pocit vyřešitelnosti je napojen na pocit postupu. Pokud hráč začne pochybovat o vyřešitelnosti hádanky, začne uvažovat nad tím, jestli pouze nemarní svůj čas a hru ukončí. Mimo pocitu postupu je někdy také vhodné prezentovat již vyřešený stav, jak je tomu například u prodeje Rubikovi kostky, která je prodávána právě ve vyřešeném stavu.

Z mého pohledu by tento bod mohla řešit i existence nápovědy, kde hráč předpokládá, že po jejím zpřístupnění hádanku vyřeší.

Pocit rostoucí obtížnosti

Ze začátku je lepší prezentovat hádanky nižší obtížnosti a postupně prezentovat hádanky složitější.

Prostor pro odpočinek

Hráč by měl mít šanci upustit od hádanky, nad kterou se trápí, a své úsilí směřovat na hádanku jinou. Hráč tak získá prostor pro odpočinek a posléze se k hádance může vrátit s čistou hlavou a nově nabytým odhodláním.

Nápovědy podporují zájem

Vyřešení hádanky s nápovědou je lepší, než nevyřešit hádanku vůbec. V dnešní době lze na internetu nalézt odpověď na téměř každou hádanku většiny alespoň středně populárních her. Není tedy důvod proč hráči odpověď za určitých podmínek neposkytnout.

Poskytnutí odpovědi

Jedním z hlavních motivací pro hraní hry typu puzzle lze považovat pocit, který se hráči dostaví po vyřešení hádanky. Tento pocit se však částečně dostaví i v případě, že na odpověď nepřišel sám. Příkladem může být například detektivka, na kterou by se dalo dívat jako na jednu velkou hádanku v knižní podobě se zveřejněnou odpovědí na konci.

Pro dodržení těchto zásad bude výsledná hra formována následujícím způsobem. Jejím základem bude hrací plocha, která bude obsahovat jednotlivé úrovně a graficky bude odlišovat již splněné od těch nesplněných. V každém momentě, s výjimkou poslední úrovně, bude mít hráč možnost řešit alespoň dvě zatím nevyřešené úrovně. Každá tato úroveň bude obsahovat nápovědu, kterou bude mít možnost hráč za cenu herní měny zpřístupnit. Herní měnu bude hráč schopný získat pomocí „minihry“, která nebude představovat logickou výzvu. Úrovně budou řazeny dle obtížnosti, kde ty nejjednodušší budou představeny jako první.

Grafická podoba úrovní se bude lišit na základě hádanky, kterou budou znázorňovat. Každá z nich ale bude obsahovat dva kontrolní prvky, kde jeden bude sloužit pro odemčení nápovědy a druhý pro návrat na hrací plochu. Celkově bude hra stylizované do tmavých barev. V rámci hry bude vůči tmavému prostředí zvolena jedna vysoce kontrastní barva, kterou budou v případě úspěšného vyřešení obarveny grafické prvky úrovně a uživatel si tak bude vědom svého úspěchu.

Jednotlivé úrovně budu postupně „prototypovat“ a testovat na uživatelích. Na základě jejich reakce budu upravovat jejich grafickou podobu a dle nutnosti i jejich samotnou náplň – představenou hádanku.

Kapitola 6

Popis jednotlivých úrovní hry

V této kapitole jsou představeny jednotlivé úrovně hry. Popis úrovní je seskupen do sekcí dle hlavního ovládacího prvku, který využívají. V jednotlivých sekcích je nejprve představena využitá technologie a následně popsán princip jednotlivých úrovní. Sekce můžou být v některých zajímavých případech proloženy i popisem či ukázkou implementace popsané problematiky. Uvědomuji si, že se v rámci některých sekcí objevuje i teorie a implementace, která by mohla rozšiřovat teoretickou kapitolu 3, nebo kapitolu 7 popisující implementaci. Přesto jsem se po rozmyšlení a konzultaci s vedoucím mé práce rozhodl, s ohledem na přehlednost a logickou návaznost mezi technologiemi a konkrétními úrovněmi, kapitolu vytvořit tímto způsobem.

V první sekci představím úrovně využívající zpracování dílčích rysů tváře, tato sekce navazuje na teoretický základ popsaný v předchozích kapitolách. Další sekce pak mohou obsahovat i stručný teoretický základ, který byl využit pro implementaci jejich hlavního ovládacího prvku.

6.1 Úrovně využívající zpracování dílčích rysů tváře

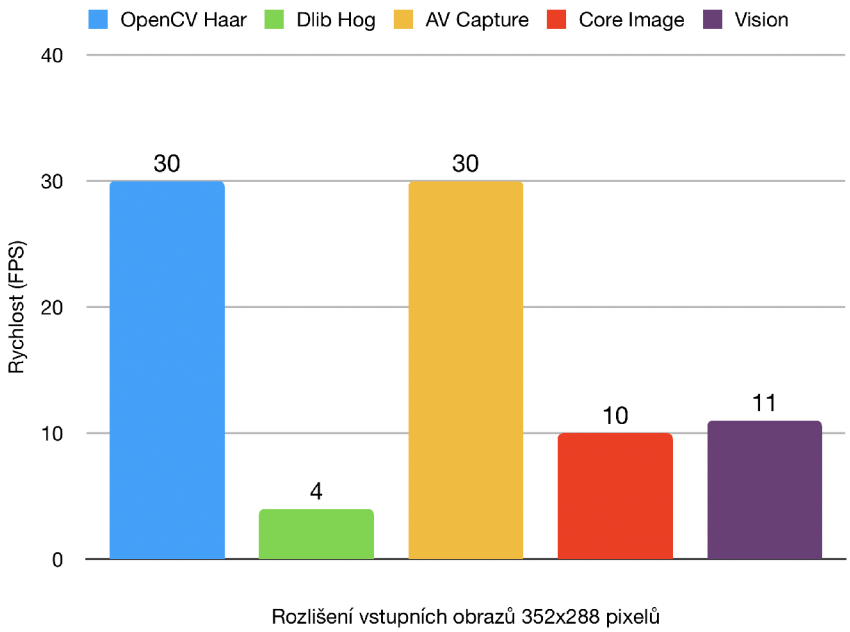
První skupina úrovní pro svůj průběh využívá detekci dílčích rysů lidské tváře. Pro detekci dílčích rysů je nejprve potřeba získat obrazová data, nalézt v nich lidskou tvář a v té následně vyhledat její dílčí rysy.

6.1.1 Detekce lidské tváře

Na základě informací uvedených v kapitole 4, jsem pro detekci lidské tváře postupně vyzkoušel pět přístupů. Detekci pomocí OpenCV, detekci pomocí Dlib a nativní detekci pomocí frameworků Vision, AV Capture a Core Image. Jelikož se dá předpokládat, že v rámci hry bude obličej ve vstupních obrazech v dostatečné velikosti ve frontální pozici a minimem okluzí, hlavní mírou vhodnosti byla stanovena rychlost a plynulost zpracování. Přístupy jsem tedy postupně testoval na zařízení iPhone 7 a zaokrouhlené hodnoty rychlosti zpracování získal zprůměrováním hodnot naměřených po dobu dvou minut detekce jednoho obličeje ve vstupním obraze. Souhrn naměřených hodnot lze vidět na obrázku 6.1.

OpenCV

Při detekci pomocí OpenCV jsem využil Haarovu kaskádu filtrů, kterou v XML připravil Prof. Dr. Rainer Lienhart. Detekce pomocí OpenCV běžela na zařízení v reálném čase



Obrázek 6.1: Srovnání rychlosti detekce lidských obličejů na zařízení iPhone 7 metodami, které poskytuje OpenCV, Dlib a nativní přístupy. V obraze ze vstupu kamery byla po dobu dvou minut detekována jedna tvář a výsledné FPS bylo získáno zprůměrováním a zaokrouhlením naměřených hodnot. Vstupní obraz byl v rozlišení 352×288 pixelů. Pro využití ve hře se jevil nejvhodněji přístup, který poskytoval framework AV Capture, který dosahoval rychlosti 30 FPS i na vstupních obrazech v HD rozlišení a detekoval tváře s většími okluzemi než přístup OpenCV Haar.

s rychlostí 30 FPS na vstupních obrazech s rozlišením 352×288 pixelů. Tento detektor velice špatně reagoval na okluze – nebyl schopný detekovat ani lidskou tvář s dioptrickými brýlemi. V případě zvýšení kvality vstupních obrazových dat na rozlišení HD (myšleno 1280×720 pixelů) nebyla detekce v reálném čase možná a dosahovala rychlosti okolo 2 FPS.

Dlib

Jako druhý přístup jsem vyzkoušel knihovnu Dlib. Zde jsem využil uvnitř knihovny dostupný detektor lidské tváře využívající HOG a SVM. Tento detektor reagoval oproti přístupu pomocí OpenCV lépe na okluze a natočení obličeje, ale nebyl schopný běžet v reálném čase ani na vstupních obrazech s rozlišením 352×288 pixelů. Na těchto vstupních obrazech dosahoval rychlosti okolo 4 FPS. Po optimalizaci práce s knihovnou Dlib v prostředí iOS, která je uvedena dále, dosahovala tato detekce rychlosti 10 FPS, která byla stále nedostačující.

Vision

Framework Vision dokázal tváře detekovat v rychlosti 11 FPS na vstupních obrazech s rozlišením 352×288 pixelů a v rychlosti 9 FPS na vstupních obrazech v HD. Velmi dobře reagoval na okluze. Tvář detekoval i v případě, že byla z poloviny zakryta.

Core Image

Nativní framework Core Image podobně jako framework Vision velice dobře reagoval na okluze a natočení hlavy a dosahoval rychlosti 10 FPS na vstupních obrazech s rozlišením 352×288 pixelů. Na vstupních obrazech v HD rozlišení dosahoval rychlosti okolo 9 FPS. Rychlost této detekce byla ve stanovených podmínkách pomalejší než pomocí frameworku Vision, což neodpovídalo předpokladu dle údajů uvedených v tabulce 4.1.

AV Capture

Posledním z vyzkoušených přístupu byl nativní framework AV Capture, což by měl být dle dříve uvedených informací 4.1 nejrychlejší přístup dostupný nativně na iOS. Tento přístup dostatečně dobře reagoval na okluze ve vstupním obraze a dokázal tvář vyhledat v rychlosti 30 FPS i na vstupních obrazech v HD rozlišení. Pro detekci lidské tváře v reálném čase na iOS byl tedy zvolen tento nativní přístup.

6.1.2 Detekce dílčích rysů lidské tváře

Pro detekci dílčích rysů lidské tváře se dle nabytých informací nabízely tři přístupy. Dva nativní s využitím frameworků Vision a Core Image a jeden s využitím knihovny Dlib. Podobně jako u detekce samotné tváře jsem postupně vyzkoušel a porovnal přístupy na zařízení iPhone 7. Přístupy jsem testoval na obrazech ze vstupu kamery s jedním obličejem. Přehled naměřených výsledků lze vidět na obrázku 6.2.

Vision

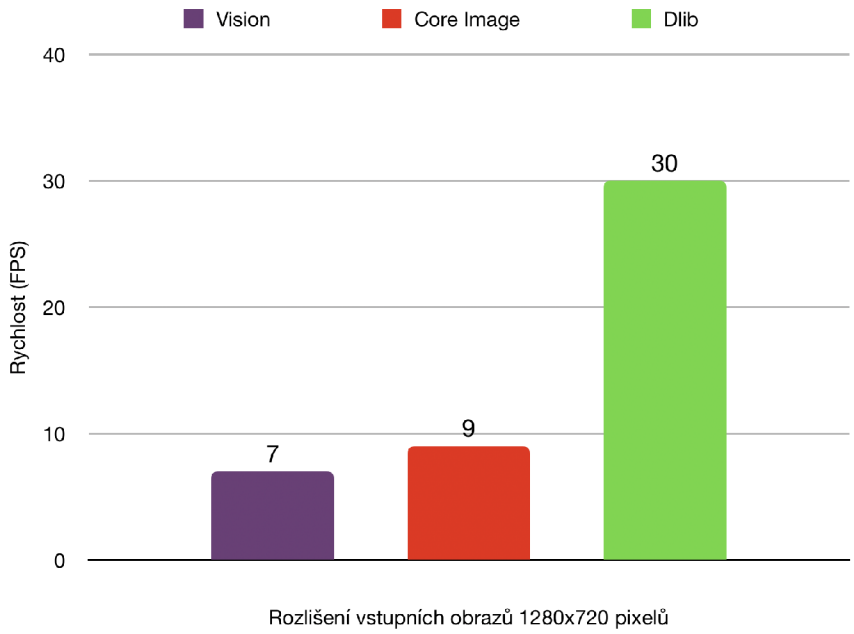
Jako první jsem vyzkoušel framework Vision. V případě použití tohoto frameworku pro detekci samotné tváře je schopný v obličeji detekovat a vymezit 12 dílčích rysů [4]. Tyto rysy dokázal detekovat velmi přesně v rychlosti 8 FPS na obrazech v rozlišení 352×288 pixelů a v rychlosti 7 FPS na obrazech v HD rozlišení. Tato rychlost se bohužel nejevila jako uspokojivá. Jelikož se jedná o nativní framework iOS, neposkytuje příliš prostoru pro optimalizace a následné urychlení detekce.

Core Image

V případě použití frameworku Core Image pro detekci samotné tváře je možné detekovat i informace o očích a ústech. Pro detekci dílčích rysů dosahoval tento framework srovnatelných hodnot jako framework Vision – na vstupních obrazech s rozlišením 352×288 pixelů rychlosti okolo 9 FPS a na obrazech v HD rozlišení 8 FPS. Tato detekce poskytovala údaje o otevřeném, nebo zavřeném stavu očí a detekovala úsměv. Vrácené výsledky ale konzistentně neobsahovaly očekávané hodnoty a skrze zastínění implementace samotné detekce zde byl malý prostor pro úpravy a optimalizace.

Dlib

Poslední vyzkoušený způsob využíval knihovnu Dlib. Ta před provedenými optimalizacemi detekovala dílčí rysy (konkrétně zmiňovaných 68 bodů) v nalezené tváři získané pomocí frameworku AV Capture v obraze v HD rozlišení s rychlostí okolo 19 FPS. Tato rychlost se stále nejevila jako optimální, ale na rozdíl od předchozích nativních přístupů poskytoval tento přístup více prostoru pro úpravu a optimalizaci zpracování. Původním záměrem



Obrázek 6.2: Srovnání rychlosti detekce dílčích rysů obličejů na zařízení iPhone 7 metodami, které poskytuje nativní framework Vision a Core Image a knihovna Dlib. V obraze ze vstupu kamery byla po dobu dvou minut detekována jedna tvář a výsledné FPS bylo získáno zprůměrováním a zaokrouhlením naměřených hodnot. Vstupní obraz byl v HD rozlišení. Lze si povšimnout, že nalezení dílčích rysů tváře pomocí frameworku Vision a Core Image dosahuje podobné rychlosti, jako jejich samotná detekce tváře. Tyto frameworky totiž pro nalezení dílčích rysů nejdříve vyhledávají samotnou tvář, kterou následně zpracují. Přístup pomocí knihovny Dlib může pro nalezení regionu tváře využít libovolnou metodu, a tak knihovna Dlib v kombinaci s frameworkem AV Capture, který byl využit pro nalezení oblasti tváří, poskytovala nejlepší výsledky.

bylo dosáhnout rychlosti 30 FPS, v které probíhala detekce samotného obličeje pomocí frameworku AV Capture. Toho bylo nakonec s využitím knihovny Dlib docíleno pomocí optimalizace, která je uvedena dále.

6.1.3 Dlib v prostředí iOS

Pro využití knihovny Dlib v prostředí iOS je potřeba vytvořit její verzi pro iOS pomocí knihovny CMake[1]. Poté tuto verzi Dlib vložit do projektu Xcode a tuto knihovnu v rámci projektu zahrnout do jeho překladové fáze. Následně je potřeba vytvořit *wrapper*, nebo-li obal této knihovny psaný v Objective-C. Jelikož je jazyk Objective-C založen na jazyku C, je možné pomocí tohoto jazyku využívat metody a datové struktury z knihovny Dlib, která je napsána v jazyce C++. Pro zpřístupnění kódu, který obsahuje *wrapper* z jazyka Swift, je potřeba vytvořit *bridging header*, který umožní přístup k třídám a dalším deklaracím napsaným v jazyce Objective-C [7].

[1]CMake je multiplatformní soubor nástrojů pro překlad software na různých operačních systémech.

Popis průběhu zpracování vstupního obrazu pro detekci dílčích rysů pomocí Dlib

Pro zpracování vstupního obrazu je nejprve potřeba tento obraz získat. K získání obrazu z kamery slouží třída `AVCaptureDeviceInput`. Pomocí ní lze nastavit jaké vstupní data bude zařízení odebírat. V rámci této třídy lze také nastavit jejich formát. Pro zobrazení obrazových vstupních dat na výstup slouží třída `AVCaptureVideoDataOutput`. Pomocí této třídy lze nastavit například zrcadlení přijatých dat, správnou orientaci zobrazených dat v závislosti na poloze zařízení a také nastavení formátu pixelů, které jsou následně ukládány do bufferu typu `CVPixelBuffer`. Ten pak udržuje vstupní data v hlavní paměti.

Pro úrovně využívající dílčí rysy obličeje jsou data získávána z přední kamery v HD rozlišení v rychlosti 30 FPS. Jednotlivé pixely jsou ukládány ve formátu `32-bit BGRA`. Tento formát byl zvolen z důvodu, že je jako jediný podporován jak na straně knihovny Dlib [18], tak ve třídě `AVCaptureVideoDataOutput` spjatou se zařízením iPhone.

Nad získaným obrazem uloženým v bufferu je provedeno vyhledání lidských tváří. Tyto nalezené lidské tváře jsou vyznačeny obdélníkem, který je ohraničuje. Pokud je nalezena právě jedna lidská tvář, je obraz společně se souřadnicemi obdélníku předán ke zpracování pomocí knihovny Dlib.

Pro získání dílčích rysů je využit již zmiňovaný model, který je schopný ve tváři naleznout 68 význačných bodů. Před použitím modelu je třeba ho načíst a převést z formátu `.dat` do objektu `shape_predictor`, se kterým je možné v rámci knihovny Dlib pracovat. Tento proces může na zařízení trvat i několik sekund a je tedy proveden hned po spuštění aplikace na vlákně s prioritou, která neblokuje plynulé chování aplikace. Model je pak v době potřeby již připraven a není třeba čekat na jeho načtení.

Optimalizace detekce dílčích rysů pomocí knihovny Dlib

Aby detekce dílčích rysů v obličeji pracovala v rychlosti 30 FPS, musel být kód práce s knihovnou Dlib optimalizován. Postup získávání dílčích rysů bylo po analýze možné rozdělit do tří nejvíce časově náročných kroků.

1. Převod obrazu z `CVPixelBuffer` do Dlib podoby typu `array2d<dlib::bgr_pixel>`. To trvalo průměrně okolo 20 ms.

2. Získání dílčích rysů z převedeného obrazu a jejich vyznačení. To trvalo průměrně okolo 22 ms.

3. Převod obrazu z `array2d<dlib::bgr_pixel>` do `CVPixelBuffer`. To trvalo průměrně 10 ms (oproti prvnímu převodu již nevyžadovalo vytváření nového pixelu).

Po testování na zařízení iPhone 7 se ukázalo, že první a třetí krok, který se samotnou detekcí nijak nesouvisí, zabírá okolo 58 % potřebného času pro zpracování. Převod obrazu mezi formáty byl implementován následujícím způsobem. Tento způsob byl inspirován volně dostupným ukázkovým kódem, který demonstruje využití knihovny Dlib v prostředí iOS.

Jak je vidět v následující ukázce implementace, obraz byl postupně procházen a jednotlivé barevné složky pixelů kopírovány do dvourozměrného pole obsahující typ `dlib::bgr_pixel`. Detekce pomocí třídy `shape_predictor` vyžaduje, aby vstupní obrazová data splňovala požadavky definované uvnitř třídy `generic_image`. Tato třída nevyžaduje konkrétní formát pixelu, podporuje všechny formáty uvedené v Dlib třídě `pixel_traits`, mezi kterými je zahrnut i použitý formát `24-bit BGR`, nebylo tedy nutné kopírovat `alpha` složku.

```
dlib::array2d<dlib::bgr_pixel> img;
long index = 0;
while (img->move_next()) {
   dlib::bgr_pixel& pixel = img->element();

   long bufferLocation = index * 4;
   char b = imageBuffer[bufferLocation];
   char g = imageBuffer[bufferLocation + 1];
   char r = imageBuffer[bufferLocation + 2];

   dlib::bgr_pixel newpixel(b, g, r);
   pixel = newpixel;

   index++;
}
```

Tento způsob převodu byl nevhodný, jelikož nedocházelo k získávání nových informací, ale pouze ke změně formátu jejich uložení. Místo kopírování hodnoty se jevilo mnohem vhodnější pouze předat ukazatel na data. Toho jsem docílil využití kompatibility Objective-C s OpenCV a následnou kompatibilitou mezi knihovnami OpenCV a Dlib. Část implementace zabývající se převodem je zachycena v následující ukázce.

```
cv::Mat mat(height, width, CV_8UC4, baseAddress,
            CVPixelBufferGetBytesPerRow(*imageBuffer));

dlib::cv_image<dlib::rgb_alpha_pixel> cimg(mat);
img = cimg;
```

Pro převod je využita třída z OpenCV s názvem `cv::Mat`. Tato třída stejně jako třída `array2d` v knihovně Dlib může reprezentovat dvourozměrná pole (dále matice). Lze ji inicializovat způsobem, který je zachycen v ukázce implementace. Stačí předat výšku a šířku matice, typ dat které ukládá, ukazatel na samotná data a krok, který značí kolik bytů zabírá jeden řádek matice. Typ dat byl nastaven pomocí konstanty `CV_8UC4`, která reprezentuje typ pro pixel obsahující 4 barevné kanály, kde je pro každý využito 8 bitů. Jelikož je BGRA v OpenCV výchozí formát uložení pixelu [27], je tento nastavený typ shodný s typem, ve kterém byla obrazová data uložena ze vstupu kamery. Tato konstrukce matice je velice rychlá, protože nevyžaduje žádné kopírování dat. Třídu `cv::Mat` lze pak použít pro inicializaci třídy `dlib::cv_image` – při převodu je potřeba dát pozor na typ pixelů, v tomto případě je odpovídající typ pro `CV_8UC4` typ `dlib::rgb_alpha_pixel`. Při tomto převodu opět nedochází ke kopírování dat. Jelikož třída `dlib::cv_image` splňuje požadavky definované uvnitř třídy `generic_image`, lze ji použít pro reprezentaci vstupních obrazových dat pro detekci pomocí třídy `shape_predictor` [36].

Pokud jsou obrazová data transformována tímto způsobem, trvá proces převodu průměrně 0.03 ms. Navíc jakákoliv změna dat je prováděna pouze na jednom místě a není nutné data transformovat nazpět. Oproti původnímu času nutnému pro celý převod, který trval

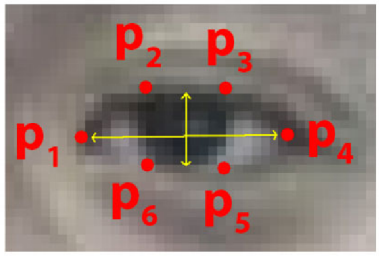
okolo 30 ms je tento způsob 1000x rychlejší. S pomocí této optimalizace nalezení dílčích rysů v obličeji probíhá v požadované rychlosti 30 FPS.

6.1.4 Zpracování získaných dílčích rysů

Jako ovládací prvky využité pro průběh hry byla mimo pozici dílčích rysů využita také detekce zavřených očí a detekce „tvaru“ úst.

Detekce zavřených očí

Pro detekci zavřených očí byl použit algoritmus implementovaný na základě článku, ve kterém je pro detekci zavřených očí využito tzv. „Eye Aspect Ratio“ (dále EAR) [42]. Tento přístup se dle článku osvědčil jako dostatečně spolehlivý a výpočetně nenáročný. Je navíc kompatibilní s přístupem vyznačení dílčích rysů v obličeji pomocí knihovny Dlib. Každé oko je vyznačeno šesti body. EAR využívá vztah mezi mezi výškou a šířkou pomyslného obdélníku, který body ohraničují. Pomyslný obdélník lze vidět na obrázku 6.3.



Obrázek 6.3: Šest bodů ohraničuje pomyslný obdélník [42].

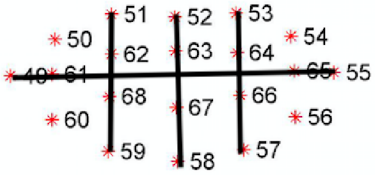
Na základě článku [42] pak tedy EAR získáme vztahem:

$$EAR = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2||p_1 - p_4||},$$

kde body p_1,...,p_6 odpovídají lokacím bodů znázorněných na obrázku 6.3. Z rovnice lze vidět, že v případě oka zavřeného bude hodnota EAR menší, než v případě oka otevřeného. Pak tedy stačí pro správnou detekci zavřeného oka vhodně stanovit prahovou hodnotu EAR, od které můžeme oko považovat za zavřené a minimální hodnotu po sobě jdoucích zpracovaných snímků, v rámci kterých se stav oka nemění. Experimentálně byla prahová hodnota EAR stanovena na hodnotu 0.24 a minimální počet po sobě jdoucích snímků byl nastaven na hodnotu 15. V průběhu souvislé detekce zavřeného oka jsou tolerovány dvě překročení prahové hodnoty EAR, tato tolerance umožňuje ignorovat drobné výchylky spojené s detekcí dílčích rysů pomocí knihovny Dlib. V případě detekce obou zavřených očí musí být uvedené podmínky splněny na obou očích zároveň.

Detekce úsměvu a otevřených úst

Podobným způsobem jako byla získána hodnota EAR, lze dle autora článku [1] detekovat i stav úst pomocí hodnoty tzv. „Mouth Aspect Ratio“ MAR. Pomyslný obdélník je nyní stanoven body, které ohraničují ústa viz 6.4.



Obrázek 6.4: Body ohraničující ústa tvoří pomyslný obdélník [1].

Na základě článku [1] pak hodnotu MAR získáme vztahem:

$$MAR = \frac{||p_{51} - p_{59}|| + ||p_{52} - p_{58}|| + ||p_{53} - p_{57}||}{3||p_{49} - p_{55}||},$$

kde body p_{49},...,p_{59} odpovídají bodům znázorněných na obrázku 6.4. Stejně jako u EAR, bylo potřeba určit prahové hodnoty MAR. Experimentálně byla pro úsměv stanovená hodnota menší než 0.24 a pro otevřená ústa hodnota větší než 0.5. Hodnoty v rozmezí mezi 0.24 a 0.5 pak značí ústa v neutrální pozici. Minimální počet po sobě jdoucích snímku i počet tolerovaných výchylek byl nastaven stejně jako u výpočtu hodnoty EAR.

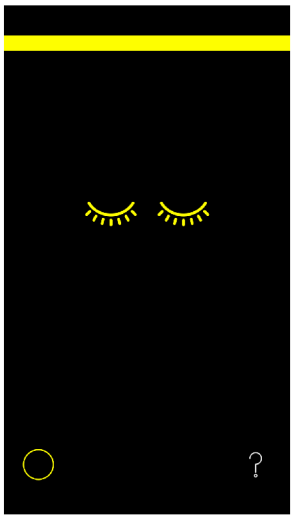
6.1.5 Využití frameworku ARKit pro získaní dílčích rysů tváře

V rámci aplikace jsem poskytl i podporu pro zařízení iPhone X a novější zařízení, které mohou pro detekci dílčích rysů lidské tváře využívat ARKit 4.1. Tento přístup poskytoval velmi dobré výsledky. Detekoval tváře i v horších světelných podmínkách a toleroval větší rozsah natočení tváře.

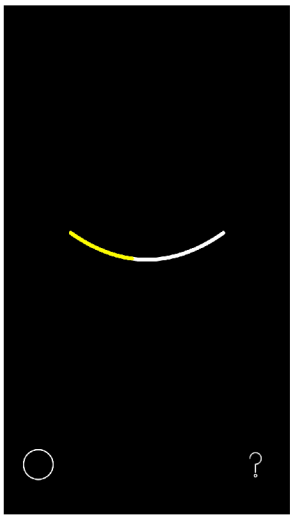
Při spuštění ARKitu s konfigurací typu `ARFaceTrackingConfiguration`, je v pravidelných intervalech na vstupu přední kamery vyhledáván lidský obličej. V případě jeho nalezení je zavolána metoda, v které je předán tzv. `ARFaceAnchor`. Ten poskytuje údaje o pozici, topologii a dílčích rysech tváře. Každý dílčí rys nabývá hodnot od 0.0 do 1.0, dle míry která odpovídá jeho hledanému výrazu. Nevýhodou toho přístupu je, že je metoda pro zpracování obličeje volána pouze v případě úspěšně nalezeného obličeje. V případě nenalezení obličeje ARKit žádnou veřejnou metodu nevolá. Pro zjištění absence obličeje na vstupu kamery je tedy potřeba si pamatoval poslední okamžik jejího úspěšného nalezení a v pravidelných intervalech kontrolovat, jestli tento časový okamžik není příliš starý.

6.1.6 Ukázka úrovní

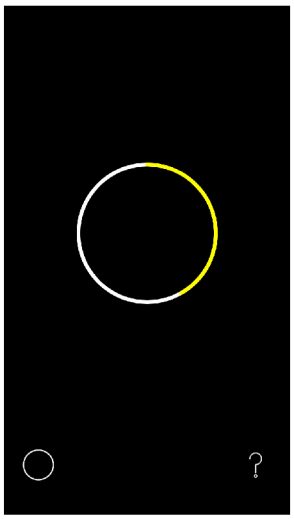
Celkově jsou v rámci hry pomocí technologie detekce dílčích rysů obličeje vytvořeny čtyři úrovně, které lze vidět na obrázku 6.5. Uživateli není dopředu vysvětleno, že je daná úroveň ovládána pomocí obličeje. Vodítkem pro vyřešení těchto úrovní je to, že kontrolní prvky značící časový interval úspěšného plnění úkonu jsou skryty v případě, že na vstupu přední kamery není detekován právě jeden obličej.



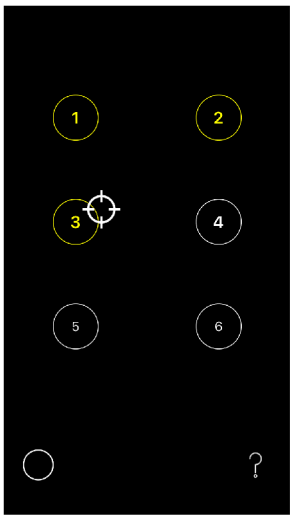
(a) Úroveň využívající detekci zavřených očí – úkolem úrovně je mít po dobu pěti sekund zavřené oči. Časomíru znázorňuje žlutý obdélník v horní části obrazovky, který postupně při detekci zavřených očí roste. V případě detekce otevřených očích, jsou místo očích zavřených uživateli obrázkem ukázány oči otevřené. Časomíra je umístěna do horní části obrazovky, aby uživatel směřoval svoje oči do míst, kde se nachází vstup přední kamery zařízení a hra tak poskytovala co nejpřesnější výsledky.



(b) Úroveň využívající detekci úsměvu – úkolem úrovně je se po dobu pěti sekund usmívat se zavřenými ústy. Úkon je pouze okrajově naznačen tvarem, které ústa v této pozici utváří.



(c) Úroveň využívající detekci otevřených úst – úkolem úrovně je mít po dobu pěti sekund mít otevřená ústa. Úkon je pouze okrajově naznačen tvarem, které ústa v této pozici utváří.



(d) Úroveň využívající detekci pozice nosu – úkolem úrovně je projít kurzorem graficky zobrazené terče ve správném pořadí. Pozice kurzoru odpovídá pozici nosu snímaného obličeje v rámci vstupního obrazu.

Obrázek 6.5: Úrovně využívající detekci dílčích rysů obličeje.

6.2 Úrovně využívající zpracování barevnosti vstupního obrazu kamery

Další skupinu tvoří úrovně využívající průměrné barvy vstupního obrazu. Získaní vstupních obrazových dat pro následovné zpracování probíhá již představeným způsobem pomocí frameworku AV Capture.

6.2.1 Získání průměrné barvy ze vstupních dat

Pro získání průměrné barvy vstupních obrazových dat je využit již zmiňovaný nativní framework Core Image. Ten poskytuje filtry pro zpracování a analýzu statického obrazu a videa. Mezi těmito filtry je zahrnut i filtr `CIAreaAverage`. Tento filtr dokáže v obraze reprezentovaném objektem typu `CIImage` vyhodnotit jeho průměrnou barvu. Průměrná barva je s pomocí filtru získána a reprezentována objektem typu `CIImage`, který je tvořen jedním pixelem, který tuto průměrnou barvu reprezentuje [4].

Přestože je typ `CIImage` spjatý s obrazovými daty, spíše než samotný obraz obsahuje „recept“ na jeho vytvoření. Obraz není vykreslen do doby, než je o to explicitně požádán. Proto je pro získání RGBA hodnot pixelu obraz nejprve potřeba vykreslit v grafickém kontextu. V rámci kontextu je třeba nastavit správný barevný prostor a formát reprezentace barevných složek pixelu.

Barevný prostor reprezentuje jakou škálu barev je zařízení schopno zobrazit. Všechna zařízení do verze iPhone 7 používají barevný prostor *sRGB IEC 61966–2–1* novější pak barevný prostor *Display-P3* [9]. Apple doporučuje na zařízení skrz kompatibilitu používat první zmíněný prostor a barevný prostor *P3* používat jen ve specifických situacích, kde můžou intenzivnější barvy zásadně vylepšit uživatelův dojem z grafického rozhraní aplikace [6]. Pro vykreslení pixelu byl zvolen barevný prostor *sRGB*.

Rozdíl mezi dvěma barvami

Zpracování barevného údaje ze vstupního obrazu bylo v rámci hry zamýšlené pro využití detekce podobných barev. Uživatel by pak tedy v reálném prostředí hledal aplikací představenou barvu. Pro umožnění takového porovnání bylo potřeba získat rozdíl těchto barev, nebo-li jejich „vzdálenost“.

Barvu lze volně chápat jako „psychofyzický“ vjem, který se projevuje během stimulu našeho zrakového systému. V principu se jedná o způsob, jakým naše sítnice interpretuje vlnovou délku vstupujícího světla. Lidské oko dokáže vnímat světlo o vlnové délce v rozmezí 380 - 750 nm [40].

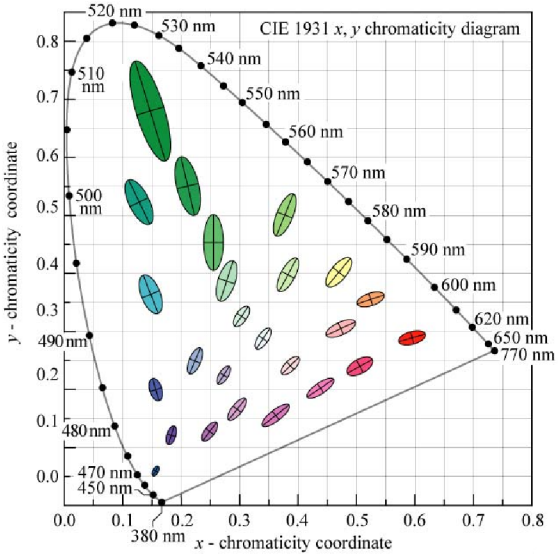
Jako první způsob porovnání dvou barev se nabízela Eukleidova vzdálenost, která by značila vzdálenost dvou bodů v trojrozměrném prostoru vymezeném složkami v barevném prostoru sRGB. Vzdálenost byla vypočtena dle následující rovnice:

$$distance = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}, \tag{6.1}$$

kde R představuje červenou barevnou složky, G zelenou a B modrou. Tento výpočet dával klamné výsledky, které o podobnosti barev z hlediska percepce lidského oka nijak nevypovídaly.

Problém určení podobnosti barev je mimo jiné řešen například v tiskařství. Zde je nutné měřit okem vnímaný rozdíl barev z důvodu standardizace a stanovení limitní tolerované odchylky barev tisknuté kopie od originálu. Vznikla tedy snaha o vytvoření barevného

prostoru, ve kterém by vzdálenost mezi body definující barvu poměrově odpovídala lidským okem vnímanému rozdílu.



Obrázek 6.6: Diagram MacAdam v barevném prostoru CIE s fixní složkou světlosti [40]. Zobrazené elipsy jsou pro názornost desetkrát zvětšeny. Elipsy ohraničují body zachycující barvu limitně podobnou barvě referenční. Takový popis jednotlivých oblastí vyžaduje funkci a není tedy „percepčně uniformní“.

K získání takového prostoru je využit matematicky definovaný barevný prostor *CIE XYZ*. Ten byl vytvořen mezinárodním úřadem Commission Internationale de l'Eclairage (dále CIE) v roce 1931. Je tvořen třemi složkami, kde složka Y představuje jas a samotná barva je pak určena složkami X a Z. Složky X a Z byly standardizovány úřadem CIE na základě statistik z experimentů, které zahrnovali lidské pozorovatele. V tomto prostoru lze vyjádřit všechny člověkem vnímané barvy, ale vyjádření rozdílu mezi těmito barvami není „percepčně uniformní“. V tomto barevném prostoru tedy nelze stanovit vzdálenost, která by okolo referenčního bodu v barevném prostoru vytvořila oblast, do které by spadaly všechny body značící lidským okem vnímané barvy stejnou mírou podobné k té referenční. Taková oblast je v rámci tohoto barevného prostoru definována pomoc tzv. *MacAdam ellipse*, pro lepší představu viz obrázek 6.6

Na základě barevného prostoru *CIE XYZ* byl v roce 1976 definován úřadem CIE barevný prostor *CIEL*a*b** (dále Lab). Tento barevný prostor je na rozdíl od prostoru *CIE XYZ* percepčne uniformní. Je tvořen třemi složkami:

1. $L*$ - Světlost, která nabývá hodnot od 0 do 100.
2. $a*$ - V intervalu od -128 do 127 značí přechod mezi zelenou a červenou barvou, kde hodnota -128 značí barvu zelenou a 127 barvu červenou.
3. $b*$ - V intervalu od -128 do 127 značí přechod mezi modrou a žlutou barvou, kde hodnota -128 značí barvu modrou a 127 barvu žlutou.

V tomto barevném prostoru lze zobrazit všechny viditelné barvy. Viditelné barvy dokonce představují pouze 40 % všech v tomto prostoru reprezentovatelných barev. Prostor

Lab je založen na principu „protilehlých barev“ – barvy které leží na opačných stranách nemohou být viděny zároveň [32].

Vypočítáním Eukleidovy vzdálenosti podle rovnice (6.2) lze získat vzdálenost ΔE_{Lab}

$$\delta E_{L_{ab}} = \sqrt{(\Delta L*)^2 + (\Delta a*)^2 + (\Delta b*)^2}, \tag{6.2}$$

která se blíží vnímanému rozdílu těchto barev lidským okem a v rámci aplikace se jevila jako dostatečná.

Pozorovatel v závislosti na hodnotě $\Delta E_{L_{ab}}$ vnímá rozdíly barev dle [32] následovně:

1. $0 < \Delta E < 1$ - pozorovatel nevnímá rozdíl,
2. $1 < \Delta E < 2$ - jen někteří pozorovatelé vnímají rozdíl,
3. $2 < \Delta E < 3.5$ - všichni pozorovatelé vnímají jemný rozdíl,
4. $3.5 < \Delta E < 5$ - pozorovatel vnímá zjevný rozdíl,
5. $5 <$ - pozorovatel vnímá odlišné barvy

Tyto data reprezentují experimentálně verifikované statistiky. V rámci hry nebylo nutné, aby uživatel našel barvy shodné. Experimentálně byly stanovené vlastní hodnoty tolerance.

Převod z barevného prostoru sRGB do XYZ

Na začátku jsou jednotlivé hodnoty V barevných kanálů normalizovány do intervalu $V \in < 0, 1 >$. Následně jsou jednotlivé kanály z barevného prostoru sRGB převedeny do jejich lineární podoby. Barvy jsou totiž v barevném prostoru sRGB kódovány pomocí „gamma korekce“, která zohledňuje nelineární vnímaní světlosti lidským okem. Tyto barvy jsou následně při zobrazení na monitoru dekódovány inverzní operací, která důvěryhodně zachycuje barvy v původních světelných podmínkách. Standardizovaná hodnota gamma pro zobrazení na zařízeních je zobecněna na hodnotu $\gamma = 2.2$. Obecnou rovnici pro převod mezi lineárním a „zakódovaným“ prostorem lze vidět v rovnici (6.3) a (6.4).

$$V_{encoded} = V_{linear}^{1/\gamma}, \tag{6.3}$$

$$V_{linear} = V_{encoded}^{\gamma}, \tag{6.4}$$

kde V znázorňuje normalizovanou barevnou složku příslušného barevného prostoru. V případě sRGB funkce převodu obsahuje navíc krátký lineární segment, který zamezuje nepodstatnému výpočtu u nízkých hodnot barevné složky V blížících se 0, kde je při linearizaci velmi malý nárůst hodnot [34]. Získání lineárních hodnot značených jako $v \in \{r, g, b\}$ lze získat ze zakódovaných složek v prostoru sRGB, značených jako $V \in \{R, G, B\}$, následovně:

$$v = \begin{cases} \frac{V}{12.92}, & \text{if } V \leq 0.04045 \\ ((\frac{V+0.055}{1.055})^{2.4}, & \text{jinak} \end{cases}, \tag{6.5}$$

nad získanými RGB hodnotami je poté provedena lineární transformace do prostoru XYZ, která odpovídá změně primárních barev:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9504 \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix}, \tag{6.6}$$

kde jsou koeficienty zvoleny s ohledem na převod z barevného prostoru sRGB do barevného prostoru XYZ s bílým bodem stanoveným standardem D65, který definoval úřad CIE a je použit v barevném prostoru sRGB [26].

Převod z barevného prostoru XYZ do Lab

Tento převod lze provést následujícím způsobem:

$$L* = 116(Y/Y_0)^{1/3} - 16, \tag{6.7}$$

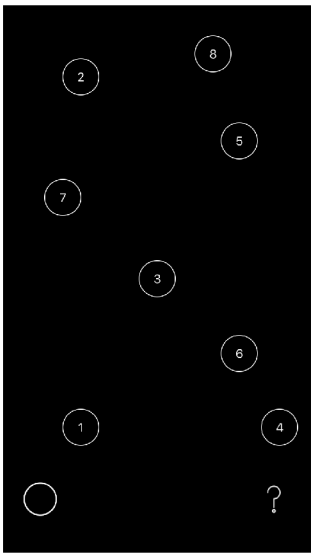
$$a* = 500((X/X_0)^{1/3} - (`y/Y_0)^{1/3}), \tag{6.8}$$

$$b* = 200((Y/Y_0)^{1/3} - (Z/Z_0)^{1/3})), \tag{6.9}$$

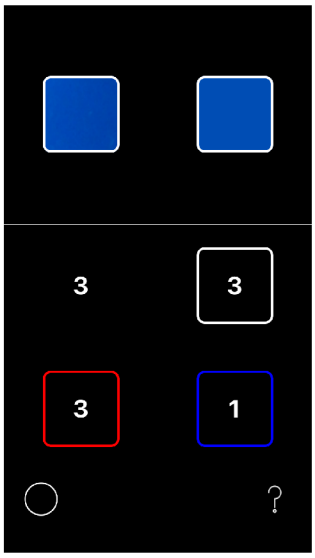
kde: $X_0 = 95.047, Y_0 = 100, Z_0 = 108.883$ jsou koeficienty stanovené úřadem CIE. Koeficienty jsou stanoveny vzhledem k hodnotě $Y_0 = 100$ a bílému bodu určenému standardem D65 [32].

6.2.2 Ukázka úrovní

V rámci hry jsou vytvořeny dvě úrovně, které využívají implementace převodu barevného prostoru popsanou v předchozí kapitole. Jedna úroveň využívá porovnávání dvou barev, druhá využívá hodnoty $L*$ z barevného prostoru Lab, která přibližně popisuje lidské vnímání světlosti. Jejich konkrétní podobu lze vidět na obrázcích 6.7.



(a) Úroveň využívající průměrné vstupní světlosti. Úkolem úrovně je postupně ve chronologickém pořadí stisknout zobrazená tlačítka. Tlačítka ale nejsou zobrazena do doby, než je průměrná světlost vstupního obrazu nižší než přednastavená prahová hodnota. Uživateli je poskytnuta nápověda postupnou změnou pozadí obrazovky do úrovní šedi, která odpovídá hodnotě vstupní světlosti.



(b) Úroveň využívající podobnosti průměrné vstupní barvy s barvami referenčními. Úkolem úrovně je postupně ve výřezu vstupního obrazu zadní kamery detekovat po dobu tří sekund barvy referenční. Referenční barvy jsou znázorněny rámečky čtverců pod bílou čarou. Rámeček obsahuje číslo, které značí zbylou potřebnou dobu detekce barvy, kterou reprezentuje. Nad bílou dělící čarou je vlevo zobrazen výřez obrazu z kamery a napravo od něj jeho průměrná barva.

Obrázek 6.7: Ukázka implementovaných úrovní, které pro svou funkcionality využívají zpracování barevnosti a světlosti vstupního obrazu.

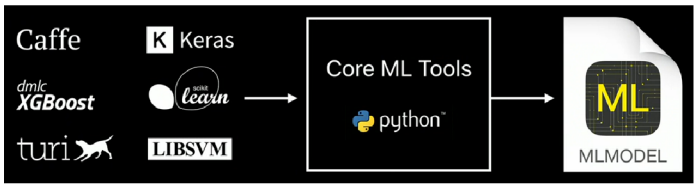
6.3 Úrovně využívající framework Core ML

S představením frameworku Core ML na WWDC 2017 [14] existuje na platformě iOS nativní způsob integrace modelů strojového učení. Toho je využito k tvorbě několika dalších úrovní výsledné hry. V této sekci je ve zkratce představen framework Core ML a následně uvedeny některé možné postupy pro vytvoření pro něj kompatibilního modelu. Na závěr jsou představeny úrovně, ve kterých byly tyto modely využity.

6.3.1 Framework Core ML a kompatibilní typy modelů

Pomocí frameworku Core ML lze model získaný pomocí strojové učení nativně integrovat do iOS aplikace. Core ML je framework optimalizovaný pro provoz přímo na zařízení za co nejmenších nároku na paměť a spotřebu energie. Díky tomu, že funguje přímo na zařízení, zamezuje možnému úniku osobních dat a není závislý na připojení k internetu.

Core ML vyžaduje model ve formátu *.mlmodel*. Ten lze získat buď jeho samotným vytvořením například v systému *Mac OS* pomocí frameworku *Create ML*, nebo převodem z modelu vytvořeném v jiném prostředí – například pomocí nástroje *Core ML Tools*. Ten dokáže modely vhodných typů převést z kompatibilních knihoven do formátu *.mlmodel*. Kompatibilní knihovny lze vidět na obrázku 6.8 [4].



Obrázek 6.8: Knihovny, ze kterých je možné vytvořené modely vhodného typu převést do formátu *.coreml* pomocí nástroje *Core ML Tools* (Zdroj obrázku: Apple WWDC2017)

V rámci aplikace jsem se rozhodl využít celkově tři modely. Všechny slouží pro vyhledání objektu v obrazových datech. K tomu je použit framework Core ML v kombinaci s frameworkem Vision, který lze využít pro analýzu obrazových dat.

Jako první jsem vyzkoušel od firmy Apple již připravený model s názvem *MobileNet*. Jedná se o model, který dokáže pomocí neuronových sítí rozpoznat dominantní objekt v obraze. Celkově dokáže vstupní data rozlišit až do 1000 kategorií. Po vyzkoušení tohoto modelu v rámci aplikace jsem se rozhodl vytrénovat modely vlastní. Jeden s využitím knihovny Keras a druhý pomocí knihovny Azure.

6.3.2 Model vytvořený pomocí knihovny Keras

Keras je knihovna napsaná v jazyku Python a je schopná fungovat nad knihovnami TensorFlow, CNTK a Theano. Tato knihovna vznikla se záměrem vytvoření uživatelsky přívětivého prostředí pro snadnou tvorbu modelů využívajících konvolučních a rekurentních vrstev. Proces trénování s využitím knihovny Keras je schopný provozu na CPU i na GPU [25]. Modely, které využívají CNN a jsou vytvořeny pomocí této knihovny, jsou kompatibilní s nástrojem *Core ML Tools* [4].

Trénovací data

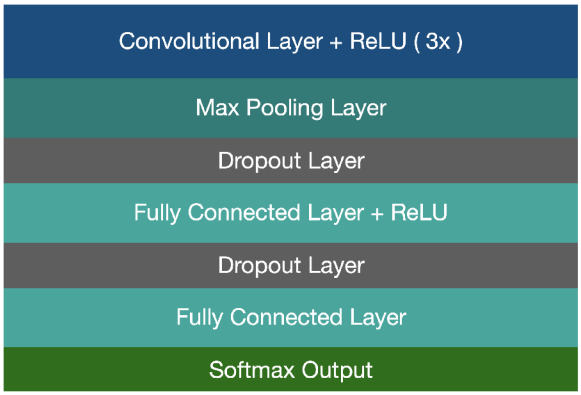
Pro trénování byla využita data z datasetu *Quick, Draw!*. Jedná se o open-source dataset kreslených skic obrázků, který pomohlo vytvořit přes 15 milionů lidí. Tento dataset obsahuje 345 kategorií a jednotlivé obrázky jsou dostupné i ve formátu *.npy*, který byl pro vytvoření modelu využit.

Pro náplň jedné úrovně jsem chtěl vytvořit model, který by byl schopný detekovat uživatelem nakreslené jablko. Vytvořil jsem tedy dvě skupiny dat, kde jednu skupinu tvořily obrázky jablek a druhou skupinu náhodně vybrané typy kreslených objektů. Každá skupina obsahovala celkově 20 000 vzorků. Při trénování model hlásil přesnost nad testovacími daty přes 90 %. Po jeho zahrnutí do aplikace, ale za jablko považoval i kulaté tvary, které jablko nepředstavovaly.

Z toho důvodu jsem trénovací data upravil a využil více kategorií, které jsou jablku podobné. Konkrétně jsou to kategorie znázorňující budík, borůvku, kruh, žárovku, hrušku a bramboru. Navíc jsem zvýšil množství vstupních dat na 30 000 vzorků u každé ze skupin, kde jedna obsahovala obrázky jablka a druhá obrázky jiného objektu.

Trénování modelu

Samotné trénování pak pomocí knihovny Keras proběhlo s následujícími parametry. Tyto parametry byly zvoleny na základě dvou článků, které pro trénování CNN modelu využívaly stejný dataset [2][21]. Parametry jsem následně experimentálně upravoval. Hodnota *batch*, která určuje nutný počet zpracovaných vzorků před aktualizací interních parametrů modelu, byla zvolena hodnota 256. Celkově trénování proběhlo ve třech epochách – každý obrázek tedy ovlivnil hodnotu vnitřních parametrů modelu třikrát [16]. Složení CNN lze vidět na obrázku 6.9. V modelu bylo na testovacích vstupních datech dosaženo přesnosti 96 %. Model byl následně převeden do formátu *.mlmodel* pomocí nástroje *Core ML Tools* a zahrnut do aplikace.



Obrázek 6.9: Ukázka složení CNN ve vytvořeném modelu pomocí knihovny Keras. Model byl trénován celkově nad 30 000 vstupními obrazy a na testovacích vstupních datech dosahoval přesnosti 96%.

6.3.3 Model vytvořený pomocí knihovny Azure Custom Vision

Azure Custom Vision je služba od společnosti Microsoft, která slouží pro vytvoření modelů pro klasifikaci obrazových dat pomocí strojového učení. Pro vytrénovaní modelu stačí poskytnout dataset obrázků rozdělených do požadovaných kategorií. Dle dokumentace je

tato služba optimalizovaná pro rychlé rozpoznání hlavních rozdílu ve vstupních obrazech a pro základní funkcionalitu modelu stačí i malý počet vzorků dat vstupních kategorií – konkrétně dokumentace uvádí 50 vstupních obrazů na kategorii. Tento způsob vytvoření modelu tedy dle dokumentace není vhodný pro detekci nepatrných změn ve vstupních datech. Samotný průběh trénovaní je pak odstíněn. V omezené podobě lze tuto službu využít zdarma a vytrénovaný model lze exportovat i do formátu *.mlmodel* [30].

Trénovací data

Pro další úroveň jsem se rozhodl pomocí služby Azure Custom Vision vytvořit model, který by byl schopný rozpoznat ze vstupního obrazu gesto ruky. Při sběru dat jsem kladl důraz na robustnost modelu vůči světelným podmínkám, použití pravé/levé ruky a vzdálenosti ruky od kamery. Celkově jsem získal fotky požadovaného gesta ruky od devíti lidí. Vedle skupiny obsahující hledané gesto ruky jsem vytvořil sadu náhodných obrázků o stejném počtu. Obě kategorie obrázků v prvním kole obsahovaly doporučených 50 vzorků. Tyto obrázky jsem následně převedl do jednotné velikosti s šířkou 300 pixelů. Po vytrénovaní tohoto modelu jsem si po jeho otestování uvědomil nedostatky takto vytvořeného datasetu. Za hledané gesto ruky byl totiž modelem považován jakýkoliv obrázek obsahující ruku.

Z tohoto důvodu jsem vytvořil další sadu obrázků, která obsahovala obrázky jiných nehledaných gest ruky. Model jsem pak trénoval v iteracích, kde jsem při každé iteraci model otestoval a na základě chybných predikcí do jednotlivých skupin přidal nové vzorky vstupních dat. Konečný počet vstupních dat obsahoval 130 obrázků v každé kategorii. Po vytrénování služba Custom Vision odhadovala přesnost 90 %, testování ale provádí na trénovacích datech, což může být údaj skrz možný jev *over-fitting* poněkud zavádějící. Samotné testování tedy proběhlo následně v rámci aplikace. Model poskytoval pro nároky hry uspokojivé výsledky. Pro robustnější model by bylo potřeba dále zvětšovat trénovací dataset.

6.3.4 Ukázka úrovní

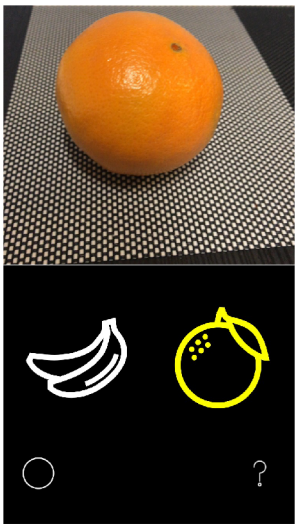
V rámci hry jsou vytvořeny tři úrovně, které využívají detekci objektu v obraze. Jejich princip byl již nastíněn při popisu použitých technologií a lze je vidět na obrázcích 6.10 a 6.11. Dvě úrovně využívají vstup obrazu z kamery a jedna úroveň postupně vyhodnocuje kresby jablka, které uživatel provádí dotykem na obrazovce zařízení.

6.4 Úrovně využívající zpracování zvuku

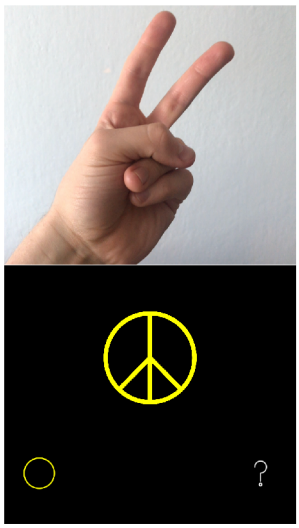
Pro další skupinu úrovní byla využita možnost na zařízení nahrávat a dále zpracovávat zvuk. Konkrétně bylo pro tvorbu úrovní využito rozpoznání řeči a informace o úrovni hlasitosti vstupního zvuku.

6.4.1 Detekce hlasitosti

Pro získání úrovně hlasitosti přímo ze vstupu zabudovaného mikrofonu lze využít třídu `AVAudioRecorder`. Pomocí této třídy lze zvuk ze vstupu nahrávat a při pořizování záznamu získávat úroveň jeho hlasitosti. Před samotným nahráváním, je třeba nastavit formát práce se zvukem v rámci aplikace ve třídě `AVAudioSession` do stavu *record*. Následně je možné spustit samotné nahrávání a v pravidelných intervalech se doptávat na aktuální úroveň vstupní hlasitosti. Nahrávka je ukládána do složky vyhrazené pro dokumenty aplikace

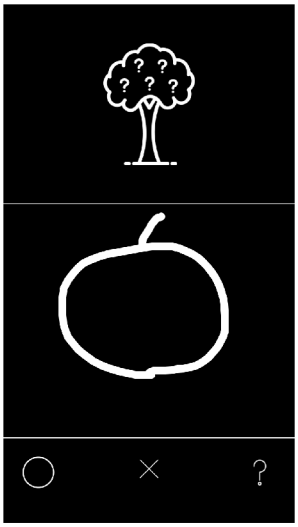


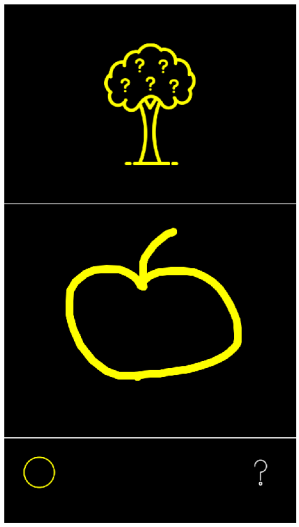
(a) Úroveň, která v obraze ze vstupu zadní kamery vyhledává banán a pomeranč. Pro úspěšné splnění úrovně je třeba detekovat postupně oba tyto objekty.



(b) Úroveň, ve které je pro její úspěšné splnění nutno v obraze ze vstupu zadní kamery detekovat představené gesto ruky.

Obrázek 6.10: Úrovně, které využívají vyhledání objektu v obraze ze vstupu kamery.





Obrázek 6.11: Úroveň hry, ve které je pro splnění nutné na obrazovku zařízení nakreslit jablko. V případě, že je nakresleno jiné ovoce i jablku podobné (jak je tomu vidět na obrázku vlevo) model správně vyhodnocuje, že se o jablko nejedná. V případě nakreslení jablka pak model obrázek úspěšně vyhodnocuje.

a v případě, že je využita pouze pro detekci vstupní hlasitosti, je potřeba ji po dokončení nahrávání explicitně smazat.

6.4.2 Detekce řeči s využitím frameworku Speech

Pro rozpoznání řeči v reálném čase přímo ze vstupu mikrofonu lze od iOS 10 využít framework Speech, který dokáže rozpoznat řeč v několika jazycích. Dle dokumentace dokáže

řeč zpracovávat přímo na zařízení, ale v některých případech je závislý na serverech společnosti Apple. Při využití tohoto frameworku je tedy nutno vždy předpokládat, že vyžaduje připojení k internetu.

Z toho důvodu jsou na jeho využití společností Apple stanoveny limity tak, aby nebyly jejich servery příliš zahlcené. Každé zařízení má tedy stanovený limit na počet dotazů za den. Pokud aplikace zasílá příliš mnoho žádosti o zpracování, může jí být přístup k této službě globálně odepřen. Navíc je tento přístup rozpoznání relativně náročný na baterii a rozpoznání řeči by nemělo kontinuálně probíhat déle, než jednu minutu. Tato jedna minuta ale není zaručena a služba pro rozpoznání řeči může být nezávisle na programátorovi ukončena i po kratším časovém intervalu [4].

Tato detekce na zařízení fungovala velice přesně a rychle a nevyžadovala zahrnutí nové knihovny. Bohužel nebylo využití tohoto přístupu v průběhu hry skrze uvedená omezení možné.

6.4.3 Detekce řeči s využitím knihovny OpenEars

Tato knihovna je alternativou k frameworku Speech. Jedná se o SDK, které je možné zdarma v rámci iOS aplikace použít pro rozpoznání řeči bez nutnosti připojení k internetu. SDK bylo navrženo přímo pro platformu iOS a je kompatibilní i s jazykem Swift.

Rozpoznání řeči pomocí této knihovny v aplikaci nedosahuje tak rychlých a přesných výsledků jako framework Speech, ale neobsahuje podobná omezení. Z tohoto důvodu byla pro rozpoznání lidské řeči v rámci hry využita tato knihovna.

6.4.4 Ukázka úrovní

Pro hry jsou vytvořeny dvě úrovně, kde jedna pro svoji náplň využívá detekci hlasitosti vstupního zvuku a druhá i zpracování řeči. V obou případech je naznačeno možné řešení úrovně pomocí zvuku grafickým ekvalizérem ve spodní části obrazovky. Úrovně s jejich popisem lze vidět na obrázku 6.12.

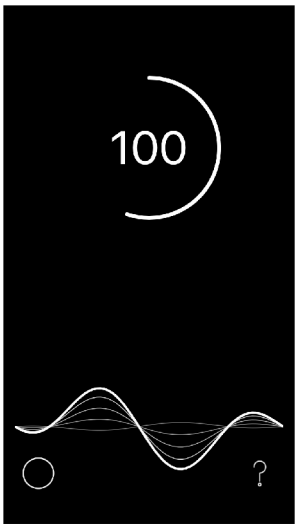
6.5 Úroveň využívající framework Core Motion

Framework Core Motion umožňuje v rámci aplikace získávat a využívat hardwarem generovaná data, které informují o pohybu fyzického zařízení a prostředí, ve kterém se vyskytuje. V rámci aplikace byla využita třída `CMMotionManager`, která slouží pro zahájení získávání a zpracování pohybových dat. Tato data byla využita pro detekci rotace zařízení u o 360° podle os, které lze vidět na obrázku 6.13.

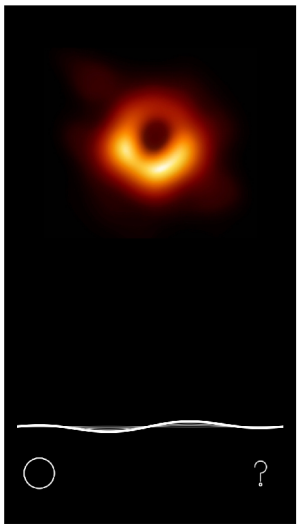
Pro detekci rotace je potřeba postupně detekovat změny rotace v jednotlivých směrech, které nabývají hodnot v rozmezí $-\pi$ až π. Ty lze pomocí třídy `CMMotionManager` detekovat, ale je třeba dát pozor na to, že rozsah hodnot rotace není napříč osami jednotný. Navíc je třeba hlídat změnu směru rotace tak, aby byla celá rotace provedena pouze do jednoho směru. Grafickou podobu úrovně lze vidět na obrázku 6.14.

6.6 Úrovně využívající další možnosti chytrých telefonů

V této sekci je uvedeno posledních pět úrovní, které využívají některé další informace získané z telefonu. Ve zkratce se jedná o detekci připojení k internetu, detekci nastavené úrovně hlasitosti, informace o tichém režimu, nastavené úrovně jasu, detekci zapojených sluchátek



(a) Úroveň využívající detekci hlasitosti vstupního zvuku. Úkolem této úrovně je po dobu 2 sekund, udržovat dostatečnou hlasitost nad prahem, který byl stanoven experimentálně. Dosáhnutí tohoto prahu značí spuštění časomíry a zobrazení hodnoty 100, která se mění v závislosti na vstupní hlasitosti. Úroveň lze tedy splnit dostatečně hlasitým projevem, nebo také „fouknutím“ do mikrofonu.



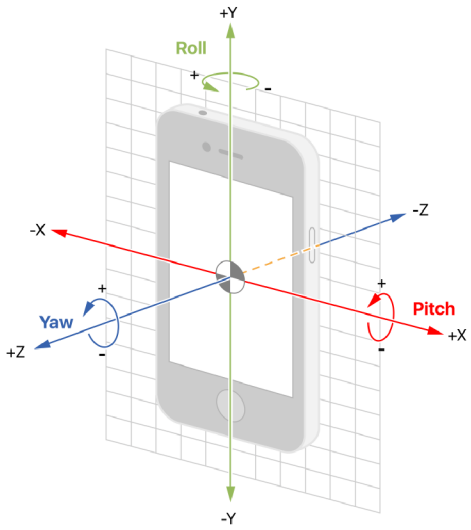
(b) Úroveň využívající detekce lidské řeči. Úkolem této úrovně je v angličtině vyslovit název objektu, který je zachycen na obrázku. Jedná se o první obrázek černé díry. Obrázek byl vědci vytvořen pomocí dat získaných ze soustavy teleskopů tzv. „Event Horizon Telescope“. Odpovědí na tuto úroveň je tedy vyslovení slova „blackhole“ [19].

Obrázek 6.12: Ukázka implementovaných úrovní, které pro svou funkcionality využívají zpracování vstupního zvuku.

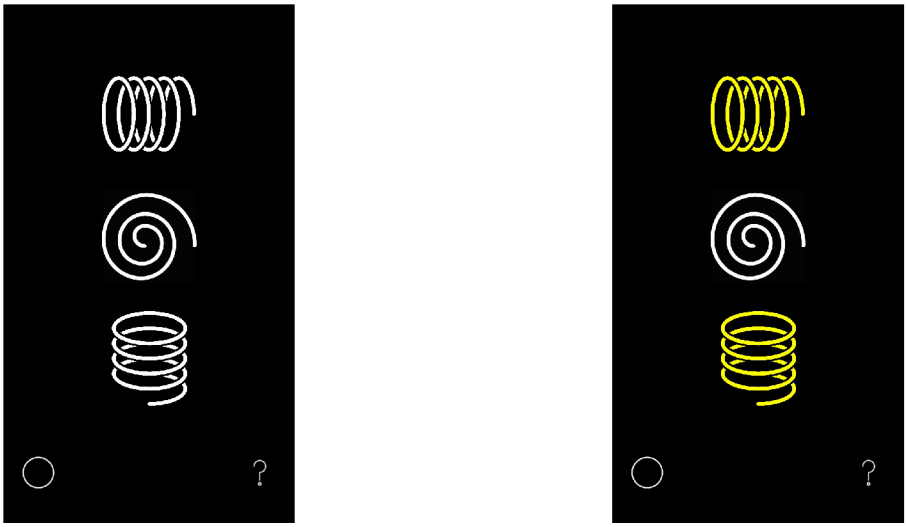
a nabíjení zařízení. Jedna úroveň také využívá „tradiční“ ovládání pomocí detekce krátkého a dlouhého stisknutí obrazovky. Při implementaci těchto úrovní byly využity frameworky systému iOS, s výjimkou detekce připojení k internetu a tichého režimu.

Pro detekci připojení k internetu je využita knihovna *Reachibility* a pro detekci tichého režimu knihovna *Mute* – systém totiž neposkytuje veřejné API pro detekci tichého režimu zařízení. Tichý režim lze zjistit pomocí přehrání zvuku a následným získáním informace o době, kterou byl přehráván. V případě, že byl zvuk přehráván velice krátkou dobu blížící se hodnotě nula lze předpokládat, že je zařízení v tichém režimu. Tato knihovna toho využívá a v pravidelných intervalech přehrává zvukovou stopu a kontroluje dobu přehrávání.

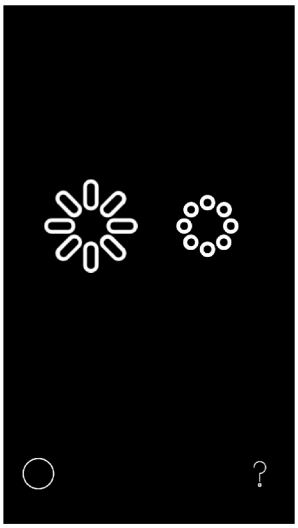
Způsob, jakým byly tyto získané informace využity pro průběh hry lze vidět na následujících obrázcích.



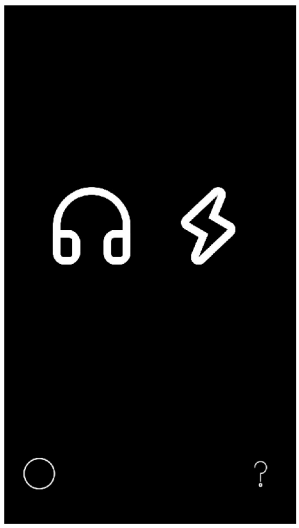
Obrázek 6.13: Ukázka označení os zařízení [4]. Pozice označující natočení zařízení na osách Roll a Yaw jsou rozděleny do dvou intervalů, kde je pozice značena v rozmezí od π do 0 a následně od 0 do $-\pi$. V tomto případě tedy stačí postupně odečítat pootočení a kontrolovat, jestli hodnoty po celou dobu rostou, nebo klesají. Pozice na ose Pitch je rozdělena do čtyř intervalů, kde dva intervaly nabývají hodnot od 0 do $\pi/2$ a dva od 0 $-\pi/2$. V rámci hry je kontrolováno, že se zařízení svým pootočením nacházelo ve všech těchto intervalech alespoň jednou.



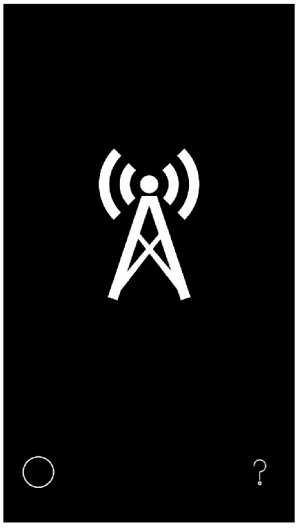
Obrázek 6.14: Ukázka úrovně hry, pro jejíž splnění je potřeba zařízení otočit o 360° podle tří os. Rotace zařízení je naznačena třemi spirálami, kde postupně provedené rotace označují příslušné spirály žlutou barvou.



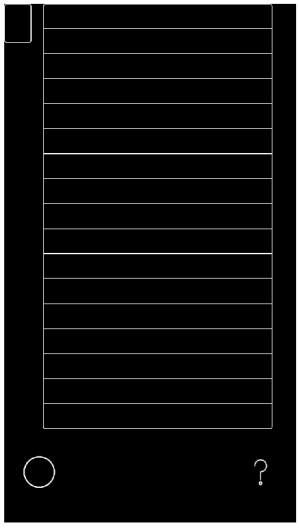
Obrázek 6.15: Úroveň, která využívá detekce nastaveného jasu obrazovky zařízení. Pro její splnění je nutné postupně v centru ovládání zařízení nastavit minimální a maximální úroveň jasu.



Obrázek 6.16: Úroveň využívající detekce připojení sluchátek a detekci nabíjení zařízení. Postupně je tedy třeba zapojit do zařízení sluchátka a nabíječku.

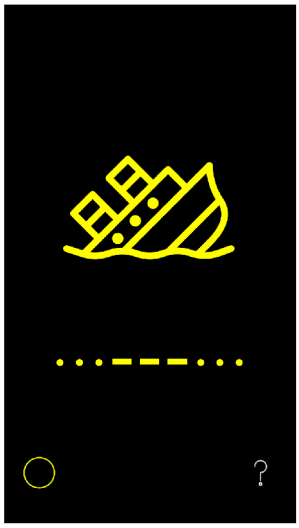


Obrázek 6.17: Úroveň, která pro svůj průběh využívá detekci připojení k internetu. Pro její splnění je potřeba zamezit zařízení jakékoliv připojení k internetu – například zapnutím letového režimu a vypnutím připojení k síti pomocí Wi-Fi.



Obrázek 6.18: Úroveň, která využívá detekci v zařízení nastavené úrovně hlasitosti a detekci tichého režimu. Pro její splnění je nutno pomocí tlačítek na zařízení postupně nastavit všechny úrovně hlasitosti a následně zapnout a vypnout tichý režim – tento úkon je značen obdélníkem vlevo nahoře, který svou pozicí odpovídá pozici tlačítka, které slouží pro aktivaci tichého režimu. Jednotlivé vodorovné obdélníky pak odpovídají konkrétní nastavené úrovni hlasitosti.

(a) Úroveň v nevyřešeném stavu.



(b) Úroveň ve vyřešeném stavu.

Obrázek 6.19: Pro vyřešení této úrovně je potřeba pomocí kombinace krátkého a dlouhého stisknutí obrazovky napsat Morseovou abecedou signál SOS.

Kapitola 7

Implementace

V této kapitole je představeno několik implementačních technik v jazyze Swift. Kapitola se nezabývá implementačními detaily jednotlivých úrovní, ale představuje techniky použité v rámci implementace celé hry. Tyto techniky se postupně projevily jako užitečné a vedly k tvorbě robustního a rozšiřitelného kódu.

7.1 Využití protokolů v jazyce Swift

Na WWDC 2015 Apple uvedl přednášku, na které byl jazyk Swift prohlášen za protokolově orientovaný [12]. Protokoly lze chápat jako předpis pro metody, proměnné a další požadavky, které zaobalují určitou funkcionalitu. Tyto předpisy pak můžou splňovat jak třídy, tak i struktury nebo výčtové typy (enum). Tyto typy jsou zavázány požadavky stanovené předpisy implementovat. O těchto typech pak můžeme prohlásit, že splňují daný protokol.

Pomocí protokolu lze řešit některé implementačních problémy lépe než čistě pomocí objektově orientovaného programování (OOP). Jedním z těchto problémů může být například problém spojený s potřebou vícenásobné dědičnosti. S růstem programu může třída vyžadovat funkcionalitu z více již existujících tříd. Jako řešení se nabízí vícenásobná dědičnost, která sebou přináší různá úskalí. Třída totiž poskytuje veškerou, i nepotřebnou, funkcionalitu svých předků, což vede ke zvýšení její komplexnosti. Toto řešení není v jazyce Swift ani možné, jelikož Swift vícenásobnou dědičnost tříd nepodporuje [10] a u struktur a výčtových typů dědičnost nepodporuje vůbec. Místo dědičnosti je vhodné využít protokoly. Třída může splňovat protokolů více a poskytnout tak jejich vlastní implementaci bez nutnosti přepisování již existující implementace.

Pomocí protokolů lze také odstínit funkcionalitu třídy, či jiného typu. Například lze v metodě jako parametr předat typ stanovený protokolem, který odstiňuje metody a proměnné skutečného typu, který tento protokol splňuje.

Samotné protokoly mohou dědit od více protokolů. Také lze přímo uvnitř protokolu definovat základní chování některého z předpisů (například základní chování požadované metody). Základní chování takovéhoto předpisu je využito v případě, že daný typ splňující protokol tento požadavek explicitně nedefinuje, jelikož ho skrze existenci základního chování definovat nemusí.

Protokol byl v aplikaci mimo jiné použit pro vymezení základních požadavků na úroveň hry. Aby bylo možné s jednotlivými úrovněmi pracovat v rámci hry jednotně, každá z nich splňuje následně vymezený protokol `GameLevel`.

```
protocol BaseWinningRequirements {

    var isSuccess: Bool { get }
}

protocol GameLevel {

    associatedtype WinRequirements: BaseWinningRequirements

    var gameRequirements: WinningRequirements { get }
}
```

Protokol `BaseWinningRequirements` stanovuje povinnost implementovat proměnnou s názvem `isSuccess` typu `Bool`. Protokol `GameLevel` vyžaduje ve své implementaci existenci proměnné typu `WinningRequirements`. Tento typ není v rámci samotného protokolu nijak vymezen a jeho skutečnou podobu definují jednotlivé třídy a struktury. Je však zavázaný splňovat požadavky definované protokolem `BaseWinningRequirements`. Pro ukázku lze splnit požadavky protokolu `GameLevel` následujícím způsobem.

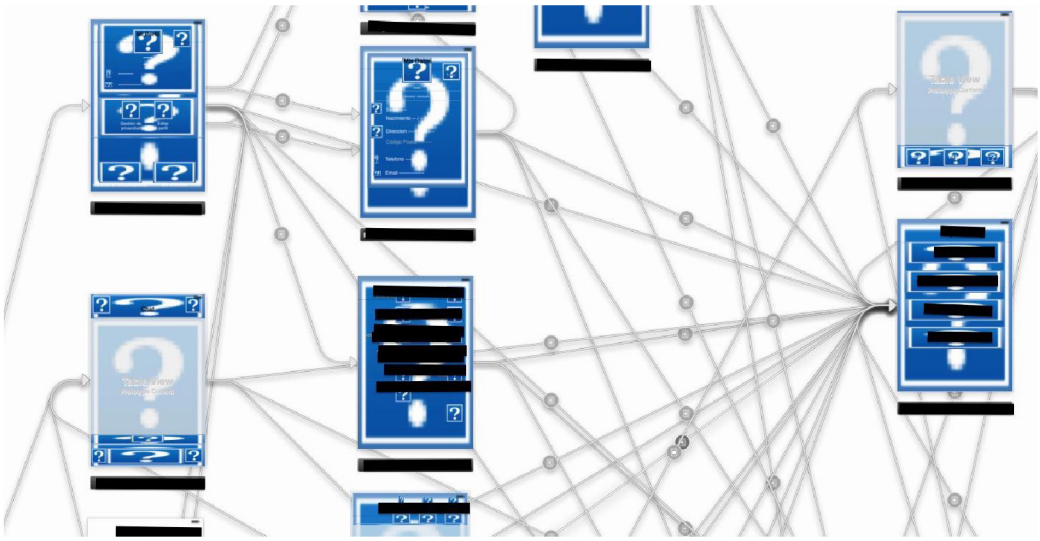
```
class RotationGameVC: UIViewController, GameLevel {

   struct RotationGameReq: BaseWinningRequirements {
       var didRoll: Bool = false
       var didPitch: Bool = false
       var didYawn: Bool = false

       var isSuccess: Bool {
           return didRoll && didPitch && didYawn
       }
   }

   var gameRequirements = RotationGameReq()
}
```

Takto vytvořená třída splňuje všechny požadavky. Obsahuje totiž proměnnou s názvem `gameRequirements`, která je typu `RotationGameReq`. Ten obsahuje proměnnou `isSuccess` a splňuje tak protokol `BaseWinningRequirement`. Takto definované protokoly dávají jednotlivým třídám velkou volnost v implementaci jejich funkcionality, a zároveň zaručují existenci protokolem vymezených proměnných a metod.

Využívaní protokolů není v rozporu s OOP a v rámci aplikace jsou dle vhodnosti využívány oba přístupy.

7.2 Problém s editorem Interface Builder a jeho možné řešení

Při vývoji na iOS se každý setká se tzv. „storyboardy“. Storyboardy slouží k vytvoření grafické podoby výsledné aplikace. Lze je vytvářet v rámci editoru *Interface Builder*. Společnost Apple tento editor popisuje jako způsob, jakým navrhnout celé grafické uživatelské rozhraní bez nutnosti psaní kódu [8]. Stačí tedy vytvořit několik oken[1] obrazovek aplikace, do oken vložit poskytnuté grafické prvky, mezi jednotlivými obrazovkami jednoduše nastavit závislosti a vše by mělo být téměř hotové. Realita ovšem při nevhodném návrhu může vypadat i tak, jak lze vidět na obrázku 7.1.



Obrázek 7.1: Ukázka, jak může vypadat storyboard v případě, že je v něm příliš mnoho prvků (Zdroj obrázku: Ted Bendixson, medium.com). Na obrázku lze vidět několik obrazovek. Každá obrazovka je spjata s třídou `UIViewController`. V této třídě je pak pomocí kódu definováno chování jednotlivých prvků, které obsahuje. Mezi jednotlivými obrazovkami jsou vytvořeny vazby, které jsou označeny univerzálními textovými identifikátory. Pro rozpoznání dané vazby v rámci kódu je nutno tento identifikátor správně přepsat. Pokud je v rámci jednoho storyboardu hodně prvků, trvá načtení takového storyboardu i několik sekund. S rostoucím počtem prvků roste také náchylnost k chybám při načítání. Chybu při načtení prvku pak označuje modrý otazník. Jakákoliv grafická úprava v takovém storyboardu se tak stává velice nepříjemnou a zdlouhavou záležitostí. Jednotlivé prvky v rámci storyboardu jsou definovány pomocí XML, které je generováno na základě grafických úprav. To způsobuje problém v případě, že ve stejném storyboardu provedou změnu dva lidé a tuto změnu oba nahrají do distribuovaného systému pro správu verzí, tedy například na GIT. Taková společná úprava může vést ke konfliktům, které lze upravit pouze pomocí úpravy zdrojového kódu XML, jelikož je storyboard v tomto stavu nemožné zobrazit v jeho grafické podobě.

Problémům spojeným se storyboardy lze například nevyužitím editoru *Interface Builder*. Jednotlivé grafické prvky lze generovat i přímo ve zdrojovém kódu. Takto vytvořené grafické uživatelské rozhraní však vede k nárůstu množství kódu, který je tím pádem méně čitelný

[1] Okno je spjato s třídou UIViewController nebo UITableViewControlle.r

a hůře se udržuje. Jiný způsob řešení je jednotlivé obrazovky rozdělit do více storyboardů. Dále upravené grafické prvky vytvořit ve vlastních souborech typu *xib* a nevytvářet je přímo v oknu ve storyboardu.

Po návrhu hry bylo zřejmé, že bude obsahovat více úrovní. Každá úroveň vyžaduje vlastní okno spjaté s třídou `UIViewController`. Zamýšlel jsem, že úrovní bude více než deset, což by za použití pouze jednoho storyboardu eventuálně vedlo k již zmíněným problémům. Pro jejich zamezení jsem v aplikaci pro každé okno vytvořil vlastní storyboard. Na storyboard se pak lze v rámci kódu odkazovat přes jeho identifikátor. Samotnému oknu je pak v rámci storyboardu přiřazen taktéž textový identifikátor. Vazby mezi jednotlivými okny v tomto případě nejsou vytvořeny v rámci storyboardu – jsou řešeny následně v rámci kódu.

S tímto řešením je, podobně jako při identifikaci vazeb, spjatý problém opisování textového identifikátoru, který je definován ve storyboardu, ale volán z kódu. Xcode pro takto vytvořené identifikátory nenabízí nápovědu a tak často dochází k typografickým chybám. Tento problém lze řešit pomocí knihovny Swiftgen. Ta řeší problém Xcode, kde se mimo vazby mezi okny a jejich identifikátory textový identifikátor používá například i pro identifikaci obrázků ve zdrojích. Řešení pak spočívá ve spuštění skriptu, který je vložen do překladové fáze projektu, a ten následně pomocí knihovny Swiftgen z textových identifikátorů vytvoří proměnné, na které se pak lze v rámci projektu odkazovat. Ukázku lze vidět v následujícím kódu.

```
// Inicializace bez knihovny Swiftgen
let storyboard = UIStoryboard(name: "Storyboard", bundle: nil)
let vc = storyboard
        .instantiateViewController(withIdentifier: "vcId")

let image = UIImage(named:"picture")

// Inicializace s knihovnou Swiftgen
let vc = StoryboardScene.Storyboard.vcId.instantiate()

let image = Asset.picture.image
```

Bez využití Swiftgen vidíme nutnost použití tří textových identifikátorů, které musíme naleznout v editoru *Interface Builder* a identifikátor pro obrázek je roven názvu souboru ve složce *Assets*. Proměnná `image` je při této incializaci rovna hodnotě *optional*. Ta značí možnou existenci obrázku. V případě, že nebyl obrázek s opsaným identifikátorem nalezen, má hodnotu `nil`.

Za použití knihovny Swiftgen není potřeba textové identifikátory vyhledávat a přepisovat. Stačí se odkázat do vygenerovaného souboru a s pomocí nápovědy ze strany Xcode danou proměnnou incializovat. V případě vytvoření obrázku pomocí Swiftgen nemá hodnotu *optional*, jelikož je jisté, že obrázek existuje.

7.3 Typ úrovně

Pro samotnou hru bylo potřeba vytvořit výčet jednotlivých úrovní. Každá úroveň má svůj identifikátor, svůj `UIViewController`, svou pozici v rámci hlavní herní plochy a sousední úrovně, které svým vyřešením zpřístupňuje. Nabízela se možnost vytvoření statického pole struktur, které by se inicializovalo při spuštění aplikace a v kódu by se na něj dalo libovolně odkazovat. Jednotlivé struktury by pak obsahovaly potřebné údaje pro danou úroveň. V rámci jazyka Swift ale existuje elegantnější řešení s využitím tzv. *computed properties* a typu *enumeration*.

7.3.1 Computed properties v jazyce Swift

Kromě tradičních *properties*, které jsou v jazyce Swift nazývány jako *stored properties* a nesou přiřazenou hodnotu, existují v jazyce Swift také *computed properties*. Ty ve skutečnosti neukládají hodnotu a mohou být využity i v typu *enumeration*. Místo uložené hodnoty poskytují tzv. *getter* a volitelný *setter* pro nepřímé získání a nastavení hodnot jiných proměnných. Pro ukázku je uveden následující kód přejatý z dokumentace k jazyku Swift [10].

```
struct Rect {
    var origin = Point()
    var size = Size()
    var center: Point {
        get {
            let centerX = origin.x + (size.width / 2)
            let centerY = origin.y + (size.height / 2)
            return Point(x: centerX, y: centerY)
        }
        set(newCenter) {
            origin.x = newCenter.x - (size.width / 2)
            origin.y = newCenter.y - (size.height / 2)
        }
    }
}
```

7.3.2 Typ enumeration v jazyce Swift

Na rozdíl od jazyka C, kde typ *enumeration* existuje také, poskytuje tento typ v jazyce Swift mnohem více možností. V jazyce Swift můžou jednotlivé případy (volný překlad z angl. *case*) nést kromě hodnoty *int*, jak je tomu v jazyce C, i hodnoty typu *string*, *character* a *floating-point* – s podmínkou, že každý takový případ má explicitně přiřazenou hodnotu daného typu.

Alternativně může mít každý případ přiřazenou hodnotu libovolného typu, která sice nereprezentuje jeho skutečnou hodnotu, ale je s ním svázaná.

Typ *enumeration* navíc lze využít podobně jako strukturu nebo třídu. Může totiž obsahovat *computed properties*, metody, definovat vlastní inicializátory a může být rozšířen

pomocí protokolů [10]. Přes jednotlivé případy lze také iterovat, stačí daný typ *enumeration* rozšířit o protokol *CaseIterable*. Tím je zpřístupněna proměnná `allCases`, se kterou lze pracovat jako s polem.

7.3.3 Využití typu enumeration a computed properties pro definici úrovní

Typ *enumeration*, který může obsahovat i *computed properties*, může být využit pro definici úrovní. Těm lze libovolně přidávat a měnit vlastnosti v jednom přehledném souboru, který řídí chování celé hry. Pro ukázku je uvedena část souboru definující vlastnosti jednotlivých úrovní hry. Jednotlivé případy nesou hodnotu typu *Int*, ta se ukázala jako vhodná a slouží pro indexaci do pole všech úrovní, které je dostupné přes proměnnou `allCases`.

```
enum GameLevelType: Int, CaseIterable {
    case eyesClosed = 0
    case fruitDetection

    var viewController: UIViewController {
        switch self {
        case .eyesClosed:
            return StoryboardScene.EyesClosed.eyesClosedVC
                    .instantiate()
        case .fruitDetection:
            return StoryboardScene.Object.objectVC
                    .instantiate()
        }
    }

    var normalizedPosition: (x: CGFloat, y: CGFloat) {
        switch self {
        case .eyesClosed:
            return(x: 0.5, y: 0.45)
        case .fruitDetection:
            return(x: 0.5, y: 0.55)
        }
    }
}
```

Kapitola 8

Vyhodnocení a testování výsledné hry

V této kapitole je představena konečná podoba hry, popsán způsob jejího průběžného testování a proces jejího zveřejnění na App Store.

8.1 Výsledná podoba hry

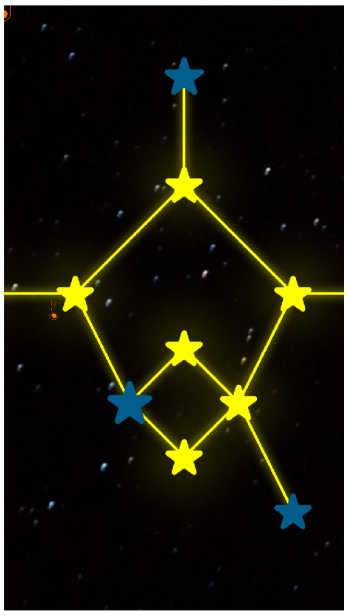
Konečná hra je složena z úrovní, které byly popsány v kapitole 6. Tyto úrovně byly spojeny do jednotného celku a společně tak tvoří výslednou hru. V této sekci je popsána navigace mezi úrovněmi, náplň již v návrhu zmiňované „minihry“ a možnost odemykání nápověd.

Jako inspiraci pro design hrací plochy, která tvoří hlavní obrazovku hry a jsou v ní postupně zobrazovány úrovně, jsem zvolil noční oblohu. Tu jsem si vybral z důvodu minimalisticky tmavě laděných grafických zpracování úrovní. Jednotlivé úrovně jsou znázorněny hvězdami. V případě, že úroveň není vyřešena, má její hvězda modrou barvu. Po úspěšném vyřešení úrovně se hvězda „rozsvítí“ a v animaci vyšle paprsky, které zpřístupní minimálně dvě další úrovně. Ukázku hrací plochy lze vidět na obrázku 8.1. V rámci hrací plochy je implementováno několik animací, díky kterým hra působí dynamičtěji.

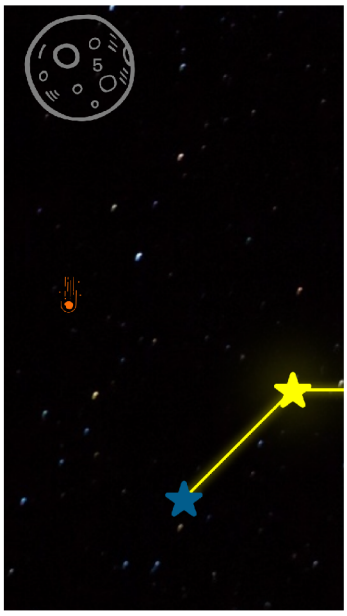
Princip „minihry“ je jednoduchý. Pro získání bodů je potřeba pomocí dotyku „sbírat“ padající obrázky meteorů oranžové barvy. Každý takto chycený meteor pak přidá jeden bod. Celkový počet bodů lze vidět uvnitř měsíce nahoře vlevo, jak lze vidět na obrázku 8.2. Tento údaj je částečně skryt úmyslně, nechtěl jsem vytvářet dojem velké podstaty těchto bodů v rámci celkové hry. Pomocí těchto bodů lze odemknout textové nápovědy úrovní. Pro odemknutí nápovědy je potřeba 50 bodů, které jde na hlavní obrazovce nasbírat zhruba za 2 až 3 minuty. Ukázku nápovědy lze vidět na obrázku 8.3.

8.2 Testování

Před vytvořením výsledné podoby hry jsem postupně implementoval prototypy jednotlivých úrovní a testoval je na uživatelích. Díky tomuto testování jsem odhalil nedostatky spojené jak s uživatelským rozhraním, tak s technologickým řešením, nebo samotným námětem úrovně. Zároveň mi tento způsob průběžného testování již v počátcích vývoje poskytl reakce uživatelů, na základě kterých jsem upravoval vývoj dalších úrovní a návrh finální podoby hry. Jednotlivé úrovně mi průběžně testovalo 12 uživatelů.



Obrázek 8.1: Ukázka hlavní obrazovky hry. Žluté hvězdy představují úrovně vyřešené, modré nevyřešené. Vyřešením úrovně se hvězda rozsvítí a zpřístupní úrovně nové. Každá úroveň sousedí minimálně s dvěma úrovněmi, které svým vyřešením zpřístupňuje.



Obrázek 8.2: Náplní zakomponované „minihry“ je „chytání“ padajících oranžových meteorů. Za každý meteor je hráči přičten jeden bod. Celkový stav bodů lze vidět šedým číslem obsaženým uvnitř měsíce vlevo nahoře.

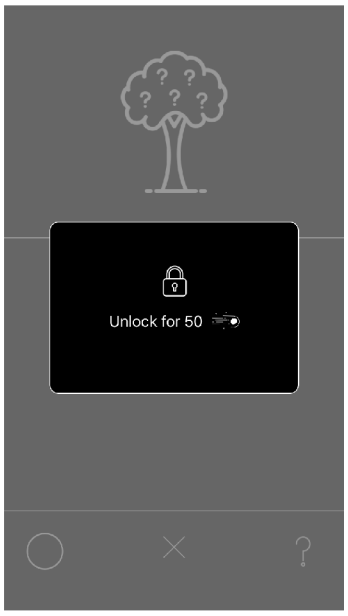
Pro příklad uvedu dvě úrovně, které byly na základě uživatelského testování graficky upraveny. Grafické změny v rámci těchto úrovní se promítly do návrhu úrovní nových. Na obrázcích 8.4 a 8.5 můžeme vidět jejich původní podobu.

Na obrázku 8.4 lze také vidět, že původní barvou, která značila úspěšné vyřešení úrovně, byla barva světle zelená. Většina uživatelů na ni ale reagovala negativně. Barva značící úspěch byla následně pozměněna na světle žlutou, která byla vnímána lépe.

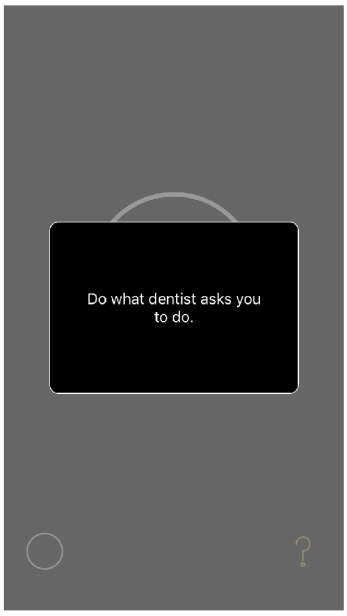
Na základě výsledků průběžného testování jsem se některé úrovně rozhodl i odstranit. Například úroveň, kde uživatel navigoval kurzor pomocí hlasu, nepřinášela radost z vyřešení. Uživatel totiž po odhalení principu řešení musel zdlouhavým způsobem navigovat kurzor po správné trajektorii. Tuto úroveň jsem tedy ze hry odstranil a na základě této zkušenosti jsem se při návrhu nových úrovní snažil, aby uživatel po odhalení principu řešení mohl úroveň vyřešit hned, nebo co nejméně nudným způsobem.

Při testování jsem také upravil úrovně, které místo grafického znázornění úrovně používaly hádanku v textovém formátu v angličtině. To vytvářelo omezení pro neanglicky mluvící uživatele a narušovalo původně čistě grafický koncept hry. Rozhodl jsem se tedy text z úrovní odstranit a upravit je tak, aby pro jejich splnění stačila pouze jejich grafická podoba.

Během testování jednotlivých úrovní jsem také sledoval, jak náročné jsou pro uživatele na vyřešení. Bohužel nebylo neobvyklé, že úroveň, na které se jeden uživatel trápil, druhý vyřešil téměř hned a naopak. S postupem času se mi ale po nabytých zkušenostech podařilo vybrat několik úrovní jednoduchých a několik obtížných. Zbylé úrovně spadají do kategorie obtížností neurčitých, jelikož se jejich náročnost skrze odlišné reakce uživatelů nedala ob-



(a) Ukázka nezpřístupněné nápovědy v úrovni vyžadující nakreslení jablka.



(b) Ukázka zpřístupněné nápovědy v úrovni detekující otevřená ústa.

Obrázek 8.3: Za body získané v „minihře“ lze v rámci úrovní odemykat nápovědy. Nápovědu pro úroveň lze zakoupit za 50 herních bodů.

jektivně odhadnout. V rámci hry pak nejprve prezentuji úrovně jednoduché a až ke konci úrovně složité.

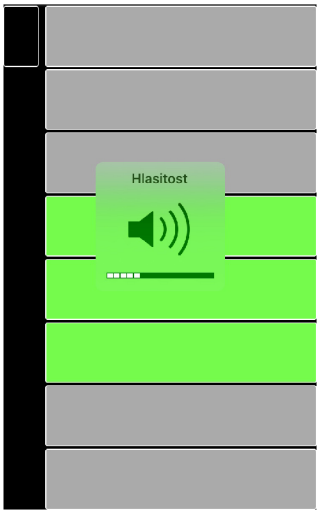
- `jednoduché`: úroveň detekující ovoce, úroveň porovnávající barvy, úroveň využívající vstupní hlasitosti, úroveň detekující nakreslené jablko
- `složité`: úroveň detekující úsměv a otevřená ústa, úroveň detekující nastavenou hlasitost zařízení, úroveň využívající zpracování světlosti vstupního obrazu

Po vytvoření výsledné hry jsem ji nejprve osobně testoval na šesti uživatelích. Předal jsem jim telefon s připravenou hrou a během testování jsem si poznamenal několik údajů.

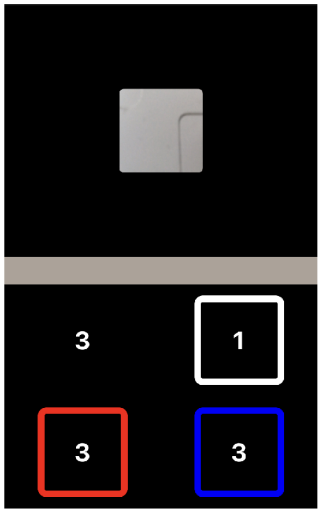
Sledoval jsem, zda-li princip hry uživatelé pochopí i bez slovního úvodu, či přečtení popisu, který bude dostupný na App Store. Většina uživatelů princip hry pochopila, ale v další verzi by hra mohla obsahovat i krátký úvod, který by ji lépe představil. Dále jsem sledoval jakým způsobem uživatelé hru prochází a kolik úrovní zvládnou vyřešit bez nápovědy. Tím jsem ověřil, zda-li je ovládání hry uživatelům zřejmé a jestli jsem úrovně správně seřadil dle obtížnosti a uživatel se nedostane do „nevyřešitelného stavu“ v samotném počátku hry.

Také jsem sledoval, jestli uživatelé pochopí princip odemykání nápověd. Ten pochopili všichni bez problému a s textovou nápovědou zvládli vyřešit i úrovně, které bez ní vyřešit nedokázali. Nápovědy jsou tedy velmi popisné, což není v rozporu s doporučením pro hru typu puzzle, jelikož jak bylo zmíněno v kapitole 5, i poskytnutí odpovědi může vést k dobrému dojmu ze hry. Dále jsem si poznamenal, kolik nápověd jednotliví uživatelé potřebovali pro vyřešení všech úrovní hry. V průměru potřebovali šest nápověd a větší polovinu úrovní tak zvládli vyřešit jen na základě jejich grafické podoby.

Na závěr jsem uživatele poprosil o upřímné ohodnocení „zábavnosti“ této hry. V průměru hra získala hodnocení 7.7/10. Jelikož uživatelé nebyli fanoušci her typu puzzle, považuji toto



Obrázek 8.4: Původně byly úrovně hlasitosti rozděleny graficky do intervalů, které počtem neodpovídaly počtu úrovní hlasitosti systému. Mohlo se tedy stát, že stisknutí bočního tlačítka pro změnu hlasitosti se nijak neprojevilo do vizuální podoby úrovně. To bylo pro některé uživatele matoucí. Graficky znázorněný interval byl tedy upraven tak, aby odpovídal tomu systémovému.



Obrázek 8.5: Tento návrh úrovně pro detekci barvy v horním čtverci zobrazoval vstupní obraz. V tomto čtverci navíc nebyl zobrazen pouhý výřez, ale byl v něm zachycen zmenšený celý vstup kamery zarovnaný do čtverce. Uživatelé při snaze zachycení vhodné barvy museli telefon dostatečně přiblížit k barvě tak, aby požadovanou barvu obsahoval téměř celý obraz snímaný kamerou. To vedlo k tomu, že si při snímání této barvy skrze bezprostřední vzdálenost telefonem stínili a veškeré barvy byly velmi tmavé. Po úpravě je průměrná barva získávána pouze z výřezu vstupního obrazu, což bylo vhodnější. Obdélník uprostřed obrazovky zobrazoval průměrnou barvu vstupního obrazu. Jeho význam ale nebyl některým uživatelům zřejmý. Zobrazování vstupní barvy bylo přesunuto do čtverce vedle snímaného výřezu vstupního obrazu, kde dávalo větší smysl.

hodnocení za úspěch. Uživatelé ke svému hodnocení dodávali, že je hra velice zaujala svou netradiční náplní a principem řešení jednotlivých úrovní. Požádal jsem je tedy, aby mi označili úroveň, která se jim zdála nejzajímavější. Uživatelé zaujaly převážně úrovně, které využívaly zpracování vstupního obrazu. Souhrn získaných hodnot během testování lze vidět v tabulce 8.6.

Následně jsem hru otestoval pomocí služby *TestFlight*. Tuto službu poskytuje Apple jako možnost, jak před samotným vydáním aplikace provést uživatelské testování a získat tak cenné připomínky. Stejně jako pro samotné vydání aplikace je pro její umístění na *TestFlight* potřeba vývojářský účet, který lze zakoupit za poplatek 99$/rok. Dále je potřeba, aby aplikace prošla schvalovacím procesem. Tento proces je podobný jako proces schvalování

	Pochopení principu hry bez poskutnytí popisu	Počet splněných úrovní bez nápovědy	První nevyřešená úroveň	Pochopení principu odemykání nápověd	Počet využitých nápověd	Zábavnost od 1 do 10	Nejzajímavější úroveň
Uživatel 1	ano	11	Rotace zařízením	ano	3	8	Detekce barev
Uživatel 2	ne	5	Vypnutí připojení k internetu	ano	8	6	Detekce ovoce
Uživatel 3	ano	12	Detekce úsměvu	ano	4	8	Detekce otevřených úst
Uživatel 4	ano	10	Gesto ruky	ano	5	9	Detekce vstupní světlosti obrazu
Uživatel 5	ano	7	Nastavení hlasitosti	ano	5	8	Detekce ovoce
Uživatel 6	ano	6	Nastavení hlasitosti	ano	9	7	Rozpoznání nakresleného jablka

Obrázek 8.6: Výsledky sledovaných údajů získaných během prvního uživatelského testování kompletní hry.

aplikace pro její vydání, toleruje ale více nedostatků. Může se tedy stát, že aplikace, která prošla schvalovacím procesem pro testování, neprojde schvalovacím procesem pro její vydání na App Store. Po umístění aplikace na *TestFlight* je možné uživatele pro její testování pozvat přes e-mail nebo pomocí odkazu [4].

Pomocí služby *TestFlight* mi hru otestovalo pět uživatelů. Na ty hra až na drobné grafické výhrady působila dobře a sklízel jsem převážně pozitivní reakce jak na způsob provedení, tak na samotnou náplň hry.

8.3 Zveřejnění hry na App Store

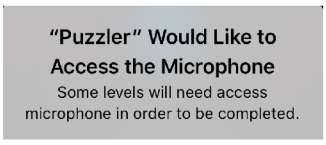
Po otestování hry a opravení několika drobných nedostatků, na které mě uživatelé upozornili, jsem nahrál hru ke schválení na App Store. Pro umístění na App Store musí každá aplikace poskytovat odkaz na webovou stránku, na které jsou uvedené informace o aplikaci a kontakt na podporu. Dále je potřeba poskytnout odkaz na dokument *Privacy Policy* vztažený k dané aplikaci. Každá aplikace také musí splňovat tzv. *App Store Review Guidelines*. Jedná se o dokument od firmy Apple uvádějící směrnice, které musí každá aplikace dodržovat. Tyto směrnice uživatelům zaručují, že aplikace na App Store splňují určitý standard, a vývojáře vedou k tomu, aby tento standard dodrželi a zvýšili šanci na úspěch své aplikace. Směrnice se týkají bezpečnosti, výkonu, designu a obchodních i právních nároků na aplikaci.

Jelikož moje aplikace používá senzory zařízení netradičním způsobem, měl jsem z některých bodů těchto směrnic obavy. Například bod 1.4.5 zmiňuje, že by aplikace neměla nutit uživatele používat zařízení způsobem, který vede k riziku jeho poškození. Obával jsem se, že úroveň, pro jejíž splnění je potřeba zařízením rotovat podle os, by mohla tuto směrnici porušovat. Další bod, který byl částečně v rozporu s jednou úrovní hry, byl bod 2.5.9, který pojednává o tom, že by v rámci aplikace nemělo být pozměněno chování standardních tlačítek dostupných na telefonu – tlačítko pro přepnutí do tichého režimu nebo tlačítka pro nastavení hlasitosti. Tyto tlačítka jsem sice využíval i jiným způsobem, ale jejich původní funkcionalitu jsem ponechal. Největší starost mi dělal bod 2.5.14, který vyžaduje, aby bylo uživateli vždy vizuálně zřejmé, kdy aplikace používá kameru nebo mikrofon. V případě mikrofonu je jeho využití v úrovních naznačeno pomocí grafického ekvalizéru. V případě

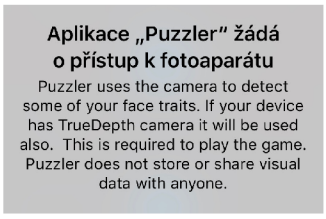
několika úrovní vyžadující vstup z kamery je její využití pouze naznačeno a uživatel tento fakt musí sám zjistit a využít ho pro vyřešení úrovně.

Jelikož mi aplikace prošla schvalovacím procesem pro testování, rozhodl jsem se ji i přes obavy umístit do schvalovacího procesu pro vydání. Vytvořil jsem pro ni jednoduchou webovou stránku a soubor obsahující *Privacy Policy*, který byl vytvořen pomocí dostupných generátorů pro mobilní aplikace. Díky tomu, že jsem žádné data o uživatelích neshromažďoval a s nikým je nesdílel, jsem neměl z právního hlediska velké obavy. Přesto, že je celý schvalovací proces velmi skryt, snažil jsem se co nejvíce zvýšit šanci na úspěšné schválení. Pro schvalovací proces jsem tedy do hry zakomponoval skrytou možnost generování bodů, za které lze odemknout nápověda, abych ulehčil případné testování aplikace.

Po několika dnech mi došel výsledek ze schvalovacího procesu. Aplikace mi byla zamítnuta. K mému překvapení nebyly důvodem jejího zamítnutí mnou předpokládané body směrnic. Dle schvalovacího procesu byl porušen bod 5.1.1, který pojednává o zpracování a shromažďování uživatelských dat. Konkrétně bylo potřeba upravit znění zprávy, která je uživateli zobrazena při žádosti o přístup ke kameře a mikrofonu. Dále bylo potřeba uvést, jakým způsobem pracuji s daty získanými *TrueDepth* kamerou a jestli moje aplikace dokáže fungovat i na zařízeních, které ji nemají. Skutečnost o zpracování dat získaných pomocí *TrueDepth* kamery bylo potřeba zmínit i v dokumentu *Privacy Policy*. Požadované úkony jsem tedy provedl, změnu formulování dotazu o přístup ke kameře lze vidět na obrázku 8.7. Následně jsem aplikaci opět zaslal ke schválení, tentokrát již schvalovacím procesem prošla a je nyní volně dostupná ke stažení.



(a) Původní znění zprávy, která byla zobrazena uživateli se žádostí o přístupu ke kameře. Ve schvalovacím procesu byla označena za nevyhovující a nedostatečně popisující záměr jejího využití. Tento obrázek byl součástí zprávy obsahující důvod zamítnutí aplikace ve schvalovacím procesu.



(b) Upravené znění zprávy, které bylo ve schvalovacím procesu následně vyhodnoceno jako dostačující.

Obrázek 8.7: Ukázka změny formulovaní žádosti o přístup ke kameře. Podobným způsobem byla upravena i žádost o přístup k mikrofonu.

Kapitola 9

Závěr

Cílem této diplomové práce bylo vytvořit hru pro chytrý telefon s operačním systémem iOS. Tato hra měla pro svou funkcionalitu využívat detekci lidské tváře a jejich dílčích rysů v obraze. Dále tato hra mohla využívat další informace, které je zařízení svými senzory schopno detekovat.

Před samotnou implementací jsem nastudoval problematiku vývoje pro iOS a algoritmy vhodné pro detekci lidské tváře a jejích dílčích rysů v obraze. Jednotlivé implementace algoritmů jsem otestoval na mobilním zařízení a pro průběh hry zvolil přístup, který byl schopný významné znaky obličeje detekovat v reálném čase. Tento přístup pro detekci využívá kombinaci nativního frameworku AV Capture a knihovny Dlib a po provedené optimalizaci je schopný dílčí rysy tváře na iPhone 7 detekovat v rychlosti 30 FPS. Následně jsem navrhl hru pro chytrý telefon, která této detekce využívá.

Hra se skládá z několika úrovní, kde každá úroveň obsahuje jednu hádanku. Jednotlivé hádanky jsem navrhl tak, aby je šlo znázornit čistě graficky bez použití textu. Úrovně lze vyřešit netradičním způsobem za využití nejnovějších možností chytrých telefonů. V rámci hry je mimo detekce rysů obličeje využito i zpracování barevnosti vstupního obrazu, zpracování zvuku, informace o pohybu zařízení, detekce objektů v obraze a další. Při tvorbě hry jsem díky rozmanitosti jednotlivých úrovní vyzkoušel a využil širokou škálu funkcionality, které moderní telefony nabízí. Jednotlivé úrovně jsem postupně testoval a na základě reakce uživatelů upravoval.

Výsledná hra s názvem Puzzler Square obsahuje 17 úrovní. Hra je navržena a vytvořena tak, aby splňovala zásady pro hru typu puzzle. Hru se mi po jejím závěrečném testování přes službu TestFlight podařilo zveřejnit na App Store, kde je volně dostupná ke stažení. Hru budu dále rozvíjet. Pomocí kombinace implementovaných technik plánuji vytvářet nové úrovně a vylepšovat její grafickou podobu. Také budu reagovat na připomínky nových uživatelů, kteří hru vyzkouší.

Literatura

[1] Agarwal, R.: *Smilefie: how you can auto-capture selfies by detecting a smile.* Srpen 2018, [Online; navštíveno 13.01.2019].
URL https://medium.freecodecamp.org/smilfie-auto-capture-selfies-by-detecting-a-smile-using-opencv-and-python-8c5cfb6ec197

[2] Andersson, M.; Arvola, M.; Hedar, S.: *Sketch classification with neural networks.* Červen 2018, [Online; navštíveno 02.01.2019].
URL https://uu.diva-portal.org/smash/get/diva2:1218490/FULLTEXT01.pdf

[3] Anzalone, L.: *Training alternative Dlib Shape Predictor models using Python.* Říjen 2018, [Online; navštíveno 10.01.2019].
URL https://medium.com/datadriveninvestor/training-alternative-dlib-shape-predictor-models-using-python-d1d8f8bd9f5c

[4] Apple, I.: *Apple Developer Documentation.* [Online; navštíveno 20.11.2018].
URL https://developer.apple.com/documentation/

[5] Apple, I.: *ARFaceTrackingConfiguration.* [Online; navštíveno 2.01.2019].
URL https://developer.apple.com/documentation/arkit/arfacetrackingconfiguration

[6] Apple, I.: *Get Started with Display P3.* [Online; navštíveno 12.01.2019].
URL https://developer.apple.com/videos/play/wwdc2017/821/

[7] Apple, I.: *Importing Objective-C into Swift.* [Online; navštíveno 13.01.2019].
URL https://developer.apple.com/documentation/swift/imported_c_and_objective-c_apis/importing_objective-c_into_swift

[8] Apple, I.: *Interface Builder Built-In.* [Online; navštíveno 21.03.2019].
URL https://developer.apple.com/xcode/interface-builder/

[9] Apple, I.: *iOS Device Compatibility Referencen.* [Online; navštíveno 18.03.2019].
URL https://developer.apple.com/library/archive/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/Displays/Displays.html/

[10] Apple, I.: *The Swift Programming Language.* [Online; navštíveno 20.03.2019].
URL https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html

[11] Apple, I.: *What's New in ARKit 2.* [Online; navštíveno 2.01.2019].
URL https://developer.apple.com/videos/play/wwdc2018/602

[12] Apple, I.: *Protocol-Oriented Programming in Swift.* 2015, [Online; navštíveno 15.03.2019].
URL https://developer.apple.com/videos/play/wwdc2015/408/

[13] Apple, I.: *An On-device Deep Neural Network for Face Detection.* Listopad 2017, [Online; navštíveno 15.01.2019].
URL https://machinelearning.apple.com/2017/11/16/face-detection.html

[14] Apple, I.: *Vision Framework: Building on Core ML.* 2017, [Online; navštíveno 16.01.2019].
URL https://developer.apple.com/videos/play/wwdc2017/506/

[15] Bermudez, L.: *Overview of Neural Networks.* Listopad 2017, [Online; navštíveno 11.01.2019].
URL https://medium.com/machinevision/overview-of-neural-networks-b86ce02ea3d1

[16] Brownlee, J.: *What is the Difference Between a Batch and an Epoch in a Neural Network?* Červenec 2018, [Online; navštíveno 29.12.2018].
URL https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

[17] Dalal, N.; Triggs, B.: *Histograms of Oriented Gradients for Human Detection. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, ročník 1, Červen 2005: s. 886–893, ISSN 1063-6919.

[18] Dlib, t.: *About Dlib.* [Online; navštíveno 21.02.2019].
URL http://dlib.net

[19] EHT: *Astronomers Capture First Image of a Black Hole.* [Online; navštíveno 20.02.2019].
URL https://eventhorizontelescope.org

[20] Geitgey, A.: *Machine Learning is Fun!* Červen 2016, [Online; navštíveno 28.12.2018].
URL https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

[21] Guo, K.; WoMa, J.; Xu, E.: *Quick, Draw! Doodle Recognition.* [Online; navštíveno 02.01.2019].
URL http://cs229.stanford.edu/proj2018/report/98.pdf

[22] Gupta, V.: *An On-device Deep Neural Network for Face Detection.* Říjen 2018, [Online; navštíveno 20.02.2019].
URL https://www.learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/

[23] Kazemi, V.; Sullivan, J.: *One millisecond face alignment with an ensemble of regression trees. 2014 IEEE Conference on Computer Vision and Pattern Recognition*, Červen 2014: s. 1867–1874, ISSN 1063-6919.

[24] Keizer, G.: *Which iPhones, iPads support Apple's iOS 12?* Červen 2018, [Online; navštíveno 25.12.2018].

URL https://www.computerworld.com/article/3282811/apple-ios/which-iphones-ipads-support-apples-ios-12.html

[25] Keras, I.: *Keras: The Python Deep Learning library.* [Online; navštíveno 20.01.2019]. URL https://keras.io

[26] Lindbloomk, B. J.: *Useful Color Equations.* [Online; navštíveno 15.01.2019]. URL http://www.brucelindbloom.com/index.html?WorkingSpaceInfo.html

[27] Mallick, S.: *Why does OpenCV use BGR color format?* [Online; navštíveno 10.01.2019]. URL https://www.learnopencv.com/why-does-opencv-use-bgr-color-format/

[28] Mallick, S.: *Histogram of Oriented Gradients.* Prosinec 2016, [Online; navštíveno 3.1.2019]. URL https://www.learnopencv.com/histogram-of-oriented-gradients/

[29] Maris, A.: *Implementation and Study of Cascaded-Regression Methods for Facial Feature Points Detection.* Červen 2015, [Online; navštíveno 10.1.2019]. URL https://www.doc.ic.ac.uk/teaching/distinguished-projects/2015/a.maris.pdf

[30] Microsoft, C.: *What is Azure Custom Vision?* [Online; navštíveno 10.04.2019]. URL https://docs.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/home

[31] Miller, C.: *On the tenth anniversary of the iPhone, here's how iOS has transformed over the years.* Červen 2017, [Online; navštíveno 25.12.2018]. URL https://9to5mac.com/2017/06/29/ios-progression-over-the-years-gif/

[32] Mokrzycki, W.; Tatol, M.: *Colour difference Delta E - A survey. Machine Graphics and Vision*, ročník 20, Duben 2011: s. 383–411, ISSN 1230-0535.

[33] Nguyen-Meidine, L. T.; Granger, E.; Madhu Kiran, L.-A. B.-M.: *A Comparison of CNN-based Face and Head Detectors for Real-Time Video Surveillance Applications. 2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*, Listopad 2018: s. 1–7, ISSN 2154-512X.

[34] Novak, J.: *What every coder should know about gamma.* Září 2016, [Online; navštíveno 15.01.2019]. URL https://blog.johnnovak.net/2016/09/21/what-every-coder-should-know-about-gamma/

[35] OpenCV, t.: *About OpenCV.* [Online; navštíveno 10.11.2018]. URL https://opencv.org/about.html/

[36] OpenCV, t.: *cv::Mat Class Reference.* [Online; navštíveno 10.01.2019]. URL https://docs.opencv.org/3.1.0/d3/d63/classcv_1_1Mat.html#a51615ebf17a64c968df0bf49b4de6a3a

[37] Pandya, M.: *Introduction to Face Detection in iOS 11 using Vision Framework.* Březen 2018, [Online; navštíveno 03.01.2019]. URL https://www.moveoapps.com/blog/face-detection-ios-11-vision-framework/

[38] Patel, S.: *SVM (Support Vector Machine) — Theory*. Květen 2017, [Online; navštíveno 10.01.2019].
URL https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

[39] Popko, A.: *Swift vs Objective C: iOS' Programming Languages Compared*. Únor 2018, [Online; navštíveno 26.12.2018].
URL https://www.netguru.com/blog/swift-vs-objective-c-ios-programming-languages-compared

[40] Poynton, C.: *Frequently Asked Questions about Color*. [Online; navštíveno 08.02.2019].
URL http://poynton.ca/PDFs/ColorFAQ.pdf

[41] Schell, J.: *The Art of Game Design - A Book of Lenses*. Morgan Kaufmann Publisher, 2008, ISBN 978-0-12-369496-6.

[42] Soukupová, T.; Čech, J.: *Real-Time Eye Blink Detection using Facial Landmarks*. Únor 2016, [Online; navštíveno 13.01.2019].
URL http://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf

[43] Srivastava, N.; Hinton, G.; Krizhevsky, A.; aj.: *Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research*, ročník 15, Červen 2014: s. 1929–1958.

[44] Stackoverflow: *Developer Survey Results*. [Online; navštíveno 26.12.2018].
URL https://insights.stackoverflow.com/survey/2017#most-loved-dreaded-and-wanted

[45] Tan, Y.: *Viral app is basically Dance Dance Revolution, but for your face*. Srpen 2017, [Online; navštíveno 12.10.2018].
URL https://mashable.com/2017/08/18/face-dance-app/

[46] Thawarel, R.: *Real-Time Face Detection and Recognition with SVM and HOG Features*. Květen 2018, [Online; navštíveno 20.10.2018].
URL hhttps://www.eeweb.com/profile/rajeevthaware/articles/real-time-face-detection-and-recognition-with-svm-and-hog-features

[47] Viola, P.; Jones, M. J.: *Robust Real-Time Face Detection. International Journal of Computer Vision*, ročník 57, č. 2, Květen 2004: s. 137–154, ISSN 1573-1405.

[48] Wang, C.-F.: *What's the Difference Between Haar-Feature Classifiers and Convolutional Neural Networks?* Srpen 2018, [Online; navštíveno 3.1.2019].
URL https://towardsdatascience.com/whats-the-difference-between-haar-feature-classifiers-and-convolutional-neural-networks-ce6828343aeb

Příloha A

Obsah CD

- `/xskota07-DP.pdf` – elektronická verze písemné zprávy
- `/poster/puzzler_poster.pdf` – plakát pro aplikaci
- `/video` – složka obsahující propagační a ukázkové video aplikace
- `/source_code` – složka obsahující zdrojový kód hry pro Xcode
- `/models/CustomVision/training_data` – složka obsahující data využitá pro vytvoření modelu pomocí služby Custom Vision
- `/models/Keras` – složka obsahující skript a data pro vytvoření modelu pomocí knihovny Keras a skript pro jeho převod do formátu *.mlmodel*