



Eötvös Loránd Tudományegyetem
Informatikai Kar
Média- és Oktatásinformatikai Tanszék

Online sakk Andoidra és iOS-re

Témavezető:

Dr. Menyhárt László Gábor
adjunktus, Ph.D.

Szerző:

Cseri Ádám
Programtervező informatikus BSc.

Budapest, 2022

Tartalomjegyzék

1. Témabejelentő	3
2. Bevezetés	4
3. Felhasználói dokumentáció	5
4. Fejlesztői dokumentáció.....	9
4.1 Tervezés	9
4.2 Implementáció	11
4.3 Android UI és nézetek implementálása.....	27
4.4 iOS UI és nézetek implementálása	31
4.5 Webszerver és kliens implementálása	34
4.6 Tesztelés	36
5. Összefoglalás.....	44
6. További fejlesztési lehetőségek	45
7. Források	46

1. Témabejelentő

A szakdolgozatom témája egy cross platformos, telefonra készített online sakk alkalmazás, aminek előnye sok más telefonos sakk játékkal szemben, hogy egy iOS és egy Android telefon tulajdonos is tud egymás ellen játszani, ezáltal nem kell ugyanolyan operációs rendszerű mobillal rendelkezünk.

Azért választottam a sakkot, mert jelenlegi formájában a 15. század óta létező, ma is zseniálisan komplex stratégiai társasjáték. Hobbi és kompetitív játék is egyben, amit emberek milliói játszanak otthon, klubokban, online. Rengetegen rendeznek sakk bajnokságokat és mérettetik meg magunkat benne, illetve sokan ezzel a taktikai játékkal fejlesztik stratégiai képességeiket is.

Az elmúlt évtizedekben az internet teljes mértékben felkapott lett, ezzel összekötve a világot. A társasjátékokat is implementálták különböző programozási nyelvekkel és az online térbe helyezték, hogy a világ bármely pontjáról tudjanak az emberek játszani egymással.

Úgy gondolom hogy a játékfejlesztés kifejezetten alkalmas különböző nyelvek és technológiák elsajátítására, gyakorlására. Egyébként kellemes szórakozás látni, ahogy a játék életre kel.

A játékot Kotlinban implementálom, a webservert (megfelelő tárhelykapacitás esetén) a <http://csadam.web.elte.hu> weboldalon lesz található. Mivel igény van a játék közbeni kommunikációra és versengésre a játékosok között, ezért különböző funkciókkal, mint például chat, statisztika, stb. bővítem az alkalmazásom.

2. Bevezetés

Szakdolgozatom célja, hogy egy összetett játék applikáció implementálásával új ismeretekre tegyek szert, kihívás elé állítsam magam, valamint az, hogy létrehozzak egy sakk alkalmazást, ami széleskörben használható a felhasználók számára.

Mindigis érdekelték a mobil applikációk, illetve az internetes kommunikáció, ezért döntöttem el, hogy egy telefonos alkalmazást fogok készíteni, aminek lesz online funkciója, például többszemélyes játék. Már csak azt kellett eldöntennem, hogy mi lenne a tökéletes applikáció. Sok sok gondolkodás után eldöntöttem, hogy egy játékot, mégpedig sakkot fogok írni, amit online is lehet játszani.

Végül azért választottam játékot, mert számomra izgalmasabb úgy tanulnom, hogy tudom, ha készen lesz, akkor kipróbálhatom, játszhatok rajta, emellett a tesztelést is érdekessé teszi. Játéknak a sakkot azért választottam, mert bár összetett játék egyszerű és egyértelmű szabályokkal rendelkezik, gyerekként sokat sakkoztam édesapámmal, illetve a családi számítógépünkön a gép ellen és nagyon megszerettem. Telefonra implementálom a játékot, mert manapság, 2022-re, a fogyasztói igények áttértek a PC-ről az okostelefonokra. Amíg régen a fiatalok asztali gépen fogyasztottak tartalmakat (például social media, youtube és játékok is), ma már az okostelefonjaink is kifejezetten alkalmasak erre, ezért sokan már telefonjaikat használják a legtöbb tartalom fogyasztására, felhasználására. Az internet pedig ma már a mindennapjaink részét képezi. Úgy gondolom, hogy a legtöbb mobil alkalmazás akkor válik igazán sikeressé, ha tud internetes szolgáltatást is nyújtani. A felhasználói interface-t pedig egyszerűen kezelhetőre akarom kialakítani, hogy mind a fiatalabb, mind az idősebb korosztály könnyedén tudja használni, ezáltal egy nagypapa is tudjon együtt játszani az unokájával, illetve tudja tanítani őt.

Azért választottam a Kotlin-t, az alkalmazásom lefejlesztésére, mert alkalmas arra, hogy multiplatformos applikációkat fejlesszünk benne a Kotlin Multiplatform Mobile Plugin segítségével [1]. Emellett kifejezetten hasonlít a nyelv a Java-ra, ami egy jó választás Android applikációk fejlesztésére, így a nyelv tanulásának ideje is alacsony lenne, de mégis egy új nyelvet ismerek meg, miközben írom az applikációt. Androidra a hivatalosan támogatott nyelv a Kotlin, ezért ha multiplatformos applikációt készítek, Androidban könnyedén és gyorsan megvalósíthatom a célokat, amit majd iOS-re is implementálok.

A játékhoz adatbázis készítését nem tartom szükségesnek, de egy jó kiegészítés. Egy jó adatbázissal több felhasználói igénynek eleget tehetünk, mert nyomon követhetik saját haladásukat a játékban, előző játékaikat visszanezethetik, illetve egy globális ranglistával egészséges kompetitív környezetet teremthetünk a felhasználóknak. Egy ilyen kiegészítés sok időt venne igénybe, ezért az applikációm jelenlegi verziójában nem fog adatbázist tartalmazni.

3. Felhasználói dokumentáció

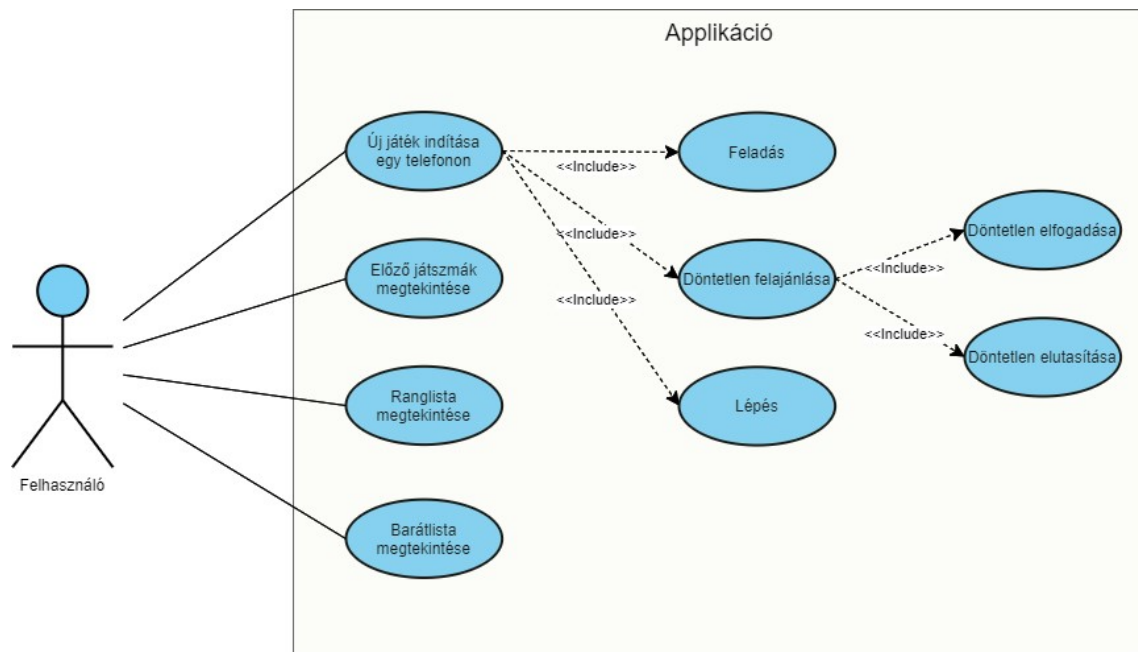
Az általam készített szoftver egy sakk alkalmazás, amit 2 ember tud együtt játszani telefonjaikon. Az alkalmazás nézeteit 2 operációs rendszeren, Androidon és iOS-en készítettem el, így mindkét interface natív. A 2 nézetet úgy implementáltam, hogy kifejezetten hasonlítsanak egymásra és lehetőség szerint ugyanazt a funkciót lássák el. A következő fejezetben több ábra látható a mobil alkalmazásról. Összehasonlítóképp egymás mellett látható az Android és az iOS implementáció.

Verziókövetelmény (Android): Android 7.1 (API 25)

Verziókövetelmény (iOS): iOS 15.2

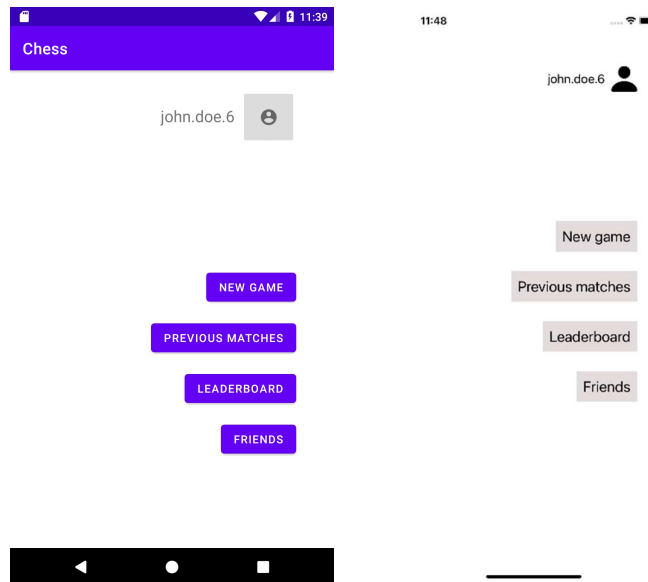
Az alkalmazás tesztelése Google Pixel 2-n és iPhone 11-en történt. Az előbb említett 2 telefon és az ezekhez a telefonokhoz hasonló felbontással és mérettel rendelkező eszközök nyújtják a legmegfelelőbb felhasználói élményt.

Az 1. ábrán látható a felhasználói esetdiagram, amely bemutatja az applikáció funkcióit. Mivel a sakk játékra lett elkészítve a UI, ezért a játék elemei kerültek implementálásra, az „előző játszmák”, „ranglista” és „barátlista” menüpontok csak mintaadatokat tartalmaznak. Ezek a menüpontok később vizuális képet adhatnak a további fejlesztési lehetőségekről, illetve az alkalmazás potenciáljáról.



1. ábra Felhasználói eset diagram

Az alkalmazás kezdőképernyője a főmenü, az alábbi 2. ábrán látható a főmenü nézete Android és iOS operációs rendszeren. Itt láthatjuk a felhasználónevünket, illetve kiválaszthatjuk, hogy mit szeretnénk csinálni: Új játékot kezdeni, az előző játszmák listájára ránézni, megtekinteni a ranglistát vagy a barátlistát.

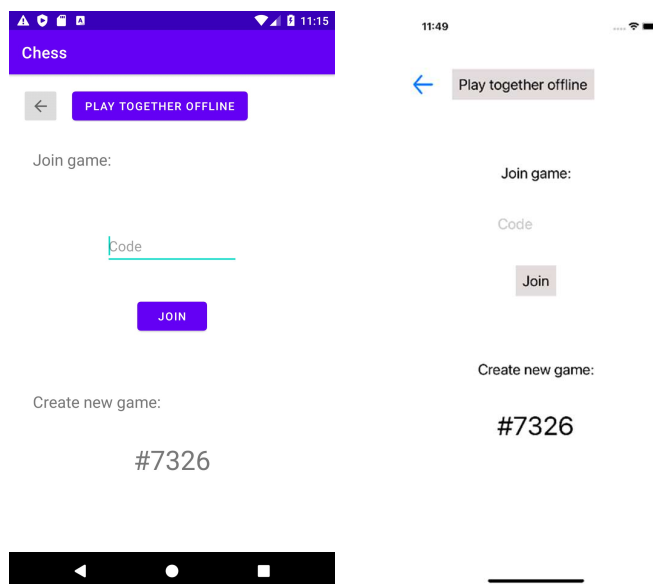


2. ábra Az alkalmazás főmenüje Android és iOS felületen

Új játék létrehozása kétféleképpen történhet: online és offline. Ezt a 3. ábra is mutatja. Az alkalmazás jelenlegi verziójában csak az offline játék kerül bemutatásra.

Ha a „Play Together Offline” gombra nyomunk, akkor 1 eszközön játszhatunk közösen. Itt nem lesz lehetőség chatelni egymással, mivel amúgyis egy térben van a 2 játékos, viszont ugyanúgy lehet feladni a játszmat és döntetlent felajánlani.

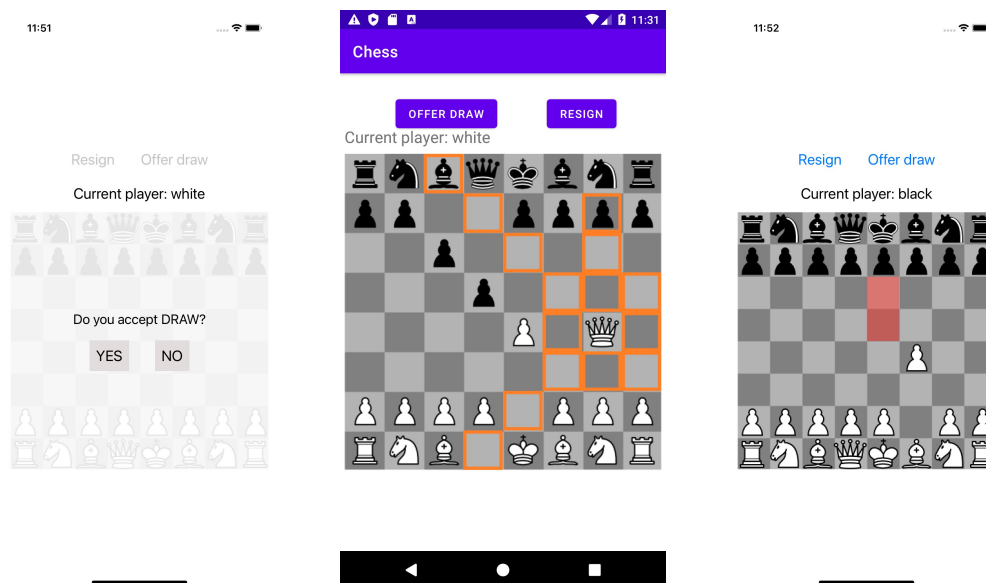
Következőképpen fog az online játék felépülni: a felületre a szerver generálni fog egy egyedi kódot, mint az a 3. ábrán is látható. Ezt a kódot be kell írnia a másik felhasználónak a „Code” beviteli mezőbe és utána rá kell nyomnia a „Join” gombra, ami elindítja a játékot a 2 játékosal.



3. ábra Új játék indítása Android és iOS felületen

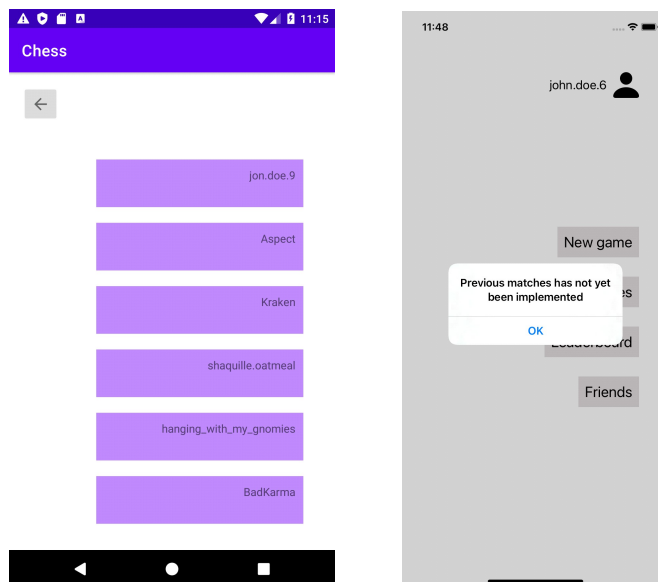
Amennyiben egy eszközön szeretne 2 felhasználó játszani, a „Play together offline” játékmódot kell kiválasztaniuk. A játékban található 2 gomb. A „Resign” a játék feladását jelenti, ezt akkor szokás megnyomni, hogyha úgy érzi a játékos, hogy nincs esélye megnyerni a játszmát. Az „Offer draw” gomb megnyomásával meg lehet kérdezni a másik játékost, hogy elfogadja-e a döntetlent. Erre igennel és nemmel válaszolhat.

A játékmenet kövei a sakk játszmák szabályait, és segíti a kezdő-, illetve tanuló játékosokat, mert egy bábú kiválasztásakor mutatja, hogy melyik mezőkre léphet vele a játékos. Ezekre láthatunk példát a 4. ábrán.



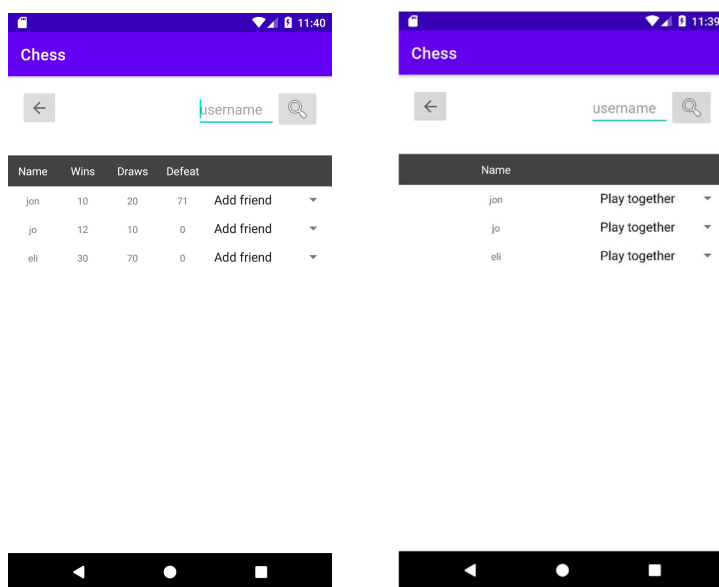
4. ábra A játék egyes elemei Android, illetve iOS telefonon, vegyesen

Az előző sakkjátszmák visszanézése menüpont véletlenszerű felhasználónevekkel van előregenerálva, a lista pedig ezekkel van feltöltve. Androidon a nevekre kattintva felugrik egy apró üzenet, ami kiírja hogy melyik névre kattintottunk. Ennek a továbbfejlesztett verziója lehet az előző játszma betöltése és visszanézése. Optimális esetben nyilakkal lehetne előre felé, illetve visszafelé lépkedni a játszma lépései között. Ide töltené be a program a lementett játszmákat, amikre kattintva visszanézhetőek lehetnek. Ezzel ellentétben ugyanezt az egyszerű tulajdonságot iOS-en nem implementáltam, mivel az applikációhoz, illetve a sakk játékhoz nem ad hozzá értelmes funkcionálisitást. Az 5. ábrán az előző játszmákra való menüpontra kattintás eseménye látható.



5. ábra Előző játszmák menüpontra kattintva előugró esemény Android és iOS felületen

Az applikáció jelenlegi verziójában a ranglista és a barátlista csak egy sablont tartalmaz, amit a 6. ábrán tekinthetünk meg. Ennek a 2 funkciónak az implementálása nem kerül a szakdolgozatba. A funkciók a „További fejlesztési lehetőségek” fejezetben kerülnek tárgyalásra.



6. ábra Ranglista és Barátlista Android felületen

4. Fejlesztői dokumentáció

4.1 Tervezés

A multiplatform applikációm elkészítéséhez a JetBrains által fejlesztett Kotlin Multiplatform Mobile SDK-t [1] tervezem használni. A multiplatformos applikációk előnye, hogy a sakk program logikáját – amit nevezzünk az 7. ábra alapján az alkalmazás logikájának és magjának – elegendő egyszer implementálnom. Az iOS, illetve Android specifikus API-kat is egymás mellett tárolhatom, ami a shared code (azaz a közös kód) része. Csak a 2 operációs rendszer különböző nézeteit kell külön-külön elkészítenem, Android esetén Kotlinban XML segítségével, iOS esetén SwiftUI-al Swift programozási nyelvben. Ez lehetővé teszi natív applikációk készítését mindkét környezetben.



7. ábra Multiplatform appok felépítése [1]

A következő osztályokból szükségesek a sakk program OOP-elvű implementációjához.

- Tábla (Board): 8x8-as táblázat, 64 mezővel (Spot). Itt helyezkedik el az összes aktív bábu (Piece). Aktív bábu, amit nem ütöttek le.
- Játék (Game): A játékmenetet vezérli. Tartalmazza követi az összes lépést, kinek a köre van jelenleg, és a játszma végeredményét is.
- Lépés (Move): egy lépést reprezentál, ami tartalmazza a kezdő- és végpontokat (Spot). Azt is nyomon követi az osztály, hogy melyik játékosnak (Player) a lépése volt.
- Bábu (Piece): Minden bábu egy mezőre (Spot) lesz helyezve. Absztrakt osztály. A kiterjesztett osztályokban (paraszt, király, királynő, futó, ló, bástya) implementálom az absztrakt műveleteit.
- Játékos (Player): A játékost reprezentálja.
- Mező (Spot): A 8x8-as tábla egy mezője, ahol lehetséges hogy egy bábu (Piece) található.

AZ ALKALMAZÁS NÉZETEI ÉS UI IMPLEMENTÁLÁSA

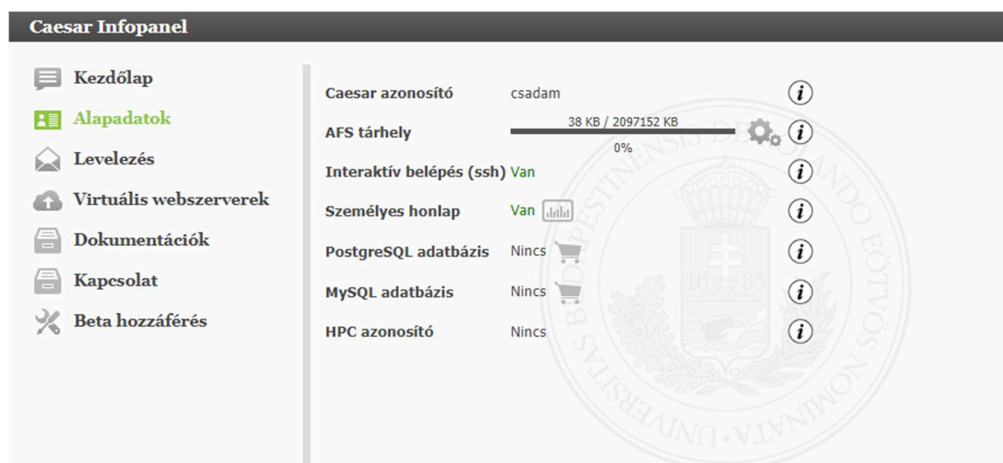
Az Androidra való integrációra és tesztelésre alkalmas az Android Studio fejlesztőkörnyezete. Az Android nézetek elkészítése számomra az elsődleges feladat, ezután iOS-re is elkészítem a nézeteket. Az iOS-re való integráció a hivatalos dokumentáció és oktatóanyag alapján elkészíthető [2]. Mivel Apple-s termékekre csak a saját rendszerükön lehet fejleszteni Xcode IDE-vel [3], ezért egy macOS-en fogom fejleszteni az applikációt a Kotlin Multiplatform Mobile (KMM) plugin segítségével [4]. Így az applikációt tesztelni tudom iOS szimulátorban.

INTERNETKAPCSOLAT MEGVALÓSÍTÁSA

A 2 játékos közötti internetkapcsolatot sokféleképpen lehet megvalósítani. A WebSocket egy erre alkalmas módszer, TCP protokollt használok a fejlesztéshez, a Socket.io könyvtár pedig fejlesztőbaráttá teszi a webszerver és a kliens elkészítését. Az online játék úgy néz majd ki, hogy egy új játék indításakor egy szobába kerül a 2 játékos, akik tudnak egymásnak üzeneteket is küldeni. A főmenüben egy globális chat funkció is megvalósítható ezáltal. Ehhez lehetőség szerint továbbra is Kotlin-t fogok használni [5], de több példa is elérhető interneten WebSocket implementációval [6].

OPCIONÁLIS: ADATBÁZIS LÉTREHOZÁSA

Ezek mellett, amennyiben elegendő idő áll rendelkezésre a további fejlesztésekre, egy SQLite lokális adatbázis a felhasználó telefonján vagy egy szerveren található adatbázis növelné a felhasználói élményt, mert az applikáció el is tárolna felhasználók számára értékes információkat, mint például: felhasználók pontszáma, barátai, játszmainak száma, illetve a játékos elérhetné a saját korábbi játszmáit, hogy visszanézhesse azokat. A lentebbi 8. ábrán látható Caesar infópanelen igényelni lehet PostgreSQL, illetve MySQL adatbázist. Ez egy jó eszköz az adatbázis létrehozására.



8. ábra Caesar infopanel segédeszközei

4.2 Implementáció

Az objektumorientált programozási módszer elveit követve implementáltam az applikációt. 2 nyelvben, Kotlinban és Swiftben készítettem el a sakk programot. A Kotlin Multiplatform Mobile környezet alkalmatlan volt multiplatformos applikációk készítésére. Mivel az egyik verzió később készült el, és mindkettő nyelvben ugyanarra a logikára támaszkodom, ezért az osztálynevek, a függvénynevek, illetve a szerepeik a programban azonosak. Egyetlen eltérés fedezhető fel. A Swift nyelvben használt Extension-öket a fejezet végén tárgyalom.

A sakkprogramozás témakörében 2 kifejezést fogok használni és rövidíteni. A Universal Chess Interface jelölést használom a lépések tárolására, röviden UCI [7]. A Forsyth–Edwards Notation (röviden: FEN) jelölést pedig széleskörben használják a sakk állásának reprezentálására [8].

A következő osztályokból áll a sakk program:

- Tábla (Board): 8x8-as táblázat, 64 mezővel (Spot). Itt helyezkedik el az összes aktív bábu (Piece). Aktív bábu, amit nem ütöttek le.
- Játék (Game): A játékmenetet vezérli. Tartalmazza követi az összes lépést, kinek a köre van jelenleg, és a játszma végeredményét is.
- Lépés (Move): egy lépést reprezentál, ami tartalmazza a kezdő- és végpontokat (Spot). Azt is nyomon követi az osztály, hogy melyik játékosnak (Player) a lépése volt.
- Bábu (Piece): Minden bábu egy mezőre (Spot) lesz helyezve. Absztrakt osztály. A kiterjesztett osztályokban (paraszt, király, királynő, futó, ló, bástya) implementálom az absztrakt műveleteit.
- Játékos (Player): A játékost reprezentálja.
- Mező (Spot): A 8x8-as tábla egy mezője, ahol lehetséges hogy egy bábu (Piece) található.

Board.class

KONSTRUKTOR

Paraméter nélküli konstruktor. Inicializáláskor egy megadott FEN jelölés alapján felépíti a táblát a `parseFEN` nevű függvény segítségével.

PÉLDÁNYSZINTŰ VÁLTOZÓK

- board: Array<Array<Spot>> – Egy 8x8-as mátrix, amelynek minden eleme egy mező (Spot). Ez reprezentálja a játéktáblát.
- side: String – A változó tartalmazza, hogy melyik játékosnak a köre van éppen. 2 értéket vehet fel: 'w' és 'b'.

- enPas: String – Jelöli hogy van e En Passant mező. Amennyiben nincs a karakterlánc tartalma egy vonal ('-'), egyéb esetben a megfelelő mező egyedi azonosítója.
- castPerm: String – Jelöli az engedélyezett sáncolásokat ('KQkq'). Amennyiben nincs engedélyezett sáncolás, a karakterlánc tartalma egy vonal ('-').
- halfMove: Number – A fél lépések számát jelöli.
- fullMove: Number – A teljes lépések számát jelöli.

PÉLDÁNYSZINTŰ FÜGGVÉNYEK

- fun toPrint()

Kirajzolja a játéktáblát. Emellé kiírja, hogy kinek a köre van, van e En Passant mező és, milyen sáncolások aktívak még.

- fun toPrintIDs()

Kirajzolja minden mező azonosítóját.

- fun parseFEN()

A program fejlesztéséhez sok sok állást és pozíciót kell megadnom, ezért beépítettem egy FEN jelölés konvertálót a programomba, ami feldolgozza, majd feltölti a programomat a jelölésben megadott adatokkal. Így sokkal könnyebben tudom tesztelni a kódomat, és megkönnyítette a fejlesztést, egyébként, alap esetben csak a kiinduló állást töltheti be.

Egy FEN jelölést 6 részre lehet bontani:

1. Van egy táblát reprezentáló része, ahol a per jelek szeparálják el a sorokat. Azaz 7 perjel található benne 8 sorral. A kisbetűs karakterek a fekete, a nagybetűsek a fehér bábukat jelölik, a számok pedig, hogy hány darab üresen hagyott mező van 2 bábu között (vagy a tábla 2 széle között).
2. 2 különböző karaktert tartalmazhat, 'w' és 'b'. Ez jelöli, hogy kinek a köre következik.
3. Az aktív sáncoló lépéseket mutatja be, a fehér és fekete játékosoknál, király és királynői oldalról. Amennyiben nincs aktív sáncoló lépés egy átvonás ('-') jel van a helyén.
4. Az aktív En Passant mezőt jelöli. Ha nincs aktív ilyen mező, akkor egy átvonás ('-') jel van a helyén.
5. A fél lépések számát jelzi.
6. A teljes lépések számát jelzi. A fekete lépése után eggyel nő az értéke.

Ezeket eltárolom az osztály adattagjaiként, és ezekkel az adatokkal inicializálva indul el a játék.

- fun makeMove()

Végrehajtja a játékos lépését, és frissíti az osztályt, illetve az asztalt a lépés szerint. A metódus elején lekérem hogy egy elfogó lépés e vagy sem. Lementem a bábút, amivel lépni

akarok, majd ahonnan lépni akarok üresre állítom. Ezután a célmezőre helyezem a bábút, akár volt rajta előzőleg bábu, akár nem. Azt, hogy saját bábút ne tudjak ütni, illetve az ütés szabályainak megfelelően lépjek ütés esetén, a program `isValidMove()` metódusában validálom. Amennyiben En Passant ütés történik, az ellenfél érintett bábuját leszedem a tábláról. Amennyiben promóció történik, kicserélem a gyalogot a játékos által választott egyik bábura. Amennyiben sáncoló lépést kezdeményeztek a királlyal, az érintett bástyát áthelyezem a megfelelő mezőre. Ezután frissítem a játéktábla adatait. Ha a fehér lépett, most a fekete következik, ellenkező esetben a fehér. Frissítem az En Passant mezőt, a sáncoló engedélyeket, a fél lépéseket, és az egész lépéseket.

- `fun replaceRook()`

Hogyha sáncoló lépés történik, áthelyezi a bástyát a megfelelő helyre. Ha a fehér játékos a királynői oldalon akar sáncolni, akkor a bal oldali bástyát a d1 mezőre helyezzük, ha a király oldalán akar sáncolni, akkor a jobb oldali bástyát az f1 mezőre helyezzük. A fekete játékosal ugyanez a helyzet, csak ott az 1-es sor helyett a 8-as sorban történik a csere.

- `fun isCastlingMove()`

Eldönti hogy a lépés sáncoló lépés e. Hogyha a fehér király el akar lépni a 'c1'-es mezőre vagy a 'g1'-es mezőre és a megfelelő oldali sáncoló engedély is megvan még, akkor engedélyezi a lépést. A feketéknél ugyanezeket a lépéseket teszteljük, csak 'c1' és 'g1' mezők helyett 'c8' és 'g8' mezőket veszünk figyelembe.

- `fun isPromotionMove()`

Ellenőrzi, hogy a lépés átváltozó lépés e. Ezt úgy teszi meg, hogy megnézi az adott lépésnek van e promotion mezője.

- `fun promotePawn()`

Kicseréli az átváltozni kívánó gyalogot a játékos által választott bábura. Királynő, bástya, ló és futó bábukra cserélheti le.

- `fun isEnPasMove()`

Eldönti, hogy a lépés egy En Passant ütőlépés volt e. Hogyha a gyalog az En Passant mezőre lép (amit csak ütéssel tehet meg), akkor igaz értéket ad vissza, egyébként hamisat.

- `fun captureEnPas()`

En Passant ütés esetén hívódik meg a metódus. Ilyenkor a bábút, amelyik ki volt téve a leütés veszélyének, levesszük a tábláról. Azaz ha a fehér egy En Passant ütő lépést hajtott végre, akkor a fekete bábút, amelyik aktiválta az En Passant mezőt, levesszük a tábláról.

- `fun updateFullMove()`

Miután a fekete bábu lépett, eggyel növeli a fullMove értékét.

- fun updateHalfMove()

Lenullázza a féllépés számlálót, hogyha egy gyalog előre haladt, vagy ha ütés volt a körben, egyéb esetben a számláló értéke eggyel nő.

- fun updateCastPerm()

Leellenőrzi, hogy a sáncoló engedélyek még aktívak e. Amennyiben a jobb oldali bástya ellépett a helyéről, a sáncolás jobbra már nem lehetséges, baloldalon szintén ugyanígy működik. Amennyiben a király ellépett a kezdő pozíciójáról, a sáncoló lépések inaktívvá válnak az adott játékosra nézve.

- fun updateEnPas()

Ellenőrzi, hogy van e aktív En Passant mező. Ha a gyalog 2-t lépett (amit csak a kezdőpozícióból tehet meg), akkor az induló és érkező mező közötti mező lesz az En Passant mező, egyéb esetben nem lesz ilyen mező.

Példa: egy gyalog az e2-ből az e4-be lép, ebben az esetben az e3-as mező En Passant mező lesz.

- fun getSpotById()

Visszaadja a mezőt, annak azonosítója alapján.

- fun getPosition()

Visszaad egy párt, a bábu pozícióját jelöli a kétdimenziós tömbben. Megkeresi a bábút, és visszaadja, hogy mely sor, mely oszlopában található.

- fun filterNotSafeMoves()

Egy lépés nem biztonságos, ha utána a király ütés alá kerülne. A metódus leellenőrzi, hogy az adott lépés után ütés alatt fog e állni a király. Lementi az állást, odamozgatja a bábút a kívánt helyre, megnézi, hogy a király így ütés alatt áll e, visszaállítja a táblát, és végül visszaadja a megmaradt lépéseket, ahová léphet a bábu.

- fun isKingUnderCheck()

Ellenőrzi, hogy a király sakkban van e. Megnézem, hogy az ellenfél bábuai közül van e bármelyik, ami ütheti a királyt, és ennek megfelelő igaz-hamis értéket ad vissza.

- fun allSpotsOfPlayer()

Összegyűjti az egyik játékos összes bábuját a játéktáblán.

- `fun printPossibleMoveList()`

Kiírja a lehetséges lépéseket a konzolra.

KONSTRUKTOR

Paraméterben vár 2 játékost, amiket eltárol, létrehoz egy új táblát, egy üres lépéslistát és a játék státuszát aktívra állítja.

PÉLDÁNYSZINTŰ VÁLTOZÓK

- player1: Player – Az egyik játékos.
- player2: Player – A másik játékos.
- board: Board – Ez a tábla osztály, ami tartalmazza a játéktáblát, és a játékhoz fontos adattagokat.
- moves: MutableList<Move> – Az összes lépést eltároljuk ami a játékban történik.
- status: GameStatus – A játék státusza: aktív, döntetlen, fehér vagy fekete győzelem.

PÉLDÁNYSZINTŰ FÜGGVÉNYEK

- fun printBoard()

Kírja a konzolra a táblát. Meghívja a tábla toPrint() metódusát.

- fun printBoardIDs()

Kiírja a konzolra a tábla azonosítóit. Meghívja a tábla toPrintIDs() metódusát.

- fun getCurrentPlayer()

Visszaadja a következő játékost. Hogyha a fehér lépése jön, akkor visszaadja a fehér játékost, egyéb esetben a feketét.

- fun isGameOver()

Leellenőrzi, hogy a játék még aktív állapotban van e.

- fun isPromotion()

Eldönti, hogy a gyalog átalakulhat e egy másik bábuvá. Amennyiben a lépés nem gyaloggal történik, a visszatérési értéke további elemzés nélkül false lesz.

- fun updateStatus()

Frissíti a játék állapotát. Hogyha egy játékosnak nincs egy lehetséges lépése sem, és a királya ütés alatt áll, akkor sakkmatt-ot kapott, azaz ő veszített. Hogyha egy játékosnak nincs egy lehetséges lépése sem, de a királya nem áll ütés alatt, akkor pattot kapott, azaz a játék döntetlen lesz.

- fun showSelectedMoves()

Kiírja a konzolra azokat a mezőket, amelyeken a highlight attribútum true értéket ad.

Move.class

KONSTRUKTOR

A konstruktor paraméterként megkapja hogy melyik játékos lépése a jelenlegi lépés, és egy UCI jelölést, ami a lépést reprezentálja. Az első 2 karakter tartalmazza, hogy melyik mezőről lép a játékos, a második 2 karakter pedig, hogy melyik mezőre lép. Az 5-ik esetleges karakter annak a bábunak a kezdőbetűjét tartalmazza, amivé át fog alakulni a gyalog.

PÉLDÁNYSZINTŰ VÁLTOZÓK

- from: String – Tartalmazza hogy melyik mezőről lép el a játékos.
- to: String – Tartalmazza hogy melyik mezőre lép a játékos.
- promotion: String – Amennyiben a lépésben átalakulás történik (a gyalog beér az ellenfele alapsorába) tartalmazni fogja a változó hogy mivé alakul át. Ha nincs átalakulás, akkor null értéket kap.
- player: Player – Tartalmazza hogy melyik játékos kezdeményezte a lépést.

abstract Piece.class

A Piece.class leszármaztatott osztályaiban gyűjtöm össze a bábuk lehetséges lépéseit. Itt ismertetem, hogy a gyalog először 2-t, majd csak egyesével tud lépni, a ló csak L alakban tud lépni, a bástya csak egyenesen és vízszintesen, a futó függőlegesen, a királynő a bástya és a futó kombinációja. Ilyen szabályokat implementálok azokban az osztályokban, amelyek a Piece osztályhoz tartoznak.

KONSTRUKTOR

Paraméterben bekéri, hogy fehér e a bábu. Megadható neki igaz és hamis érték, illetve üresen is hagyhatjuk. Ekkor null értéket vesz fel. Azért hagytam meg az isWhite adattagnak a null értéket, mert amennyiben egy mezőn nincsen bábu, egy üres adattaggal töltöm fel a mezőnek a bábuját. A következőkben nem tárgyalom a leszármaztatott osztályokban lévő konstruktor hívásokat, hiszen azok megegyeznek ezen konstruktor hívással.

PÉLDÁNYSZINTŰ VÁLTOZÓK

- isWhite: Boolean – Minden egyes leszármaztatott osztálynak van egy színe. Ez határozza meg, hogy melyik játékos irányíthatja a bábút.
- char: Char – Minden egyes leszármaztatott osztálynak van egy egyedi karaktere. A fehérek nagybetűsek, a feketék kisbetűsek, a bábu nélküli mezőknek pedig egy kötőjelet tartalmaz. P – gyalog, K – király, Q – királynő, B – futó, N – ló, R – bástya. A fejlesztést nagyban megkönnyíti a vizuális reprezentáció, ezért alkalmaztam ezeket a karaktereket. A továbbiak alosztályokban nem tárgyalom a példányszintű változókat, mivel mindengyiknek a leírása megegyezik az imént említettel.

PÉLDÁNYSZINTŰ FÜGGVÉNYEK

- fun possibleMoves()

Minden származtatott osztálynak tartalmaznia kell egy lehetséges lépések metódust. Összegyűjti az összes lehetséges lépést és egy listában visszaadja, hogy mely mezőkre tud lépni az adott bábu.

- fun possibleCaptureMoves()

Minden származtatott osztálynak tartalmaznia kell egy lehetséges ütőlépések metódust. Összegyűjti az összes lehetséges ütést és egy listában visszaadja, hogy mely mezőkre tud lépni az adott bábu ahol van ellenséges bábu.

King.class : abstract Piece.class

A király lépésének szabályait tárgyalom ebben az alfejezetben. A király lehetséges lépései a lentebbi, 9. ábrán láthatóak.

PÉLDÁNYSZINTŰ FÜGGVÉNYEK

- fun possibleMoves()

Először lekérem a király pozícióját a sakktáblán, közvetlenül ezután pedig összegyűjtöm az összes mezőt, ahol ellenséges bábu tartózkodik. Ezután megvizsgálom, hogy üres e a mező a közvetlen környezetében. Amelyik mező üres, azt hozzáadom a lehetséges lépéseihez.

Amennyiben a feltételek adottak és sáncolhat a király ezeket is hozzáadjuk mint lehetséges lépések a listánkhoz.

- fun possibleCaptureMoves()

Megvizsgálom, hogy a király közvetlen környezetében van e ellenséges bábu. Amelyik mezőn van ellenséges bábu, hozzáadom az elfogólépések listához.

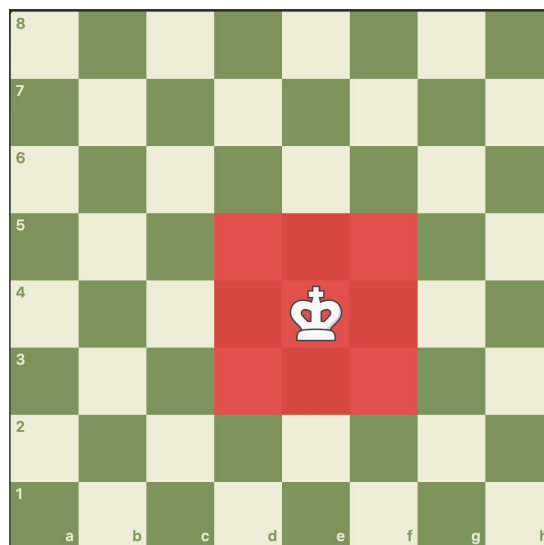
- fun isKingSpotUnderAttack()

Azt vizsgálja, hogy a király kiinduló mezője sakkban van e. Erre azért van szükség, mert a király csak akkor sáncolhat, hogyha a király nincs sakkban, illetve a mezők, amelyen áthalad és amelyre érkezik sem lehet sakkban.

A működése: A királyt a paraméterben megadott mezőre mozgatom, megnézem, hogy így ütés alatt áll e, majd visszaállítom a táblát, és visszaadom, hogy ütés alatt állt e a király a megadott mezőn.

- fun collided()

Visszaadja, hogy a paraméterben megadott mező üres e.



9. ábra A király lehetséges lépései

Queen.class : abstract Piece.class

A királynő lépésének szabályait tárgyalom ebben az alfejezetben. A királynő lehetséges lépései a lentebbi, 10. ábrán láthatóak. A királynő a bástya és a futó kombinációja, ezért a lépések szabályai kifejezetten hasonlóak, mint az utóbbi 2 bábutípusnál leírtaké.

PÉLDÁNYSZINTŰ FÜGGVÉNYEK

- fun possibleMoves()

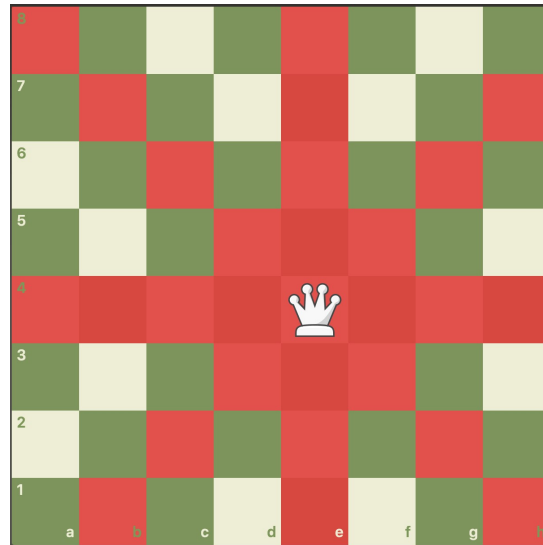
A királynő mozoghat felfelé, lefelé, balra, jobbra, illetve a 2 átló mentén, amíg akadályba nem ütközik. Az akadály egy bábu vagy a tábla széle lehet. Ezeket az üres mezőket adja vissza a metódusunk, mint lehetséges lépések.

- fun possibleCaptureMoves()

Az ütésnél azt vizsgálom, hogy amikor akadályba ütközik a királynő, az a tábla széle, vagy egy ellenséges bábu. Hogyha egy ellenséges báburól van szó, akkor a mezőt, amelyen áll a lehetséges ütőlépések listájához adom hozzá.

- fun collided()

Visszaadja, hogy a paraméterben megadott mező üres e.



10. ábra A királynő lehetséges lépései

Rook.class : abstract Piece.class

A bástya lépésének szabályait tárgyalom ebben az alfejezetben. A bástya lehetséges lépései a lentebbi, 11. ábrán láthatóak.

PÉLDÁNSZINTŰ FÜGGVÉNYEK

- fun possibleMoves()

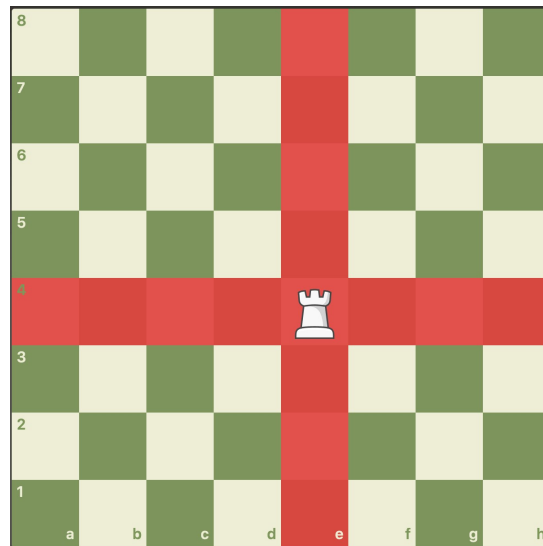
A bástya esetében mozoghat felfelé, lefelé, balra és jobbra, amíg nem ütközik akadályba. Az akadály egy bábu vagy a tábla széle lehet. Ezeket az üres mezőket adja vissza a metódusunk, mint lehetséges lépések.

- fun possibleCaptureMoves()

Az ütésnél azt vizsgálom, hogy amikor akadályba ütközik a bástya, az a tábla széle, vagy egy ellenséges bábu. Hogyha egy ellenséges báburól van szó, akkor a mezőt, amelyen áll a lehetséges ütőlépések listájához adom hozzá.

- fun collided()

Visszaadja, hogy a paraméterben megadott mező üres e.



11. ábra A bástya lehetséges lépései

Knight.class : abstract Piece.class

A ló lépésének szabályait tárgyalom ebben az alfejezetben. A ló lehetséges lépései a lentebbi, 12. ábrán láthatóak.

PÉLDÁNYSZINTŰ FÜGGVÉNYEK

- fun possibleMoves()

Lekérjük a ló koordinátáit a táblán. Ezután 4 részre osztjuk a metódusunkat: mozgás felfelé, lefelé, balra és jobbra.

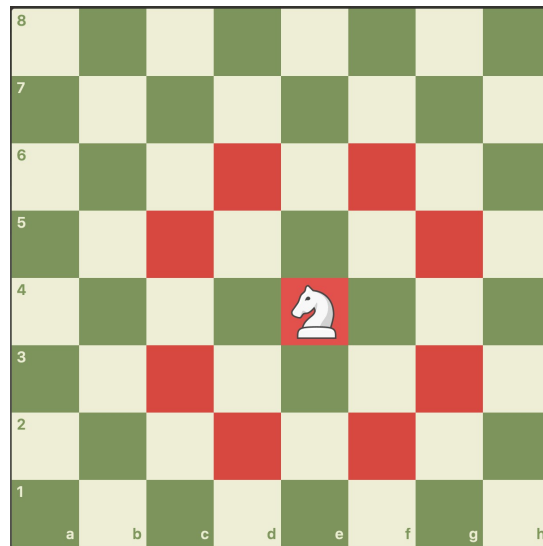
Megnézzük, hogy 2 sorral feljebb van e még mező. Ha van, akkor megnézzük, hogy van e 1 oszloppal balra üres mező. Ha van, akkor hozzáadjuk azt a mezőt a lehetséges lépésekhez. Ezután megnézzük, hogy van e 1 oszloppal jobbra üres mező. Ha van, akkor ezt a mezőt is hozzáadjuk a lehetséges lépésekhez. Így jön ki az L alak. Ugyanezt vizsgálom lefelé, balra és jobbra is.

- fun possibleCaptureMoves()

A lehetséges elfogó lépések annyiban különböznek a lehetséges lépés metódustól, hogy itt kifejezetten azt vizsgálom, van e ellenséges bábu az adott mezőn. Csak akkor adom hozzá a listához, hogyha van.

- fun collided()

Visszaadja, hogy a paraméterben megadott mező üres e.



12. ábra A ló lehetséges lépései

Bishop.class : abstract Piece.class

A futó lépésének szabályait tárgyalom ebben az alfejezetben. A futó lehetséges lépései a lentebbi, 13. ábrán láthatóak.

PÉLDÁNYSZINTŰ FÜGGVÉNYEK

- fun possibleMoves()

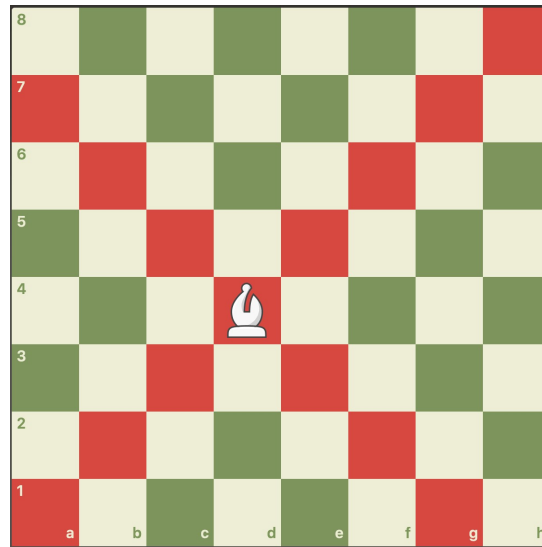
A futó esetén az átlókat kell megfigyelnünk. A főátlón, illetve a mellékátlón balra és jobbra megyünk, amíg nem ütközik akadályba. Az akadály egy bábu vagy a tábla széle lehet. Ezeket az üres mezőket adja vissza a metódusunk, mint lehetséges lépések.

- fun possibleCaptureMoves()

Az ütésnél azt vizsgálom, hogy amikor akadályba ütközik a futó, az a tábla széle, vagy egy ellenséges bábu. Hogyha egy ellenséges báburól van szó, akkor a mezőt, amellyen áll a lehetséges ütlőlépések listájához adom hozzá.

- fun collided()

Visszaadja, hogy a paraméterben megadott mező üres e.



13. ábra A futó lehetséges lépései

Pawn.class : abstract Piece.class

A gyalog lépésének szabályait tárgyalom ebben az alfejezetben. A gyalog kiinduló helyzete a lentebbi, 14. ábrán látható. Ha innen elmozdult, akkor csak 1 lépést haladhat előre, egyébként 2-t. Amikor a gyalog 2 lépést tett meg, aktiválódik az En Passant mező (az a mező, amelyiket átugrott), és ez a következő körben üthető lesz.

PÉLDÁNYSZINTŰ FÜGGVÉNYEK

- fun possibleMoves()

Lekérjük a gyalog koordinátáit a táblán, majd eldöntjük, hogy melyik irányba megy. A feketék lefelé, a fehérek felfelé mennek. Ezután a következő feltételeket ellenőrizzük: Ha a gyalog előtti mező nem üres, akkor nem léphet a bábu. Ha továbbmegy a metódus, az azt jelenti, hogy az előtte lévő mező tiszta, ezután ha nem mozdult még el a gyalog, akkor megvizsgáljuk, hogy a második mező is üres e előtte, ha igen, akkor az előtte lévő 2 mezőt visszaadja, azaz, mindkét mezőre léphet a gyalog. Egyéb esetben pedig csak 1-et léphet előre.

- fun possibleCaptureMoves()

Az elfogólépéseknél is lekérjük a gyalog koordinátáit, majd eldöntjük, hogy melyik irányba megy. Azok a gyalogok, amelyek az a-g oszlopok között helyezkednek el, üthetnek jobbra, hogyha van ott ellenséges bábu, vagy a mező egy En Passant mező. A b-h oszlopok között elhelyezkedő gyalogok pedig balra is üthetnek, amennyiben van ott ellenséges bábu, vagy En Passant mező.

- fun isEmptySpot()

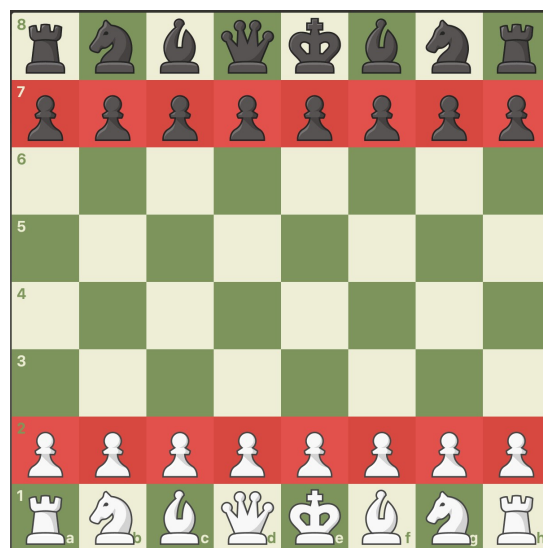
Ellenőrzi, hogy a paraméterben megadott mező üres e.

- fun differentColors()

Ellenőrzi, hogy a paraméterben megadott bábu színe különbözik e a szóban forgó gyalog színétől.

- fun hasMoved()

Ellenőrzi, hogy a gyalog elmozdult e a kiinduló mezőjéből. Ez, mint a 14. ábrán is látható, a fehérenél a második sor, a feketénél a hetedik sor.



14. ábra A gyalogok elhelyezkedése a játék kezdetén

Empty.class : abstract Piece.class

Az üres (bábu nélküli) mezők piece adattagjának egy úgynevezett üres osztályt adok meg. Ezt az osztályt az indokolja, hogy többet tud, mint a null érték, ezzel meggyorsítva a fejlesztés folyamatát.

PÉLDÁNYSZINTŰ VÁLTOZÓK

- char: Char – Az üres adattagnak is reprezentálásra kell kerülnie a konzolon, amit egy kötőjellel meg is teszek.

PÉLDÁNYSZINTŰ FÜGGVÉNYEK

- fun possibleMoves()

Mivel egy bábu nélküli mezőről nem lehet ellépni, ezért egy üres listát ad vissza.

- fun possibleCaptureMoves()

Mivel egy bábu nélküli mezőről nem lehet ellépni, ezért egy üres listát ad vissza.

Player.class

KONSTRUKTOR

Egy példány létrehozásakor bekéri paraméterként a játékos színét.

PÉLDÁNYSZINTŰ VÁLTOZÓK

- isWhite: Boolean – Egy true/false értéket vár. Az érték konstans, a játék végéig ugyanazt a színt fogja reprezentálni.

Spot.class

KONSTRUKTOR

Egy példány létrehozásakor kér egy bábút, hogyha van rajta bábu. Ha nincs, akkor nem kell megadni, hogy milyen bábu áll a mezőn, alapértelmezetten a piece üres lesz. Illetve inicializáláskor a mező megkapja a statikus adattagtól a saját azonosítóját és frissíti a következő aktuális azonosítóra.

PÉLDÁNYSZINTŰ VÁLTOZÓK

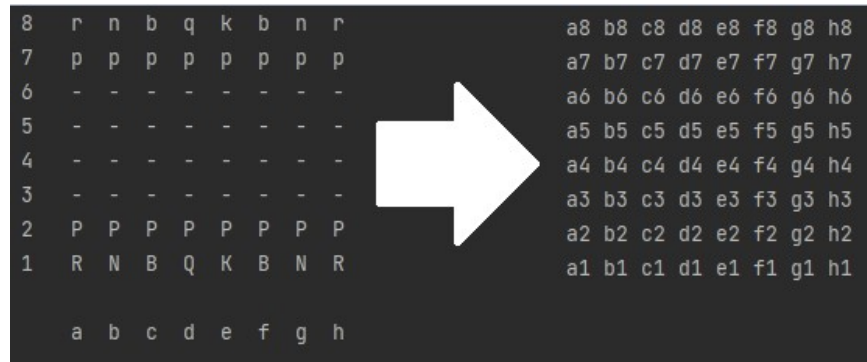
- piece: Piece – Az adott mezőn megtalálható bábút tartalmazza. Tartalma úgy változik ahogy más és más bábuk lépnek a mezőre vagy lépnek le róla. Tartalmazhat egy bábút vagy lehet üres.
- id: String – Minden egyes mezőnek van egy egyedi azonosítója. Ez nem változik, konstans érték.
- highlighted: Boolean – Minden mező értéke alapértelmezetten hamis. Amennyiben egy kijelölt bábu léphet egy mezőre, ennek a mezőnek igaz lesz az értéke, ideiglenesen.

OSZTÁLYSZINTŰ (STATIKUS) VÁLTOZÓK ÉS FÜGGVÉNYEK

Ezt Kotlinban társ objektumként (companion object) hozom létre, mivel a nyelvben nem létezik a Javában ismert static kulcsszó.

- `setId: MutableList<String>` – Egy 2-elemű lista, amelykiosztja a mező azonosítóját. Az első elem egy karakter (a-h), a második elem egy szám (1-8).
- `fun updateIdSetter()`

A karakter ASCII számához hozzáad 1-et, amennyiben eléri a `h` karaktert, visszaállítja `a`-ra és eggyel csökkenti a második karaktert, ami egy szám.



15. ábra Egy egy mező konstans karakterlánc a következőképpen tevődik össze

Data.swift Extension

A `Data.kt` és a `Data.swift` fájlok egyaránt adatokat tartalmaznak. Itt tárolódnak a FEN pozíciók konstansai, amiket különböző tesztelésekhez használtunk, az `Enum` osztályunk, illetve a nyelvek sajátosságaival itt foglalkozunk. A Swift programozási nyelv felépítése végett bővítményeket (Extension) használnak a fejlesztők. Ebben a fejezetben ezeket a bővítményeket tárgyalom.

STRING OSZTÁLY

- `Substring`

Egy `String` karakterlánc valahány elemét adja vissza. Ez a sok nyelvben implementált `String`-eken való `substring` művelet. Ezt egy `subscript`-el valósítottam meg, így ha egy `str` nevű változónak valahány elemét akarjuk visszakapni, a következőképpen néz ki a kódunk: `str[x..<y]`, ahol „x” és „y” számok, és az előbbi kezdő, utóbbi pedig végső helyi értéke a szövegrészletnek.

CHARACTER OSZTÁLY

- `inc()`

Egy karakter ASCII értékét növeli 1-el. Az ASCII táblában a kis „a” betűt 97-es érték jelöli, erre meghívva a függvényt, „b” betűt kapunk vissza, aminek 98-as értéke van.

- `dec()`

Egy karakter ASCII értékét csökkenti 1-el. Az ASCII táblában a kis „b” betűt 98-es érték jelöli, erre meghívva a függvényt, „a” betűt kapunk vissza, aminek 97-as értéke van.

ARRAY OSZTÁLY

- `findIndexNS()`

Egy egymásba ágyazott (Nested) tömbből kikeresi, hogy a paraméterül kapott objektum, melyik sorban helyezkedik el, ezt a számot adja vissza.

PIECE OSZTÁLY

- `contains()`

Az első paraméterül kapott tömbből kikeresi a második paraméterül kapott elemet. Ha van ilyen elem a tömbben, akkor igaz, egyébként hamis értéket ad vissza.

4.3 Android UI és nézetek implementálása

Az alkalmazás megjelenítésének implementálása Android Studio-ban készült. Az alkalmazás nézeteit a Google Pixel 2-n teszteltem, így különböző felbontással és dpi-vel rendelkező telefonokon eltérhet a nézet. Az API verziója 25-ös (Android 7.1), ez azt jelenti, hogy régebbi Android verziókon is fut az alkalmazás.

A fejlesztés során a „res>layout” könyvtárban tárolom a megjelenéshez szükséges XML nézeteket. Minden egyes Activity XML fájl egy oldal nézetét tartalmazza. Az `item.xml` állományok egy lista egy-egy példányát reprezentálják, ami lehet a sakktábla egy mezője, vagy egy egyszerű listaelem.

A „res>drawable” könyvtár különböző képeket és ikonokat tartalmaz. Itt található az alkalmazás ikonja [9], illetve a sakktábla és a sakkbábuk [10] is itt kerülnek tárolásra.

A „kotlin” könyvtárakban található az Android UI forráskódja és a shared code, ami a sakk program logikáját tartalmazza. A „kotlin>chess” könyvtár tartalmazza a különböző nézetekhez hozzárendelt kódot. Itt találhatóak a menük és a menüpontok navigációjának kódjai is. Az `OfflineBoardAdapter` a sakktábla és a bábuk vezérlésére és állapotának frissen tartására szolgál, itt kezeljük le az eseményeket. A `PrevMatchRecyclerViewAdapter` azért felel, hogy az előző játszmákat betöltse a program, illetve azok interaktívak legyenek.

A játék UI részét az `InOfflineGameActivity`-ben és az `OfflineBoardAdapter`-ben implementáltam.

`InOfflineGameActivity.class`

VÁLTOZÓK

- spotsRecyclerView: RecyclerView
- promoButtons: List<View>
- gameOverIds: Pair<View, TextView>
- proposals: List<View>
- adapter: OfflineBoardAdapter
- game: Game
- offerScreen: View

FÜGGVÉNYEK

- fun onCreate()

A függvény először összegyűjti a releváns attribútumokat:

- a gyalog előléptetésére szolgáló gombokat és panelt
- a döntetlen felajánlását megjelenítő nézetet, és a hozzá tartozó igen/nem gombokat
- a feladást jelző gombot
- a játék vége nézetet és szöveget, amit aszerint módosítunk, hogy ki nyert

Végül létrehozza a játékot és az adaptert, a listanézetnek pedig átadja a sakktáblát tartalmazó adaptert és rács nézet-et állít be (GridLayoutManager)

- fun btnOffer()

A döntetlen felajánlását tartalmazó nézetet megjeleníti.

- fun btnBackToMain()

Visszairányítja a felhasználót a főmenübe.

OfflineBoardAdapter.class

KONSTRUKTOR

Lekéri a kontextust, a játék objektumot, előléptető gombokat, a játék vége nézetet és szöveget, illetve a döntetlent felajánló nézetet.

VÁLTOZÓK

- promoButtonList: List<View>
- spots: MutableList<Spot>()
- context: Context
- game: Game

- gameOverScreen: View
- gameOverText: TextView
- proposals: List<View>
- allMoves: MutableList<List<Spot>>()
- fromSpot: Spot
- toSpot: Spot
- showPanel: Boolean

ALOSZTÁLY

- class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView)

Ez az osztály tartalmazza egy-egy mező nézetét. A konstruktorban lekéri a res>layout könyvtár spot_item.xml nézetben található állományát. A spot_item.xml fájlban találhatóak a mezők általános paraméterei, mint például a méretük. A child tartalmazza a képet, a spot pedig a relatív nézetet, amelyben a child található.

FÜGGVÉNYEK

- fun onCreateViewHolder()

Az InOfflineGameActivity.kt fájlban található Game osztályból és az in_offline_game_activity.xml fájlban található sakktáblából (mezőkből) létrehoz egy nézetcsoporthoz, amely a játéktáblát alkotja. Ezt visszaadja egy általunk létrehozott ViewHolder osztályleszármazottjaként.

- fun onBindViewHolder()

Itt történik az algoritmus logikájának megírásának lényegi része. Első lépésként kikapcsolom a nézet újrahazsnosítását (ami egyébként a RecyclerView egyik előnye), mert ez a funkció megakadályozza a mezők frissítését. Majd beállítom a nézetet a setLayout függvény segítségével, ezáltal minden bábuval rendelkező mező kap egy, a bábuját ábrázoló képet. Végül 3 eseménykezelőt hívok meg: a gyalog előléptetéséért felelős eseménykezelőt, a gombok-ra való kattintásra vonatkozó eseménykezelőt („Resign” és „Offer draw”) és a mezők eseménykezelőit.

Egy mezőre az onClickListener a következő lépéseket hajtja végre: Először kijelöl egy mezőt a játékos (az interakcióról eldönti a program, hogy saját bábu kijelölése vagy lépés történt), utána frissítjük a játékállást, és a játék végén pedig bejelentjük, hogy ki nyert.

- fun setLayout()

A mezőn található bábu alapján kiválasztja a megfelelő képet, majd beállítja. Hogyha a mező ki van emelve, azaz ha a kiválasztott bábu léphet az adott mezőre, akkor egy sárga, üres belsejű négyzettel színezzük ki a mezőt.

- fun selectImage()

Minden spot.piece.char értékhez hozzárendelünk egy sakk figurát.

- fun promoPanelController()

A showPanel igaz értéke esetén felugrik a gyalog cseréjére szolgáló panel. A játékos kiválaszthatja, hogy melyik bábu szeretné lecserélni a gyalogját. Erre a gyaloggal a játéktábla végén kerül sor. A showPanel értékét hamisra állítja a függvény, ezáltal eltűnik a panel, és folytatható tovább a játék.

- fun selectSpotToMovePromo()

Amennyiben ki van emelve a mező, amelyikre a korábban kiválasztott bábuval szeretne lépni a játékos, hozzárendeljük a mezőt, mint célmező („toSpot”), és megjelenítjük az előléptetésre szolgáló mezőt, azaz a „showPanel” állapot értékét igazra állítjuk.

Ha a kapott érték tartalmazza, hogy mire akarja lecserélni a játékos a bábuját, akkor végre is hajtjuk a lépést és lecseréljük a sakk bábút. Ezután kitöröljük a lépéshez szükséges értékeket és kijelöléseket, hogy egy új lépés következhesen.

- fun proposalController()

A proposalController irányítja, hogy mi lesz, ha a felhasználó megnyomja a „Resign”, illetve az „Offer draw” gombokat. Ha a Resign gombra kattint, akkor feladja a játékot, ha az Offer draw gombra kattintás után a másik játékos elfogadja a döntetlent, akkor a játék véget ér és az eredmény döntetlen lesz.

- fun proposal()

A játék státuszát állítja át, hogyha a játékot nem akarják folytatni a felek, akkor a game.status-t aktívról átállítja.

- fun isGameOver()

Visszaad egy igaz/hamis értéket a játék státuszáról.

- fun announceGameEnd()

Megjeleníti a játék vége nézetet, és kiírja, hogy melyik játékos nyert (döntetlen esetén jelzi a döntetlen állapotát).

- fun move()

Ez a függvény kezeli le a lépéseket. Először kiválaszt a játékos egy bábút, ha a sajátja, akkor azzal kijelöli a lehetséges lépéseit a bábunak, ha nem az övé, de van kijelölve bábuja, akkor megpróbál odalépni. Amennyiben adottak a feltételek a lépés megtörténik. A gyalog

lecserélésére szolgáló lépés egy egyedi eset, ezért ezt az esetet a `selectSpotToMovePromo()` függvényhívással kezeljük le.

- `fun selectSpot()`

A paraméterben kapott mező tartalmazza, hogy melyik bábuval szeretnénk lépni. A bábu összes lehetséges lépését kiemeli, amennyiben van lépése, illetve van engedélye lépni az imént kiemelt bábuval.

- `fun selectSpotToMove()`

Amennyiben a kiválasztott mező ki van emelve, a korábban kiválasztott sakkfigurát erre a mezőre helyezi, azaz végrehajtja a lépést, majd a kijelöléseket pedig törli.

- `fun showPromoPanel()`

Megjeleníti az előléptető panelt.

- `fun hidePromoPanel()`

Eltünteti az előléptető panelt.

Emellett az osztály még tartalmaz egy getter-t, ami a mezők számát adja vissza és egy setter-t, ami beállítja a mezőket és frissíti az adaptert.

4.4 iOS UI és nézetek implementálása

Az alkalmazás megjelenítésének implementálása az Apple sajátkészítésű fejlesztőkörnyezetében készítettem el, Xcode-ban. Az alkalmazás nézeteit a iPhone 11-re teszteltem, így különböző felbontással és mérettel rendelkező telefonokon eltérhet a nézet. Az Apple iOS 15.2-es verzióját használtam a teszteléshez.

Az iOS-hez használt kód struktúrája szimplisztikus, egyszerűbb felépítéssel rendelkezik. A projekt 3 részre bontható: „shared code”, „View” és „Assets”.

A shared code tartalmazza a sakk programot, amit Swift-ben implementáltam. Itt található a játék logikája. A View tartalmazza a felhasználói interface-t. Lényegében a menü és a játék nézete itt található, illetve az interakciók implementálása (pl: bábu mozgatása) itt került implementálásra. A játék irányítására szolgáló függvényeket és kódot elkülönítettem a nézetektől egy GameLibrary nevű fájlba, ezáltal egyrészt a nézetek átláthatóbbak, másrészt egy letisztultabb kódázist létrehozva, ami a játék esetleges továbbfejlesztését könnyíti meg. Az Assets tartalmazza azokat a képfájlokat és ikonokat, amiket az applikációhoz használtam fel.

A shared code-ot tartalmazó kódázisról a fejlesztői dokumentáció 4.1. fejezetében található leírás.

ÁLLAPOTOK

- highlight: Array<Color>
- currentColor: String
- toggle: Bool
- toggle2: Bool
- allMoves: Array<Array<Spot>>
- fromSpot: String
- toSpot: String
- showPanel: Bool
- gameOver: Bool
- gameOverText: String
- offerDraw: Bool

EGYÉB VÁLTOZÓK

- columns: Array<GridItem>
- game: Game

NÉZET

Az inicializáláskor hozzuk létre a játék osztályt és hívjuk meg a „getCurrentColor” függvényt, aminek segítségével kiírjuk, hogy melyik játékos lépése következik. Ez először mindig a fehér lesz.

2 gomb található a nézetben. Az egyik a Resign, amivel a jelenlegi játékos feladja a játékot, ezáltal vége a játéknak. A másik az Offer draw, amivel felugrik a döntetlen elfogadásáról szóló panel a másik játékosnak. Ha a játékos elfogadja a döntetlent, akkor a játéknak vége, döntetlennel zárul. A gombok ki vannak kapcsolva, amennyiben a gyalog előléptetésére létrehozott panel aktív, vége a játéknak, vagy a döntetlenről kell döntenie egy játékosnak.

A következő elem egy szöveg, amely jelzi, hogy melyik játékosnak kell lépnie. Ez minden lépés után frissül.

A szöveg alatt a játéktábla található. Itt helyezkednek el a mezők, melyeken bábuk találhatóak, amikkel interakciót végezhetünk. Ha egy saját bábuva kattintunk, azt kiválasztjuk, ezáltal kiemelődnek a mezők, amellyekre léphetünk a bábuval. Ha egy ilyen kiemelt mezőre nyomunk, akkor odamozgatjuk a sakk bábunkat. Az interakció után frissítjük a játék állapotát.

FÜGGVÉNYEK

- func proposal()

A proposalController irányítja, hogy mi lesz, ha a felhasználó megnyomja a „Resign”, illetve az „Offer draw” gombokat. Ha a Resign gombra kattint, akkor feladja a játékot, ha az Offer draw gombra kattintás után a másik játékos elfogadja a döntetlent, akkor a játék véget ér és az eredmény döntetlen lesz.

- func promotion()

A showPanel igaz értéke esetén felugrik a gyalog cseréjére szolgáló panel. A játékos kiválaszthatja, hogy melyik bábra szeretné lecserélni a gyalogját. Erre a gyaloggal a játéktábla végén kerül sor. A showPanel értékét hamisra állítja a függvény, ezáltal eltűnik a panel, és folytatható tovább a játék.

- func announceGameEnd()

Megjeleníti a játék vége nézetet, és kiírja, hogy melyik játékos nyert (döntetlen esetén jelzi a döntetlen állapotát).

- func isGameOver()

Visszaad egy igaz/hamis értéket a játék státuszáról.

- func updateGameState()

A játék nézetében 2 változást figyelő állapot található.

A toggle állapot akkor frissül, ha lépett egy játékos. Ezáltal update-eli a jelenlegi játékos kiírást, ami jelzi, hogy kinek a lépése következik.

A toggle2 állapot akkor frissül, ha új bábra nyomunk. Ezáltal frissül a játéktábla is és kirajzolódnak a bábu lehetséges lépései a játéktáblára.

- func getCurrentColor()

Visszaadja egy tájékoztató szöveget, amely tartalmazza, hogy jelenleg melyik játékos van. 2-féle szöveget adhat vissza:

1. „Current player: white”
2. „Current player: black”

- func selectSpot()

A paraméterben kapott mező tartalmazza, hogy melyik bábuval szeretnénk lépni. A bábu összes lehetséges lépését kiemeli, amennyiben van lépése, illetve van engedélye lépni az imént kiemelt bábuval.

- `func selectSpotToMovePromo()`

Amennyiben ki van emelve a mező, amelyikre a korábban kiválasztott bábuval szeretne lépni a játékos, hozzárendeljük a mezőt, mint célmező („toSpot”), és megjelenítjük az előléptetésre szolgáló mezőt, azaz a „showPanel” állapot értékét igazra állítjuk.

- `func selectSpotToMove()`

Amennyiben a kiválasztott mező ki van emelve, a korábban kiválasztott sakkfigurát erre a mezőre helyezi, azaz végrehajtja a lépést, majd a kijelöléseket pedig törli.

- `func setLayout()`

Meghívja a `selectImage` függvényt, és képpé alakítja a választott sakkfigurát. Ezt a képet vissza is adja.

- `func selectImage()`

Minden `spot.piece.char` értékhez hozzárendelünk egy sakk figurát.

4.5 Webszerver és kliens implementálása

A webszervert és a klienst a Node JS-ben Socket.io library [11] segítségével implementáltam le. A Socket.io-val alacsony késleltetésű, kétirányú és eseményalapú kommunikációt tesztek lehetővé a kliens és a szerver között. A szerveren 2 játékos kapcsolattartását valósítom meg. A localhost 3000-es portján fut.

Szerver

A `server.js` a szerver funkcionalitását, a websocket-ek küldését és fogadását tartalmazza. Itt kezelem le az eseményeket és a szerver felé érkező kéréseket, innen indítom a kliensek felé érkező kéréseket, tárolom és kezelem az algoritmus szempontjából releváns adatokat. A `handler.js` tartalma leszűkül a tárolás és adatok kezelésének hogyanjára. Függvényeket és függvénydefiníciókat tartalmaz, amiket a `server.js`-ben használlok.

SERVER.JS

Kliens által küldhető kérések:

- `create-room`

Ezzel a kéréssel lehet új szobát létrehozni. A kliens automatikusan hozzáadódik a szobához.

- join-room

Ezzel a kéressel lehet csatlakozni egy szobához. Meg kell adni a szoba azonosítóját is a csatlakozáshoz. Ekkor a szoba állapota zárt lesz, elindul a játék, és egy kérést küld mindkét 2 kliensnek, ami a játék során használt színt tartalmazza. A szoba létrehozója lesz mindig a fehér színű játékos.

- leave-room

Ezzel a kéressel lehet elhagyni a szobát. A kliens, a programból való kilépése esetén ugyanúgy törlésre kerül a szobából.

- sync-action

Ezzel a kéressel a kliens akcióját szinkronizálja. A kérés tartalmazza a végrehajtott eseményt is. Minden lépés után érkezik egy ilyen kérés a szerver felé. Ilyen akció a lépés, a döntetlen felajánlása és a feladás is. A szerver egy action-sent kérést küld a másik játékosnak, hogy update-elje a másik játékos állapotát is. Tehát miután lépett az „A” játékos, ezzel a kéressel a „B” játékos állapotát szinkronizálja.

- send-message

Ezzel a kéressel 2 kliens üzenetet küldhet egymásnak. Az üzenet tartalmát tartalmazza a kérés. A szerver egy message kérést közvetít a szoba mindkét tagjának, és kiírja a konzolra az üzenetet.

Szerver által küldhető kérések:

- room-is-full

A szobába való csatlakozás esetén, ha megtelik a szoba (2 fő fog bent tartózkodni), akkor a szobában tartózkodó kliensek room-is-full kérést kapnak, ami tartalmazza a játékosok színeit. Ezzel a kéressel indul el a játék.

- action-sent

Ez a kérés hajtja végre a másik kliens akcióját.

- message

Itt íródik ki az üzenet a 2 kliens számára.

HANDLER.JS

Objektumok:

- Random

Ez az objektum tartalmazza a véletlenszerű 4 számjegyű szobaazonosító generálását.

- Room

Ez az objektum tartalmazza a listát, amiben tárolom a szoba azonosítóját, a játékosok socket azonosítóját, és az állapotát. Az állapot nyílt és zárt lehet, ami azt befolyásolja, hogy lehet e csatlakozni a szobához. A szobák kezelésére szolgáló függvényeket itt implementáltam.

Kliens

A kliens egy demo változata érhető el. Ezt szintén node js-ben implementáltam, viszont a kész kliensoldalnak az applikációhoz való integrálásához szükséges Kotlinban és Swiftben is implementálnom, az Android/iOS UI-e alatt. Létezik library ezekre a nyelvekre is [12].

Ebben a verzióban az események a konzolra íródnak ki.

4.6 Tesztelés

A tesztelésnél Android és iOS környezetben is ugyanazokat a unit tesztek végzettem el. Ezért továbbra is nem szedem ketté a Tesztelés fejezetét, mindkét esetben ugyanazokat az ellenőrzéseket végzem.

A tesztelés során Kotlin-ban JUnit5-öt használtam, ennek előnyeit felhasználva belső osztályokkal csoportosítottam a teszteseteket, ezáltal egy-egy tesztelő osztály tartalmaz belső osztályokat, amelyek nevei megegyeznek azzal a metódussal, amelyiket tesztelem. Belső osztályok nem léteznek XCTest-ben, ezért itt kommentekkel tudtam növelni az átláthatóságot. Ezen kívül Kotlinban az átláthatóság érdekében a tesztelésre létrehozott metódusok nevei szóközzel vannak elválasztva és rövid leírásokat tartalmaznak arról, hogy mi célt szolgál a metódus léte. Ez Swift-ben alsóköötőjellel van helyettesítve.

Tesztesetek száma: 244

BoardTest.class

A BoardTest osztály a Board osztály tesztelésére van létrehozva. Itt vizsgálom például a sakk játék különböző attribútumait, azaz a játéktáblának egy-egy elemének visszaadását, a lépések, a sáncolás és a jelenlegi játékost frissítem itt.

A FEN jelöléseket a PerseFEN alosztályban ellenőrzöm. Egy FEN jelölés 6 különböző részből tevődnek össze és a tesztelésnél is fontos szerepe van különböző pozíciók gyors beállításához, ezért kifejezetten fontos a függvény megfelelő működése.

A `getSpotById(id)` függvényt a tesztelés során sokat használtam, ezért kifejezetten fontos megbizonyosodni róla, hogy jól működik-e.

A tesztelő osztály felépítése az alábbi 16. és 17. ábrákon látható.

✓ Test Results	103 ms
✓ BoardTest	103 ms
✓ ParseFEN	33 ms
✓ check if full moves are correctly set()	18 ms
✓ check if En Passant square is correctly set()	1 ms
✓ check if half moves are correctly set()	0 ms
✓ check if the side is correctly set()	1 ms
✓ check if the castle permission is correctly set()	1 ms
✓ check if the board is correctly set()	12 ms
✓ ReplaceRook	23 ms
✓ check if replacing rook for castling works on white queen side()	22 ms
✓ check if replacing rook for castling works on black queen side()	0 ms
✓ check if replacing rook for castling works on black king side()	0 ms
✓ check if replacing rook for castling works on white king side()	1 ms
✓ IsCastlingMove	3 ms
✓ check if the algorithm recognizes black queen side castling()	1 ms
✓ check if the algorithm recognizes black king side castling()	1 ms
✓ check if the algorithm recognizes white king side castling()	0 ms
✓ check if the algorithm recognizes white queen side castling()	1 ms
✓ IsPromotionMove	2 ms
✓ check if return value true when a promotion move is passed()	1 ms
✓ check if return value false when not a promotion move is passed()	1 ms
✓ PromotePawn	3 ms
✓ check if black pawn gets promoted to bishop()	1 ms
✓ check if black pawn gets promoted to knight()	0 ms
✓ check if white pawn gets promoted to queen()	0 ms
✓ check if black pawn gets promoted to queen()	1 ms
✓ check if white pawn gets promoted to bishop()	1 ms
✓ check if white pawn gets promoted to rook()	0 ms
✓ check if white pawn gets promoted to knight()	0 ms
✓ check if black pawn gets promoted to rook()	0 ms
✓ IsEnPasMove	3 ms
✓ check if En Passant capture move is not accepted when moving with pieces other than pawn()	1 ms
✓ check if En Passant capture move is not accepted when it's not En Passant square()	2 ms
✓ check if En Passant capture move is accepted()	0 ms
✓ CaptureEnPas	12 ms
✓ check if after executing En Passant capture move the enemy's piece disappears()	12 ms
✓ UpdateFullMove	2 ms
✓ check if counter increases by 1 when black moved()	2 ms
✓ check if counter does not increase when white moved()	0 ms

16. ábra Board Test osztály struktúrája

✓ UpdateHalfMove	2 ms
✓ check if counter resets to 0 when pawn moved()	0 ms
✓ check if counter increases by 1 when move was not by pawn and was not a capture move()	1 ms
✓ check if counter resets to 0 when it was a capture move()	1 ms
✓ UpdateCastPerm	3 ms
✓ check if 4 castle permissions are enabled when met with criteria()	0 ms
✓ check if black castling permission is disabled when king moved away()	0 ms
✓ check if white queen side castle permission is disabled when rook moved away()	1 ms
✓ check if black king side castle permission is disabled when rook moved away()	1 ms
✓ check if black queen side castle permission is disabled when when rook moved away()	0 ms
✓ check if white castling permission is disabled when king moved away()	0 ms
✓ check if white king side castle permission is disabled when rook moved away()	0 ms
✓ check if it returns no castling permission when there is no castling permission available()	1 ms
✓ UpdateEnPas	7 ms
✓ check if white player advances with pawn only 1 square there is no En Passant square()	1 ms
✓ check if white player moved with pieces other than pawn there is no En Passant square()	1 ms
✓ check if black player advances with pawn only 1 square there is no En Passant square()	1 ms
✓ check if black player moved with pieces other than pawn there is no En Passant square()	1 ms
✓ check if white player advances with pawn 2 squares there is En Passant square()	3 ms
✓ check if black player advances with pawn 2 squares there is En Passant square()	0 ms
✓ GetSpotById	1 ms
✓ check if function returns the correct spots()	1 ms
✓ GetPosition	0 ms
✓ check if function returns the correct positions()	0 ms
✓ FilterNotSafeMoves	5 ms
✓ check if function returns the moves where the King will be safe()	5 ms
✓ IsKingUnderCheck	2 ms
✓ check if king is in check works()	1 ms
✓ check if king is not in check works()	1 ms
✓ AllSpotsOfPlayer	2 ms
✓ check if it selects all the spots where the black player has a piece()	1 ms
✓ check if it selects all the spots where the white player has a piece()	1 ms

17. ábra Board Test osztály struktúrája

GameTest.class

A GameTest osztály a Game osztály tesztelésére van létrehozva. Az osztályban a konstruktort és azokat a függvényeket tesztelem, amelyek a játék logikájára hatással vannak. Így tehát a konzolra kiíró függvényeket (printBoard, printBoardIDs, showSelectedMoves) kihagyom, ezek csupán a fejlesztés megkönnyítésére, illetve a kód működésének reprezentálására lettek implementálva.

A konstruktorban tesztelem, hogy a játékosok helyesen lettek e beállítva, a lépéslista alapértelmezetten üres e és, hogy a játék státusza aktív e.


A 'GetCurrentPlayer' alosztályban tesztelem, hogy a most lépő játékost adja e vissza.

Az 'IsGameOver' alosztályban tesztelem, hogy a metódus a megfelelő játéktárust adja e vissza.

Az 'IsPromotion' alosztályban tesztelem, hogy a gyalogokat csak akkor cserélhetjük le, mikor az utolsó sorból lépnek az utolsó sorba. Ellenőrzöm, hogy fekete bábú léphet e visszafelé, egyéb bábú lecserélhető e, illetve „nagy” bábuk nem cserélhetőek e le. Nagy bábunak nevezzük az összes, gyalogon kívüli bábút.

Az 'UpdateStatus' alosztályban tesztelem, hogy a játék státuszát mindig korrekten állítja e be a metódus. Patt helyzet esetén döntetlen, ha a fekete ad sakk mattot, akkor a fekete nyer, ha pedig a fehér ad sakk mattot akkor a fehér nyer.

A tesztelő osztály felépítése az alábbi 18. ábrán látható.



✓ Test Results	70 ms
✓ GameTest	70 ms
✓ Constructor	28 ms
✓ check if the move list is empty()	26 ms
✓ check if the players are correctly set()	1 ms
✓ check if the status is active()	1 ms
✓ GetCurrentPlayer	2 ms
✓ check if the player can be set to black()	1 ms
✓ check if the starting player is white()	1 ms
✓ IsGameOver	3 ms
✓ check if the game status is active, the game is not over()	2 ms
✓ check if the game status is not active, the game is over()	1 ms
✓ IsPromotion	33 ms
✓ check if only at the end of the table can a pawn be promoted()	30 ms
✓ check if the black pawn can step back to be promoted()	1 ms
✓ check if a big piece can't be promoted()	1 ms
✓ check if other pieces can't be promoted()	1 ms
✓ UpdateStatus	4 ms
✓ check if the status will be 'Black_Win' at black mate()	2 ms
✓ check if the status will be 'White_Win' at white mate()	1 ms
✓ check if the status will be 'Draw' at stalemate()	1 ms

18. ábra GameTest osztály struktúrája

MoveTest.class

A MoveTest osztály a Move osztály tesztelésére van létrehozva. Mivel az osztály csak egy konstruktort tartalmaz, ezért a MoveTest-en belül is csupán az értékek konzisztenciáját fogom vizsgálni.

Egy 'Constructor' nevű belső osztályban vizsgálom a Move osztály adatait. Azt, hogy átadott játékos és a lekérdezett játékos egyenlő e, a honnan és a hova lépések a megfelelő értéket mutatják e, és a gyalog cseréje esetén a megfelelő karaktert tárolja e az osztály. A tesztelő osztály felépítése az alábbi 19. ábrán látható.

✓ Test Results	32 ms
✓ MoveTest	32 ms
✓ Constructor	32 ms
✓ check if the 'promotion' value is correct()	31 ms
✓ check if the 'from' value is correct()	0 ms
✓ check if the 'to' value is correct()	0 ms
✓ check if the player is set()	1 ms

19. ábra MoveTest osztály struktúrája

PieceTest.class

A PieceTest osztály a Piece kotlin fájl tesztelésére van létrehozva. A fájlban található a Piece absztrakt osztály, és az ebből leszármaztatott osztályok, amelyeket a programomban használok: Pawn, King, Queen, Rook, Bishop, Knight és Empty. A PieceTest osztályban, ezáltal nem az absztrakt Piece-t, hanem a belőle leszármaztatott osztályokat tesztelem.

Minden egyes leszármaztatott osztályhoz létrehoztam egy-egy teszt alosztályt. Ezekben az alosztályokban a konstruktort, a lehetséges lépéseket (PossibleMoves) és a lehetséges elfogólépéseket (PossibleCaptureMoves) tesztelem.

A PossibleMoves minden olyan lépés, amely nem számít ütésnek. Itt tesztelek olyan eseteket, amikor ellenőrzöm, hogy a bábu nem tud-e kimenni a pályáról, vagy nem tud-e rálépni vagy átlépni bábukát. Az Empty osztály alapértelmezetten üres listát ad vissza.

A PossibleCaptureMoves minden olyan lépés, amely ütést fejez ki. Itt tesztelek olyan eseteket, amikor ellenőrzöm, hogy a bábuval csak az ellenfél bábuját ütheti le. Az Empty osztály alapértelmezetten üres listát ad vissza.

A tesztelő osztály felépítése az alábbi 20. és 21. ábrákon látható.

✓ Test Results	81 ms
✓ PieceTest	81 ms
✓ King	59 ms
✓ Constructor	28 ms
✓ check if the character of white king is 'K'()	27 ms
✓ check if the character of black king is 'k'()	1 ms
✓ PossibleMoves	30 ms
✓ check if castle permission works well()	26 ms
✓ check if king can step without going out of table()	1 ms
✓ check if king can step without hitting any enemy piece()	1 ms
✓ check if king can step without hitting any friendly piece()	2 ms
✓ PossibleCaptureMoves	1 ms
✓ check if king can not capture friendly pieces()	0 ms
✓ check if king can capture enemy pieces()	1 ms
✓ Queen	4 ms
✓ Constructor	1 ms
✓ check if the character of white queen is 'Q'()	1 ms
✓ check if the character of black queen is 'q'()	0 ms
✓ PossibleMoves	1 ms
✓ check if queen can go horizontal, vertical and diagonal without going through or hitting any friendly piece()	0 ms
✓ check if queen can go horizontal, vertical and diagonal without going out of table()	1 ms
✓ check if queen can go horizontal, vertical and diagonal without going through or hitting any enemy piece()	0 ms
✓ PossibleCaptureMoves	2 ms
✓ check if bishop can not capture friendly pieces()	1 ms
✓ check if bishop can capture enemy pieces()	1 ms
✓ Rook	5 ms
✓ Constructor	2 ms
✓ check if the character of white rook is 'R'()	1 ms
✓ check if the character of black rook is 'r'()	1 ms
✓ PossibleMoves	3 ms
✓ check if rook can go horizontal and vertical without going through or hitting any enemy piece()	1 ms
✓ check if rook can go horizontal and vertical without going through or hitting any friendly piece()	1 ms
✓ check if rook can go horizontal and vertical without going out of table()	1 ms
✓ PossibleCaptureMoves	0 ms
✓ check if bishop can not capture friendly pieces()	0 ms
✓ check if bishop can capture enemy pieces()	0 ms

20. ábra PieceTest osztály struktúrája

✓ Knight	6 ms
✓ Constructor	2 ms
✓ check if the character of black knight is 'n'()	1 ms
✓ check if the character of white knight is 'N'()	1 ms
✓ PossibleMoves	3 ms
✓ check if knight can jump in an L shape without hitting any enemy piece()	1 ms
✓ check if knight can jump in an L shape without going out of table()	1 ms
✓ check if knight can jump in an L shape without hitting any friendly piece()	1 ms
✓ PossibleCaptureMoves	1 ms
✓ check if knight can capture enemy pieces()	1 ms
✓ check if knight can not capture friendly pieces()	0 ms
✓ Bishop	4 ms
✓ Constructor	1 ms
✓ check if the character of black bishop is 'b'()	1 ms
✓ check if the character of white bishop is 'B'()	0 ms
✓ PossibleMoves	2 ms
✓ check if bishop can go diagonal without going out of table()	0 ms
✓ check if bishop can go diagonal without going through or hitting any enemy piece()	1 ms
✓ check if bishop can go diagonal without going through or hitting any friendly piece()	1 ms
✓ PossibleCaptureMoves	1 ms
✓ check if bishop can not capture friendly pieces()	0 ms
✓ check if bishop can capture enemy pieces()	1 ms
✓ Pawn	0 ms
✓ Constructor	0 ms
✓ check if the character of black pawn is 'p'()	0 ms
✓ check if the character of white pawn is 'P'()	0 ms
✓ PossibleMoves	0 ms
✓ check if pawn can advance 2 steps forward from starting position()	0 ms
✓ check if pawn can advance only 1 square from every other spots than the starting position()	0 ms
✓ check if pawn can not advance when blocked from front()	0 ms
✓ check if pawn can advance 1 step when blocked from the 2nd square()	0 ms
✓ PossibleCaptureMoves	0 ms
✓ check if pawn can't attack forward()	0 ms
✓ check if left diagonal capture move works()	0 ms
✓ check if the En Passant square is capturable()	0 ms
✓ check if right diagonal capture move works()	0 ms
✓ Empty	2 ms
✓ Constructor	2 ms
✓ check if the empty piece's character is empty()	1 ms
✓ check if the possible move list is empty()	1 ms

21. ábra PieceTest osztály struktúrája

PlayerTest.class

A PlayerTest osztály a Player osztály tesztelésére van létrehozva. Itt csak a konstruktort ellenőrzöm, hogy az értékek konzisztensek maradnak e a háttérben.

A ``check if the constructor works well`` függvényben meghívom a Player osztályt és az `'isWhite'` argumentumnak átadok először egy igaz, utána pedig egy hamis értéket, majd lekérdezem.

A tesztelő osztály felépítése az alábbi 22. ábrán látható.



✓ Test Results	32 ms
✓ PlayerTest	32 ms
✓ Constructor	32 ms
✓ check if the constructor works well()	32 ms

22. ábra PlayerTest osztály struktúrája

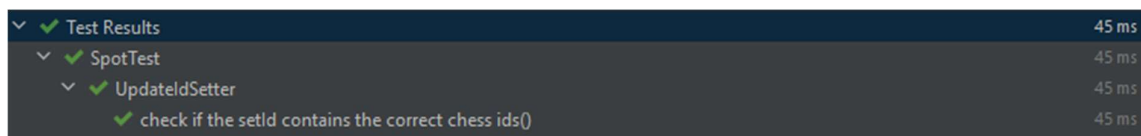
SpotTest.class

A SpotTest osztály a Spot osztály tesztelésére van létrehozva. Az `'updateIdSetter'` metódust működését ellenőrzöm.

A ``check if the setId contains the correct chess ids`` függvényben vizsgálom, hogy a megfelelő azonosítókat kapják meg a mezők. Eredetileg 64 mezőt tartalmaz egy sakktábla ezért ennyi mezőt hozok létre. Az elvárt eredményt manuálisan táplálom egy listába és ellenőrzöm, hogy az egymás után következő azonosítók a helyes sorrendben jönnek e. Az azonosítók felépítése:

a8, b8, ..., h8,
a7, b7, ..., h7,
...,
a1, b1, ..., h1

A tesztelő osztály felépítése az alábbi 23. ábrán látható.



✓ Test Results	45 ms
✓ SpotTest	45 ms
✓ UpdateIdSetter	45 ms
✓ check if the setId contains the correct chess ids()	45 ms

23. ábra SpotTest osztály struktúrája

5. Összefoglalás

Az egyetemi tanulmányaim alatt rengeteg hasznos dolgot tanultam, amit a későbbiekben programozóként hasznosítani fogok tudni. Mivel mindigis szerettem tanulni, és új ismereteket szerezni, ezért egyértelmű volt számomra, hogy olyan témát választok, amelyben valami újdonságot, valami olyan dolgot alkotok, melyet egyetemen nem volt lehetőségem tanulni. Ettől függetlenül hatalmas segítség volt számomra az, hogy részt vettem az egyetemi webprogramozás kurzusokon, amik alapot adtak a mobilra való fejlesztéshez, illetve a szakmai gyakorlatom elvégzése pedig tapasztalatot adott arra, hogy a szakdolgozatomban is hasonló alapossággal járjak el.

A szakdolgozatom rávilágított arra, hogy egy saját projekt, legyen az startup alapítása, vagy akármilyen más egyedi hobbiprojekt, mennyi munkát rejt magában. Mikor elgondoltam a szakdolgozatom témáját, azt hittem, hogy minden a tervek szerint működhet és könnyedén elkészítem az applikációmát bőven határidőn belül. A Kotlin Multiplatform Mobile Plugin-hez léteznek nagyon érdekes tutorial-ok, amelyek rendeltetésszerűen működnek, és bár sok problémát oldottam meg a szakdolgozatomban, a multiplatformra való implementálást máshogyan kellett megközelítenem, mivel sem a tutorial-ok segítségével, sem magamtól nem sikerült vele érdemi projektet létrehoznom.

Sikerélménnyel tölt el, hogy a játékot sikeresen elkészítettem, és a szakdolgozatomat teljesítettem, emellett azt is megtanultam, hogy az apró rejtett problémák összessége nagy hatással van a kiadás dátumára. Ezáltal is tapasztaltabban kerülök ki az egyetemről, és a jövőben jobban ráláthatok arra, hogy egy egyszerűnek tűnő feladat megoldása is mennyi munkával járhat.

6. További fejlesztési lehetőségek

- A Webszerver tárolhatná az adatokat egy adatbázisban és onnan be is olvashatná, ahelyett hogy csak a memóriában tárolódnának.
- A barátlista és a ranglista egy adatbázishoz lehetne integrálva.
- Globális chat funkció hasznos lenne arra, hogy emberek keressenek/kereshessenek maguknak ellenfeleket az applikáción belül.
- Egy match maker beépítése azért hasznos, mert véletlenszerűen összesorsolhatna egy másik játékosal, így nem kellene üzeneteket küldeni egy játék elindításához.

7. Források

- [1] Ábra 1. forrása, illetve a KMM Plugin ismertetője: https://kotlinlang.org/lp/mobile/?utm_medium=link&utm_source=product&utm_campaign=ASKMMI&utm_content=0.3.2
- [2] iOS integráció tutorial: <https://kotlinlang.org/docs/kmm-integrate-in-existing-app.html>
- [3] iOS fejlesztői követelmények: <https://aws.amazon.com/mobile/mobile-application-development/native/ios/#:~:text=To%20develop%20iOS%20apps%2C%20you,use%20to%20write%20iOS%20apps.>
- [4] KMM: <https://plugins.jetbrains.com/plugin/14936-kotlin-multiplatform-mobile>
- [5] Kotlin WebSocket: <https://kotlinlang.org/api/latest/jvm/stdlib/org.w3c.dom/-web-socket/>
- [6] Chat program WebSocketsen (Node.js): <https://socket.io/get-started/chat>
- [7] UCI jelölés: https://en.wikipedia.org/wiki/Universal_Chess_Interface
- [8] FEN jelölés: https://en.wikipedia.org/wiki/Forsyth-Edwards_Notation
- [9] App ikon: https://www.flaticon.com/free-icon/chess-board_3522641
- [10] Sakk bábuk: https://commons.wikimedia.org/wiki/Category:PNG_chess_pieces/Standard_transparent
- [11] Socket.io weboldala: <https://socket.io>
- [12] Socket.io library-k implementálása különböző nyelveken: <https://socket.io/docs/v4/>