

Stredná priemyselná škola informačných technológií
Ignáca Gessaya, Medvedzie 133/1, Tvrdošín

Inteligentná Rubikova kocka

Stredná priemyselná škola informačných technológií
Ignáca Gessaya, Medvedzie 133/1, Tvrdošín

Inteligentná Rubikova kocka

Študijný odbor: 3918 M technické lýceum

Trieda: IV. C

Konzultant práce: Ing. František Benčík



ZADANIE VLASTNÉHO PROJEKTU

pre

praktickú časť odbornej zložky maturitnej skúšky

forma b)

v školskom roku 2022/2023

Názov vlastného projektu: Inteligentná Rubikova kocka

Meno riešiteľa: Branislav Mateáš

Trieda: IV.C

Študijný odbor: 3918M technické lýceum

Meno spoluriešiteľa: -

Meno konzultanta: Ing. František Benčík

Cieľ vlastného projektu: Analýza Bluetooth komunikácie dostupných inteligentných Rubikových kociek a aplikácií dodávaných s kockami. S využitím týchto znalostí bude cieľom tohto projektu vytvoriť vlastnú aplikáciu pre jej ovládanie.

Stručná anotácia:

Pôjde o aplikáciu pre operačný systém Android s týmito vlastnosťami:

- vizualizácia pohybov a označovanie ťahov
- meranie času a počet krokov riešenia
- uloženie a zobrazenie dosiahnutých výsledkov

Spôsob využitia: Možnosť budúceho využitia dekodovanej komunikácie pre budúce projekty (možnosť ovládania Rubikovou kockou napríklad).

Finančné, technologické a priestorové nároky na realizáciu projektu: 100€

Čestné vyhlásenie a poďakovanie

Vyhlasujem, že som zadanú prácu vypracoval samostatne, pod odborným vedením vedúceho práce **Ing. Františka Benčíka** a používal som len literatúru uvedenú v práci.

Súhlasím s tým, že práca bude uložená v školskej knižnici a môže byť zapožičaná záujemcom o jej preštudovanie.

Tvrdošín, 10. 2. 2023

podpis

Abstrakt

MATEÁŠ, Branislav: Inteligentná Rubikova kocka. Stredná priemyselná škola informačných technológií Ignáca Gessaya, Tvrdošín: 2023. 49 strán, 41 067 znakov.

Kľúčové slová: Flutter, Unity, Rubikova kocka, Bluetooth, Android

Obsah

Úvod	6
1 Rubikova kocka.....	7
1.1 História.....	7
1.2 Speedcubing.....	7
1.3 Nami využívaný model	8
2 Využitie nástroje a technológie	9
2.1 Flutter.....	9
2.2 Dart	10
2.3 Unity	11
2.4 Bluetooth.....	12
2.5 Android	13
2.6 IntelliJ IDEA.....	14
3 Dekódovanie Bluetooth komunikácie	16
3.1 Úvodná analýza komunikácie	16
3.2 Prieskum a dekódovanie oficiálnej aplikácie.....	17
3.3 Analýza .dll súborov	19
4 Tvorba aplikácie a 3D modelu	25
4.1 Nastavenie prostredia a vytvorenie základného projektu	25
4.2 Konverzia Bluetooth dekódera do jazyku Dart.....	26
4.3 Napojenie Rubikovej kocky a Bluetooth-u.....	27
4.4 Tvorba Rubikovej kocky v prostredí Unity	31
4.5 Prepojenie Unity modelu s aplikáciou	34
4.6 Tvorba grafickej predlohy pre aplikáciu.....	35
4.7 Mód Timer	36
4.8 Mód Solver	38
4.9 Mód Stats	40
Záver.....	44
Príloha A.....	A
Príloha B.....	B
Príloha C.....	C

Úvod

Rubikova kocka je jedným z hlavolamov, ktorý už niekoľko desaťročí fascinuje ľudí všetkých vekových kategórií. Jej riešenie vie byť nie len náročné pre náš mozog, ale aj obohacujúce – ručné zaznamenávanie časov či vykonaných ťahov však môže byť únavné. Naším cieľom pre tento projekt bude preto rozanalyzovať Bluetooth komunikáciu s našou inteligentnou Rubikovou kockou a na jej základe vytvoriť aplikáciu pre operačný systém Android spríjemňujúcu jej riešenie.

V minulosti si riešenie Rubikovej kocky vyžadovalo ručné zaznamenávanie a bolo k dispozícii len veľmi málo nástrojov, ktoré by pri tomto procese pomáhali. S rozvojom technológií však máme teraz možnosť vyvíjať aplikácie, ktoré môžu ako pomôcť pri riešení kocky, tak aj spríjemniť tento zážitok.

Finálnym produktom tohto projektu bude teda aplikácia s vizuálne príťažlivým užívateľským rozhraním na vizualizáciu a označovanie ťahov, spoločne s možnosťou merať a zaznamenávať čas, rovnako ako aj s možnosťou ukladať počet krokov, ktorý následne poskytneme k zobrazeniu užívateľovi na neskoršiu analýzu. Veríme, že naša aplikácia k inteligentnej Rubikovej kocke – alebo ako sme si ju nazvali – Cubio, bude cenným nástrojom pre každého, kto sa zaujíma o riešenie tohto kultového hlavolamu.

1 Rubikova kocka

Hlavalom v tvare kocky, ktorý v roku 1974 vynášiel Maďar Erno Rubik, je populárnym fenoménom aj v dnešnej dobe. Tento vynález sa stal obľúbeným a ikonickým symbolom pre oblasti ako matematika či problem-solving. Najznámejšou je variant $3 \times 3 \times 3$, ktorý budeme aj my využívať, populárnymi sú však aj varianty $4 \times 4 \times 4$ (Rubikova pomsta), $5 \times 5 \times 5$ a i. Rubikova kocka viedla aj k vynájdeniu ďalších podobných hlavalomov, ako sú napr. Pyraminx a Skewb.

Jedným z najfascinujúcejších aspektov Rubikovej kocky je jej jednoduchosť. V podstate je to len kocka zložená z menších kociek, ktoré sa dajú otáčať a posúvať, čím sa menia farby na jednotlivých stenách väčšej kocky. Napriek svojej jednoduchosti si však vyžaduje kombináciu logiky, rozpoznávania vzorov a zručnosti. Uvažujúc štandardnú $3 \times 3 \times 3$ kocku, táto kocka môže mať až približne 43 triliónov rôznych stavov. [1]

1.1 História

História Rubikovej kocky je tiež veľmi zaujímavá. Ako sme už spomenuli, v roku 1974 ju vynášiel Erno Rubik, maďarský sochár a profesor architektúry. Pôvodne vytvoril kocku ako učebnú pomôcku, ktorá mala pomôcť jeho študentom pochopiť trojrozmernú geometriu. Zámerom bolo ale aj umenie. Kocka mala ukazovať ostré kontrasty človeka: problémy a ich riešenie, poriadok a chaos, jednoduchosť a komplexnosť... Až neskôr si Erno začal uvedomovať potenciál svojho vynálezu. Zaujímavosťou je, že pôvodne sa kocka nevolala Rubikova, ale ako ju tento sochár pôvodne nazval - „Zázračná kocka“. Podľa odhadov je dokonca Rubikova kocka najpredávanejšou "hračkou" histórie. [2]

1.2 Speedcubing

V poslednej dobe populárnym aspektom Rubikovej kocky je disciplína zvaná speedcubing, čo ako už názov napovedá, je činnosť, ktorej cieľom je vyriešiť kocku čo najrýchlejšie. Speedcubing sa stal súťažným športom s mnohými svetovými rekordmi a súťažami, ktoré sa konajú po celom svete. Súčasný svetový rekord v riešení Rubikovej kocky drží Yusheng Du z Číny, ktorý ju v novembri 2021 vyriešil za 3,47 sekundy.

1.3 Nami využívaný model

Po diskusii s konzultantom sme sa rozhodli pre model inteligentnej Rubikovej kocky značky Giiker, a to konkrétne model **Giiker Super Cube i3S Light retro**.



Obr. 1.3: Giiker Super Cube i3S Light retro

2 Využité nástroje a technológie

Dôležitou súčasťou každého projektu je výber správnych technológií. Pre tvorbu mobilnej aplikácie môžeme vybrať z rozsiahleho spektra programovacích jazykov a frameworkov (framework – súbor knižníc, nástrojov a postupov, ktoré poskytujú štruktúru na vytváranie a údržbu softvérovej aplikácie). Tieto riešenia možno rozdeliť na dva druhy – multiplatformné a natívne.

Multiplatformné, alebo inak nazývané, cross-platform riešenia, dovoľujú vývojárom tvoriť softvér pre viacero platforiem (oba Android, tak aj iOS) s využitím jedného zdrojového kódu. Príkladmi takýchto riešení sú technológie Flutter či React Native, ale aj mnoho ďalších (Ionic a pod.). Siahnutie po takomto type technológie nám značne ušetrí čas a zdroje v porovnaní s natívnymi riešeniami. Treba však podotknúť, že je to na úkor rýchlosti, ktorej rozdiel je však v dnešnej dobe pri bežných projektoch zanedbateľný. Aj vďaka tomu siahajú po týchto technológiách každým rokom viac a viac programátorov.

Na druhej strane, natívny vývoj mobilných aplikácií znamená vytváranie samostatných aplikácií pre každú z platforiem. A to pomocou špecifických programovacích jazykov a nástrojov pre každú platformu – aplikácie vyvinuté pre systém iOS vytvárame za pomoci jazykov Swift či Objective-C, zatiaľ čo pre Android využívame jazyky ako Kotlin alebo Java. Výsledkom sú potom zväčša optimalizovanejšie aplikácie, pretože umožňujú ich tvorcom zlepšený prístup k nízkoúrovňovým nástrojom ako sú GPS, fotoaparát a iné, a rôznym ďalším, častokrát systém-špecifickým funkciám v porovnaní s hybridnými riešeniami.

2.1 Flutter

Aj napriek tomu, že sme sa rozhodli vyvíjať aplikáciu pre operačný systém Android (respektíve sme nemali možnosť vyvíjať na základe absentujúceho hardvéru pre Apple iOS), zvolili sme si práve tento framework.

Medzi hlavné dôvody patria možnosť neskôr distribuovať našu aplikáciu na ďalšie operačné systémy ako iOS, no po novom aj Windows či Linux. Nemôžeme taktiež zabudnúť ani na webové rozhranie. Ďalším z dôvodov bola podpora potrebných

technológií – primárne interakcia s Bluetooth rozhraním a 3D modelom. Neposledným dôvodom bola aj už nadobudnutá skúsenosť s touto technológiou.

Flutter je teda open-source framework vytvorený spoločnosťou Google slúžiaci na tvorbu hybridných aplikácií pre platformy Android, iOS, Linux, macOS, Windows, Google Fuchsia a web za pomoci jedného kódu. [3]

S prvou verziou zverejnenou v máji 2017, jedným z argumentov proti tejto technológii je práve jej krátkovekosť v porovnaní so svojimi konkurentmi. Medzi programátormi má však veľkú obľubu. Flutter je využívaný nie len na malé projekty, ale je taktiež využívaný korporáciami ako Google, BMW či eBay, no využívajú ho aj mnohé ďalšie. [4]

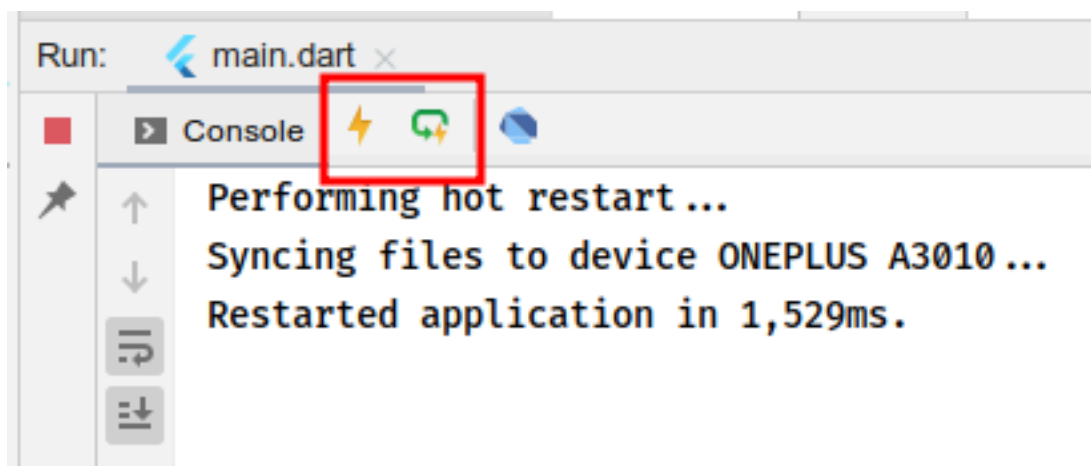
Jednou z doposiaľ nespomenutých a obrovských výhod tejto technológie taktiež patria dve predvytvorené sady widgetov pre rýchlu a uľahčenú tvorbu našej aplikácie. Sú nimi knižnice Material Design a Cupertino – obe implementujúce dizajnový štandard pre oba rozhrania systémov –Android i iOS.

Je dôležité spomenúť, že Flutter je postavený na programovacom jazyku Dart. Okrem metód tohto jazyka teda využíva napríklad aj jeho package manager – nazývaný 'pub' (možno preto využiť ľubovoľný Dart balíček bez akýkoľvek obmedzení).

2.2 Dart

Programovací jazyk Dart je dôležitou súčasťou úspechu Flutter-u ako takého. Flutter beží na virtuálnom stroji napísanom práve v jazyku Dart. Táto skutočnosť umožňuje Flutter aplikácii pri vývoji využívať napríklad Just-In-Time (JIT) kompiláciu.

Tento typ kompilácie nám prináša pri vývoji funkciu zvanú "Hot Reload", umožňujúcu rýchlo obnoviť aplikáciu po zmene kódu (trvajúcu zhruba okolo jednej sekundy). Pri použití funkcie Hot Reload môžu preto vývojári vidieť zmeny v kóde okamžite po ich uložení, čo mimoriadne uľahčuje a urýchľuje proces vývoja aplikácie. Hot Reload tiež umožňuje programátorom pridávať, meniť alebo odstraňovať widgety a robiť úpravy v kóde bez nutnosti reštartovania aplikácie. Rovnaký efekt, avšak aj s reštartovaním stavu aplikácie, možno dosiahnuť za použitia funkcie "Hot Restart" (trvajúcej okolo 7 sekúnd).



Obr.2.2: Hot Reload a Hot Restart

Vo všeobecnosti možno teda uviesť, že Dart je programovací jazyk vytvorený taktiež spoločnosťou Google, podobajúci sa syntaxou k tzv. C-jazykom (C, Objective-C, C++ či C#). [5] Verzia 1.0 bola zverejnená v novembri 2013, pričom medzi jeho typické charakteristiky patrí fakt, že je objektovo orientovaný, relatívne jednoducho naučiteľný a spustiteľný v rôznych prostrediach, so širokou komunitou silne podporujúcou tento open-source projekt. [6]

2.3 Unity

Dôležitým rozhodnutím, ktoré sme museli urobiť bol spôsob, ako zakomponovať 3D model Rubikovej kocky do prostredia našej aplikácie. Bo dlhej rešerši sme pristúpili k použitiu herného engine Unity.

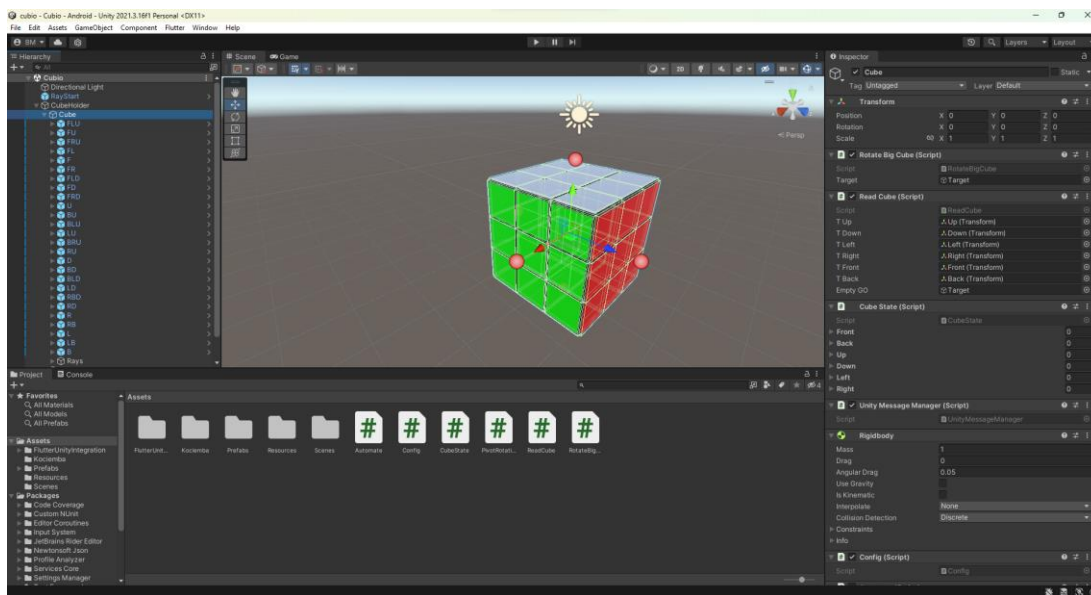
Unity je populárny herný engine, ktorý umožňuje vývojárom vytvárať 2D a 3D hry či modely pre rôzne platformy. Je známy svojím jednoduchým používaním a komplexným súborom funkcií, vďaka čomu je obľúbenou voľbou pre nezávislých aj profesionálnych vývojárov. Unity obsahuje výkonný editor, ktorý umožňuje jednoduchú tvorbu herných prostriedkov, fyzikálnych simulácií a skriptov. Okrem toho veľká a aktívna komunita okolo tohto programu poskytuje vývojárom množstvo zdrojov a podpory. Práve táto podpora bola pre nás kľúčová.

Spočiatku sme rozmýšľali nad programom Blender, no kľúčová vec, ktorá by nám chýbala, je komunikácia s našou aplikáciou pri vykonávaní pohybu kocky. Samotný program nijako nevie komunikovať a dynamicky meniť, napr. aktuálne farby kocky,

ale len vytvára 3D model, ktorý vie následne zobrazit' v našej aplikácii. Hľadali sme preto ďalej.

Potenciálnym riešením sa javilo využiť oficiálnu sadu funkcií pre prácu s 3D vydanú v posledných verziách nami využívaného frameworku Flutter. Najväčším problémom však bolo, že kvôli novosti týchto funkcií sme prakticky nenašli žiadne konkrétne príklady, tak ani žiadnu kvalitne zdokumentovanú prácu využívajúcu tieto metódy.

Po niekoľkých dňoch sme však pri dekompilácii oficiálne dodávanej aplikácie (kapitola 3.2) natrafili na autorské riešenie využívajúce práve engine Unity. Začali sme preto hľadať možnosti implementácie tohto engine v kombinácii s Flutter-om. Nakoniec sa nám podarilo nájsť na package 'flutter_unity_widget' (https://pub.dev/packages/flutter_unity_widget) poskytujúci plnú, obojstrannú komunikáciu s engineom samotným. Prirátajme k tomu výborne spracovanú oficiálnu dokumentáciu a komunitnú podporu na komunitnom Discord serveri a perfektné riešenie bolo na svete.



Obr.2.3: Prostredie programu Unity

2.4 Bluetooth

Bluetooth je jedna z technológií, ktorú možno využiť k bezdrôtovej komunikácii medzi zariadeniami. Rovnakú technológiu využíva aj náš model našej Rubikovej kocky. Poďme si teda predstaviť túto metódu.

Bluetooth bol vyvinutý švédskou spoločnosťou Ericsson v roku 1994 ako spôsob pripojenia mobilných telefónov k slúchadlám bez potreby použitia káblov. V súčasnosti sa Bluetooth používa v naozaj širokej škále zariadení vrátane smartfónov, prenosných počítačov, reproduktorov, slúchadiel, a v našom prípade aj vrátane inteligentných – smart Rubikových kociek.

Využívajúca rádiovú frekvenciu 2,4 GHz na prenos dát, jednou z jej hlavných výhod tejto technológie jej nízka spotreba energie a relatívne nízka cena na implementáciu v porovnaní s inými metódami bezdrôtovej komunikácie. Pre zvýšenie bezpečnosti sú údaje navyše zašifrované. Azda jej najväčšou nevýhodou je jej relatívne malé pokrytie. [7]

Údaje sa teda posielajú vzduchom – ako sme už spomenuli, za pomoci rádiových vln. V každom zariadení využívajúce túto technológiu je modul, ktorý obsahuje rádiový vysielateľ a prijímač, ako aj malý mikrokontrolér, ktorý riadi proces komunikácie.

Aby sa zabránilo rušeniu inými zariadeniami prenášajúcimi údaje, využíva Bluetooth tzv. "frequency hopping" - Ide o rýchle prepínanie rádiovej frekvencie, na ktorej sa prenášajú údaje, aby práve k rušeniu nedochádzalo.

2.5 Android

Podobne ako pri Bluetooth technológii, tak ani tu nebola pre nás príliš voľba. Ako sme už spomínali vyššie, vzhľadom k tomu, že sme sa rozhodli vytvoriť mobilnú aplikáciu, možno považovať za jediné dva relevantné operačné systémy pre telefóny Android a iOS. Ideál by bol implementovať aplikáciu pre oba systémy, no keďže sme nedisponovali adekvátnym hardvérom od značky Apple, nie je pre nás možné ani vyvíjať pre ekosystém tejto značky. Android bol preto jediná logická voľba. Implementácia pre jeden operačný má však aj výhodu, a to, že výber dodatočných balíkov, je jednoduchší, lebo nemusia využívať plnú funkcionality oboch systémov.

Založený firmou Google, tento open-source operačný systém postavený na Linuxovom základe je určený primárne pre smartfóny a tablety. Podľa portálu Statista [8] je systém Android tým najzastúpenejším na mobilnom trhu a to s takmer s výškou 71,8% k poslednému štvrtroku roku 2022. S tým, že je omnoho jednoduchšie vyvíjať aplikácie pre tento ekosystém a fakt, že je tento ekosystém otvorený, sa nemožno

čudovať obrovskému množstvu vyvinutých aplikácií. Odhady tvrdia, že ich sú už skoro 3 milióny. [9]

2.6 IntelliJ IDEA

Voľba toho správneho prostredia je pre programátora niečo ako pre stolára dláto, s ktorým pracuje. Rozhodli sme sa preto venovať v tejto kapitole práve voľbe vývojového prostredia a prečo sme siahli po systéme od firmy JetBrains.

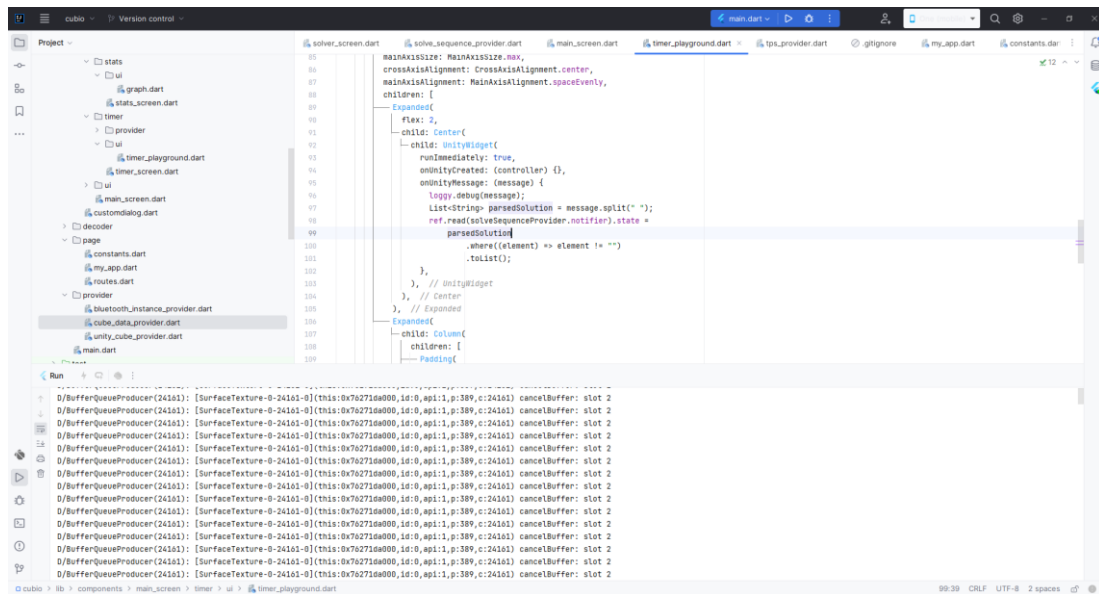
Pri tvorbe mobilných aplikácií máme spravidla výber medzi tromi najpopulárnejšími možnosťami čo sa vývojárskych prostredí týka. Sú nimi produkty Android Studio, Visual Studio Code a práve IntelliJ IDEA.

Vývojárske prostredia alebo ako moderne nazývané, IDE, nám môžu ponúkať nespočetne funkcií. Najväčším priekopníkom v tejto oblasti je práve IDE Android Studio, ktoré je doporučeným riešením od Android tímu samotného. Azda najlepšou funkciou sú priamo implementované, virtuálne Android simulácie – emulátory. Obrovské množstvo funkcií si však vyžaduje svoju daň, a to v podobe veľkého využitia zdrojov počítača, primárne čo sa RAM týka.

Pokiaľ by sme chceli ušetriť operačnú pamäť, jasnou voľbou je pre nás v takom prípade Visual Studio Code udržiavané firmou Microsoft. Najväčšiu prednosť tvorí plne zadarmo, navyše vysoko nastaviteľné prostredie, ktoré možno využiť na vývoj v kombinácii s takmer akýmkoľvek programovacím jazykom (za pomoci rozšírení). Aj keď je to pre niektorých určite prívetivá vlastnosť, z našej skúsenosti nám skôr prišlo, že v prípade písania kódu v jazyku Dart, resp. frameworku Flutter, pôsobí vývoj trochu neprehľadne a chaoticky v porovnaní so svojou konkurenciou.

V nedávnej dobe sme narazili taktiež na prostredie IntelliJ IDEA spravované korporátom JetBrains. Sú autormi mnohých ďalších IDE – známymi sú produktmi ako PhpStorm alebo PyCharm a i. Konkrétne, prostredie IntelliJ IDEA je zamerané primárne na vývoj v Jave a celkovo na tvorbu mobilných aplikácií. Niektorí by mohli namietat, že na rozdiel od prostredia VS Code má IntelliJ aj prémiovú verziu. Výhodou však je, že bezplatná verzia má len veľmi málo obmedzení, a pokiaľ študent vlastní ISIC kartu, možno celý prémiový balík firmy JetBrains využívať zadarmo. Podobne sme teda vykonali aj my. Páčilo sa nám lepšie manažovanie operačnej

pamäti, ako aj inteligentné dopĺňanie či tipy k nášmu kódu urýchľujúce vývoj, spoločne s robustnou sadou klávesových skratiek.



Obr.2.6: Prostredie programu IntelliJ IDEA

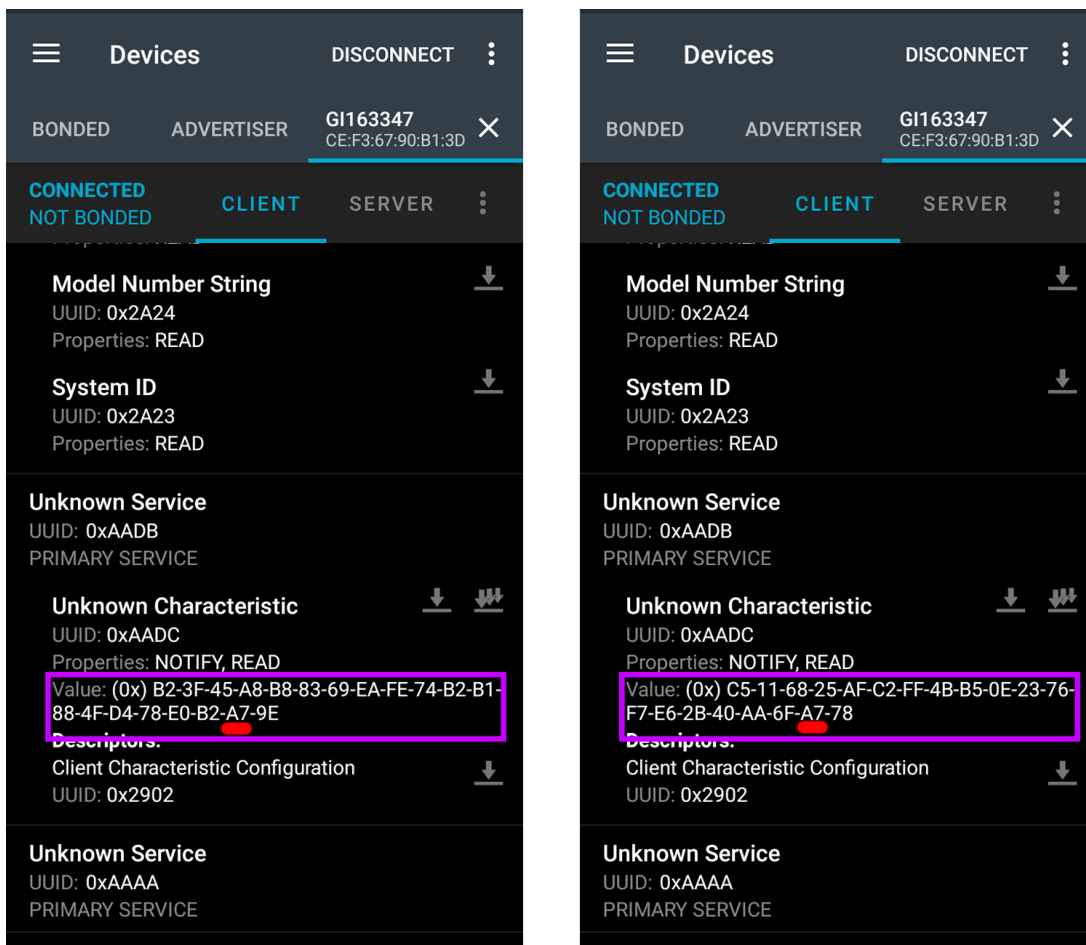
3 Dekódovanie Bluetooth komunikácie

Po naštudovaní si princípu fungovania technológie Bluetooth sme sa vrhli na dekódovanie komunikácie našej inteligentnej Rubikovej kocky. Našťastie to však nebudeme robiť úplne "od nuly", ale náš základ postavíme na článku umiestnenom na platforme Medium (<https://medium.com/@juanancaramunt/xiaomi-mi-smart-rubik-cube-ff5a22549f90>), kde autor rieši veľmi podobný problém na inteligentnej Rubikovej kocke inej značky. Týmto mu chceme poďakovať.

3.1 Úvodná analýza komunikácie

Logickou vecou, na ktorú sme sa snažili prísť bolo, či sa pri komunikácii nevyskytujú určité vzory, ktoré možno odsledovať. Potrebovali sme preto program, ktorý by mohol "napichnúť" a čítať dáta, ktoré naša kocka posiela. Využili sme preto aplikáciu z Google Store a to s názvom "nRF Connect for Mobile".

Čakali sme teda, čo nám kocka ukáže. Skúšali sme vykonať rôzne sekvencie ťahov – ako rovnaké, tak aj rozdielne – očakávali sme, že po rovnakých ťahoch dostaneme rovnaké hodnoty, avšak nestalo sa. Zaujala nás ale jedna konštanta - A7 - na 19. pozícii 20-položkového zoznamu bytov, ktorá sa objavila vždy na rovnakom mieste, bez ohľadu na náš ťah či sekvenciu.

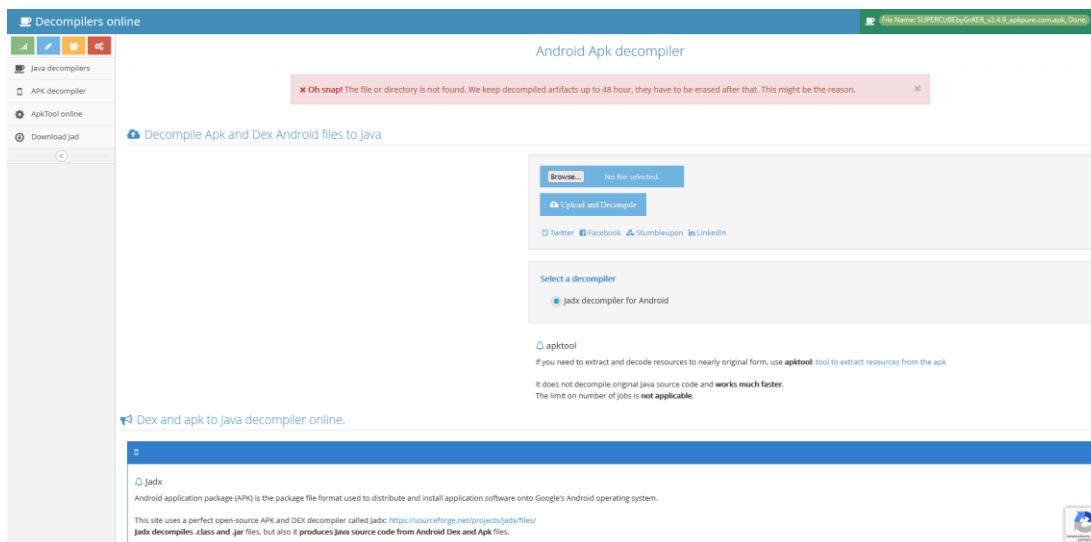


Obr.3.1: Odpoveď kocky po vykonaní rovnakého ťahu

Údaje, ktoré sme dostali, nám teda toho okrem jednej konštanty, ktorej význam aj tak nepoznáme, moc toho nenapovedali. Bolo treba nájsť preto určitý kus kódu, ktorý by túto komunikáciu prekladal do rozumného a aspoň trochu zrozumiteľného, deterministického tvaru – znamenajúc, ak som spravil rovnaký ťah, mal by som dostať podobný (ideálne rovnaký) výstup. Logickým postupom je preto preskúmať oficiálnu aplikáciu dodávanú výrobcom kocky.

3.2 Prieskum a dekódovanie oficiálnej aplikácie




Bol sa čas pustiť do dekódovania oficiálnej aplikácie. Ako autor spomína vo svojom článku, najviac sa mu osvedčil Java Decompiler zo stránky <http://www.javadecompilers.com/apk>.



Obr.3.2.1: Vzhľad prostredia nami využíteho Java Decompilera

Zo stránok tretích strán (primárne zo stránky ApkPure) sme si zistili aktuálnu verziu aplikácie a stiahli jej najnovšiu verziu. Niektorí by mohli namietat', prečo sme nestiahli aplikáciu z Google Store ako sa má. Dôvody boli dva. Nami žiadaný .apk súbor nie je možné stiahnuť do počítača, nie to stiahnuť jeho staršie verzie.)

Dekompilovali sme teda najnovšiu verziu oficiálnej aplikácie (v tej dobe verzia 2.6.15). Natrafili sme však na problém. Autor referuje vo svojom článku cestu "resources\assets\bin\Data\Managed", ktorá by mala obsahovať .dll súbory, ktoré využíva Unity engine. Niečo však nesedelo. V tomto priečinku sa nachádzali iba nasledovné súbory:

 Metadata	16. 10. 2022 11:03	Priečinkov súborov
 Resources	16. 10. 2022 11:03	Priečinkov súborov
 etc	16. 10. 2022 11:03	Priečinkov súborov

Obr.3.2.2: Obsah priečinka s cestou "resources\assets\bin\Data\Managed"

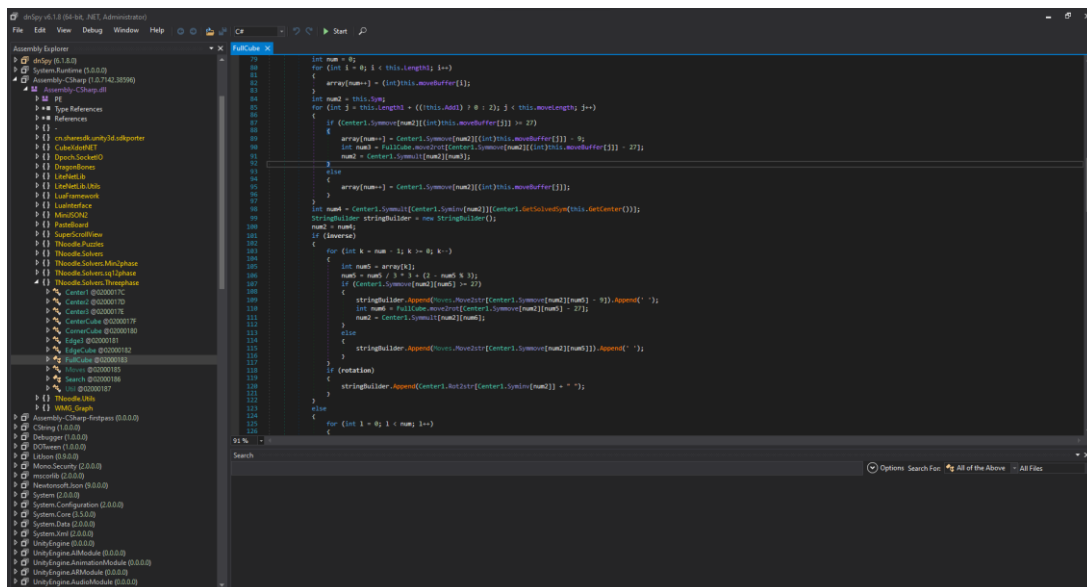
Skúsili sme preto logicky preskúmať aj ostatné cesty, či sa náhodou autor nezmýlil. Opak je však pravdou a prítomné boli len .unity3d a .dat súbory, čo značilo, že príloha, konkrétne model kocky, bol v oficiálnej aplikácii zabalený a zašifrovaný, čo nám znemožnilo sa k modelu, a teda aj dekompilátoru Bluetooth komunikácie, dostať. Rozhodli sme sa preto skúšať dekompilovať staršie verzie aplikácie a dúfať, že

nájde verziu, kde bude tento model prístupný k nahliadnutiu (.dll súbory sú totiž jediné z prístupných a čitateľných pre nás ako externých užívateľov čo sa unity týka. Problém autora bol, že taktiež nespomenul meno oficiálnej aplikácie – my sme za ňu považovali aplikáciu SUPERCUBE - by GiiKER).

Operácia nakoniec dopadla úspešne. Je kľúčové, aby sme siahli po verzii aplikácie NIŽŠEJ ako 2.5.0 – to znamená verzie 2.4.9 a nižšie – práve po verzii 2.4.9 sme siali aj my. A hľa, priečinok plný .dll súborov, presne tak, ako autor spomínal. Tieto súbory však nejde len tak čítať, nutnosťou je využiť externý softvér.

3.3 Analýza .dll súborov

Spomínali sme externý program. Jeho názov je dnSpy. Slúži na debugovanie či dekompilovanie takýchto registrov využívaných práve našim enginom. Je však častejšie využívaný ako .NET dekompilér. Výskumníci v oblasti kybernetickej bezpečnosti využívajú napríklad tento program bežne pri analýze najrôznejších malvérov a iných škodlivých softvérov postavených práve na frameworku .NET. [10]

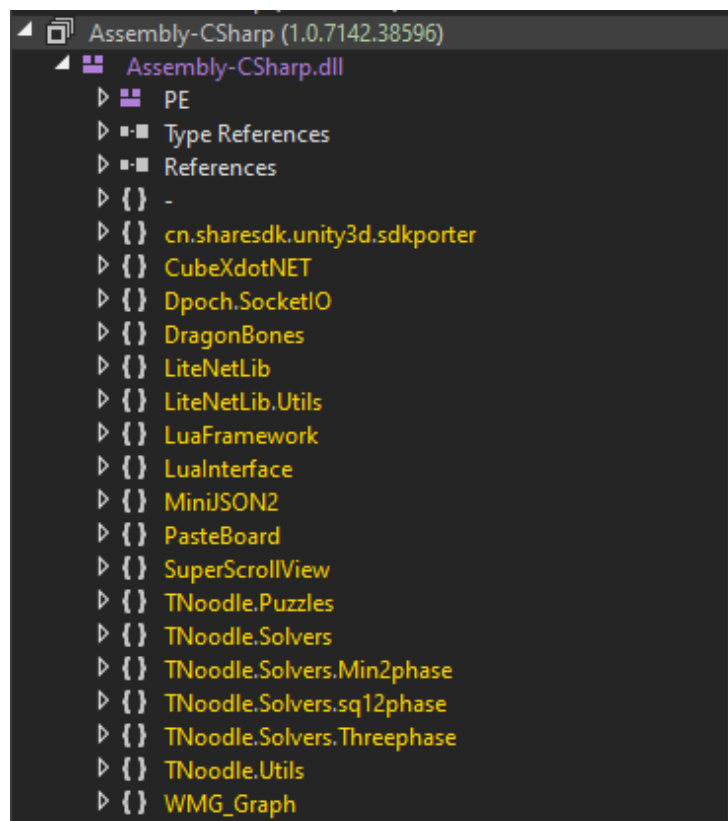


Obr.3.3.1: Prostredie programu dnSpy

Čo teda hľadáme? To sme ešte úplne nevedeli. Indície toho, čo sme mali robiť, boli buď vyhľadať pole o dĺžke 20 položiek, prípadne nájsť niekde povestnú konštantu A7. Tvorca článku nám však trochu uľahčil pátranie. Museli sme nájsť triedu BluetoothDecoder – konkrétne jednu z jeho metód zvanú ConverseToPaperType, ktorá prijíma práve zoznam dlhý 20 položiek. Niekde sme sa začínali dostávať.

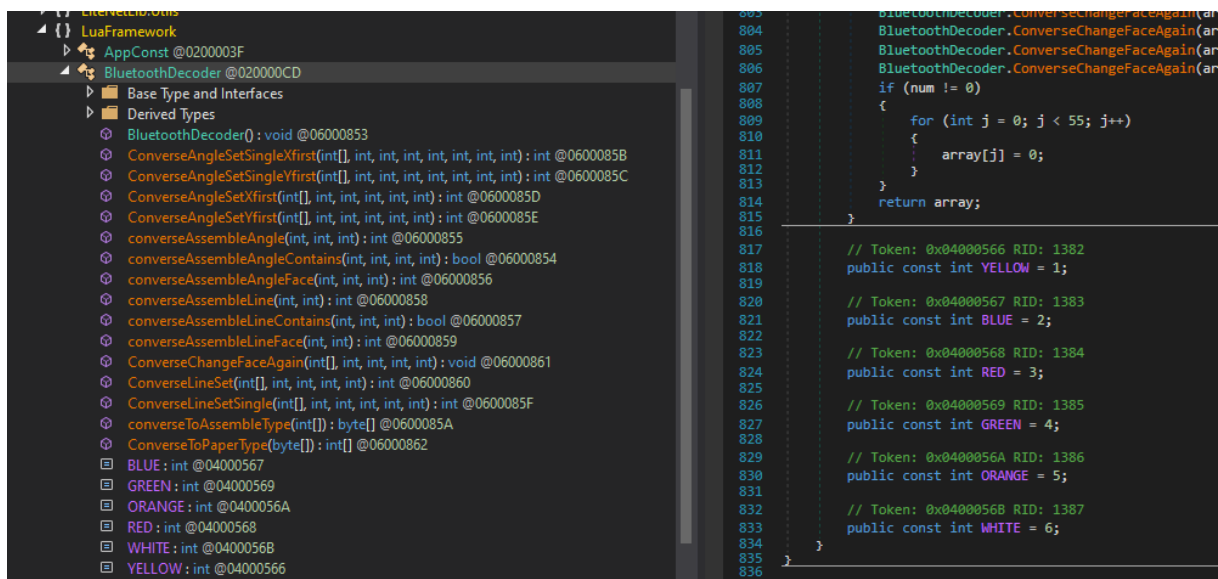
Po štarte programu otvoríme už nami známu cestu (z kapitoly 3.2) a vyberieme celý obsah tohto priečinka (je dobré predtým v menu vybrať File > Close all, až potom načítať naše súbory pre lepšiu prehľadnosť).

Postupným prechádzaním registrov sme došli k cieľu – k triede BluetoothDecoder. Ľavým panelom si nájdeme položku Assembly-CSharp (1.0.7142.38596). Položku rozbalíme. Následne rozbalíme aj podpoložku s názvom Assembly-CSharp.dll. Mali by sme vidieť niečo nasledovné:



Obr.3.3.2: Stav po rozbalení položky Assembly-CSharp

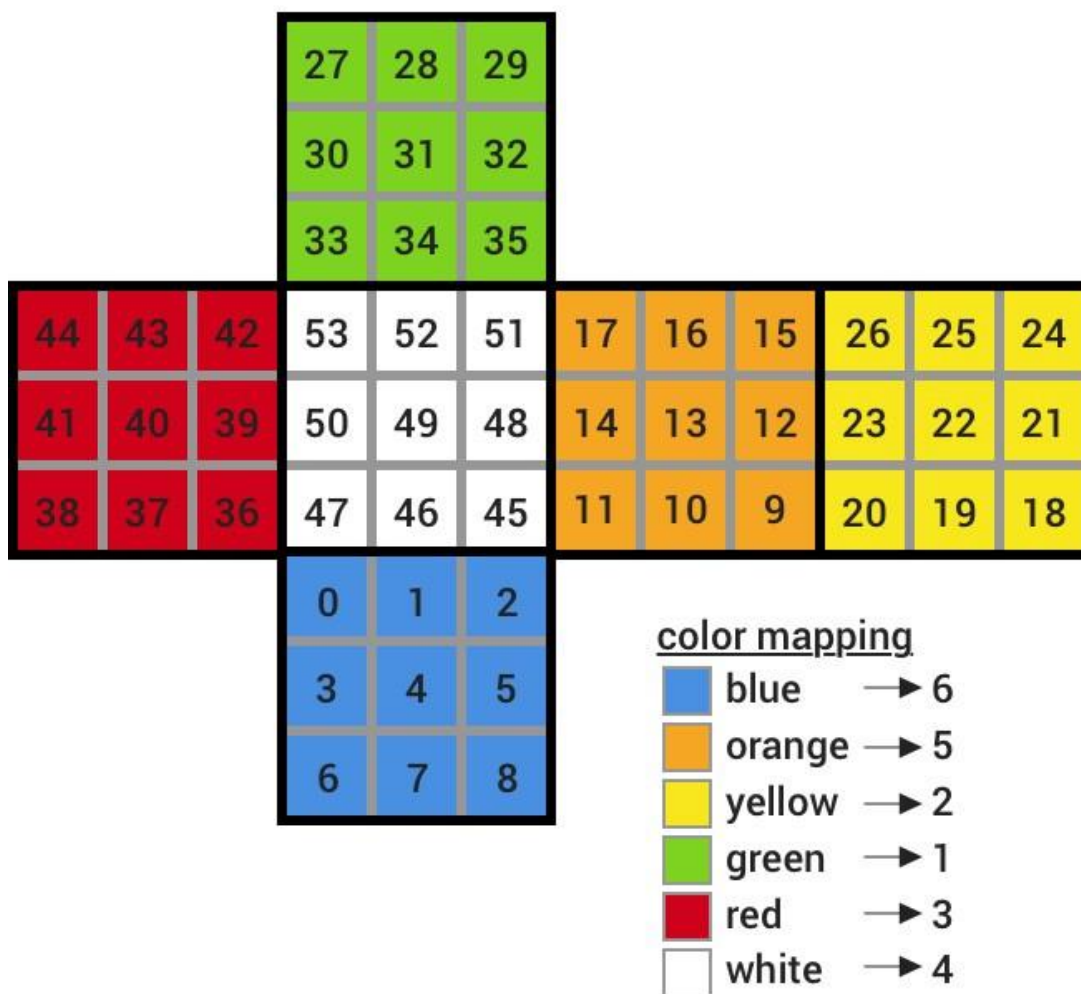
Pokračujeme. Expandujeme položku s názvom LuaFramework v ktorej sa nachádza naša cieľová trieda. Po kliknutí na BluetoothDecoder môžeme vidieť kompletný zdrojový kód triedy a po jej rozbalení jej jednotlivé metódy či konštanty, ako kódy farieb.



Obr.3.3.3: Obsah triedy BluetoothDecoder

Toto ale nie je všetko. Dôležitá bude pre nás ešte jedna trieda. Doposiaľ máme triedu BluetoothDecoder a jej metódu ConvertToPaperType, čo do ľudského prekladu možno preložiť ako funkciu, ktorej úlohou je vrátiť list dlhý 55 položiek – prvá položka bude predstavovať číslo, či konverzia prebehla úspešne, resp. či vstup bol validný, kde 0 predstavuje úspech. Následných ďalších 54 položiek predstavuje jednotlivé farby kocky podľa nasledovnej legendy.

each element of the array represents the color a square



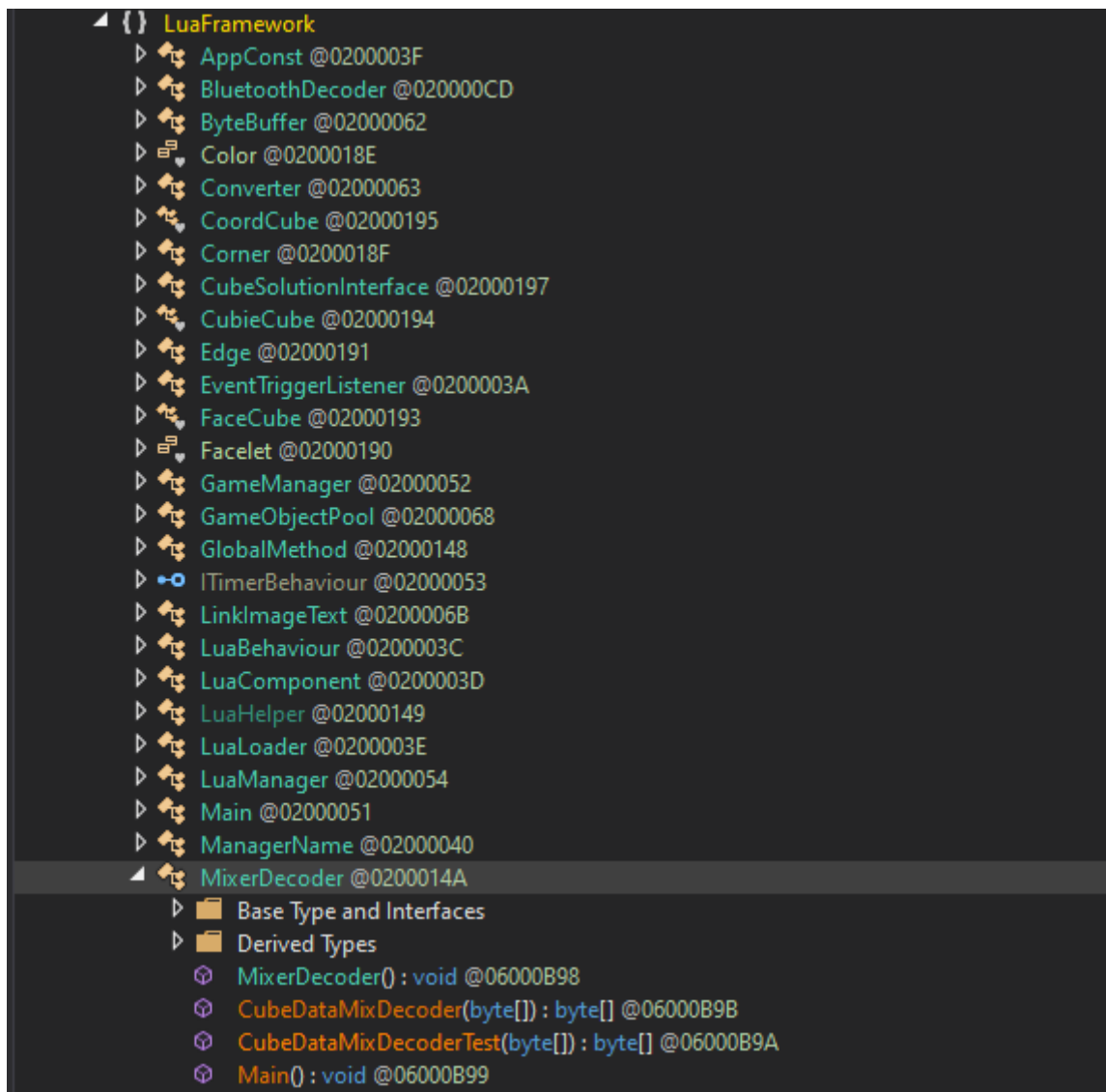
Obr.3.3.4: Legenda kódovania položiek "papierového typu"

Keď sme túto metódu prekonvertovali do programovacieho jazyka Dart (kapitola 4.2) a začali ju debugovať, hodnota s indexom 0 nanešťastie nenaberala hodnotu nula. Vari nám ešte niečo chýba? Áno! Treba si uvedomiť, že nemôžeme posielat' surový byteArray rovno do metódy converseToPaperType, musíme ešte túto "surovú" komunikáciu poslať ešte jednej funkcii.

Ňou je funkcia, lepšie ju nazvať metódou triedy MixerDecoder (rozdielna od triedy BluetoothDecoder), nazývaná CubeDataMixDecoder, ktorá konvertuje komunikáciu na síce list s rovnakou dĺžkou ako pôvodný vstup, no s týmto výstupom už možno slobodne pracovať a odvádzať práve napr. ako vyzerá náš "papierový typ", na základe čoho môžeme posúdiť, či už naša kocka bola vyriešená a pod. (túto vlastnosť využijeme aj neskôr pri tvorbe našej aplikácie). Finálnym dôkazom, že trieda

MixerDecoder je využívaná je fakt, že je v nej ako v jedinej nami spomínaná konštanta A7 – a to vo forme jej desiatkovej hodnoty – 167.

Trieda MixerDecoder je mimochodom rovnako ako trieda BluetoothDecoder dieťaťom položky LuaFramework.



Obr.3.3.5: Obsah triedy MixerDecoder

Na záver tejto kapitoly by som ešte chcel podotknúť jednu zaujímavosť. Je zaujímavé sledovať, že dve rôzne hodnoty, ktoré sme dostali po vykonaní rovnakého pohybu kocky vytvárajú rovnaký výstup, hoci vstupné hodnoty sú rôzne.

Výstup1: [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]

Výstup2: [6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4]

24

4 Tvorba aplikácie a 3D modelu

V dnešnej digitálnej dobe sa vytváranie a navrhovanie aplikácií a 3D modelov stalo nevyhnutnou zručnosťou pre mnohé odvetvia. Od vývoja videohier až po architektonický dizajn, schopnosť vytvárať interaktívne a vizuálne ohromujúce aplikácie a modely sa stala nevyhnutným nástrojom pre profesionálov i nadšencov.

Cieľom tejto kapitoly je preto primárne priblížiť postup možnosti replikovať podobný projekt aj u seba a sprevádzať tak potenciálneho riešiteľa aj s prípadnými problémami, na ktoré by mohol pri tvorbe výsledného produktu naraziť.

4.1 Nastavenie prostredia a vytvorenie základného projektu

Vytvorenie základného projektu vo frameworku Flutter v prostredí IntelliJ IDEA je relatívne jednoduchý proces, ktorý možno vykonať v niekoľkých krokoch. Predtým, než začneme, je dôležité zabezpečiť, aby boli v počítači správne nainštalované Flutter - presnejšie Flutter SDK a samozrejme aj IDE IntelliJ.

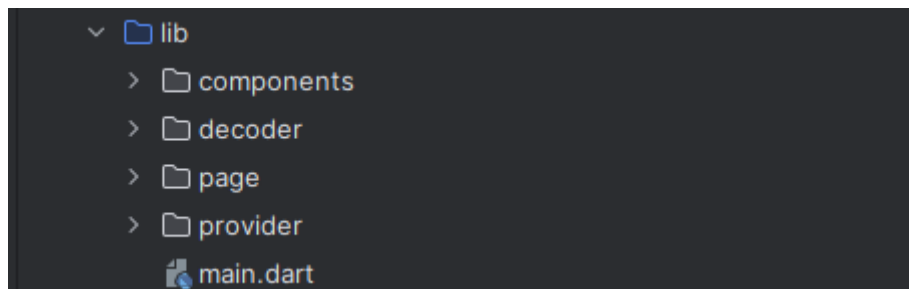
Pre nastavenie všetkého potrebného, vrátane vytvorenia základného projektu, možno využiť článok zo stránky <https://www.geeksforgeeks.org/how-to-install-and-setup-flutter-in-intellij-idea/>. Je vhodné spomenúť, že využívame nové UI tohto prostredia, tieto kroky sa preto na staršej verzii môžu mierne líšiť - ako prepnúť na nové UI možno nájsť na adrese <https://www.jetbrains.com/help/idea/new-ui.html>.

Ideme vytvárať kód pre operačný systém Android, ako ho ale budeme testovať? Možno si za pomoci Android Studio vytvoriť virtuálnu Android simuláciu, ale výhodnejším riešením sa nám javí využiť reálne zariadenie pripojené k vášmu počítaču, primárne kvôli výkonu.

Všetko čo stačí je sa prekliknúť nastavení nášho zariadenia a podľa smartfónu povoliť vývojárske nastavenia, kde vyberieme a následne zaškrtneme položku "USB Debugging" – alebo "Zapnúť ladenie USB", pokiaľ máme nastavenia slovensky. Následne stačí pripojiť telefón k PC. Malo by sa objaviť okno, kde stlačíme možnosť "Áno". Gratulujem, sme pripravený vyvíjať reálnu aplikáciu.

4.2 Konverzia Bluetooth dekódera do jazyku Dart

Medzičasom sme si v našom projekte vytvorili základnú priečinkovú štruktúru, konkrétne lokalizovanej v priečinku lib.



Obr.4.2.1: Priečinková štruktúra priečinku lib

Následne sme pristúpili k prekonvertovaniu Bluetooth dekódera, ktorý sme našli v oficiálnej aplikácii v podobe jazyku C#. Ak chceme dekódovať reťazec, ktorý nám bude kocka posielat' za pomoci Bluetooth-u v našej aplikácii, budeme musieť prekonvertovať tento dekóder do rovnakého jazyka, akým bude písaná naša aplikácia – a keďže u nás to je Flutter, budeme prepisovať do jazyka Dart.

Vytvorili sme si teda triedu CommunicationDecoder a začali do nej prepisovať metódy originálneho kódu (Pozn.: Vytvárame triedu namiesto samostatných metód, pretože sa to považuje za "good-practice"). Ťažili sme z toho, že oba jazyky sú si veľmi podobné.

Aj napriek tomu, proces nie je možné veľmi urýchliť. Žiaľ sa nám nepodarilo nájsť žiaden nástroj, ktorý by naše tvrdenie vyvracal, jediným ako-takým riešením sa zdá siahnuť po umelých inteligenciách ako ChatGPT od firmy OpenAI, ktorej výstupy však častokrát obsahujú množstvo chýb. Určitá znalosť je tu preto tak stále potrebná.

Výhodou však je, že nepotrebujeme konvertovať všetky funkcie, napríklad testovacie. Možno preto postupovať podľa našej šablóny, kde vidno, ktoré funkcie budú pre nás potrebné:

```

import 'dart:typed_data';

class CommunicationDecoder {
  CommunicationDecoder();

  // HLAVNE FUNKCIE
  List<int> bytesToCubeOutputData(Uint8List mixData) {
    ...
  }

  List<int> cubeOutputToPaperType(List<int> cubeOutput) {
    ...
  }

  // POMOCNE FUNKCIE
  int _converseAngleSetXFirst(Uint8List cube, angle, angleFace, f1, f2, f3) {
    ...
  }

  int _converseAngleSetSingleXFirst(cube, angleFace, p1, p2, p3, c1, c2, c3){
    ...
  }

  int _converseAngleSetYFirst(cube, angle, angleFace, f1, f2, f3) {
    ...
  }

  int _converseAngleSetSingleYFirst(cube, angleFace, p1, p2, p3, c1, c2, c3) {
    ...
  }

  int _converseLineSet(cube, line, lineFace, p1, p2) {
    ...
  }

  int _converseLineSetSingle(cube, lineFace, p1, p2, c1, c2) {
    ...
  }

  void _converseChangeFaceAgain(cube, a1, a2, a3, a4) {
    ...
  }
}

```

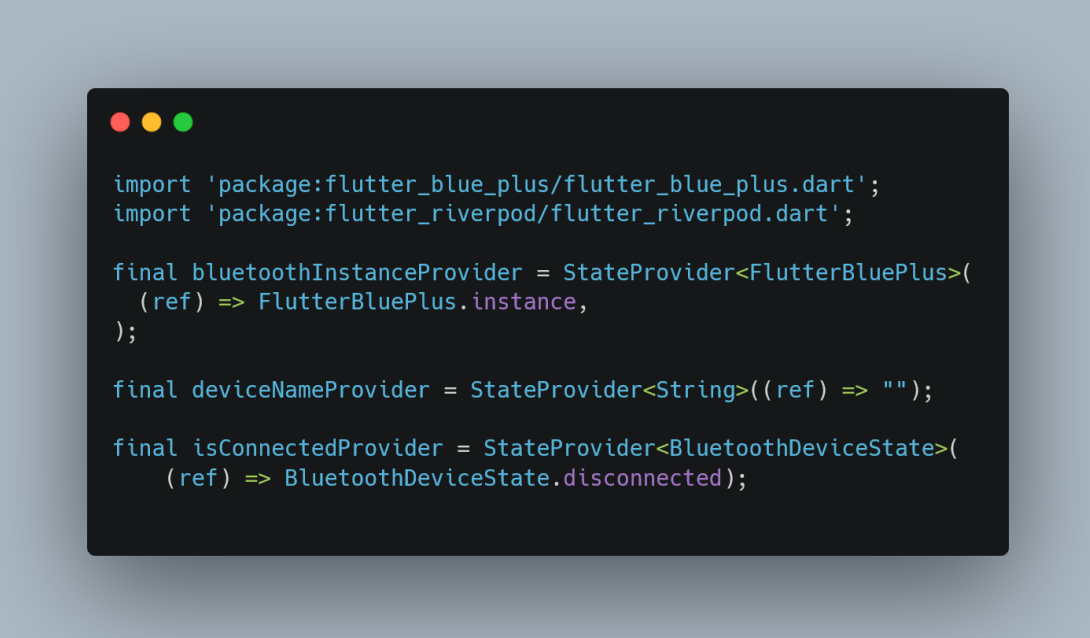
Obr.4.2.2: Nutné funkcie na prekonvertovanie

4.3 Napojenie Rubikovej kocky a Bluetooth-u

Keďže dekódér už máme, je načas prepojiť komunikáciu s kockou k našej aplikácii. Pôvodne sme pracovali s Flutter balíkom `flutter_blue` (https://pub.dev/packages/flutter_blue), no kvôli faktu, že už takmer dva roky je tento plugin bez nových commitov, presedlali sme na balík `flutter_blue_plus` (https://pub.dev/packages/flutter_blue_plus), čo je prakticky ten istý package, akurát že si ho na starosť zobrala komunita, ktorá ho priebežne aktualizuje.

Ako každý iný plugin, aj tento inštalujeme príkazom `flutter pub get <nazov balíka>`, následne postupujeme podľa dokumentácie poskytnutej na stránke pluginu (Pozn.: V prípade problémov možno kontaktovať autorov na ich komunitnom Discord serveri.). Vytvorili sme si samozrejme inštanciu triedy `FlutterBluePlus`, avšak inak, ako odporúčajú autori. Využili sme tzv. providery s pomocou pluginu `riverpod` (<https://pub.dev/packages/riverpod>).

Na čo nám ale slúžia providery? Providery nám centralizujú state – stav našej aplikácie. Umožňujú prístup k nemu z ktoréhokoľvek miesta v aplikácii, čím zjednodušujú jeho správu a robia kód prehľadnejším. Je dobré si tento odstavec zapamätať, pretože providery budeme ešte široko v našej aplikácii využívať.

A screenshot of a code editor with a dark background and light-colored text. The code defines three Riverpod providers for Bluetooth functionality. The first provider, `bluetoothInstanceProvider`, is a `StateProvider<FlutterBluePlus>` initialized with `FlutterBluePlus.instance`. The second provider, `deviceNameProvider`, is a `StateProvider<String>` initialized with an empty string. The third provider, `isConnectedProvider`, is a `StateProvider<BluetoothDeviceState>` initialized with `BluetoothDeviceState.disconnected`.

```
import 'package:flutter_blue_plus/flutter_blue_plus.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

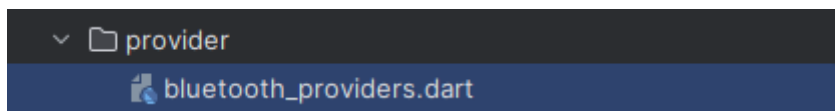
final bluetoothInstanceProvider = StateProvider<FlutterBluePlus>(
  (ref) => FlutterBluePlus.instance,
);

final deviceNameProvider = StateProvider<String>((ref) => "");

final isConnectedProvider = StateProvider<BluetoothDeviceState>(
  (ref) => BluetoothDeviceState.disconnected);
```

Obr.4.3.1: Bluetooth providery

Iste ste si všimli, obsahom kódu nie je len jeden provider, ale ešte ďalšie dva, jeden ukladajúci meno zariadenia a druhý stav, či je zariadenie stále pripojené (čo neskôr využijeme na vrátenie sa k vyhľadávaniu dostupných zariadení, pokiaľ sa kocka odpojí, napr. kvôli neaktivite). Toto všetko je obsahom jedného súboru `bluetooth_providers.dart`, ktorý bude obsahovať práve všetky bluetooth-related providery.



Obr.4.3.1: Obsah súboru lib/provider

Čo sa grafického spracovania týka, odrážali sme sa od UI poskytnutého v Example sekcii pluginu flutter_blue_plus. Náš výsledný dizajn si možno prehliadnuť v prílohe A vo forme obrázku (príloha A, B, C). Samozrejme, možno preskúmať aj zdrojový kód dodávaný s dokumentáciou.

Budeme používať však ešte jeden provider. Nie ako doteraz, StateProvider, ale StateNotifierProvider, ktorý drží v okrem hodnoty, častokrát aj chovanie, čo sa stane napríklad, keď sa aktualizuje hodnota.

```

final cubeDataProvider = StateNotifierProvider<CubeNotifier, List<int>>(
    (ref) => CubeNotifier(ref),
);

class CubeNotifier extends StateNotifier<List<int>> with UiLoggy {
    final Ref ref;

    // This original state will only store the original value (rawData)
    CubeNotifier(this.ref) : super([]);

    List<int> processedData = [];
    List<int> cubeColors = [];

    final CommunicationDecoder _communicationDecoder = CommunicationDecoder();

    int _callNum = 0;

    String _lastMove = "-";

    void process(List<int> rawData) async {
        ...
    }

    void _setLastMove() {
        ...
    }

    int _getNibble(List<int> val, int i) {
        ...
    }

    bool get isSolved {
        ...
    }

    String get lastMove {
        ...
    }
}

```

Obr.4.3.2: CubeNotifier

Prvé riadky patria deklarácií, volaniu konšuktora, a nastavením pomocných premenných. všetky verejné premené možno z takéhoto providera dostať príkazom `ref.read()`, `ref.watch()` alebo `ref.listen()`. Nový `StateNotifierProvider` v dnešnej verzii už ale napríklad neponúka starý príkaz `notifyListeners()`, ktorým sme kedykoľvek mohli všetkým listenerom aktualizovať stav. Dneska už musíme priradiť novú hodnotu premennej `state`, ktorá v tomto prípade predstavuje surové hodnoty kocky. Až po priradení novej hodnoty dôjde k oboznámení listenerov. Dáva si preto dávať pozor, lebo opačne môže dôjsť k nesprávnym updatom UI.

Funkcia `process` konvertuje za pomoci `CommunicationDecoder` triedy naše surové dáta to, čo chceme, t.j. buď spracovaný 20-položkový zoznam, alebo 55-položkový zoznam oboznamujúci nás o poradí farieb na našej kocke.

Ďalej funkcia `_setLastMove()`. Tá zisťuje zo spracovaného 20-položkového listu posledný ťah, ktorý následne priradzuje aj svojej lokálnej premennej. Funkcia `_getNibble()` je pomocnou funkciou tejto funkcie.

`lastMove` a `isSolved` nazývame gettery. Tie využívame, keď nechceme, aby mohol užívateľ mohol meniť dáta z inej triedy, čo by porušovalo pravidlá kvalitného kódu. Aké by to bolo, keby mohol užívateľ zmeniť posledný ťah kocky podľa seba?

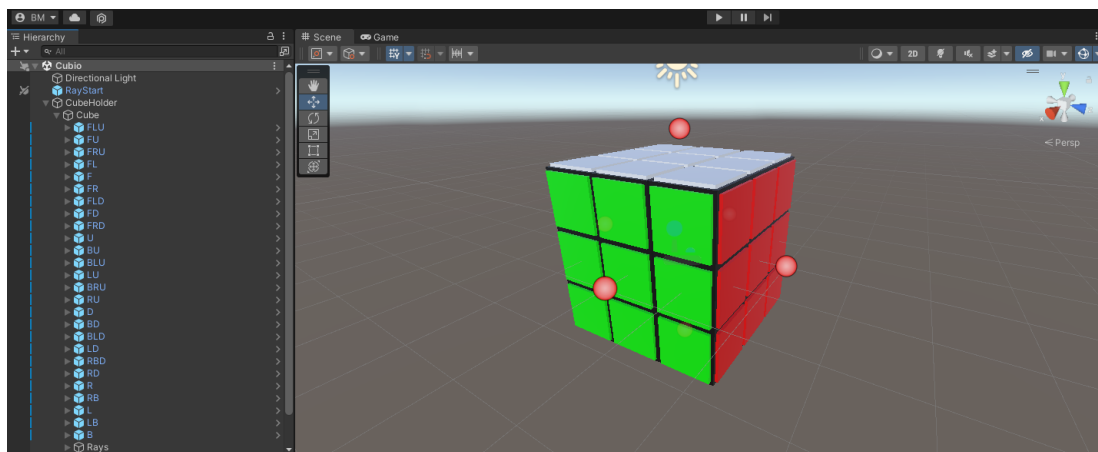
4.4 Tvorba Rubikovej kocky v prostredí Unity

Bolo načase vytvoriť 3D model našej kocky, a keďže pre nás bolo Unity úplne novou skúsenosťou, začali sme hľadať relevantné tutoriály k našej problematike. Jeden z nich ale zaujal viac ako iné. A to tutoriál od autora Megalomobilezo - Let's Make & Solve a Rubik's Cube in Unity (https://www.youtube.com/watch?v=JN9vx0veZ-c&list=PLuq_iMEtykn-ZOJyx2cY_k9WkixAhv11n).



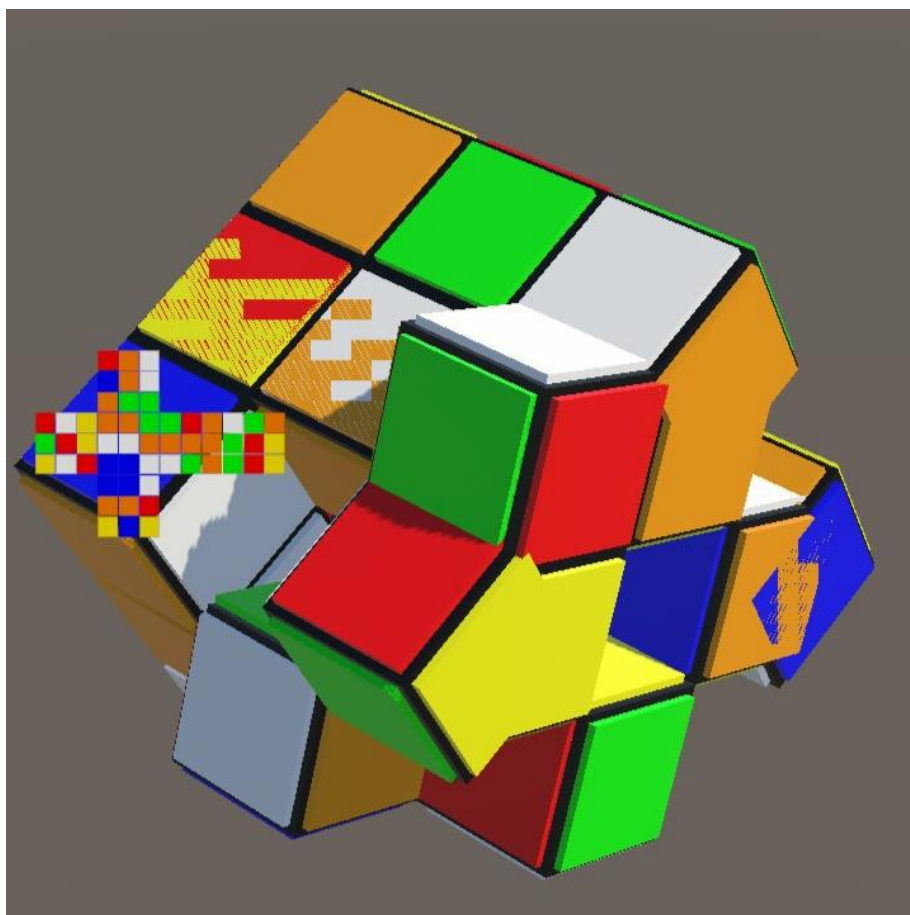
Obr.4.4.1: Vzhľad využitého Youtube tutoriálu

Po sedemdielnej inštrukčii, ktorá nám ukázala naozaj mnohé princípy prostredia Unity bol naším výsledkom už takmer hotový produkt. Avšak len navonok. Technický stav nebol dokonalý.



Obr.4.4.2: Medzivýsledok po dokončení inštruktážneho videa

Vo výslednej verzii sme okrem detailov ako nastavenie cieľového počtu snímok na 60 za sekundu pre lepšiu plynulosť, upravovali aj mnohé z metód autora, keďže pre náš projekt jednoducho neboli potrebné. Príkladmi sú napríklad funkcia rotovania strán myšou, funkcia Shuffle, CubeMap, a do určitej miery aj funkcia Solve, keďže naša finálna implementácia je značne odlišná od originálu. Najväčší problém autorovho riešenia bolo ale to, že pri neustálom klikaní tlačidla sa kocka pri pohybe do seba zrazí, vid' obrázok nižšie:



Obr.4.4.3: Okrajový prípad autorovej implementácie

Naše riešenie? Vyriešili sme to trochu inak, než je zvykom. A to pomocou časovača a funkcie DoMove, ktorá sa stará o vykonávanie pohybov. Funguje na podobnom princípe, ako keď autor v príručke generovali zoznam pohybov pri funkcii Shuffle alebo Solve.

Vytvorili sme si funkciu MyMove, ktorej budeme posilať z aplikácie ťah, ktorý náš modul vykoná. Aktuálny zoznam nevykonaných pohybov ktorý v sebe trieda Automate obsahuje preto vždycky rozšírime a necháme približne každých pol sekundy vykonať funkciu DoMove.

Okrem toho sme si aj vytvorili funkciu SetupPosition, ktorú taktiež voláme z našej aplikácie. Parametrom je prvotné rozloženie kocky, ktoré treba poslať Unity kontroléru za pomoci našej aplikácie .

Aby sme ušetrili čo najviac času kým sa naša kocka nachystá, vo funkcii dočasne vypneme animáciu prechodu. Následne nám zostáva úloha dopočítať ťahy, ako sa k

takémuto úvodnému stavu možno dostať. Využijeme na to autorovu C# implementáciu Kociemba algoritmu.

Kociembov algoritmus je dobre známa metóda na riešenie Rubikovej kocky. Algoritmus využíva kombináciu teórie skupín a vyhľadávacích algoritmov na určenie optimálnej postupnosti ťahov a hoci je relatívne náročný na pochopenie a implementáciu, považuje sa za jeden z najrýchlejších algoritmov.

Posielanie stavu kocky (obrázok 3.3.4) do tejto C# implementácie si však vyžaduje špeciálne poradie jednotlivých stien oproti poradiu, ktoré posielala naša kocka cez Bluetooth interface. Správne poradie indexov pre úspešné vrátenie ťahov k vyriešeniu kocky je nasledovné:

[53, 52, 51, 50, 49, 48, 47, 46, 45, 11, 14, 17, 10, 13, 16, 9, 12, 15, 0, 1, 2, 3, 4, 5, 6, 7, 8, 18, 19, 20, 21, 22, 23, 24, 25, 26, 42, 39, 36, 43, 40, 37, 44, 41, 38, 35, 34, 33, 32, 31, 30, 29, 28, 27]

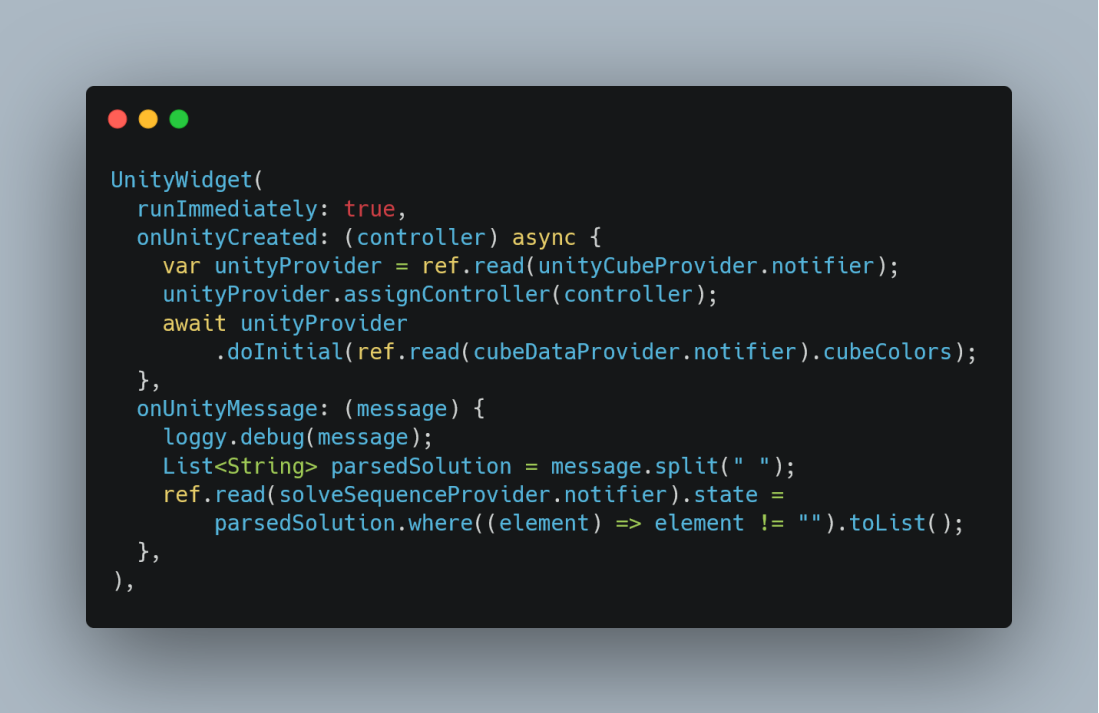
V prípade volania SetupPosition však nedostaneme ťahy z vyriešenej, na požadovaný, úvodný stav našej kocky. Ako to teda opraviť? Je to prekvapivo jednoduché – stačí si uvedomiť, že stačí len obrátiť ťahy, takže napríklad z ťahu R bude R'. Toto urobíme so všetkými ťahmi a výsledný zoznam obrátime.

4.5 Prepojenie Unity modelu s aplikáciou

Keďže bol náš model hotový, bolo ho načase prepojiť aj s našou aplikáciou. Ako sme už spomínali vyššie, toto prepojenie medzi technológiami Unity a Flutter sme dosiahli za pomoci Flutter balíka flutter_unity_engine (https://pub.dev/packages/flutter_unity_widget), ktorého prehľadnú dokumentáciu sme nasledovali.

Radi by sme najmä ale vypichli, aby ste sledovali verziu, ktorú používate. S novšími verziami tohto jazyka je nutné použiť verziu ^2022.2.0 a nie ^2022.1.0+7, pretože verzia +7 je špecificky pre staršie verzie tejto technológie.

Pre vykreslenie nášho modelu v aplikácii je potrebné vložiť do našej aplikácie widget poskytnutý balíkom s názvom UnityWidget. V prípade, že niečo neprebehlo v poriadku, odporúčame napísať na komunitný server, prípadne si znovu pozrieť dokumentáciu.



```

UnityWidget(
  runImmediately: true,
  onUnityCreated: (controller) async {
    var unityProvider = ref.read(unityCubeProvider.notifier);
    unityProvider.assignController(controller);
    await unityProvider
      .doInitial(ref.read(cubeDataProvider.notifier).cubeColors);
  },
  onUnityMessage: (message) {
    loggy.debug(message);
    List<String> parsedSolution = message.split(" ");
    ref.read(solveSequenceProvider.notifier).state =
      parsedSolution.where((element) => element != "").toList();
  },
),

```

Obr.4.5: Konkrétna implementácia UnityWidget-u

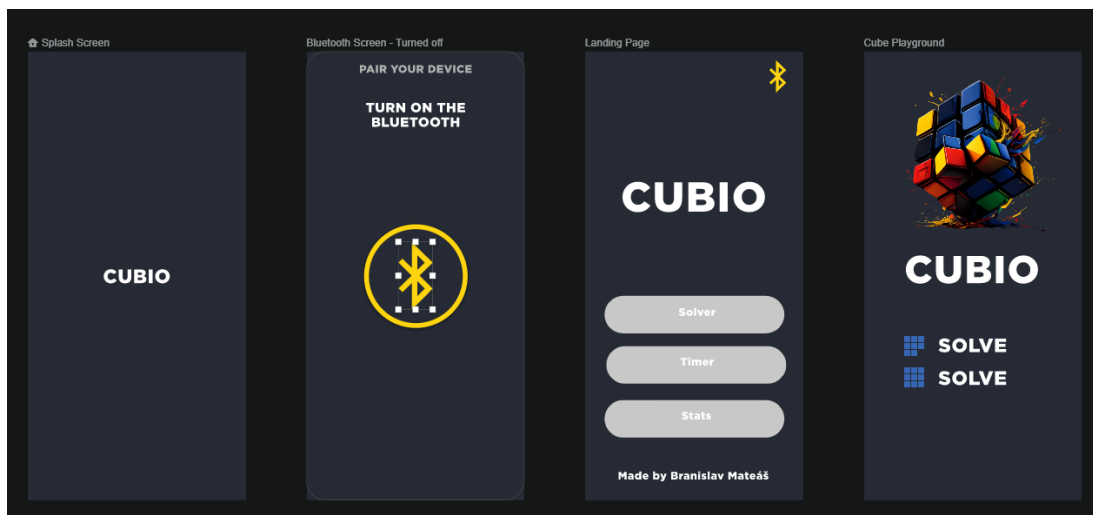
Kód z obrázku je riešením jedného zo špecifických problémov – prednačítanie nášho modelu, aby sme nemuseli zbytočne čakať po kliknutí na konkrétny. V opačnom prípade riskujeme znechutenie užívateľa. Kód z obrázku 4.5 sme teda vložili do kódu nášho menu, ale tak, aby nebol užívateľovi viditeľný. Týmto riešením sme ušetrili zhruba 5 sekúnd faktom, že sa model naštartuje už v menu, na pozadí.

Ďalší problém, ktorý by sme chceli riešiť je to nepekne sivé pozadie za našou kockou. K tejto veci existuje skvelý blog post priamo od autorov. Prvotne nám to síce nechcelo fungovať, ale neskôr sme zistili, kde je problém. Následné riešenie sme zdieľali medzi komunitu na adrese <https://github.com/juicycycleff/flutter-unity-view-widget/issues/190>.

4.6 Tvorba grafickej predlohy pre aplikáciu

Predtým, než sme začali tvoriť dizajn našej aplikácie, bolo dobre sa niečím inšpirovať. O to lepšie si pripraviť určité "nástrely" – odborne nazývane, Wireframe-y, toho, ako by približne mohla vyzeráť naša aplikácia.

Využili sme k tomu bezplatný dizajnový softvér nazývaný InVision Studio. Jedná sa prakticky o alternatívu nástroja Figma s tým, že je zadarmo a je taktiež kompatibilný so systémom Windows.




Obr.4.6: Náčrt vzhľadu našej aplikácie

Ako môžeme vidieť v porovnaní s výsledkom (príloha A, B, C), táto verzia bola skutočne iba náčrt a myslíme si, že finálna verzia urobila značný pokrok čo sa dizajnu týka. Špecificky by sme radi upozornili na funkciu, kedy sú farby tlačidiel v našej aplikácii generované náhodne, tak, že každý raz, čo sa aplikácia otvorí, bude farba menu tlačidiel, ale aj napríklad nájdených Bluetooth zariadení, iná. Navyše obrázok kocky, ktorý možno vidieť, je unikátny, vygenerovaný umelou inteligenciou Midjourney.

4.7 Mód Timer

Jeden z troch hlavných režimov, na ktorom budeme na tomto projekte pracovať, je režim Timer – časovač. Užívateľ zamieša kockou a následne potvrdí. Po nasledujúcom ťahu sa mu spúští čas, ktorý beží, až svoju kocku nevyrieši. Môže tento čas však resetovať vrátením sa o krok späť. Po úspešnom zložení kocky sa objaví okno oznamujúce úspešné zloženie kocky, po ktorého odkliknutí môžeme vidieť svoje štatistiky za toto zloženie – Počet otočení za sekundu, čas a počet pohybov, ktoré sme potrebovali na vyriešenie tohto hlavolamu. Vo vrchnej časti obrazovky je samozrejme kocka priebežne vizualizujúca vykonané ťahy.

Najprv sme začali s "medzistránkou". Tá obsahuje pokyny, čo sa bude diať. Čo by sme radi dali k pozornosti ohľadom tejto stránky je ako v celej aplikácii, hýbanie sa medzi jednotlivými stránkami za pomoci "Named Routes". V praxi to znamená, že každá stránka má svoj jedinečný identifikátor, vďaka ktorému sa vieme dostať na stránku z prakticky akéhokoľvek miesta. Pre lepšiu prehľadnosť si odporúčame vytvoriť externý súbor.



```
import 'package:cubio/components/bluetooth_list/bluetooth_list.dart';
import 'package:cubio/components/main_screen/connected_device/connected_device.dart';
import 'package:cubio/components/main_screen/main_screen.dart';
import 'package:cubio/components/main_screen/solver/solver_screen.dart';
import 'package:cubio/components/main_screen/stats/stats_screen.dart';
import 'package:cubio/components/main_screen/timer/timer_screen.dart';
import 'package:cubio/components/main_screen/timer/ui/timer_playground.dart';
import 'package:flutter/material.dart';

final Map<String, Widget Function(BuildContext)> routes = {
  '/': (context) => const BluetoothList(),
  '/menu': (context) => const MainScreen(),
  '/connected': (context) => const ConnectedDevice(),
  '/solver': (context) => const SolverScreen(),
  '/timer': (context) => const TimerScreen(),
  '/timer-playground': (context) => const TimerPlayground(),
  '/stats': (context) => const StatsScreen(),
};
```

Obr.4.7.1: Named Routes

Čo sa týka hlavnej stránky módu, ako sme už spomenuli, vložíme UnityWidget, ktorý po načítaní scény dostane pridelený kontrolér. Následne môžeme volať metódu doInitial, ktorá pošle príkaz Unity, aby vykonala metódu SetupInitial vysvetlenú vyššie (pre bližšie info o komunikácii odporúčam článok <https://stackoverflow.com/questions/65775363/flutter-unity-communication-issue>).



```
await state!.postMessage("Cube", "SetupPosition", stateReceived);
```

Obr.4.7.2: Posielanie správy Unity enginu

Dolnú polovicu tvoria už spomínané stopky. Počas vývoja sme sa u stopiek presunuli z pôvodného lokálneho manažmentu stavu na riverpod, aby čas, ktorý užívateľ potreboval, bol jednoduchšie prístupný aj napríklad sekcii pre štatistiky. Niektorí by si mysleli, že stopky sa obnovujú každú stotinu či tisícinu. To však nie je pravda. Flutter to však rieši prekresľovaním. Tak rýchlym, že my si to ani nevšimneme – v našom projekte to je 10 milisekúnd. Kebyže neprekresľujeme, videli by sme na displeji len prvotný stav našich stopiek.

Ešte k dialógovému oknu. Existuje veľa metód pre Flutter, ako na túto činnosť. My sme sa rozhodli siahnuť po šablóne, ktorú sme následne zveľaďovali, a to z webu <https://medium.flutterdevs.com/custom-dialog-in-flutter-7ca5c2a8d33a>. Jeho volanie závisí od toho, či je kocka vyriešená. Ako to odsledovať? Podobne ako som už spomínal, bude to za pomoci príkazu `ref.watch()`. Tento príkaz nastaví aktívne počúvadlo na stav nášho providera, a to konkrétne na getter `isSolved`. Treba si však dať pozor, build sa volá veľmi veľa krát, preto treba ošetriť, aby sa vždy otvorilo iba jedno okno. Treba pozorne sledovať dokumentáciu jednotlivých balíkov, pretože ak chceme využívať, je potrebné náš widget nastaviť na `ConsumerWidget` alebo `ConsumerStatefulWidget` (rovnaký rozdiel ako medzi `Stateless` a `Stateful` widgetom, akurát, že využívajúce riverpod - <https://stackoverflow.com/questions/5329618/stateless-vs-stateful>). Bacha na miznutie tohto okna. Je potrebné vložiť `"Navigator.of(context, rootNavigator: true).pop();"` pre korektné zatvorenie okna.

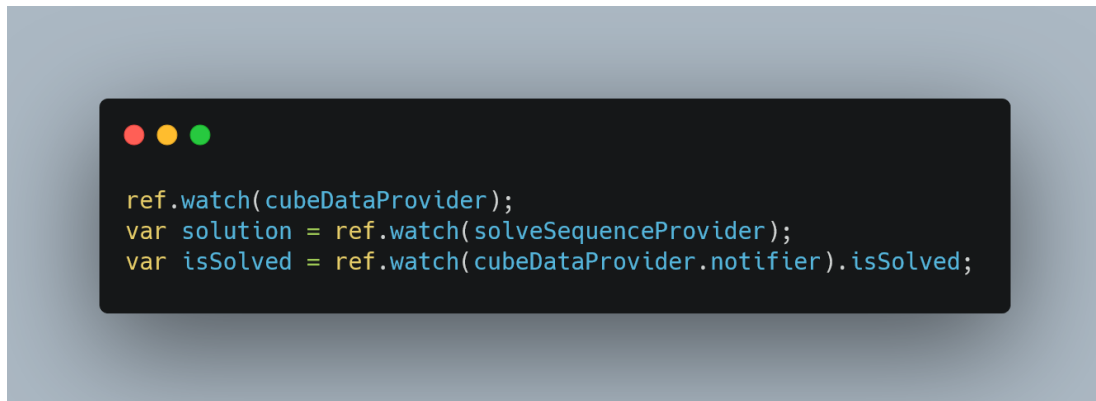
Výsledná implementácia tohto módu je znázornená na obrázku v prílohe C.

4.8 Múd Solver

Druhým z módov bude Solver – mód pre vyriešenie Rubikovej kocky. Po skupine ťahov a pauze aspoň pol sekundy nám Unity model pošle sekvenciu k vyriešeniu. Spodnú sekciu tvoria dva údaje – posledný ťah a práve spomínanú sekvenciu. Tú vykresľujeme za pomoci balíka `carousel_slider` (https://pub.dev/packages/carousel_slider).

Nastavenie je značne jednoduchšie než pri prvom režime. Po nainštalovaní balíka sa môžeme odraziť od príkladu v dokumentácii. Čo budeme nastavovať budú porcia položky vzhľadom k zariadeniu, pomer strán a minimálna výška. S nadobudnutými znalosťami už zrejme vieme ako sa nám podarilo dostať k priebežnej hodnote posledného ťahu. Treba vytvoriť nový provider špecificky pre sekvenciu ťahov vzhľadom na fakt, že tá bude chodiť v inakších

intervaloch ako ťahy kocky, kebyže tento krok nepodnikneme, naše riešenia budú o krok meškať. Samozrejmosťou je potreba nastaviť obidva listenery na `ref.watch()`. Vhľadom k tomu ale, že ešte sledujeme, či kocka je vyriešená, opäť treba počúvať na hodnotu `isSolved`.



Obr.4.8.1: Listenery v móde Solver

Tento režim je však primárne o Unity implementácii. Solve nie je funkcia s konštantnou rýchlosťou, preto sa nám dlho stávalo, že sa kocka sekla sama do seba ako autorovi na obrázku 4.4.3. Po dlhom skúšaní sme však prišli na správnu implementáciu. A to čakať, kým sa aspoň pol sekundy nevykoná žiaden iný ťah. Pol sekundy nie je fixných, vieme to meniť za pomoci premennej, avšak hodnota pol sekundy vyzerá najstabilnejšie.



Obr.4.8.2: Implementácia kombinácie Solve a DoMove v metóde Update


```

private void Solve()
{
    // Vymedzeju opätovné volanie ďalšej inštancie Solve
    CubeState.doSolve = false;

    // Prečíta aktuálny stav kocky (vyšle "lasery")
    readCube.ReadState();

    // Prekonvertuje aktuálne farby na String reprezentujúci stav
    string moveString = cubeState.GetStateString();

    // Vyhľadávanie riešenia za pomoci Kociemba algoritmu
    string info = "";
    string solution = Search.solution(moveString, out info);

    // Pohyby oddeľuje medzerou
    string optimizedSolution = string.Join(" ", OptimizeSolution(solution));

    // Posiela riešenie Flutteru
    Manager.SendMessageToFlutter(optimizedSolution);

    // Reštart premennej sledujúcej čas od posledného pohybu
    secondsWithoutMove = 0f;
}

```

Obr.4.8.3: Výsledná metódy Solve

Podobne ako inde, aj tento mód možno nájsť – obrázok vpravo v prílohe B.

4.9 Mód Stats

Ako naša anotácia ukazovala, naším cieľom je užívateľovi ukázať jeho štatistiky. Budú nás zaujímať štatistiky počtu ťahov riešiteľa, jeho priemerný počet ťahov za jednu sekundu a logický čas, za ktorý kocku vyriešil. Vytvorili sme si preto v projekte tri providery, ktoré budú držať tieto hodnoty (je ale pravdou, že stopwatch_provider neadrží len hodnotu, ale organizuje celé jeho správanie).

Na tvorbu grafov budeme využívať Flutter package `fl_chart` (https://pub.dev/packages/fl_chart), ktorý ponúka širokú škálu najrôznejších typov grafov, s nimi úzko spojenými funkcie a dizajnové možnosti viac než dostačujúce pre náš účel. Navyše ponúka aj šablóny už vopred vytvorených grafov. Po jednom z nich sme siahli aj my.

Je potrebné však aj získané dáta pre graf ukladať. Na to použijeme zasa plugin `Shared_preferences`, slúžiaci na ukladanie jednoduchých dát do úložiska nášho zariadenia (https://pub.dev/packages/shared_preferences).

Implementácia samotných grafov je taktiež veľmi jednoduchá. Pokiaľ chceme vykresľovať body na grafe, stačí upraviť len časť textu zobrazeného na obrázku 4.9.1.:



```
...
lineBarsData: [
  LineChartBarData(
    spots: const [
      FlSpot(0, 3),
      FlSpot(2.6, 2),
      FlSpot(4.9, 5),
      FlSpot(6.8, 3.1),
      FlSpot(8, 4),
      FlSpot(9.5, 3),
      FlSpot(11, 4),
    ],
    isCurved: true,
  ),
],
...

```

Obr.4.9.1: Kód zodpovedný za vykresľovanie bodov na grafe

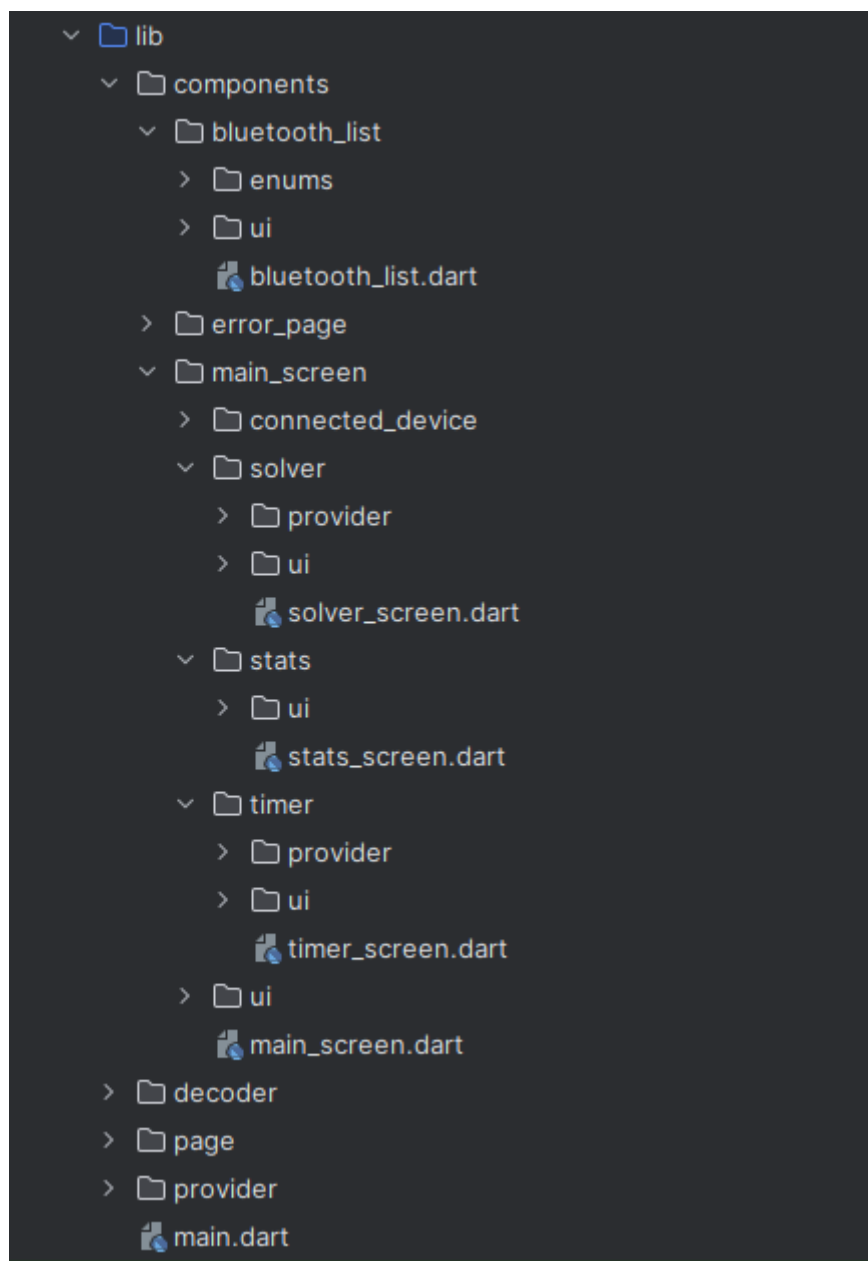
V našom prípade sme to poňali ako argument triedy `LineChart`, ktorá sme premenovali z pôvodného `LineChartSample2`. Dôvod využiť argument je prostý. Môžeme použiť jednu triedu na všetky potrebné grafy bez nutnosti duplikácie. V takomto prípade sme ale boli nútení pridať ešte ďalšie povinné argumenty ako `MinX`, `MaxX`, `MinY`, `MaxY`, `gradientColors`...

Spomenuli sme `gradientColors`. Tento balíček nám taktiež ponúka možnosť vytvárať grafy s farebnými prechodmi – gradientmi. Pre nováčikov v tomto jazyku by sme však chceli poukázať na jeden jav. Pokiaľ máme HEX kód farby a nevieme ako ju prekonvertovať do "Flutter tvaru", stačí umiestniť pred tento kód prefix `0xff`, pričom nezáleží, či danú vec napíšeme malými alebo veľkými písmenami.



Obr.4.9.2: Kód zodpovedný za farby pozadia grafov

Kedy ale budeme ukladať dáta? Rozhodli sme sa, že túto činnosť vykonáme vtedy, keď nám vyskočí dialóg oznamujúci nám úspešné vyriešenie našej kocky. Je však dôležité pamätať na fakt, že nemožno ukladať len poslednú nadobudnutú hodnotu, ale najprv je nutné získať už uložené dáta vo forme `List<String>`, tento zoznam následne rozšíriť o náš novú položku a potom ju až následne uložiť do pamäti. Záverečný výsledok našej implementácie týchto grafov možno vidieť na obrázku v prílohe C.



Obr.4.9.3: Záverečná priečinková štruktúra projektu

Záver

Ako už bolo spomenuté, cieľom tohto projektu bolo rozanalyzovať Bluetooth komunikáciu dostupných inteligentných Rubikových kociek a vytvoriť vlastnú aplikáciu na ich ovládanie. Cieľ sa podarilo splniť a výsledkom je aplikácia pre systém Android, ktorá ponúka rôzne funkcie vrátane vizualizácie a označovania pohybu, merania času, s možnosťou ukladať a zobrazovať výsledky.

Množstvo detailov a funkcií sa do tejto práce nepodarilo vložiť a opísať, preto v prípade záujmu je k práci dodávaný aj zdrojový kód.

V budúcnosti chceme, aby bola aplikácia zverejnená aj na obchode Play, čím získame väčší feedback a komunitnú podporu projektu. Je však stále čo zlepšovať. Napríklad implementácia kocky s gyroskopom - jej komunikácia je šifrovaná a jej pohyb si vyžaduje neustále prekresľovanie pozície 3D modelu kocky v reálnom čase. Vzhľadom k časovej tiesni sme ale túto funkciu nestihli implementovať.

Projekt každopádne zdôraznil význam zručností riešenia problémov a kritického myslenia, ako aj integrácie technológií do každodenného života. Vlastná aplikácia bude slúžiť ako cenný nástroj pre každého, kto má záujem o riešenie Rubikovej kocky, pričom ukazuje potenciál využitia technológií na zlepšenie tradičných skúseností s riešením hlavolamov.

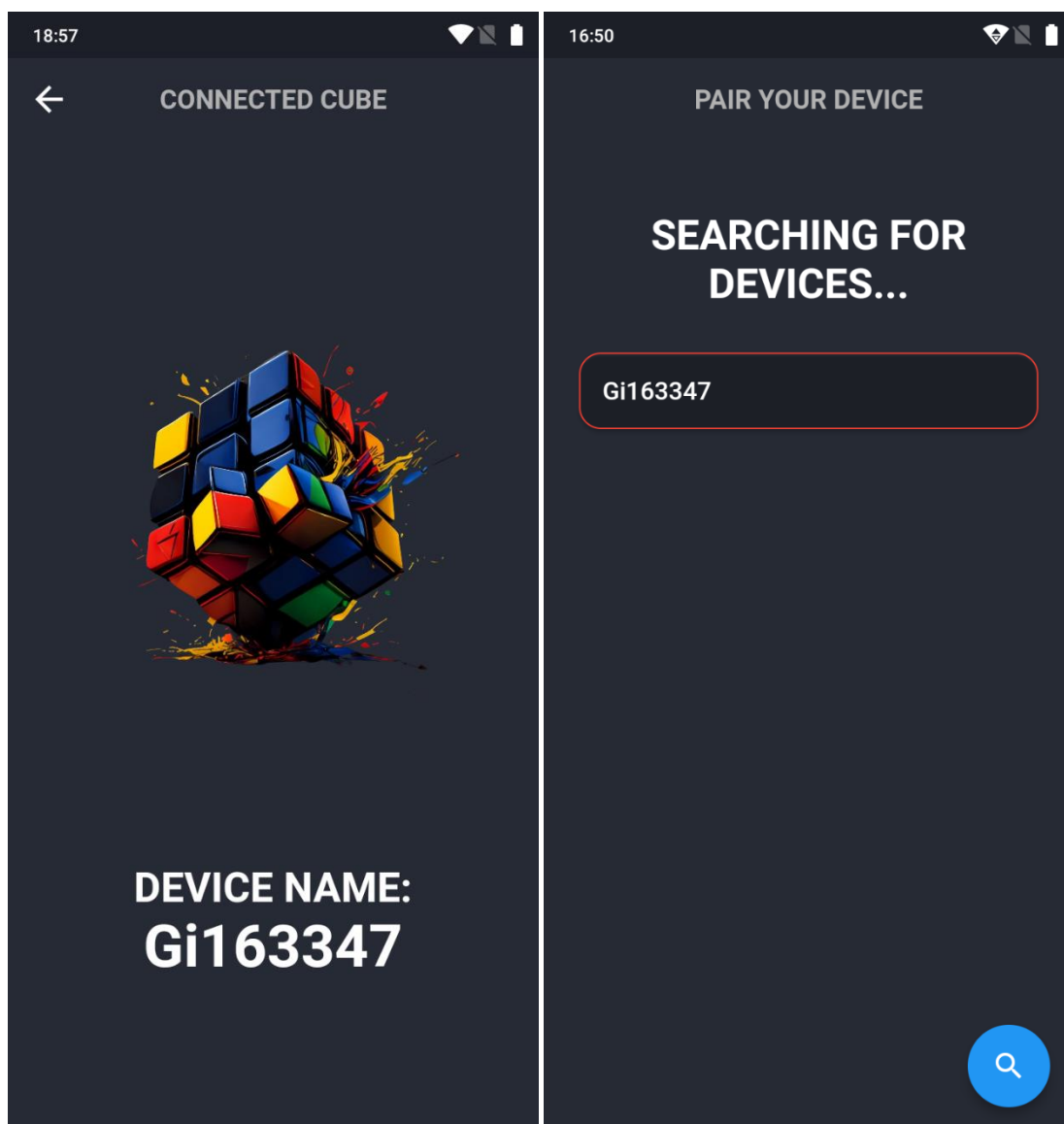
Okrem toho má dekodovaná komunikácia použitá v projekte potenciál aj na budúce využitie v iných projektoch, napríklad možnosť ovládania iných zariadení pomocou Rubikovej kocky. To poukazuje na dôležitosť skúmania a dekódovania komunikačných protokolov a otvára nové možnosti inovácie a vývoja.

Použitá literatúra

- [1] https://sk.wikipedia.org/wiki/Rubikova_kocka
- [2] <https://magazin.lionline.sk/rubikova-kocka/>
- [3] [https://en.wikipedia.org/wiki/Flutter_\(software\)](https://en.wikipedia.org/wiki/Flutter_(software))
- [4] <https://flutter.dev/>
- [5] [https://en.wikipedia.org/wiki/Dart_\(programming_language\)](https://en.wikipedia.org/wiki/Dart_(programming_language))
- [6] <https://mindmajix.com/dart-vs-javascript>
- [7] <https://sk.wikipedia.org/wiki/Bluetooth>
- [8] www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/
- [9] <https://www.appbrain.com/stats/number-of-android-apps>
- [10] <https://www.bleepingcomputer.com/news/security/trojanized-dns-py-app-drops-malware-cocktail-on-researchers-devs/>

Príloha A

Fotky z výslednej aplikácie – časť 1.



Príloha B

Fotky z výslednej aplikácie – časť 2.



Príloha C

Fotky z výslednej aplikácie – časť 3.

