Maturitní práce

Informatika

Aplikace pro pomoc při memorizaci šachových zahájení

Martin Vanko
4.C
Gymnázium Jana Keplera

2025

Obsah

1 Úvod

Zadání Androidová aplikace k učení se šachových zahájení. Uživatel bude mít na výběr z různých zahájení. Vždy bude odehrávat tahy, a jakmile se netrefí do teorie, zahájení se restartuje. Uživatel si takto bude pořád dokola opakovat teorii, dokud se ji nenaučí.
Aplikace nebude učit ideje zahájení – bude sloužit pouze jako memorizační pomůcka.

Proč? Šachy jsem začal hrát v roce 2021. Začal jsem být touto legendární hrou docela fascinovaný hned ze začátku, byla to aktivita, ve které jsem viděl smysl. Nedělalo mi problém řešit tisíce šachových úloh, čtení všemožných knih a studování partií starých mistrů, ale učení se zahájení. Všechny dostupné nástroje jsou určené k učení se idejí 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 pseudolegálních tahů je napsaná v jazyku C. Dotazy do Sqlite3 databáze jsou napsané v jazyku SQL. Na popsání statické části Bylo použito XML. Některé obrázky jsou v SVG formátu, což je taky druh XML. Pro editaci kódu bylo využito Android Studio od společnosti JetBrains a Visual Studio Code od společnosti Microsoft. JNI bylo využito k propojení generátoru tahů. Pro vytvoření obsahu databáze byl využit script v jazyce Python.

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 ostrova 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 (*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. Některé obrázky se nepovedlo najít na internetu, musely být napsáné ručně. Proto je SVG popsána jako zvolená technologie.

JNI JNI (*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. Jednoduchá 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. Guido van Rossum ho vytvořil v roce 1991. Při vývoji aplikace byl využit pro psaní scriptů při tvoření obsahu databáze. Zahájení byly ve formátu pgn a pomocí scriptu byly převedeny na pole FEN řetězců.

Android Studio Android Studio je integrované vývojové prostředí (IDE) od JetBrains určené pro tvorbu androidových aplikací. Tento IDE byl pro vývoj vskutku esenciální. Velmi důležitý byl editor layoutů, který ukazoval v reálném čase provedené změny. Android Studio ukládá automaticky lokální historii projektu, což nejednou vhod.

Visual Studio Code Visual Studio Code (VS Code) je multiplatformový textový editor od Microsoftu. Editor byl využit výhradně k napsání souboru pseudo_move_generation.h. Dále k vyzkoušení funkcí psané v jazyce Kotlin a psaní scriptů v jazyce Python.

3 Teoretická část

3.1 Pseudo_move_generation.h

Existuje řada způsobů, jak šachovnici reprezentovat. Například bychom mohli mít 32 figurek reprezentovaných pomocí barvy, typu a příslušné souřadnice na šachovnici. To je ale neefektivní a v závislosti na implementaci i prostorově náročné. Problém tedy je, jak co nejefektivněji výše zmíněné údaje reprezentovat. Pro řešení tohoto problému bylo zvoleno dvanáct 64bitových slov. Každý typ a barva figurky dostane vlastní slovo.

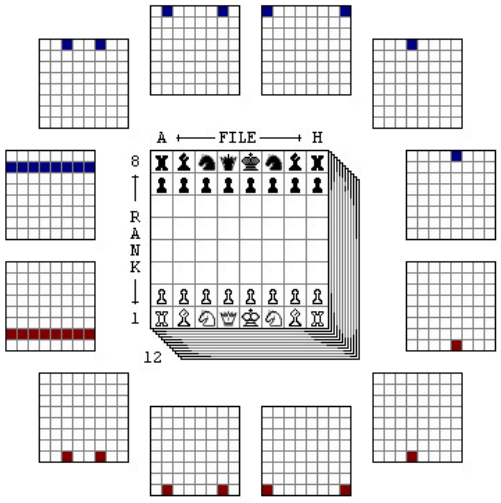

Figure 1: Bitboard

Pozice samotných figurek bude značená nastavenými 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. Protože typ unsigned long long nezaručí vždy 64 bitů, kód je vytvořen takto:

```
typedef uint64_t Bitboard;
```

Kromě toho, že je to velmi prostorově úsporné, to zároveň umožní jednoduše provádět na slovech bitové operace.

Ukládání tahů Potom bylo potřeba uložit tahy. Jeden tah se dá popsat políčky odkud a kam a 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, potřebujeme tedy 6 bitů na reprezentaci políčka odkud a 6 bitů na reprezentaci políčka kam. Zbývají 4 bity na reprezentaci typu tahu.

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

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

Čte se zleva doprava. Malá písmena jsou černé figurky a velká písmena bílé. Na konci řetězce následuje ukazatel, kdo je na tahu a práva na rošády. Pomlčka značí možné políčko pro tah en passant. 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 Bylo potřeba mnoho konstant. Nejdůležitější jsou 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;
```

Následující kód zabrání koni, aby v tomto příkladu 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 Funkce pseudo_moves_generator(char* fen) nejdříve načte notaci FEN a nastaví tak jednotlivé bitboardy. Potom vytvoří:

```
Bitboard pieces = king|queen|rook|knight|bishop|pawn;
```

Kód výše kontroluje, jestli figurka jedné barvy nebere figurku stejné barvy, což by bylo ilegální. Potom byly vytvořeny funkce pro generování tahů jednotlivých figurek.

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

Hlavní funkce projde 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 jádru celé aplikace leží komunikace mezi knihovnou v jazyku C a JVM. JNI je zkratka pro Java Native Interface, způsob jak volat nativní kód z Javy. Nejdříve se definuje funkce jstringToChar, která převede Java String na pole znaků. V Javě jsou řetězce objekty, nejsou to pouze pole znaků jako v C.
Potom se definuje samotná funkce getPseudoMoves(). Argumenty jsou

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

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

3.3 Databáze

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

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

3.3.1 .SCHEMA

Samotné schéma 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, které uvidí uživatel. Má vlastní soubor s názvem název_aktivity_layout.xml, kde je definovaný její layout. Potom má každá aktivita svůj soubor NázevAktivity.kt, což je třída, která dědí ze super třídy AppCompat-Activity.

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

V zdrojovém kódu aktivity jsem přepsal (override) různé metody z rodičovské třídy AppCompatActivity. Nejdůležitější taková metoda je:

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

V kódu výše je argument savedInstanceState poslední stav aktivity. Další důležitou metodou je 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 a provede animaci v souboru logo_enter.xml, následně otevře MainActivity.

3.4.2 MainActivity

Každá aplikace potřebuje hlavní aktivitu. Nemusí se nutně jmenovat MainActivity. Jedná se o první aktivitu, která se otevře (kromě animací). Hlavní aktivita aplikace obsahuje pouze komunikaci s databází a dále slouží jako rozcestník. Zajímavý kousek kódu je 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, streak se resetuje zpět na jedničku.
Obrázek plamene se mění s rostoucím streakem. Byla vytvořena funkce:

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

Tato funkce si bere jako argument instanci objektu ImageView, sprite objektu, hodnoty RGB a alpha. Nebude se měnit barva plamene, pouze alpha. Alpha říká, jak je ImageView průhledný, a argument může být z otevřeného intervalu celých čísel $(0, 255)$. Byla potřeba 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 posloupnost poroste, a $n = currentStreak$. $k \in \mathbb{R}$. Ideální hodnoty pro účel aplikace jsou z intervalu $(0, 1)$.

3.4.3 OpeningsRoot

Toto je rozcestník pro výběr zahájení. V hlavní aktivitě jsou dvě tlačítka ImageButton, jedno pro zahájení bílého a druhé 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 otevřena aktivita OpeningsRoot a přidá se extra informace o barvě. Dále se vytváří ImageButton pro každé zahájení. A kdyby náhodou zahájení nemělo jméno nebo obrázek, použije se základní text a obrázek.

3.4.4 Openings

Zde se kreslí figurky a šachovnice. Figurky se kreslí zvlášť dle vybrané barvy. Stejně jako v rozcestníku OpeningsRoot byly nějaké informace navíc, zde je k dispozici barva hráče, název zahájení v databázi a název zahájení. Takto se vypočítá začátek šachovnice:

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

Kód nebude fungovat na displejích, které mají podobně dlouhé strany. Pro ty se bude muset v budoucnu 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. To samé platí pro většinu dalších tříd.

Vlastnictví

- `val squareCoordinates: Array<IntArray>`: 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 objektů `ImageView`, které jsou využity k reprezentaci figurek.
- `var turn: Boolean`: Určuje, zda je na tahu bílý (`true`), nebo černý (`false`).

- `var plyCounter: Int`: Jedno ply je půltah. Pokud bílý zahraje v prvním tahu e4, ply bude 1.
- `castling rights`:

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

  Práva na rošádu a jejich permutace.
- `var enPassantSquare = false`: Uvádí 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 řetězci FEN. Používá se na zobrazení nápovědy. Vrací pole Integerů `[odkud, kam]`.
- `getPlayedMoveAlgebraicWithPiece(fen1: String, fen2: String): String`
  Detekuje provedený tah mezi dvěmi pozicemi FEN. Namísto [36, 53] vrací hodnotu e4 pro tah pěšce z e2 na e4.
- `pseudoLegalMoves(fen: String): MutableList<Move>`
  Získává pseudolegální tahy pomocí nativní funkce `getPseudoMoves`. Pro rošádu a speciální tahy jsou využity flagy uvedené výše.
- `drawPieces(...)`
  Vykreslí figurky na šachovnici podle zápisu FEN zvlášť pro každou stranu.

3.5.2 ChessPiece

Třída `ChessPiece` popisuje jednotlivé figurky, především jejich vizuální stránku. Většina logiky je ošetřena jinde. Asi nejdůležitější částí je logika při události 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 objekt ImageView, který 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 objekt `ImageView` na zadaných souřadnicích a nastaví jeho výšku a šířku.
- `makePieceClickable()`
  Nastaví události kliknutí pro figurku:
  - Po kliknutí se zvýrazní políčko zeleným čtvercem, pokud to nastavení dovoluje.
  - Zobrazí dostupné tahy pomocí `GreenSquareFactory`, pokud to nastavení dovoluje.
  - Když se uživatel dotkne políčka, zkontroluje se, zda je políčko mezi legálními tahy. Potom se aktualizuje pozice a FEN v instanci ChessBoard.
  - Zvukové efekty pomocí třídy SoundPlayer.

3.5.3 GreenSquareFactory

Třída `GreenSquareFactory` tvoří vizuální efekty ke znázornění legálních tahů, označení nápovědy, správných a špatných tahů.

Vlastnictví

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

Metody

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

- `addDot(squareNumber: Int)`
  Přidá zelenou tečku na políčko, kde je pseudolegální 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. Metoda byla přidaná, aby bylo možné dát uživateli najevo, že zahájení pokazil. Čtverec postupně mizí pomocí animace `fadeOut`.

- `addGreenSquare(squareNumber: Int)`
  Využívá se pro zvýraznění vybrané figurky nebo nápovědy. 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.

Vlastnictví

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

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

4 Návod k použití

Instalace Aplikaci lze nainstalovat dvěma způsoby. Uživatel si může aplikaci sám sestavit nebo stáhnout nejnovější verzi z Githubu s url:
https://github.com/Martin49232/chessProject/releases.

- `AndroidStudio` Uživatel si stáhne AndroidStudio na adrese
  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 připojí své zařízení se systémem Android k počítači pomocí možností pro vývojáře (Developer Options) v nastavení. Druhá možnost je spustit emulátor a zařízení emulovat.

- `Github` Uživatel si stáhne nejnovější vydání aplikace do počítače a následně si je přetáhne do telefonu, nebo si je stáhne z Githubu do telefonu přímo z url https://github.com/Martin49232/chessProject/releases.
  Poté bude potřebovat instalátor Package Installer, pomocí kterého aplikaci nainstaluje.

Použití Po nainstalování zapněte aplikaci. Otevře se hlavní okno a dostanete na výběr ze zahájení pro černého a bílého. Dále můžete otevřít nastavení nebo stisknout tlačítko hodin, které otevře poslední zahrané zahájení. Po výběru strany přejdete do okna pro výběr samotného zahájení. Po výběru zahájení se otevře učicí program. Zde má uživatel tlačítko zpět, tlačítka pro pohyb mezi odehranými tahy a tlačítko pro nápovědu. Posledně, text nad šachovnicí je pgn zápis zahájení.

5 Ukázka z aplikace

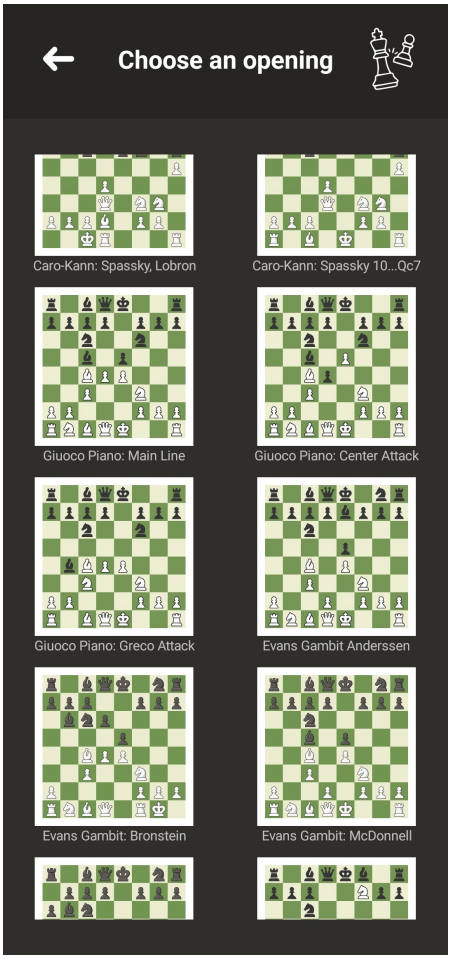

Figure 2: OpeningsRoot

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 orientaci na výšku, a to pouze u obdélníkových rozměrů obrazovek.

Přešlapy Mohlo se využít více knihoven. Když jsem mazal poslední commit, povedlo se mi smazat si práci z celého víkendu. Naštěstí většinu jsem měl stále lokálně uloženou.

Rozšíření

- `Přidání obchodu pro nákup nových zahájení` Aplikaci bych si přál vydat s několika zahájeními zdarma, po vyzkoušení by si další zahájení uživatel musel za pár korun pořídit.
- `Responzivnost` Více responzivní layout a podpora orientace na šířku.
- `Rozmanitost` Přidání výběru více stylů šachovnic a figurek v nastavení
- `Aktualizace` Android Studio často hlásí, že nějaká funkce začne být u nových zařízení nefunkční, ale zatím funguje. To bude potřeba v budoucnu upravit. Například mapování jména zahájení k R.Drawable.image.
- `Obrázky zahájení` Do budoucna bude potřeba předělat systém pro ukazování obrázků zahájení. Pokud jednou bude aplikace ukazovat stovky zahájení, bude obrovská a v tomto rozlišení velmi pomalá.
- `TODO` Ve zdrojového kódu se nachází pár komentářů //TODO. Jde o funkcionalitu, které nebylo do této doby potřeba, ale do budoucnosti bude nezbytná. Například funkce ke generování pseudolegálních tahů dokáže pracovat s tahem en passant, zatímco aplikace ho ukázat nedokáže. Nenarazil jsem zatím na zahájení, které by tento vzácný tah obsahovalo, ale jednou to bude potřeba implementovat. Takových případů je více.
- `Pseudo_move_generation.h` Funkce obsahuje dost cyklů. Je prostorově velice úsporná, časově zas tolik není. Do budoucna bude potřeba odebrání většiny cyklů.

7 Zdroje

https://developer.android.com/docs
https://developer.android.com/reference
https://stackoverflow.com https://cs.wikipedia.org/wiki/Java_Native_Interface
https://www.chessprogramming.org
https://www.overleaf.com
https://www.chess.com
https://lichess.org
https://www.geeksforgeeks.org
https://www.w3schools.com
https://android-developers.googleblog.com
https://kotlinlang.org/docs/home.html
https://stockfishchess.org
https://developer.android.com/ndk