Maturitní Práce

Informatika

Aplikace pro pomoc při memorizaci šachových zahájení

Martin Vanko
4.C
Gymnázium Jana Keplera

2025

Obsah

1 Úvod **2**

2 Použité technologie **2**

3 Teoretická část **4**
3.1 Pseudo_move_generation.h 4
3.2 JNI 8
3.3 Databáze 9
3.3.1 .SCHEMA 9
3.4 Aktivity 9
3.4.1 SplashActivity 10
3.4.2 MainActivity 10
3.4.3 OpeningsRoot 11
3.4.4 Openings 11
3.5 Třídy 11
3.5.1 ChessBoard 11
3.5.2 ChessPiece 13
3.5.3 GreenSquareFactory 13
3.5.4 SoundPlayer 14

4 Návod k použití **14**

5 Ukázka z aplikace **16**

6 Závěr **17**

1 Úvod

Abstract Vytvoření androidové aplikace, prozatím pouze pro mě, jejíž účel je pomoc při procesu memorizace šachových zahájení.

Proč? Šachy jsem začal hrát v roce 2021. Začal jsem být toutu legendární hrou docela fascinovaný hned ze začátku, byla to aktivita, ve které jsem viděl smysl. Nedělalo mi problém řešit tisíce šachových úloh, či čtení všemožných knih a studování partií starých mistrů, ale učení se zahájení. Všechny dostupné nástroje jsou určené k učení se ideí za tahy, ale neexistuje nástroj který by mi pomohl se samotnou memorizací zahájení.

2 Použité technologie

Většina kódu samotné aplikace byla napsána v Kotlinu. funkce na generování pseudo legálních tahů je napsaná v jazyku C. Queries do Sqlite3 databáze jsou napsané v SQL. Na popsání statické části bylo použito XML. Různé prvky např. šachovnice byly napsány v SVG formátu, což je taky druh XML. Jako editor kódu bylo využito Android Studio od společnosti JetBrains a Visual Studio Code od společnosti Microsoft. Pro propojení generátoru tahů bylo použito JNI. Pro vytvoření obsahu databáze byl využit pythonový script.

XML XML (*eXtensible Markup Language*) je markupový jazyk a zároveň souborový formát určený pro strukturované ukládání a přenos dat. V aplikaci jsou layouty jednotlivých aktivit popsané v XML. Výhodou XML je čitelnost nejen pro počítač, ale především pro programátora.

Kotlin Kotlin, podle ruského ostrava Kotlin, je moderní staticky typovaný programovací jazyk vyvinutý společností JetBrains, který běží na JVM (Java Virtual Machine) a je plně interoperabilní s Javou. Byl vybrán pro jeho čitelnost a preferovanost oproti Javě.

C C je nízkourovňový procedurální programovací jazyk vytvořený Dennisem Ritchiem v Bell Labs v 70. letech 20. století. Tento jazyk byl vybrán pro jednoduchost práce s pamětí a bitovou manipulací. Pro efektivní generování tahů byl potřeba precizní nástroj.

SQL SQL, nezkráceně *Structured Query Language*, je deklarativní jazyk pro správu relačních databází (jako SQLite3), standardizovaný organizací ISO/IEC. Byl využit pro psaní dotazů do databáze.

SVG SVG (*Scalable Vector Graphics*), je vektorový grafický formát založený na XML, specifikovaný W3C. Na začátku bylo velkou otázkou, jak uložit sprity, tlačítka, šachovnici atd. a nakonec byl vybrán formát SVG, protože je naprosto ideální. Aplikace pracuje s jednoduchými tvary a díky SVG se vyhne rozpixelovaným spritům. zároveň jsou SVG soubory velmi modulární, například co se jednoduchosti změny barvy týče. Nějaké obrázky se nepovedlo najít na internetu, musely tak být napsány ručně. Proto je SVG popsána jako zvolená technologie.

JNI JNI, nezkráceně Java Native Interface je rozhraní, které umožňuje volat nativní funkce (funkce v C, C++). Používá JNIEnv* pro komunikaci s JVM. Bylo využito pro komunikaci mezi Pseudo_move_generation.h a zbytekem aplikace. Základní JNI metoda vypadá takto:

```
JNIEXPORT void JNICALL Java_HelloWorld_printHelloWorld(JNIEnv *env,
    jobject obj) {
    printf("Hello, World!\n");
}
```

Tato funkce je bez argumentu, nic nevrátí a vytiskne do standardního výstupního proudu "Hello, world!".

Python Python je vysokoúrovňový, interpretovaný programovací jazyk, který je známý svou jednoduchostí a čitelností. Byl vytvořen Guido van Rossum v roce 1991. Při vývoji aplikace byl využit pro psaní scriptů při tvoření obsahu databáze. Zahájení byly ve formátu pgn a pomocí scriptu byly převedy na pole FEN řetězců.

Android Studio Android Studio je integrované vývojové prostředí (IDE) od JetBrains určené pro tvorbu Androidových aplikací. Tato IDE byla pro vývoj vskutku esenciální. Velmi důležitý byl editor layoutů, který ukazoval v reálném čase provedené změny. Android Studio ukládá automaticky lokální historii projektu, což přišlo, ne jednou, velmi vhod.

Visual Studio Code Visual Studio Code (VS Code) je multiplatformový textový editor od Microsoftu. Je docela lightweight. Dost tedy záleží na pohledu, oproti Android Studio je velmi lightweight, ale ve srovnání s Vimem určitě ne. Byl použit výhradně pro napsání souboru pseudo_move_generation.h. Potom na vyzkoušení několika funkcí v Kotlinu a psaní scriptů v pythonu.

3 Teoretická část

3.1 Pseudo_move_generation.h

Existuje spousta způsobů, jak šachovnici reprezentovat. Mohlo by být 32 figurek reprezentovaných pomocí barvy, typu a příslušné souřadnice na šachovnici. To je ale dosti neefektivní a, zavisící na implementaci, i dosti prostorově náročné. Problém tedy je jak, co nejefektivněji, výše zmíněné údaje reprezentovat. Pro řešení tohoto problému bylo zvoleno dvanáct šedesáti čtyř bitových slov. Každý typ a barva figurky dostane vlastní slovo.

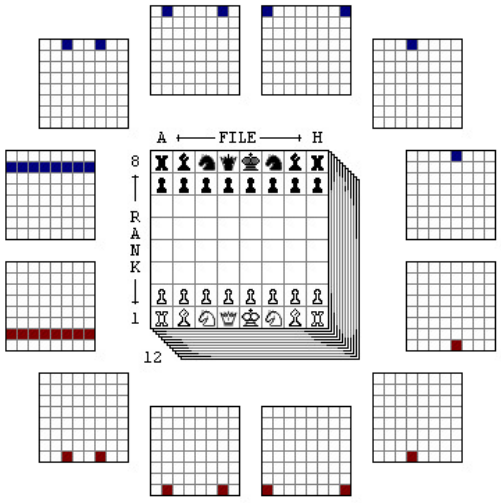

Figure 1: Bitboard

Pozice samotných figurek bude značená setnutými bity. A políčko a8 bude první bit a políčko h1 bude šedesátý čtvrtý bit. Pro samotnou reprezentaci slov byl využit datový typ unsigned long long zkráceně ULL. V kódu:

```
typedef uint64_t Bitboard;
```

Protože unsigned long long nezaručí vždy 64 bitů. Kromě toho, že je to velmi prostorově úsporné, tak to zároveň umožní jednoduše provádět na slovech bitové operace.

Ukládání tahů Potom bylo potřeba uložit tahy. Jeden tah se dá popsat políčkem odkud, kam a potom značkou pro speciální tahy např. en passant nebo promoce.

```
/*
0   0000   quiet moves
1   0001  double pawn push
2   0010   king castle
3   0011   queen castle
4   0100   captures
5   0101   ep-capture
8   1000   knight-promotion
9   1001   bishop-promotion
10  1010   rook-promotion
11  1011   queen-promotion
12  1100   knight-promo capture
13  1101   bishop-promo capture
14  1110   rook-promo capture
15  1111   queen-promo capture
*/
```

2^6 je 64 tedy potřebujeme 6 bitů na reprezentaci políčka odkud a 6 bitů na reprezentaci políčka kam. Potom nám zbývají 4 bity na reprezentaci typu tahu.

FEN (Forsyth-Edwards Notation) definuje pozici tak, že řádek po řádku říká jaká figurka je na kterém políčku. Začíná na a8 a jde na h1. Čísla značí prádná políčka. Například: `starting_position_fen`:

```
"rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
```

Čte se zleva doprava. A malé písmena jsou černé figurky a velké písmena bílé. Na konci stringu následuje ukazatel toho, kdo je na tahu. Potom práva na rošády, zase malé jsou černé figurky a naopak. Pomlčka je na místě pro možné en passat políčko. Předposlední znak je počet polovičních tahů od posledního pohybu pěšcem nebo sebrání figurky. Poslední znak je počet tahů, začíná na 1 a inkrementuje se po tahu černého.

Konstanty Byla potřeba spousta konstant. Nejdůležitější jsou asi bitboardy typu: `not_a_file`, `not_h_file`, `not_eigth_rank` atd. neboť ty zaručují, že figurka nepřeteče na druhou stranu šachovnice.

```
//0 1 1 1 1 1 1 1
//0 1 1 1 1 1 1 1
//0 1 1 1 1 1 1 1
//0 1 1 1 1 1 1 1
//0 1 1 1 1 1 1 1
//0 1 1 1 1 1 1 1
//0 1 1 1 1 1 1 1
//0 1 1 1 1 1 1 1
const Bitboard not_a_file = 0xfefefefefefefefeull;
//1 0 1 1 1 1 1 1
//1 0 1 1 1 1 1 1
//1 0 1 1 1 1 1 1
//1 0 1 1 1 1 1 1
//1 0 1 1 1 1 1 1
//1 0 1 1 1 1 1 1
//1 0 1 1 1 1 1 1
//1 0 1 1 1 1 1 1
const Bitboard not_b_file = 0xFDFDFDFDFDFDFDFDull;
Bitboard not_ab_file = ~(~not_a_file|~not_b_file);
Bitboard eastSoEa = ((knight << 10)&not_ab_file)&~ friendly_pieces;
```

Zabrání koni aby v tomto příkladě skončil na a1, což by byl ilegální tah.

```
//0 0 0 0 0 0 0 0    //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 0 0    //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 0 0    //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 0 0    //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 0 0 -> //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 1 0    //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 0 0    //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 0 0    //1 0 0 0 0 0 0 0
               Illegal

//0 0 0 0 0 0 0 0    //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 0 0    //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 0 0    //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 0 0    //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 0 0 -> //0 0 0 0 0 0 0 0
//0 0 0 0 0 1 0 0    //0 0 0 0 0 0 0 0
//0 0 0 0 0 0 0 0    //0 0 0 0 0 0 0 1
//0 0 0 0 0 0 0 0    //0 0 0 0 0 0 0 0
                Legal
```

Globální proměnné

- Bitboardy pro jednotlivé figurky.

```
Bitboard white_king = 0ULL;
Bitboard white_queen = 0ULL;
Bitboard white_rook = 0ULL;
           ...
           ...
           ...
```

  Bitboardy začínají na nule a budou nastaveny, až se načte FEN.

- Buffer pro ukládání tahů a počítadlo tahů:

```
    Move pseudo_moves[512];
    short pseudo_moves_counter = -1;
```

- Informace o Pozici:

```
    char turn = 0;
    unsigned char castling_rights = 0;
    Bitboard en_passant_square = 0ull;
```

Makra pro manipulaci s bity

- `bitset`, `bitclear`, `bitflip`, `bitcheck`: Operace s jednotlivými bity.

```
#define bitset(bitboard, nbit) ((bitboard) |= (1ull<<(nbit)))
                                ...
```

Definice směrů a typů tahů

- Směry pohybu. Pro jednoduchost byly očíslovány světové strany a jejich diagonální kombinace.
- Typy tahů (flagy): `QUIET_MOVES` (standardní tah), `CAPTURES` (sebrání figurky), `KING_CASTLE` (královská rošáda), `EP_CAPTURE` (en passant), atd.

Hlavní funkce pseudo_moves_generator(char* fen) nejdříve načte FEN notaci a nastaví tak jednotlivé bitboardy. Potom vytvoří:

```
Bitboard whitePieces =
    white_king|white_queen|white_rook|white_knight|white_bishop|white_pawn;
```

Pro kontrolu jestli, bílá figurka nebere druhou bílou figurku a vice versa. Což by bylo ilegální. Potom byly vytvořeny funkce pro generování tahů jednotlivých figurkek.

```
static Bitboard pseudo_king_moves(Bitboard king, Bitboard
    friendly_pieces, Bitboard opponent_pieces);
                                    ...
```

Hlavní funkce projde skrze všechny bitboardy, vytvoří tahy a přidá je do bufferu. Nakonec se tahy překopírují z bufferu do pole, kde je první element počet tahů.

3.2 JNI

V srdci celé aplikace leží komunikace mezi knihovou v C a JVM. JNI je zkratka pro Java Native Interface, neboli způsob jak volat nativní kód z Javy. Nejdříve se definuje funkce jstringToChar, která převede Java String na pole charakterů. V Javě jsou stringy objekty, nejsou to jen pole charakterů jako v C.
Potom se definuje samotná funkce getPseudoMoves(). Argumenty jsou

- Java Native Environment Pointer
- jobject reprezentuje objekt, který tu funkci volá
- fenString je samotný FEN argument.

Potom je zavolána samotná funkce pseudo_moves_generator(fen), vytvoří javaArray shortů, vrátí paměť a metoda vrátí pole tahů.

3.3 Databáze

Pro ukládání dat byla zvolena SQLite3 databáze. Pro účel aplikace by určitě bohatě postačil textový soubor, co se optimalizace týče. SQL je ale praktické a do budoucna lehce škálovatelné.

Pro komunikaci s databází byl zvolen balíček android.database.sqlite.SQLiteOpenHelper a byla napsána třída, která extenduje SQLiteOpenHelper. Potom byly napsány jednotlivé funkce na exekutování dotazů.

3.3.1 .SCHEMA

Samotné schema databáze vypadá takto:

```
CREATE TABLE openings (id INTEGER PRIMARY KEY
    AUTOINCREMENT,opening_name TEXT UNIQUE,fen_array TEXT);
CREATE TABLE sqlite_sequence(name,seq);
CREATE TABLE settings (chessboard TEXT, legal_moves INT, sound_effects
    INT);
CREATE TABLE last_played (table_name TEXT, color INT, name TEXT);
CREATE TABLE streak (streak INT, last_timestamp INT);
```

3.4 Aktivity

Každá aktivita je jedno okno, co uvidí uživatel. Má vlastní jménot_aktivity_layout.xml soubor, kde je definovaný její layout. Potom má každá aktivita svůj soubor JménoAktivity.kt, což je třída, která dědí ze super třídy AppCompatActivity.

```
    class MainActivity : AppCompatActivity()
```

V zdrojovém kódu aktivita jsou overridůté různé metody z rodičovské třídy AppCompatActivity. Nejdůležitější taková metoda je:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
}
```

Kde argument savedInstanceState je poslední stav aktivity. Další důležitou metodou by byla metoda finish(), díky které má aplikace skvělé animace.

3.4.1 SplashActivity

To je aktivita, která v sobě skrývá pouze úvodní animaci. Otevře se jako první, provede animaci popsanou níže. Kód logo_enter.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true">

    <!-- Fade in effect -->
    <alpha
        android:duration="500"
        android:fromAlpha="0.0"
        android:toAlpha="0.6" />

    <!-- Scale up effect -->
    <scale
        android:duration="500"
        android:fromXScale="0.5"
        android:fromYScale="0.5"
        android:toXScale="0.9"
        android:toYScale="0.9"
        android:pivotX="50%"
        android:pivotY="50%" />
</set>
```

3.4.2 MainActivity

Každá aplikace potřebuje nějakou hlavní aktivitu. Jedná se o první aktivitu, který se otevře. Shodou okolností první aktivita aplikace je SplashActivity, ta se ale úplně nepočítá. Hlavní aktivita aplikace obsahuje pouze komunikaci s databází a dále slouží jako rozcestník. Jediný zajímavý kus kódu je asi práce s ukazatelem streaku.

Streak Streak vždy začíná jedničkou. Resetuje se, pokud rozdíl posledního zapsání streaku a nejnovějšího otevření aplikace je více než dva dny. Pokud se rozdíl dnů stane záporným, také se resetuje zpět na jedničku.
Asi poslední zajímavá věc je, jak se mění obrázek plamene s rostoucím streakem. Byla vytvořena funkce:

```
changeSvgFillColor(streak, R.drawable.streak_flame, 255,255,255, (255 *
    (1 - exp((-0.05* currentStreak)))).toInt())
```

Která bere jako argument instanci ImageView objektu, sprite objektu, hodnoty RGB a aplha. Nebude se měnit barva plamene z estetických důvodů a bude se měnit pouze alpha. Alpha říká, jak je ImageView průhledný a argument může být z otevřeného intervalu celých čísel $(0, 255)$. Byla potřeba tedy posloupnost, která bude mít vlastní limitu v nevlastním bodě $\lim_{n\to\infty} a_n = 255$, zvolena byla posloupnost $a_n = 255 \times (1 - e^{-k\times n})$, kde k je parametr určující, jak rychle funkce poroste a $n = currentStreak$. $k \in \mathbb{R}$, ale ideální hodnoty pro účel aplikace jsou z intervalu $(0, 1)$.

3.4.3 OpeningsRoot

Toto je rozcestník pro výběr zahájení. V main activity jsou dvě ImageButtons, jedno pro zahájení bílého a jedo pro zahájení černého, s kódem:

```
val whiteOpenings = findViewById<ImageButton>(R.id.white)
       whiteOpenings.setOnClickListener {
           val intent = Intent(this, OpeningsRoot::class.java)
           intent.putExtra("color", 0)
           this.startActivity(intent)
           finish()
       }
```

Kód říká, že po kliknutí bude otevřen OpeningsRoot aktivita a přidá se extra informace o barvě. Dále se vytváří ImageButton pro každé zahájení. A kdyby náhodou zahájení nemělo jméno nebo obrázek, tak se použije základní text a obrázek.

3.4.4 Openings

Zde se kreslí figurky a šachovnice. Figurky se kreslí zvlášť, dle vybrané barvy. Stejně jako OpeningsRoot přišel s nějakou informací navíc, tak zde je k dispozici barva hráče, jméno zahájení v databázi a jméno zahájení. Takto se vypočítá začátek šachovnice:

```
val screenRatio = screenWidth.toFloat()/screenHeight.toFloat()
val chessB\endoardStartY= round((screenHeight/(screenRatio*7)))
```

Což nebude fungovat na hodně čtvercových displejích, ale pro ty se bude muset stejně vytvořit samostatný layout.

3.5 Třídy

3.5.1 ChessBoard

Třída `ChessBoard` se využívá jako komunikátor mezi figurkami a zbytkem aplikace. Každá figurka dostane jako argument instanci šachovnice a to samé většina dalších

tříd.

Vlastnictví

- `val squareCoordinates: Array<IntArray>`: Je matice všech 64 polí a jejich souřadnic na obrazovce.
- `val squareSize:Int`: Velikost jednoho políčka v pixelech.
- `var fen: String`: Zápis aktuální pozice v FEN viz. výše.
- `var pieces: MutableList<ImageView>`: List `ImageView` objektů, které jsou využity pro reprezentaci figurek.
- `var turn: Boolean`: Určuje, zda je na tahu bílý (`true`), nebo černý (`false`).
- `var plyCounter: Int`: Jedno ply je půl tah. Pokud bílý zahraje v prvním tahu e4, tak ply bude 1.
- `castling rights`:

```
var whiteKingsideCastle = true
          ...
```

  Práva na rošádu a jejich permutace.
- `var enPassantSquare = false`: Říka souřadnice pole, kam je možný tah en passant.

Metody

- `generateFen(color: Int): String`
  Vytvoří FEN z pole pieces a vrátí ho jako String.
- `getPlayedMove(fen1: String, fen2: String): IntArray`
  Detekuje provedený tah mezi dvěma FEN řetězci. Používá se na zobrazení hintu. Vrací pole integerů`[odkud, kam]`.
- `getPlayedMoveAlgebraicWithPiece(fen1: String, fen2: String): String`
  Detekuje provedený tah mezi dvěmi FEN pozicemi. Vrací namísto [36, 53] e4 pro tah pěšec z e2 na e4.
- `pseudoLegalMoves(fen: String): MutableList<Move>`
  Získává pseudolegální tahy pomocí nativní funkce `getPseudoMoves`. Pro rošádu a speciální tahy jsou využity flagy viz. výše.
- `drawPieces(...)`
  Vykreslí figurky na šachovnici podle FEN zápisu. Pro obě strany zvlášť.

3.5.2 ChessPiece

Třída `ChessPiece` popisuje jednotlivé figurky, především jejich vizuální stránku, logika je ošetřena jinde. Asi nejdůležitější částí je logika při eventu kliknutí na jednotlivé figurky.

Vlastnictví

- `ID`: Každá figurka má vlastní ID, které odpovídá číselným zdrojům spritů figurek (např. `R.drawable.black_king`).
- `imageView`: Samotný obrázek, který je vidět na obrazovce. Toto je ImageView, které je v poli ChessBoard.pieces.
- `chessBoard`: Každá figurka má jako argument instanci šachovnice, aby si mohla sáhnout například na souřadnice políček nebo další atributy.

Metody

- `drawPiece(x: Float, y: Float, width: Int, height: Int): ImageView`
  Vytvoří `ImageView` na zadaných souřadnicích a nastaví její výšku a šířku.
- `makePieceClickable()`
  Nastaví klikací události pro figurku. Určitě se jedná o největší monstrozitu tohoto projektu:
  - Po kliknutí se zvýrazní políčko zeleným čtvercem. Pokud to nastavení dovoluje.
  - Zobrazí dostupné tahy pomocí `GreenSquareFactory`. Pokud to nastavení dovoluje.
  - Když uživatel zmáčkne políčko, tak se zkontroluje, zda je v legálních tazích. Potom se aktualizuje pozice a FEN v instanci ChessBoard.
  - Zvukové efekty pomocí třídy SoundPlayer.

3.5.3 GreenSquareFactory

Třída `GreenSquareFactory` tvoří vizuální efekty pro znázornění legálních tahů, označení hintu, správných a špatných tahů.

Vlastnictví

- `greenSquareList`: Seznam `ImageView` prvků reprezentujících vizuální efekty (tečky, kroužky, čtverce).
- `chessBoard`: Odkaz na instanci šachovnice pro přístup k souřadnicím políček.

Metody

- `addHollowCircle(squareNumber: Int)`
  Přidá průhledný zelený kroužek na políčko s daným indexem. Používá se pro označení možného sebrání figurky. Argument je číslo políčka, pomocí ChessBoard se zjistí souřadnice.

- `addDot(squareNumber: Int)`
  Přidá zelenou tečku na políčko, kde je pseudo legální tah. Zobrazuje se pouze pokud je v nastavení povoleno zobrazování tahů. Argument je číslo políčka, pomocí instance ChessBoard se zjistí souřadnice.

- `addRedSquare(coordinates: FloatArray)`
  Vytvoří červený čtverec na zadaných souřadnicích. Byla přidaná, aby bylo jak dát uživateli najevo, že zahájení pokazil. Čtverec postupně mizí pomocí animace `fadeOut`.

- `addGreenSquare(squareNumber: Int)`
  Využívá se pro zvýraznění vybrané figurky nebo hintu. Argument je číslo políčka, pomocí instance ChessBoard se zjistí souřadnice.

- `removeSquares()`
  Odstraní všechny aktivní vizuální efekty z layoutu a vyčistí `greenSquareList`.

3.5.4 SoundPlayer

Třída `SoundPlayer` slouží k přehrávání zvukových efektů. Používá balíček android.media.SoundPool. Dále byly vytvořeny metody pro přehrávání jednotlivých efektů.

Hlavní atributy

- `soundPool`: Instance `SoundPool` přehrává zvuky.

- `dbHandler`: Odkaz na databázi pro kontrolu nastavení zvukových efektů.

4 Návod k použití

Instalace Jsou dva způsoby jak si momentálně aplikaci nainstalovat. Uživatel si může aplikaci sám postavit, nebo ji stáhnout nejnovější release z githubu s url: https://github.com/Martin49232/chessProject/releases.

- `AndroidStudio` Uživatel si musí stáhnout zde https://developer.android.com/studio/. Po instalaci uživatel klikne na New Project a vybere možnost clone repository. Zadá url https://github.com/Martin49232/chessProject a vytvoří projekt. Uživatel potom bude muset připojit své androidové zařízení k počítači pomocí developer options v nastavení. Druhá možnost je spustit emulátor a zařízení emulovat.

- `Github` Uživatel si bude potřebovat stáhnout openingo.apk do počítače a následně si ji přetáhnout do telefonu, nebo si ji stáhnout z Githubu do telefonu rovnou z url //github.com/Martin49232/chessProject/releases. Poté bude potřebovat package installer. Ten aplikaci nainstaluje.

Použití Po nainstalování zapněte aplikaci. Otevře se hlavní okno a uživatel dostane na výběr ze zahájení pro černého a bílého, dále může otevřít nastavení nebo zmáčknout tlačítko hodin, které otevře poseldní zahrané zahájení. Po vybrání strany se uživatel dostane do okna pro výběr samotného zahájení. Po vybrání zahájení se otevře učící program.

5 Ukázka z aplikace

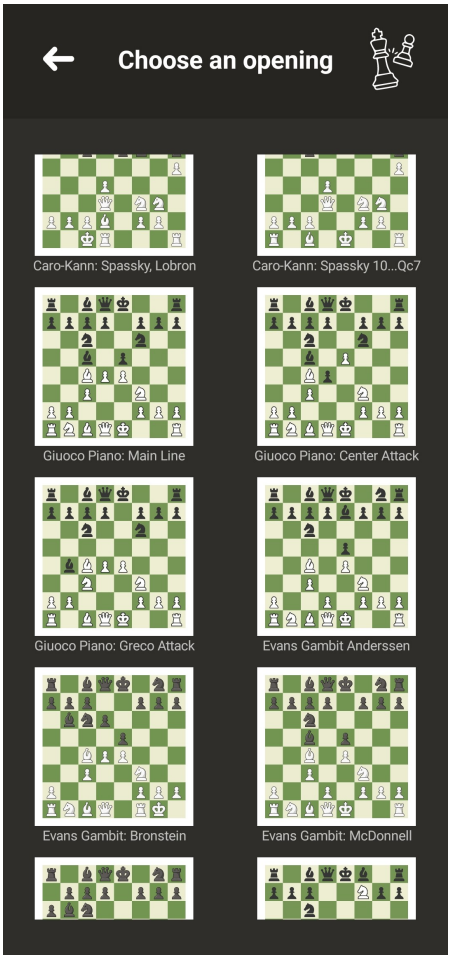

Figure 2: OpeningsRoot

Figure 3: Openings

6 Závěr

Aplikace je funkční a spustí se bez chyb. Jediný větší problém je podpora obrazovek se zvláštními tvary. Do budoucna bude potřeba přidat layout_landscape.xml pro jednotlivé aktivity. Momentálně podporuje pouze portrétovou orientaci a to pouze u obdélníkových rozměrů obrazovek.

Přešlapy Mohl jsem využít více knihoven. Když jsem mazal poslední commit, tak se mi povedlo smazat si práci z celého víkendu. Od té doby co jsem s prací začal, tak jsem se mnohé naučil a předchozí kód mi přijde nedostačující, už ale není v mojí kapacitě celou aplikaci přepsat.

Rozšíření Přidání obchodu pro nákup nových zahájení. Aplikaci bych si přál vydat s několika zahájeními zdarma, další po vyzkoušení by si uživatel musel za pár korun pořídit. Udělání layoutu více responzivní a podpora landscape orientace. Přidání výběru více stylů šachovnic a figurek v nastavení. Android Studio pořád hlásí, že nějaká funkce začne být u nových zařízení nefunkční, ale zatím funguje. Tedy to bude potřeba napravit. Například mapování jména zahájení k R.Drawable.image. Do budoucna bude potřeba předělat systém pro ukazování obrázků zahájení. Pokud jednou bude aplikace ukazovat stovky zahájení bude aplikace zaprvé obrovská a zadruhé v tomto rozlišení velmi pomalá.