**Szakdolgozat**

**Gábris Attila**

**Budapest 2023**

**Eötvös Loránd Tudomány Egyetem**

**Informatika Kar**

**Programtervező informatikus szak**

**Nappali tagozat**

**Sakk Dolgozat**

Konzulens:

**Dr. Kozsik Tamás**

Phd, docens

Készítette:

**Gábris Attila**

Informatika hallgató

**Budapest**

**2023**

Köszönet nyilvánítás

Tartalom

# Bevezetés

Ahogyan az a témabejelentőben is olvasható egy sakk program elkészítésére vállalkoztam.

Ebben elsősorban a feladatban rejlő informatikai és matematikai kihívás vonzott, másodsorban pedig szerettem volna tisztelegni az eddigi sakkozói múltam előtt.

# Felhasználói dokumentáció

## Célközönség

Bárki aki szeretne egy jót sakkozni. Természetesen tisztában kell lenni azzal, hogy ez a játék valószínűleg nem kerül nagyobb nézőközönség elé, így ha szűkebb csoportot akarok megnevezni akkor szerintem a játék gondolkodására kifejlesztett algoritmusom felkeltheti néhány inkább matematikával, modellezéssel foglalkozó szakember figyelmét.

## Üzembe helyezés

<https://github.com/gabrisattila/Chess.git> . Ennek a linknek a birtokában bárki élvezheti a játék nyújtotta élvezeteket. A szükséges attribútumok egy telepített jdk 19, min. 4 GB RAM a megfelelő minőség megtartása végett és 1 MB szabad tárhely. A legegyszerűbb üzembe helyezés a következő képen zajlik:

* A fenti linken elérhető a github repository-t, ezt a Code<> gombra kattintás után letölthető az összecsomagolt file. (.zip állomány)
* Az ezutáni futtatáshoz két út is vezethet. Az egyik megoldásban a játékos saját kezűleg fordítja majd futtatja a programot. Amennyiben megvan hozzá a telepített jdk környezet. (min. java 19) A project mappán belül a Main.java fájl elérési útja a következő: src\main\java\classes\Main.java Ezt fordítva majd futtatva minden gond nélkül igénybe vehető a játék nyújtotta kihívás. Windows operációs rendszer alatt annyi előnye van a játékosnak, hogy a mappában található egy chess.exe file, aminek a futtatásával megspórolja ezeket a lépéseket.
* A másik opció a java fejlesztő környezeten keresztül történő futtatás. Tudom ajánlani az Intellij Idea környezetet mely a piacon szereplő legfelhasználó barátabb ide. Letöltése után nyissa meg a letöltött .zip-ben található mappát mint IntelliJ projekt fájlt és a további szükséges beállításokat bízza rá az ide-re.

## Felhasználási útmutató

A játéknak alapvetően három lehetséges használati mód van. Első sorban mérkőzés a számítógép ellen, nézni ahogy az önmaga ellen játszik, teszt módban felrakni különböző állásokat, különböző méretű táblákon. Játszani csak a sakk szabályainak\* megfelelően lehet. Az játéknak nincs kicsinyített ablakos változata.

## Felhasználói utak

A felhasználónak az első amit ki kell választania az a játék mód, amit használni akar. (Játék, Ai vs Ai, Teszt) Alább részletesen olvasható ezeknek az opcióknak a részletes leírása.

### Játék

Ezt a funkciót választva a játékosnak először választania kell, hogy sötéttel vagy világossal szeretne játszani. Vagy folytatja egy már megkezdett játszmáját. Ez utóbbi esetben megjelenik az elmentett állások listája, amelyből lehet választani. Miután ez megtörtént a játéktáblán megjelenik az állás és mindent lehet ugyanolyan formában folytatni ahol abba kellett hagyni.

Ha nem ezt választjuk egy új játék kezdődik amelyben a választott színnel játszunk. (TODO Ha lesz óra ide beszúrni) A meccs során lehetőség nyílik a pozíció elmentésére (mentés gomb) (ha lesz akkor az órák megállítására) az adott mérkőzés újrakezdésére, (óra újraindul, nincs szín választás) új játék indítására, ekkor nem csak az óra indul újra hanem ismételten kell oldal választanunk és kilépésre is az adott játék funkcióból. (Kilépés gomb) Továbbá természetesen a játék feladására és döntetlen ajánlására is.

### Ai vs Ai

Ebben az esetben gyakorlatilag nem kell mást tennünk mint hátradőlve nézni ahogy a gép saját maga ellen alkalmazza a stratégiáját. Ebből a lehetőségből is ki lehet természetesen lépni, megállítani azt, elmenteni a megállított játszmát.

### Teszt

Itt pedig egy előre meghatározott méretű táblán, tetszőlegesen felrakott állásban tudunk kedvünk szerint ide-oda rakni figurákat, tetszés szerint elemezni az azok által kialakított helyzetet. Alkalom nyílik arra, hogy egy elmentett állást végig gondoljunk.

# Fejlesztői dokumentáció

## Bevezetés

Az alap ötlet és az azt megvalósító megoldás három alapprobléma köré épül. Amelyek a következők:

* Az exponenciális műveletigényű algoritmus - mely az alapját képezi az AI-nak - hatékony megvalósítása.
* A számítások hirtelen növekedése miatt ugyanúgy hirtelen keletkező nagymennyiségű adat tárolása, hatékony elemzése.
* Végül pedig egy olyan felhasználói felület létrehozása mely a fentiek ellenére képes a játékosnak élvezetes játékélmény nyújtani.

Vegyük ezeket sorra.

Először is a későbbiekhez is elengedhetetlen ismerni a minimax algoritmust\*. Ugyanis az AI gondolkodásának ez képezi az alapját. Bármilyen állásból kiindulva a megfelelő információk (melyik oldallal kell gondolkodnia, em-passant lépés lehetőség van-e jelenleg, melyik oldal(ak)ra sáncolhat a királya, sakkban van-e) és a szabályok ismeretében, legenerálja az adott pozícióból keletkezhető összes másikat. Ez eléggé könnyen belátható időn belül a lehetőségek számának már csak becslésétől is kombinatorikus robbanáshoz vezethet. Szükséges tehát egy értékelő algoritmus mellyel adott - emberi szemmel triviálisan - felesleges állások további végigszámolásától kíméljük meg a számítógépet. Erről bővebben a … fejezetben írok

Az másodikra már előttem is készítettek megoldást. Erre kutatásaim közben, hál’ Istennek, rábukkantam. Ez az úgynevezett FEN string\* melyben egyetlen karakterláncon belül tudjuk eltárolni az összes szükséges információt adott játék adott állásáról. Ennek hasznáról szintén később lesz terjengősebben szó.

A harmadik problémán történő törpölés adja meg a választ arra miért is a Java nyelvet választottam. Az eddigiek alapján szerintem látható, hogy ha azt akarjuk, hogy a játékélmény ne az adott számítógép különböző vonatkozó kapacitásainak függvényében alakuljon, kettő vagy három párhuzamosan futó szálat kell beépítenünk a programba. Ezt, bár természetesen megoldhatnánk más programozási nyelvben is, a tanulmányaim során ennek a nyelvnek a szálkezelő módszereivel ismerkedtem meg leginkább. (TODO Ide valami hathatós érvet megkérdezni még)

## Megoldási Terv

* Az minimax és annak vágóalgoritmusa.
  + Miért kell szálakban gondolkodnunk?
  + Thread fogalma
  + Gyakorlati jelentősége
  + Jelenleg tapasztalni kívánt előnye)
  + A döntési algoritmus miben léte, az ötlet kiindulási alapja (rizsa about the eredet of sakk and how it’s corelates with my idea (Apa szövegének felvezetése))
* Miért használok technically singleton megvalósítást.
  + Mi is a singleton. (Singleton definition)
  + Annak jelentősége az exponenciálisan növekvő adatmennyiség leküzdésében.
  + Ehhez kapcsolódóan a körvonalazódó osztály architektúra.
  + Milyen szálakhoz kapcsolódó egyéb szükséges elemeket hívok még segítségül?
    - BlockingQueue definition
* Az alap megjelenítés
  + Annak kapcsolata a szálakkal
  + Invoke later

## Megvalósítás

### Szálak

A program szálakra bontása szükséges volt, mert ha szekvenciálisan dolgoznánk ki a megoldást az elvezet ahhoz a problémához ami a következő módon jelentkezik: kiválasztok, majd lépek egyet a figurával a készített GUI felületen. Ekkor a program agyában ez tudatosul és el is kezdi számolni mi lehet a megfelelő válaszlépés. De ezalatt a játékos nem lát semmilyen változást a táblán, a lépés vizualizációja elmarad. Ez azért van mert a java swing-ben megvalósított GUI by default külön szálon fut, és nem végzi el a repaint műveletet egészen addig míg az összes általa meghívott plusz művelet le nem zárul. Ez esetünkben pedig ugye az általában gigantikus méreteket öltő minimax algoritmus. Így szakszerű átgondolás után adódik az ötlet, hogy a programot legalább kettő, vagy több párhuzamosított szállal valósítsuk meg. Ennek megfelelően lesz egy EDT azaz Event Dispatch Thread osztályom, mely kezeli a grafikus interfész változásait, és csak a megfelelő műveletek lezajlása után ütemezi be a gép gondolkodását implementáló szálat. Kiemelt fontosságnak örvend, hogy az AI elindítása az invokeLater\* metóduson keresztül történik, mely garantálja, hogy a kalkuláció az után induljon el, miután a repaint metódus már megtörtént.

TODO Tehát milyen szálak vannak, mik is azok a szálak (def.)

### Folyamat diagrammok

A képek alapján szépen megkonstruált diagrammok, majd azok részletes végigvezetett magyarázata.

A képen szöveg, számítógép, Megjelenítőeszköz, multimédia látható

Automatikusan generált leírásA képen szöveg, Megjelenítőeszköz, számítógép, multimédia látható

Automatikusan generált leírás

### Komponensek

A főbb komponensek megalkotása közben a vezérlő elvem az egyszerűség volt. Kettő fő komponenst különítünk el. Az egyiket pedig három alrészre bontjuk. Ez a két fő rész a megjelenítési és a logikai réteg. Az utóbbi tartalmazza alap struktúrát jelentő ősosztályokat, azok általánosabb leírását, műveleteiket, egy külön egységet, melyben a projekten belül bárhol felhasználható változókat, metódusokat szedtem össze, és egy harmadik, az AI megvalósítását végző részt.

### Komponens diagrammok

### Osztály architektúra

A szép, elkülönített működést megvalósító szálakkal ellentétben a projektben szereplő objektumok felettébb egymásra épülőek. Egész egyszerűen azért, mert bármilyen bonyolultan is igyekszünk elképzelni a megvalósítást, a sakk mégis csak egy táblából, a mezőkből melyek a táblán vannak és az azokon álló figurákból áll. Így hát jó ötlet ezeket alapul venni a programunk struktúrájának felépítésekor és megfelelő leszármazási rendszert kialakítani az egy csoportba tartozó típusoknak, legyenek azok akár a megjelenítés, akár a háttérlogika elemei.

TODO beszélni a singleton-okról és a programban betöltött fontos szerepükről

### Osztály diagrammok

### Ai vs Human & Ai vs Ai részletezés

# Tesztelés

## Egységtesztek

### Paraméterezhetőség tesztelése

### Exception-ök tesztelése

### Egységtesztek megfelelő mozgásokra

### Átfogó teszt mélységekre

### Időzítés megfelelő működésének tesztje

# Összegzés