

Széchenyi István Egyetem  
Gépészmérnöki, Informatikai és Villamosmérnöki Kar  
Informatika Tanszék

# **SZAKDOLGOZAT**

**Schneider Krisztián**

**Gazdaságinformatikus BSc szak**

2019



**SZÉCHENYI**  
ISTVÁN  
**EGYETEM**



# **SZAKDOLGOZAT**

## **Pókerezni képes mesterséges intelligencia fejlesztése**

**Schneider Krisztián**

**Gazdaságinformatikus BSc szak**

**2019**

# FELADAT-KIÍRÓ LAP SZAKDOLGOZATHOZ

## Hallgató adatai:

Név: Schneider Krisztián

Neptun-kód: S7M4GR

Szak: Gazdaságinformatikus BSc

Specializáció: -

## Cím: Pókerezni képes mesterséges intelligencia fejlesztése

## Feladatok leírása:

No Limit Texas Hold'em-et játszani képes mesterséges intelligencia fejlesztése. A szoftver a ma létező online pókertermek valamelyikében legyen képes póker asztalok felismerésére, azokon a lapok, illetve az előtte történt akciók, valamint a zsetonok mennyiségének feldolgozására. Ezek figyelembevételével hozzon döntést a megfelelő játékról és a döntést hajtsa is végre.

A megvalósítandó feladat lépései:

- Szakirodalom megismerése robotic process automation (RPA) és pokerrel kapcsolatos mesterséges intelligencia témakörében
- Rendszer-architektúra megtervezése
- Proof-of-concept deszkamodell elkészítése, amely intelligencia nélkül képes a felület navigálására
- Poker mesterséges intelligencia alapjainak kidolgozása, integrálása a korábbi deszkamoddellel
- Teljes rendszer tesztelése, validálása

Győr, 2019. május 17.

---

Dr. Csapó Ádám Balázs  
Egyetemi docens

---

Dr. Hatwágner F. Miklós  
Egyetemi docens, mb tanszékvezető

# SZAKDOLGOZAT ÉRTÉKELŐ LAP

## Hallgató adatai:

név: Schneider Krisztián

Neptun-kód: S7M4GR

szak: Gazdaságinformatikus BSc

specializáció: -

tagozat: nappali

## A szakdolgozat adatai:

cím: Pókerezni képes mesterséges intelligencia fejlesztése

nyelv: magyar

típus: nyilvános

A szakdolgozat bírálatra bocsátható.

## A bíráló:

név:

munkahely:

beosztás:

\_\_\_\_\_  
dátum

\_\_\_\_\_  
Dr. Hatwágner F. Miklós  
Egyetemi docens, mb tanszékvezető

## A bíráló javaslata:

\_\_\_\_\_  
dátum

\_\_\_\_\_  
érdemjegy

\_\_\_\_\_  
Dr. Hatwágner F. Miklós  
Egyetemi docens, mb tanszékvezető

## A belső konzulens javaslata:

\_\_\_\_\_  
dátum

\_\_\_\_\_  
érdemjegy

\_\_\_\_\_  
Dr. Csapó Ádám Balázs  
Egyetemi docens

## A ZVB döntése:

\_\_\_\_\_  
dátum

\_\_\_\_\_  
érdemjegy

\_\_\_\_\_  
aláírás (ZVB elnök)

## NYILATKOZAT

Alulírott, Schneider Krisztián (S7M4GR), Gazdaságinformatikus BSc szakos hallgató kijelentem, hogy a „Pókerezni képes mesterséges intelligencia fejlesztése” című szakdolgozat feladat kidolgozása a saját munkám, abban csak a megjelölt forrásokat, és a megjelölt mértékben használtam fel, az idézés szabályainak megfelelően, a hivatkozások pontos megjelölésével.

Eredményeim saját munkán, számításokon, kutatáson, valós méréseken alapulnak, és a legjobb tudásom szerint hitelesek.

Győr, 2019. május 17.

---

hallgató

# KIVONAT

## Pókerezni képes mesterséges intelligencia fejlesztése

Nagyon sok kutatót foglalkoztat, hogy olyan szoftvereket legyenek képesek létrehozni, amelyek az emberek képességeihez hasonló szinten képesek stratégiai játékokat játszani. A póker kifejezetten sok kihívást tartogat a mesterséges intelligencia szempontjából, ugyanis egy sztochasztikus, nem teljes információjú játék. A dolgozat bemutatja a pókert, annak szabályait és a mögötte rejlő sajátosságait. Ismertetve lesznek ismertebb pókert játszó MI fejlesztések, melyek a maguk idején kimagaslók voltak a területen. Ezt követően bemutatja a tervezés, fejlesztés és a tesztelés folyamatát egy ilyen mesterséges intelligencia létrehozása során. Az alkalmazás elkészítése során használt technológiák a Java nyelv, SQLite és a Tesseract OCR volt. A dolgozatban bemutatott megvalósítás, egy tudásalapú rendszer, amely kombinálja a szabályalapú és a formula alapú döntéshozást. A preflop játék során előre definiált laptartományok alapján hoz döntést a megadott szabályok alapján, azt követően pedig egy formula alapú döntéshozás történik, a nyerési esély kiszámításával és a többi rendelkezésre álló információ használatával. A befejező része a dolgozatnak az alkalmazás teszteléséről szól, valamint az eredmények értékeléséről. A PokerGenius alkalmazás volt használva a valós játékot szimuláló tesztek elvégzéséhez, ahol az elkészített alkalmazás a PokerGenius-nak 8 beépített pokerbot-ja ellen játszott, mely során képes volt folyamatosan megfelelő döntések meghozatalára, sok esetben nyert ezen botok ellen. Összességében az alkalmazás képes volt a tervezettnél megfelelő működésre, hasonló szinten volt képes játszani, mint az ellenfélnek használt MI megoldások.

# **ABSTRACT**

## **Developing a poker playing agent**

Numerous researches have worked on developing software agents which are capable or replicating a humans ability to play strategic games. Poker in particular poses many challenges from an artificial intelligence standpoint due to it being a stochastic, incomplete information game. This thesis describes poker, its rules and the underlying properties and strategies of the game and introduces a number of famous poker agents which at the time of their development, were leading edge solutions. It then moves through the process of architecture design, development and testing of an artificially intelligent poker playing agent. A number of technical development tools were utilised to create the application; these include Java, SQLite and Tesseract. The thesis then describes how the application employs a knowledge based system combining a rule based and formula based approach. A set of pre defined ranges were used for the preflop play and combined with other play information including competitor actions and cards in hand to make decisions based on pre defined rules. A formula based approach is then utilised to calculate the winning chance and make play decisions based on all of the known information. The concluding part of the thesis analyses testing methodologies and overall results of the software agent. Pokergenius was utilised for real world game play testing and against 8 of Pokergeniuses inbuilt players the software agent was able to consistently and accurately make decisions to play poker and in many cases won against a number of the bots. Overall, the application was able to successfully operate as designed.

# TARTALOMJEGYZÉK

1	BEVEZETÉS .....	1
2	PÓKER ÁTTEKINTÉSE .....	3
2.1	Póker kutatócsoportok .....	3
2.1.1	Carnegie Mellon egyetemi kutatócsoport.....	3
2.1.2	University of Alberta Poker Research Group.....	4
2.2	A játék nehézsége az MI szempontjából .....	4
2.2.1	Nem teljes információjú sztochasztikus játék.....	4
2.3	Póker szabályai.....	6
2.3.1	Kezek erőssége .....	6
2.3.2	No Limit Texas Hold'em.....	8
3	PÓKERT JÁTSZANI KÉPES MI LÉTREHOZÁSÁNAK MÓDJAI .....	11
3.1	Felépítésének lehetséges lépései .....	11
3.2	Adatok gyűjtése.....	12
3.2.1	PokerTracker.....	13
3.2.2	HHSmithy adatbázis ( <a href="https://www.hhsmithy.com">https://www.hhsmithy.com</a> ) .....	13
3.2.3	Az adatok tárolása .....	14
3.3	Ellenfelek modellezése .....	14
3.3.1	Sklansky féle osztályozás .....	14
3.4	Kezek értékelése.....	15
3.4.1	Pokersource Poker-Eval .....	16
3.4.2	Foldem Hand Evaluator.....	16
3.5	Odds kalkulálása .....	16
3.5.1	Effektív nyerési esély .....	16
3.6	A póker AI tesztelése.....	17
3.6.1	Meerkat API.....	18



3.6.2	Open Meerkat Poker Testbed .....	18
3.6.3	PokerSnowie .....	18
3.7	Jelenlegi megvalósítási módszerek .....	19
3.7.1	Tudásalapú rendszerek .....	19
3.7.2	Monte-Carlo szimuláció .....	19
3.7.3	Nash egyensúly stratégia .....	20
3.7.4	Evolúciós tanulás .....	20
3.7.5	Eset-alapú döntéshozás (Case Based Reasoning) .....	21
4	AZ ALKALMAZÁS FEJLESZTÉSÉNEK BEMUTATÁSA .....	22
4.1	A felhasznált technológiák ismertetése .....	22
4.1.1	Java .....	22
4.1.2	SQLite .....	24
4.1.3	Tesseract .....	26
4.1.4	JDBC .....	26
4.2	Fejlesztési dokumentáció .....	27
4.2.1	Osztályok .....	30
4.3	Rövid működési összefoglaló .....	62
4.4	Tesztelés .....	63
4.4.1	Eredmények .....	64
5	FELHASZNÁLÓ DOKUMENTÁCIÓ .....	66
5.1	Rendszerkövetelmények .....	66
5.2	PokerGenius beállítása .....	66
5.3	Az alkalmazás elindítása .....	68
6	ÖSSZEFOGLALÁS .....	69
6.1	További fejlesztési lehetőségek .....	69
7	FELHASZNÁLT IRODALOM .....	70
8	Melléklet .....	72

## ÁBRÁK JEGYZÉKE

1. ábra: Online pókerjátékosok száma .....	1
2. ábra: Játékok fajtái .....	5
3. ábra: Royal Flush .....	6
4. ábra: Színsor .....	6
5. ábra: Póker .....	7
6. ábra: Full .....	7
7. ábra: Szín .....	7
8. ábra: Sor .....	7
9. ábra: Drill .....	7
10. ábra: Két pár .....	8
11. ábra: Pár .....	8
12. ábra: Magas lap .....	8
13. ábra: Póker asztal .....	9
14. ábra: PokerTracker .....	13
15. ábra: Játékos típusok .....	15
16. ábra: PokerStove .....	17
17. ábra: PokerSnowie .....	18
18. ábra: Java programok fordítása és futtatása virtuális gépen .....	23
19. ábra: PokerGenius .....	27
20. ábra: Az elkészült alkalmazás osztálydiagrammja .....	29
21. ábra: A rangeHelper() metódus által generált szöveg .....	31
22. ábra: Kidolgozott laptartományok Equilab-ban .....	31
23. ábra: Card osztály .....	35
24. ábra: Hand osztály .....	36
25. ábra: Deck osztály .....	36
26. ábra: Player osztály .....	37
27. ábra: PlayerPlayed osztály .....	38
28. ábra: PlayerAI osztály .....	41
29. ábra: Board osztály .....	43
30. ábra: HandCombination osztály .....	46

31. ábra: Gui osztály .....	50
32. ábra: ScreenReader osztály .....	51
33. ábra: Pénzösszeg példa .....	53
34. ábra: readScreen() metódus folyamatábrája .....	54
35. ábra: Actions osztály .....	56
36. ábra: preflopAction() és flopAction() metódus folyamatábra .....	58
37. ábra: turnAction() és riverAction() folyamatábra .....	59
38. ábra: Profitot mutató grafikon.....	64
39. ábra: Különböző botok eredményei .....	64
40. ábra: Játék kiértékelését mutató grafikonok .....	65
41. ábra: PokerGenius Window fül beállítása .....	66
42. ábra: PokerGenius options fül beállítása .....	67
43. ábra: PokerGenius Options/Background image beállítása .....	67
44. ábra: PokerGenius kitörlendő játékos helye .....	68
45. ábra: PokerBot gui.....	68

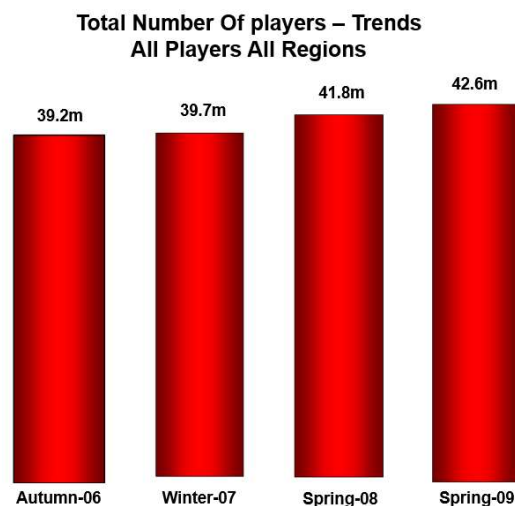
## FORRÁSKÓDOK JEGYZÉKE

1. kódrészlet: Card osztály konstruktora .....	35
2. kódrészlet: equity metódus .....	38
3. kódrészlet: equityInnerCalculation() metódus.....	40
4. kódrészlet: calcPlayerType() metódus .....	42
5. kódrészlet: addCard() metódus .....	43
6. kódrészlet: setBoardType() metódus.....	44
7. kódrészlet: Kombinációk előállítása és értékelése.....	49
8. kódrészlet: findWindow() metódus.....	51
9. kódrészlet: Pot összegét tartalmazó képmetszet készítése .....	52
10. kódrészlet: Játékosok betöltése az adatbázisból .....	55
11. kódrészlet: raisePot() metódus.....	56

# 1 BEVEZETÉS

A stratégiai játékokat játszó mesterséges intelligenciák fejlesztése mindig is különös figyelmet kapott a kutatóktól, és nagyon sok nagyszerű kutató szentelte az idejét az olyan játékok, mint például a sakk tanulmányozásának, és nagyszerű eredményeket is sikerült elérniük ezen a területen, sok játék esetén az emberek képességeit is meghaladó mesterséges intelligenciák létrehozásával. Azonban a sztochasztikus hiányos információval rendelkező játékok nagyon nagy kihívásokat nyújtanak. Az ilyen típusú játékok során, számolni kell különböző fajta problémákkal, ilyenek többek között a kockázat menedzsment, a nem megbízható vagy hiányos információk és a megtévesztés, mely kifejezetten nehezíti ezeket a játékokra a megfelelő játékos modell felépítését.

A kétezres évek óta a póker igencsak nagy népszerűségnek örvend, gyakorlatilag az egész világon ismert játék, és az online pókertermek hatalmas piacot jelentenek a mai napig. A kétezres évek első évtizedének végére, hihetetlen számú online aktív pókerjátékos volt, a számuk a mai napig igen jelentős.[1]



**1. ábra: Online pókerjátékosok száma**

<http://pokerplayersresearch.com/>

Mivel a póker során is csak hiányos információk állnak a játékosok rendelkezésére, egy igen jelentős kihívás lett a mesterséges intelligencia területén, ami motiválta a kutatókat a területen. Nagyon sok különféle megközelítéssel próbálták létrehozni a tökéletes játékot játszó póker játékost, és jelentős áttöréseket is sikerült elérni a területen.

A mesterséges intelligencia területén a kutatás az utóbbi évtized folyamán jelentősen felgyorsult. Az okostelefonok hétköznapivá válásával, a mesterséges intelligencia nagyon jelentős lett minden ember számára.

Szerepük van már szinte az élet minden területén, sokszor a felhasználók nem is tudják milyen számítások vannak egy-egy általuk használt szoftver nyújtotta szolgáltatások mögött. Ha csak GPS-ünk segítségével el szeretnénk jutni valahova, a modern valós idejű forgalomkövető rendszerek komoly szimulációkat végeznek, hogy az adott forgalmi viszonyok mellett mely út a leggyorsabb számunkra. A manapság igen népszerű közösségi média platformokon gyakran találkozhatunk reklámokkal, aminek tartalmát szintén egy mesterséges intelligencia határozza meg, az adott felhasználóra szabottan, ami számára valószínűleg figyelemfelkeltő lesz. Sok ajánlási rendszeren alapuló szolgáltatás is használja, az online piacterek a vásárlási szokásaink alapján jelenítenek meg nekünk termékeket, a különböző média streaming szolgáltatók pedig a felhasználó által fogyasztott tartalmak alapján állítják össze a számára érdekesnek szánt további javaslataikat. Újabbnak számító területként megjelentek például az autonóm járművek fejlesztésével foglalkozó vállalatok, ami jelenleg talán a leginkább finanszírozott szegmense az MI fejlesztéseknek.

A fenti okok miatt is választottam szakdolgozatom témájának egy pókert játszani képes mesterséges intelligencia fejlesztését. Így betekintést nyerhetek a dolgozat készítése során részletesebben is ebbe a területbe, az itt megszerzett tudást később is sikerrel hasznosíthatom. A pókerrel kapcsolatban elég jó ismeretekkel rendelkezem, és számomra mindig is egy érdekes és szórakoztató játék volt, így ideális terület volt a dolgozat megvalósítása szempontjából.

Céлом egy olyan alkalmazás fejlesztése, amely egy amatőr pókerjátékost képes legyőzni, ehhez a szükséges információkat képesnek kell lennie leolvasni a kijelzőről, majd azoknak birtokában megfelelő döntést hoznia.

## 2 PÓKER ÁTTEKINTÉSE

A póker elnevezés összességében azokra a játékokra vonatkozik, ahol kártyával játszunk, és különböző tétet rakunk meg a kimenetelre. Ebben a részben ez a játék lesz ismertetve, különösen az a fajtája, ami a dolgozat témája, a No Limit Texas Hold'em.

### 2.1 Póker kutatócsoportok

#### 2.1.1 Carnegie Mellon egyetemi kutatócsoport

Az egyetemen 2004-ben hozták létre a kutatócsoportot, aminek célja egy pókert játszó, a legjobb emberi játékosok legyőzésére képes MI létrehozása volt.

Sokáig úgy gondolták, hogy a póker összetettsége, és a túl sok hiányzó információ a játék során lehetetlenné teszi, hogy valaha egy gép győzedelmeskedjen a legjobb játékosok felett, ahogy ezt sokáig egyébként például a sakk esetében is gondolták. A póker egyes fajtáiban hamar elérték, hogy szinte tökéletesen játszó szoftvereket fejlesszenek, ám a No Limit Holdem sajátos jellege miatt hatalmas kihívásnak bizonyult. Sokan dolgoztak azon, hogy a legjobb játékosokat megverni képes mesterséges intelligenciát fejlesszenek, megrendezésre került éves szinten a Humans VS AI verseny, ahol a legjobb játékosok játszanak Heads Up formátumban a legfejlettebb MI-k ellen, ahol sokáig győzedelmeskedtek is. 2007-ben azonban, a Libratus névre hallgató MI legyőzte a legjobb játékosokat is.[2][3]

Libratus elődje Claudico volt, de Libratus sokkal fejlettebbnek bizonyult. Claudico 2-3 millió, Libratus pedig 15 millió mag órányi számítási feladatokról lett felépítve, mindkettőt a 'Bridges' nevezetű Pittsburgh városában található szuperszámítógépen hozták létre. Készítői elmondásai alapján, Libratus-nak nincs saját beépített stratégiája, hanem egy algoritmus segítségével dolgozza ki a stratégiát. A verseny egy 200 ezer dolláros főnyereményért folyt, minden játékos kapott 20 ezer dollárt, a többi pénz pedig a teljesítményük alapján oszlott el. Napközben Libratus a játékosok ellen játszott, esténként pedig tovább fejlesztette a játékát a napközbeni játékok elemzésének segítségével, és fejlesztői is folyamatosan javítottak rajta az észrevételeik alapján. Az AI első naptól fogva végig vezetett az emberi játékosok ellen. 120 ezer leosztás lett lejátszva Libratus ellen, hogy statisztikailag szignifikánsak legyenek az eredmények. A megmérettetés végén Libratus 1,766 250 millió zsetonnyi vezetéssel

győzedelmeskedett, a nagyvak 100 dollár volt a verseny folyamán. Így Libratus 14.7 nagyvak/100 leosztás győzelmi rátát mutathatott fel, ami kiemelkedően magasnak számít.[4]

## **2.1.2 University of Alberta Poker Research Group**

Talán a legismertebb pókerrel foglalkozó egyetemi kutatócsoport. A munkájukat még a 90-es években kezdték a területen, és rengeteg különböző póker MI iteráció került ki a csoport munkájából. A két legújabb Cepheus és a DeepStack.[5]

### **2.1.2.1 Cepheus**

Ez volt az első olyan pókert játszani képes program, amely képes volt a Limit Texas Hold'em Heads up változatán győzedelmeskedni emberi játékosok ellen. Egyfajta Nash egyensúly stratégiát követve játszott, 2015-ben publikálták.[6]

### **2.1.2.2 DeepStack**

A DeepStack nevezetű MI már a No Limit variánshoz készült, amely megoldása sokkal nehezebb feladat, mint a Limit változaté, sokkal nehezebb matematikai számításokkal az optimális stratégia kidolgozása.

A program egy többcélú algoritmust használ, amely használható más nem teljes információjú játékok esetén is. Heurisztikus keresés használatával hoz döntéseket, és ehhez előtte mélytanulás segítségével tanították be a megfelelő döntések meghozatalára. Képes volt emberi játékosok ellen is győzedelmeskedni.[7]

## **2.2 A játék nehézsége az MI szempontjából**

### **2.2.1 Nem teljes információjú sztochasztikus játék**

A stratégiai játékokat két paraméter alapján tudjunk jellemezni: determinisztikus vagy nemdeterminisztikus, és a rendelkezésre álló információ mennyisége.

A determinisztikus játékok esetén a játék következő helyzete csak a játékosokon múlik, külső események nem befolyásolják azt. Ha a játék nemdeterminisztikus, akkor külső események is befolyásolják a végkimenetelt, például véletlenszerű események. Az olyan játék amelyek végkimenetelét véletlenszerű események is befolyásolják, sztochasztikusnak is nevezzük.[8]



Az olyan játékok esetén, ahol minden információ a rendelkezésünkre áll, ezek alapján jól kalkulált döntést vagyunk képesek hozni, mivel számunkra nincsenek nem ismert tényezők, amik a döntésük helyességét befolyásolhatnák. Az olyan játékok esetén, ahol az információ nem teljes, mindig van egy bizonyos mennyiségű információ, amelyet nem ismer minden játékos, ezért nem mindig tudjuk megállapítani, hogy éppen mi a leghelyesebb döntés az adott helyzetben.



**2. ábra: Játékok fajtái**

### 2.2.1.1 Jelentősége a mesterséges intelligencia szempontjából

A stratégiai játékok számítógépek segítségével történő játszásának olyan hosszú története van, mint körülbelül magának a számítástudománynak. Ez a terület nagyon jó kutatási felületet biztosított az érdeklődő szakembereknek, és sok híres kutató is kipróbálta magát ezen a területen, és próbálkoztak AI-t építeni a sakkhoz, vagy más játékokhoz. Ilyenek voltak többek között Alan Turing, Donald Knuth, Alan Newell, Herbert Simon vagy Neumann János is.

A póker fontos a mesterséges intelligencia területén, mert teljesen újfajta kihívásokat nyújt, ellentétben az olyan játékokkal, mint pl. a sakk. A póker esetében a játékosok próbálják meghatározni mások döntését, úgy, hogy nem áll rendelkezésre a lapjaikkal kapcsolatban információ. Ez pedig egy igen komplex kihívássá teszi a pókert. Az itt elért eredmények felhasználhatók más területen is, például a tőzsén, ahol szintén úgy kell megfelelő döntést hozni, hogy nem vagyunk birtokában minden információnak.[9]

## 2.3 Póker szabályai

Ez egy kártyajáték, ahol a játékosok tétet raknak meg arra, hogy az ő lapkombinációjuk a legerősebb. Minden megtett tét a potba kerül, és a játék végén az a játékos akinél a legerősebb kéz van, ő nyeri meg a potban található összeget. Van egy másik módja is a pot elnyerésének, még pedig az, hogy az összes többi játékost dobásra kényszerítjük, ilyenkor az egyetlen játékos, akinél még lapok vannak lesz a leosztás nyertese.

### 2.3.1 Kezek erőssége

A póker kéz 5 lapból áll, és ezek határozzák meg, hogy ki mennyire erős az adott körben. Póker esetében nem feltétlen szükséges, hogy az összes lap a mi kezünkben legyen, ugyanis vannak közös lapok is, amelyek az asztalon találhatóak. Ezeket minden játékos felhasználhatja. Az olyan póker variációkban, ahol vannak közös lapok, a játékos a kezét úgy állíthatja össze a saját és a közös lapokból, hogy az a legerősebb legyen.

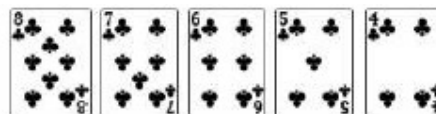
A következőkben bemutatásra kerülnek a lehetséges póker kezek, a legerősebbtől kezdve csökkenő sorrendben:

**Royal Flush:** Ez a lehető legjobb kombináció, az 5 lapos póker esetében. Egy ász, királyt, dámát, bubit és 10-es lapot tartalmaz, mindegyik egy színből.



3. ábra: Royal Flush

**Színsor:** Öt egyszínű sorba rendezhető lapból áll. Ha két ilyen találkozik, a legmagasabb lap dönt. Ha egyforma, akkor döntetlen (osztozás a nyereményen) van.



4. ábra: Színsor

**Póker:** Négy ugyanolyan számozású vagy jelű lapból és egy akármilyen másik lapból áll. Ha két ilyen találkozik, a magasabb póker nyer.



5. ábra: Póker

**Full:** Három ugyanolyan számozású vagy jelű lapból és két másik ugyanolyan számozású vagy jelű lapból áll. Ha két ilyen találkozik, a magasabb drill nyer. Ha egyforma, a magasabb pár nyer.



6. ábra: Full

**Szín:** Öt ugyanolyan színű lapból áll. Ha két ilyen találkozik, a legmagasabb lap dönt. Ha egyforma, a második legmagasabb dönt, és így tovább.



7. ábra: Szín

**Sor:** Öt sorba rendezhető lapból áll. Ha két ilyen találkozik, a legmagasabb lap dönt. Ha egyforma, a színerősség dönt.



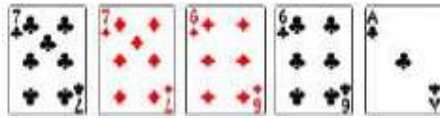
8. ábra: Sor

**Drill:** Három ugyanolyan számozású vagy jelű lapból és két akármilyen másik lapból áll. Ha két ilyen találkozik, a magasabb drill nyer. Ha egyforma, a magasabb semleges lap, majd az alacsonyabb dönt.



9. ábra: Drill

**Két pár:** Kétszer két ugyanolyan számozású vagy jelű lapból áll. Ha két ilyen találkozik, a magasabb pár, majd az alacsonyabb, majd a semleges lap erőssége dönt.



10. ábra: Két pár

**Pár:** Két ugyanolyan számozású vagy jelű lapból és három akármilyen másik lapból áll. Ha két ilyen találkozik, a magasabb pár nyer. Ha egyforma, a semleges lapok döntenek.



11. ábra: Pár

**Magas lap:** Bármelyik kéz, amelyik nem tartozik a fentiek közé. A legmagasabb lap alapján van rangsorolva, ha az egyezik akkor a második legmagasabb és így tovább.



12. ábra: Magas lap

### 2.3.2 No Limit Texas Hold'em

A No Limit Texas Hold'em egy olyan pókervariáció, amely használ közös lapokat.

Minden játék elején, 2 lap kerül kiosztásra minden játékosnak. Egy osztó játékos kerül kijelölésre, őt az osztó gomb fogja jelölni. Az osztó pozíciója az óramutató járásával megegyezően forog, minden egyes leosztás után. Ez után az osztótól balra lévő 2 játékos beteszi a vakokat. Az első játékos tőle balra a kisvak, az utána következő pedig a nagyvak. A kisvak a minimális tét felét, míg a nagyvak a minimális tétet beteszi. A lent látható képen pár játékos pozíció elnevezése:



**13. ábra: Póker asztal**

<https://blog.coinpoker.com/wp-content/uploads/2018/08/CHP-VALUE-RISES-54-744x389.png>

Az első játékos, aki cselekszik a kör folyamán, az, aki a nagyvaktól balra található. A következő mindig a tőle balra található játékos lesz. Minden játékos a következő lépéseket hajthatja végre:

- Licitál: Pénzt rak a potba.
- Megad: Tartja a licitet, ugyanannyit tesz be, mint a legmagasabb tét.
- Emel: Magasabbra emeli a licitet.
- Passzol: Ha nem történt licit, lehetséges passzolni, anélkül hogy pénzt tenne a potba.
- Dob: Eldobja a lapjait így feladja a potért a küzdelmet.
- All-in: Speciális emelési fajta. Itt a játékos a rendelkezésre álló összes összeget a potba rakja.

Egy játékosnak ahhoz, hogy folytathassa a potért való küzdelmet, mindig meg kell adnia az éppen legmagasabb licitet, vagy azt tovább kell emelnie. A No Limit Texas Hold'em esetén nincs a licitalásnak felső határa, ezért akár a rendelkezésükre álló teljes összeggel is emelhetnek egy körben. A minimális összegű emelés mindig a nagyvak mérete.

Miután minden játékos all-in ment, passzolt vagy megadta a tétet, a kör befejeződött. Összesen 4 licitalási kör van a játék során. Minden egyes körben, új közös lapok kerülnek az asztalra. A 4 kör a következő:

- Flop előtti: Itt nincsenek közös lapok még.
- Flop: 3 közös lap kerül az asztalra.
- Turn: A 4. közös lap kerül az asztalra.

- River: Az 5. és egyben utolsó közös lap is az asztalra kerül.

A river után, miután minden játkos végzett a licitálással, utána következik a mutató. Itt a játékosok sorban megmutatják a kezüket, és a legerősebb kéz elnyeri a potot. Ha több ugyanolyan erősségű kéz van, akkor a pot elosztásra kerül közöttük. Egy játékos, ha tudja, hogy az ő keze biztosan nem a legerősebb, akkor a mutatóskor mikor ő következik be is dobhatja a lapjait mutató nélkül.

Ha egy körben minden játékos eldobja a lapjait kivéve egy, akkor ő nyeri a potot.

Vannak továbbá még különleges helyzetek is, például ha egy játékos all-in megy, de a többiek még tovább emelnének, akkor számukra egy külön pot lesz létrehozva, ugyanis minden játékos csak a saját tétjének megfelelő összeget nyerhet el a többektől, de ez most részletesen itt nem kerül megvitatásra.

### 3 PÓKERT JÁTSZANI KÉPES MI LÉTREHOZÁSÁNAK MÓDJAI

A mesterséges intelligencia területe azzal foglalkozik, hogy olyan intelligens gépeket alkossanak. Alkalmazásuk nagyon sokféle lehet, a játékokon át a hangfelismerésig, a gépi látásig, a szakértői rendszerekig és az önvezető autókig is terjed. Mai hozzájárulása modern világunkhoz gyakorlatilag elengedhetetlen.

Kezdetben az MI kutatások fő célja egy olyan rendszer létrehozása volt, ami képes az emberek intelligenciáját elérni, vagy azt akár túl is szárnyalni. Ahogy teltek az évek, azonban rájöttek, hogy ez a feladat jelenleg lehetetlennek bizonyul.

Azonban mivel Moore törvénye alapján, a számítási kapacitása a CPU-knak 2 évente megduplázódik, ha ilyen ütemben folyik a technológiai fejlődés, az is lehet, hogy hamarosan sikerülhet ezt a célt is elérni. Ezzel elérkezne az emberiség a technológiai szingularitáshoz, tehát az emberek okosabb gépeket hoznának létre, mint saját maguk.[10]

Az MI kutatások egyik fontos területe a játékok. Nagyon sok játék van, ami nagyon érdekes kihívásokat nyújtott, vagy nyújt a mai napig. Klasszikus játékok, mint pl. a sakk is komoly kihívást jelentett. Jelentős eredményeket sikerült elérni ezen játék esetén is. Az egyik leghíresebb munka a Deep Blue volt, egy számítógép, amely képes volt legyőzni az akkori legjobb sakkozót egymás utáni játékok során.

A póker az elmúlt évtizedben jelentős szerepet tölt be az MI kutatás területén. Ez egy radikálisan új kihívást jelent a többi játékhoz képest. A sakk esetén például a két játékos minden információval rendelkezik a játék aktuális helyzetével kapcsolatban. Így lehetségessé válik egy sakk döntési fa felépítése. A póker ezzel ellentétben a játékosok számára zárt lapokat is tartalmaz, így sokkal nehezebb felépíteni egy döntési fát.

#### 3.1 Felépítésének lehetséges lépései

Egy olyan fajta MI létrehozása, amely az emberi játékosok által játszott leosztásokon alapul, a következő lépésekből hozható létre.

- Megfelelő mennyiségű adat gyűjtése póker leosztásokról, és azok végkimeneteléről
- Megtalálni azokat a pontokat a játék során, melyek jelentősek befolyásolják a játékosok döntéseit
- Felépíteni egy stratégiát a kigyűjtött leosztások segítségével, amely optimális lehet számunkra
- Létrehozunk a kívánt mesterséges intelligenciát
- Megnézzük, hogy az általunk betanított stratégiákat megfelelően hajtja-e végre és el tudjuk-e vele érni a kívánt hatást
- Aktívan teszteljük a megírt programot, és figyeljük a gyengeségeit, és aktívan próbáljuk javítani az elért eredményeket

### 3.2 Adatok gyűjtése

Ahhoz hogy lemodellezhessünk egy stratégiát, először is szükséges hogy adatokat gyűjtünk olyan játékokról, amelyeket emberi játékosok játszottak egymás ellen. Szerencsére nagy mennyiségű adat érhető el az interneten. A megfelelő minőségű adatoknak azonban vannak kikötései:

- Egy adott játékosról nagy mennyiségű információnak kell rendelkezésre állni, így célzottan képesek lehetünk tanulni egy játékos stratégiájából. Minél több leosztás áll rendelkezésre az illetőről, annál pontosabb képet kapunk az általa használt stratégiáról
- Fontos, hogy valódi pénzben történt játékokból származzanak. A virtuális pénzes játékokban sok ember nem játszik teljes komolysággal, sokkal könnyebben kockáztatnak, alapvetően az ilyen környezetben teljesen más fajta játék figyelhető meg. A számukra optimálisnak vélt játékot sokkal inkább akkor játsszák az emberek, ha a saját pénzük a tét, és ha esetleg valódi pénzt nyerhetnek, ha jól teljesítenek

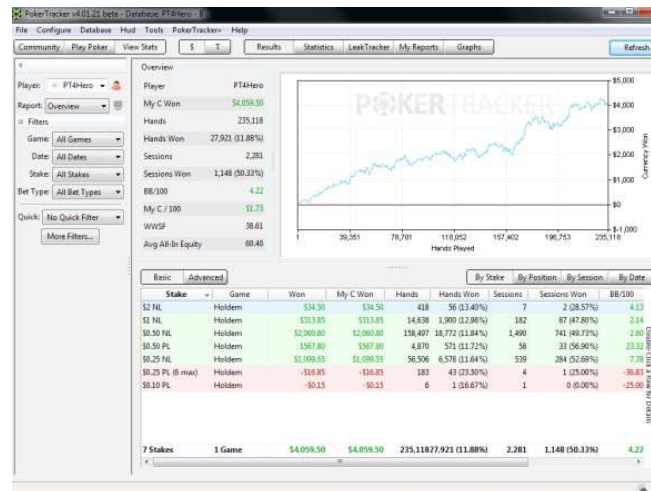
A következő források lehetnek hasznosak a leosztások gyűjtése során:



### 3.2.1 PokerTracker

A PokerTracker egy szoftver, mely online játék közben használható. Segítségével az általunk játszott leosztások adatait gyűjti be folyamatosan. Fontos azonban, hogy jogi szabályok miatt csak az általunk játszott leosztások adatait gyűjti, ha mi nem játszunk aktívan épp az asztalon csak nézzük, azok a leosztások nem kerülnek mentésre.

Viszont ha rendelkezésünkre állnak leosztások, azok betölthetők a program adatbázisába. Az alkalmazás igen hasznos, mert különféle elemzéseket végezhetünk a segítségével, akár a saját játékunkról, akár másokéről is. Képes megmondani, hogy mely pozíciókban, illetve melyik licitkörben hogyan teljesítünk, és az általa tárolt adatbázisban, a többi játékoshoz képest milyenek a statisztikáink, így segít megtalálni az optimális stratégiát.[11]



14. ábra: PokerTracker

### 3.2.2 HHSmithy adatbázis (<https://www.hhsmithy.com>)

Ezen az oldalon lehetőségünk van leosztásokat vásárolni valós játékokból, amiket különböző online kaszinókban egymás ellen játszottak az emberek. Kiválaszthatjuk, hogy mely kaszinóból szeretnénk leosztásokat kapni, milyen játéktípusban, és milyen magas limitről. Ez után megadjuk a kívánt mennyiséget, és a kifizetett összeg után a leosztásokat megkapjuk emailben.

### 3.2.3 Az adatok tárolása

Fontos tudni azt, hogy a különböző termékből származó leosztások formátuma nem megegyező. Ezért, ha ezeket különböző forrásokból fel szeretnénk használni, akkor szükségünk lesz egy egységes formátumra, melyre átalakítva a későbbiekben adatbázisban tárolhatjuk őket.

A legegyszerűbb megoldás talán az említett PokerTracker használata, ugyanis az fel van készítve a különböző oldalakról származó leosztások kezelésére, és az általa használt adatbázisba nagyon egyszerűen betölthetők, és ott el is tudjuk végezni a szükséges elemzéseket.

## 3.3 Ellenfelek modellezése

Ahhoz hogy egy jó játékost hozzunk létre, fontos hogy az ellenfelek lépéseit folyamatosan nyomon kövessük, és változtassunk a stratégián az ő döntéseik alapján. Ez egy fontos dolog, ugyanis ha egy játékos statikus taktikát alkalmaz, könnyen kiismerhetővé válik, így hamar felállítható lesz ellene egy optimális stratégia. Ezen ok miatt, egy jó játékos folyamatosan figyeli az ellenfeleit, és időről-időre változtat a saját játékán. Az ellenfelek modellezésének fő célja, hogy gyorsan megállapítsuk, hogy milyen stratégiát alkalmaznak, és a lehető legjobb játékmóddal reagáljunk rá.

### 3.3.1 Sklansky féle osztályozás

A játékosok osztályozása egy jó módszer az ellenfeleink modellezésére. Ennek lényege, hogy a játékosokat csoportokba rakjuk a játékstílusuk alapján. A kritériumok melyek alapján a csoportosítás történik a következők: milyen gyakran blöfföl, milyen kezekkel hív, milyen agresszív a játékos, milyen gyakran ad meg stb. Az egyik legismertebb osztályozási módszer a Sklansky féle osztályozás.[12]

Egy fontos osztályozási paraméter például az, hogy a játékos a kezek hány százalékával játszik. Egy játékos feszesnek mondható, ha a kezek 28%-ával vagy kevesebbel játszik, és lazának, ha ennél többel.

A másik fontos szempont az agresszió faktor. Az agresszió faktor a következő képlet segítségével mondható meg:

$$AF = \frac{(\text{hívások száma} + \text{emelések száma})}{(\text{megadások száma})} \quad (1)$$

Ha a játékos agresszió faktora nagyobb mint 1, akkor ő agresszív játékosnak számít, ha pedig alacsonyabb akkor passzív játékosról beszélhetünk.

Az agresszió faktort kombinálva a megjátszott kezek számával 4 típusú játékost kapunk:

- Feszés agresszív
- Feszés passzív
- Laza agresszív
- Laza passzív

Starting Hands	loose	<b>loose-passive (Calling Station)</b> <ul style="list-style-type: none"> <li>Plays many hands (&gt;25-40%)</li> <li>Plays passive (few bets, many calls)</li> <li>Minimal share of winning players</li> </ul>	<b>loose-aggressive (LAG)</b> <ul style="list-style-type: none"> <li>Plays many hands (&gt;25-40%)</li> <li>Plays aggressive (many bets, few calls)</li> <li>Average share of winning players; high winnings, but also high losses possible; high variance</li> </ul>
	tight	<b>tight-passive (Rock)</b> <ul style="list-style-type: none"> <li>Plays few hands (&lt;15-20%)</li> <li>Plays passive (few bets, many calls)</li> <li>Low share of winning players</li> </ul>	<b>tight-aggressive (TAG)</b> <ul style="list-style-type: none"> <li>Plays few hands (&lt;15-20%)</li> <li>Plays aggressive (many bets, few calls)</li> <li>High share of marginal winning players with many multitableting regs</li> </ul>
		passive	aggressive
		Betting Pattern	

15. ábra: Játékos típusok

<https://thepokerbaffer.com/2015/09/23/poker-plays-you-can-use-lags/>

### 3.4 Kezek értékelése

Az egyik legfontosabb dolog amire egy póker AI-nak szüksége van, az egy kézürtékelő. Egy kézürtékelő fog egy póker kezet és azt egy egyedi integer-hez társítja hozzá. Mindezt úgy, hogy hogy egyenlő integer-hez társított kezek egyforma erősségűek, és egyébként mindig a magasabb az erősebb. Mindenképpen szükséges, hogy a lehető leggyorsabb kézürtékelők egyikét használjuk, mert egy AI több ezer kezet is kiértékelhet egy-egy döntése előtt, és a gyors döntéshez elengedhetetlen a gyors kiértékelés.

### 3.4.1 Pokersource Poker-Eval

Pover-Eval egy C library, amely a póker kezek kiértékeléséhez lett létrehozva. Minden kiértékelés végén a kézhez egy számot társít. Az általános ötlet lényege, ha a kapott szám kisebb, mint az ellenfeled kezének kapott száma, akkor veszítettél. Sok pókervariációt támogat, és talán ez a leggyakrabban használt kézértékelő. A C nyelv miatt könnyebb beépíthető C vagy C++ nyelven írt AI-ba.[13]

### 3.4.2 Foldem Hand Evaluator

A Foldem Hand Evaluator egy gyors, 7 lapos kézértékelő, kb. 4.5 millió kéz kiértékelésére képes másodpercenként. Előnye hogy kevés memóriát használ, ezért bármilyen számítógépen gördülékenyen használható. További előnye, hogy Java nyelven írt, ezért könnyen használható Java-ban írt AI esetén.[14]

## 3.5 Odds kalkulálása

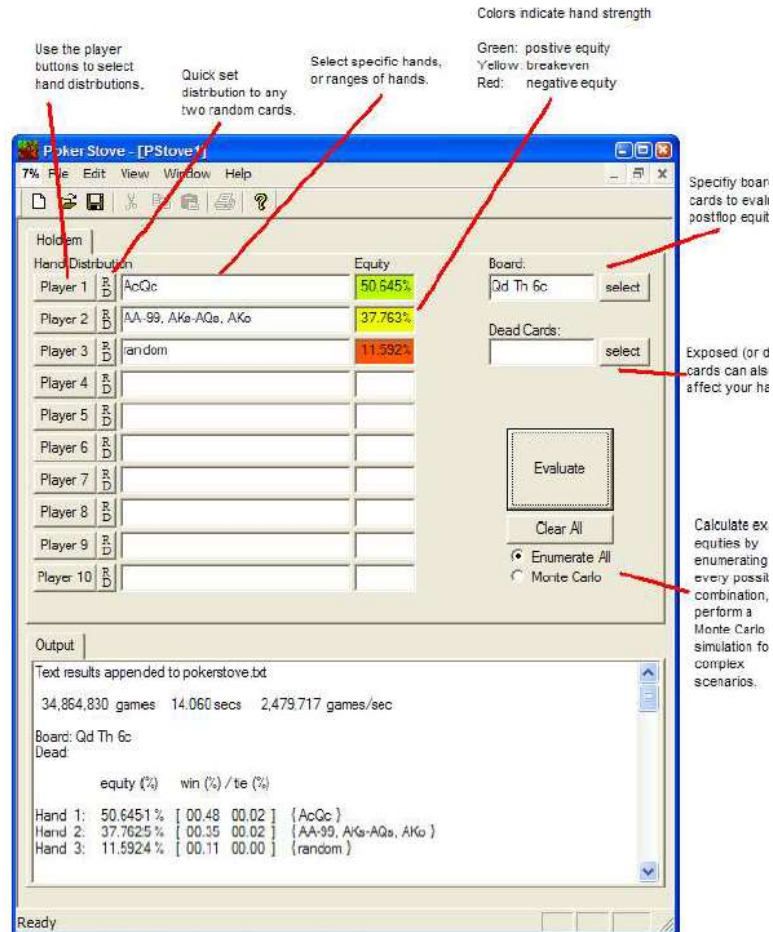
A póker AI a valóságban nem a kezeket fogja kiértékelni, mivel nem ismeri az ellenfelek lapjait. Ezért az AI a nyerési esélyt (Odds) fogja kiszámolni. Ennek a módszere, hogy kiértékeli a saját kezét, és azt összeveti a lehetséges kezeivel az ellenfeleknek. Ennek segítségével jó döntést lehet hozni a következő akciót illetően.

### 3.5.1 Effektív nyerési esély

Az effektív nyerési esély, annak a valószínűsége, hogy az adott kéz bizonyos számú ellenfél esetén mekkora eséllyel fogja megnyerni a potot. Ehhez az szükséges, hogy az összes lehetséges kézkombinációt figyelembe kell venni, a nem ismert lapok alapján, ezek azok, amik nem nálunk, illetve nem is az asztalon vannak, ugyanis ezek lesznek az ellenfeleknek. Ez a közelítés viszont azon alapul, hogy senki nem fogja már bedobni a lapját.

Probléma még, hogy az összes kombináció kiszámítása nagyon időigényes feladat. Ezért a Monte Carlo módszert szokás alkalmazni. Ezen módszer esetén, egy fix számú, véletlenszerű végkimenetel lesz generálva, ami megközelítőleg elég jó képet nyújt a nyerési esélyeinkről, ha elég nagy számú esetet veszünk figyelembe.

A nyerési esély kiszámítására különféle szoftverek is rendelkezésre állnak. Ezek közül az egyik legnépszerűbb a PokerStove, mely segítségével könnyen kiszámolható az adott szituációban a kezünk nyerési esélyei.[15]



**16. ábra: PokerStove**

<https://github.com/andrewprock/pokerstove>

### 3.6 A póker AI tesztelése

Fontos, hogy az elkészült szoftverünket utána megfelelő körülmények között tudjuk tesztelni, hogy hatékonyan fényt tudjunk deríteni a gyengeségeire, hogy azokat utána javítani tudjuk.

### 3.6.1 Meerkat API

A Meerkat API egy gyakran használt API a póker botok tesztelésére. A Poker Academy biztosítja és lehetőséget ad a botok készítőinek, hogy beletöltsék a saját fejlesztésű AI-t, és összemérjék bármelyik AI-al szemben, ami a Meerkat segítségével készült. Mindehhez egy minőségi GUI-t is kínálnak, és egy jól dokumentált Java interfészt.[16]

A program nyomot követi az összes leosztást, eltárolja azokat, és különféle grafikonok megjelenítésére is képes, mely segítségével tovább elemezhető az általunk elkészített bot.

Hátránya, hogy a felhasználói interfész miatt elég lassan szimulálja a leosztásokat. További probléma még, hogy a szimulációkat nem képes elindítani emberi felügyelet nélkül.

### 3.6.2 Open Meerkat Poker Testbed

Ez a Meerkat API egyfajta változata, de annál sokkal gyorsabb, ugyanis eltávolították róla a felhasználói interfész nagyrészét. Támogatja a fix és a no limit hold'em változatokat is. Szintén eltárolja a játék során a log fájlokat, valamint ez is képes grafikonok megjelenítésére.

### 3.6.3 PokerSnowie

A PokerSnowie egy online pókerjátékosok számára készült szoftver, melynek segítségével javíthatnak a saját játékuikon. Az MI neurális hálózatok használatával készült, önmaga ellen játszotta a leosztásokat, így lett egyre erősebb játékos. Az online pókertermek hasonló grafikus felületet használ, és több játékváltozat is választható.[17]



**17. ábra: PokerSnowie**  
<https://www.pokersnowie.com>

## **3.7 Jelenlegi megvalósítási módszerek**

### **3.7.1 Tudásalapú rendszerek**

A tudásalapú rendszerek lényege, hogy az adott terület szakértője által nyújtott tudás szükséges a megfelelő működéséhez.[18]

#### **3.7.1.1 Szabályalapú szakértői rendszerek**

Egy egyszerű szabályalapú rendszer nagyon sok döntésből áll, a program futása során sok elágazás lesz. A játék során létrejövő összes szituációt le kell fedni ezeknek az elágazásoknak, és megfelelő döntéseket kell hozniuk. Hátránya, hogy kell egy szakértő a témában, és olyan játékok, mint a póker esetén, ahol nagyon sok lehetséges helyzet állhat elő, az ilyen rendszer megvalósítása nagyon időigényes feladat lehet.[18]

#### **3.7.1.2 Formula alapú rendszerek**

Ez hasonló a szabályalapú rendszerhez. Azonban itt nem kell annyi döntési ágat létrehozni, hanem bizonyos input paraméterek alapján, számítások segítségével a döntések számát a rendszer lecsökkenti, így sok helyzet, ami a szabályalapú rendszer esetén külön volt kezelve, itt egy csoportba kerülnek. Egy ilyen megközelítés lehet például, ha a nyerési esélyt kiszámítjuk a rendelkezésre álló információk alapján, és ennek függvényében választjuk ki az adott helyzetben a megfelelő reakciót.[18]

#### **3.7.1.3 A két módszer együtt**

Sok esetben a két fent említett módszert kombinálják, és így készítik el a megfelelő MI-t. A leggyakoribb módszer, hogy a preflop játékhoz, amely egyszerűbb és sokkal kevesebb döntésből áll, elkészíthető egy szabályalapú rendszer, majd a jóval összetettebb és bonyolultabb posflop játékhoz már egy formula alapú megvalósítást használnak.[18]

### **3.7.2 Monte-Carlo szimuláció**

Az olyan játékok esetén, mint a póker, nem áll minden információ a játékosok rendelkezésére. Itt jön segítségül a Monte-Carlo módszer, ahol a nem ismert változókat egy szimulációs technikával előállítják, majd ezt kiértékelik. Ezt a folyamatot ismétlik újra és újra egymás után, aminek a végezetével megfelelő közelítésű következtetéseket tudnak levonni.[19]

### 3.7.3 Nash egyensúly stratégia

A Nash egyensúly egy olyan játékelméleti koncepció, amely bizonyítja, hogy többjátékos játékok esetén, mindig van egy olyan stratégia, amelyet követve mindig előnyben leszünk ellenfeleinkkel szemben, akkor is ha ők közben változtatnak a stratégiájukon. A No Limit Texas Hold'em sok változója miatt, nagyon nehéz a játékszituációkat így modellezni, ezért csak néhány helyzetben alkalmazható. Volt már ilyen stratégiát használó rendszer kifejlesztve, nem volt hibátlan, de a legtöbb ellenféllel szemben, akik nem igazodtak rendesen a vakok változásaihoz, és a gyengébb lapokhoz heads up játék esetén, 5-40%-os előnnyel rendelkezett velük szemben.[20]

Különböző módszerekkel a Nash egyensúly stratégia tovább fejleszthető:

#### 3.7.3.1 Legjobb válasz

A legjobb válasz stratégia azon alapul, hogy minden stratégiára van egy lehető legjobb ellenstratégia. Mivel a legjobb játékosok változtatják a stratégiájukat a játék folyamán, az AI-nak szükséges észlelnie ezeket a változásokat a játék folyamán, ha kompetitív akar lenni. A probléma, hogy ehhez nagy mennyiségű információra lenne szükség minden játékosról, ami általában nem áll rendelkezésre. [21]

#### 3.7.3.2 Korlátozott Nash válasz

Ezen módszer esetén, a játék során játékosprofilokat kell felépíteni az ellenfeleinkről. Ameddig az nem ismert, megfelelő adatok hiányában, addig a Nagy egyensúly stratégiát kell követni. Amint az ellenfél kategorizálva lett, hozzá igazított döntéseket hozunk, így kevésbé lesz kiszámítható a játékunk.

### 3.7.4 Evolúciós tanulás

Ez a tanulási mód neurális hálózatokra jellemző. A természetes szelekciót utánozzák, mostanában egyre gyakrabban használt az MI területén is. „A genetikai algoritmusok sztochasztikus keresési algoritmusoknak tekinthetők, ahol a paraméterter feltérképezése és az iterációnként egyre jobb megoldások előállítása az eddigi eljárásoktól gyökeresen eltérő módon történik.”.[22]

A lényege, hogy evolúciós neurális hálózatokat hozunk létre, ahol játékosokat generálunk, melyek egymás ellen játszanak. Ha valamelyik játékosnak elfogy a pénze, ő kiesik



és a helyére a legjobban teljesítő játékos tulajdonságaihoz hasonló játékos lesz generálva, és folytatódik tovább. Sok leosztás és iteráció után, egy egészen jó játékos is lehet a végeredmény, azonban sok dolog hiányozni fog belőle, amire emberi játékosok képesek, ilyen lehet például a blöffölés képessége, valamint teljesen váratlan helyzetekre nem valószínű, hogy megfelelően fog reagálni.

### 3.7.5 Eset-alapú döntéshozás (Case Based Reasoning)

A módszer lényege, hogy a korábbi tapasztalatok alapján próbál döntést hozni az adott új döntési helyzetben is. Az alapja, hogy eltárol régebbi eseteket a hozzájuk tartozó helyes megoldással együtt, és mikor döntést kell hoznia, akkor az éppen aktuálishoz hasonló helyzetet keres a tárolt adatok között és annak segítségével választ megoldást. [23]

4 lépésből áll:

- **Megszerzés:** A célproblémához hasonló eset megszerzése a memóriából.
- **Újra felhasználás:** Az adott megoldás átalakítása, hogy a jelenlegihez megfelelő legyen.
- **Kiértékelés:** Az előző lépés során létrejött megoldás kiértékelése szimulációs vagy valós teszttel. Ha szükséges változtat a megoldáson.
- **Tárolás:** Miután a megfelelő megoldás megtalálása sikeres volt, azt eltárolja a memóriában.

## 4 AZ ALKALMAZÁS FEJLESZTÉSÉNEK BEMUTATÁSA

A következő fejezetek során bemutatásra fog kerülni az általam készített alkalmazás fejlesztésének lépései. Bemutatásra kerülnek az általam felhasznált technológiák a folyamat során, valamint az elkészült program részletes dokumentációja osztályonként lebontva. Végezetül a tesztelési folyamat és annak eredményei is ismertetve lesznek.

### 4.1 A felhasznált technológiák ismertetése

#### 4.1.1 Java

Az általam választott programozási nyelv a Java lett. Fontos volt egy olyan programozási nyelv választása, amely objektumorientált. A Java-ra azért esett a választásom, mert a legtöbb ismeretem ezen objektumorientált nyelven van, az egyetemi évek folyamán kötelező tárgy keretében is tanultam.

##### 4.1.1.1 Java története

A nyelv fejlesztése 1991-ben kezdődött Patrick Naughton és James Gosling által. „Céljuk egy hatékony, architektúra-független, kis erőforrásigényű rendszer kifejlesztése volt, melyet egyszerű elektronikai eszközökben szerettek volna felhasználni.”[24]

Az ötletük alapja a következő modell volt: A Java Virtual Machine – JVM nevezetű hipotetikus gép segítségével olyan kód létrehozása, amely azt követően minden operációs rendszeren és architektúrán képes működni, amely rendelkezik ezzel az interpreterrel.

Ez a hordozhatóság nagyon fontos volt, ugyanis akkoriban a gyors technológiai fejlődés miatt nagyon nagyfokú architekturális eltérések voltak a hardverek között. Így ez a tulajdonsága képes volt ellensúlyozni az akkori gyengeségeit, mint például a lassúság, így a nyelv képes volt gyorsan elterjedni.

„A virtuális gép a Java alkalmazások futtatására képes, általában szoftveresen megvalósított működési környezet. A virtuális gép feladata a Java osztályokat tartalmazó 'class' állományok betöltése, kezelése, valamint az azokban található – Java kódban adott – műveletek gépi kódú utasításokká történő átalakítása.”[24]

A JVM képezi a hidat a fizikai vas és a Java classok között. A fizikai gépben lévő erőforrásokhoz való hozzáférést is a JVM kezeli a programok számára.

Terjedését nagyban segítette az internet térhódítása, ugyanis a JVM segítségével a weblapok esetében is képesek voltak Java programok futtatására, így a fejlesztők képesek voltak az addig statikus weblapokon dinamikus tartalmak megjelenítésére is.

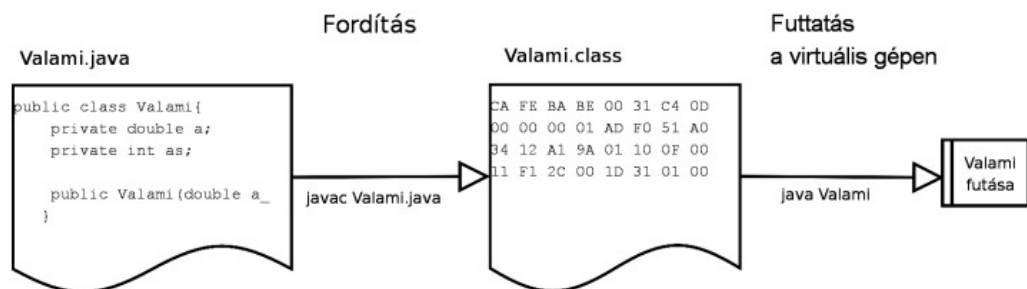
A Java programozási nyelvet végül 1995-ben adta ki a Sun Microsystems. Az 1999-ben megjelenő Java2 esetében már 3 külön részre bontották a Java technológiát. ME (Micro Edition) a beágyazott rendszerekhez, SE (Standard Edition) a desktop alkalmazásokhoz és EE (Enterprise Edition) a nagyteljesítményű, vállalati, illetve a kliens-szerver alkalmazásokhoz.

Jelenleg a Java SE 12 az aktuális verzió, napjainkban pedig több mint 2.5 milliárd Java képes eszköz van forgalomban.

#### 4.1.1.2 Java programok és a virtuális gép szerkezete

„Mint minden programozási nyelvhez, úgy a Java nyelvhez is szükséges valamilyen fejlesztői környezet, amely a forrásprogramok fordítását, illetve futtatását lehetővé teszi. A Java fejlesztői csomag (Java Software Development Kit – SDK) telepítése, és a szükséges környezet beállítása után az elkészült forráskódot lefordíthatjuk.”[24]

A JVM előzetesen lefordított bájtkódú bináris állományokkal dolgozik. Lényege, hogy ugyanaz a Java forrásprogram, bármilyen fordító használata esetén ugyanazt a bájtkódot állítja elő. A bájtkódot a virtuális gépen futtatva pedig azonos eredményt is kapunk. Így a Java működése teljesen platformfüggetlen lesz.



18. ábra: Java programok fordítása és futtatása virtuális gépen

A fenti ábrán jól látható a Java programok futtatásának folyamat. Először a forráskódot egy .class kiterjesztésű bájtkóddá alakítjuk, majd ezt tudjuk futtatni utána a JVM-en.

Fő előnyei:

- Egyszerű: A nyelv a C++ egy egyszerűbb nyelvi változata, sokkal kevesebb nyelvi eszközt tartalmaz.

- Objektumorientált: A Java tisztán objektumorientált nyelv.
- Robosztus: Már fordítási időben képes a programozói hibák kiszűrésére.
- Biztonságos: A kezdetek óta támogatja a hozzáférések és jogosultságok kezelését. Támogatja a hálózatos és elosztott rendszerek használatát. Az osztályok virtuális gépbe töltésénél hibaellenőrzés történik.
- Hordozható: Adott forrás esetén minden Java fordító ugyanazt a tárgykódot állítja elő.
- Architektúra-független: Az egyes architektúrák és operációs rendszerekhez elkészített virtuális gépek alakítják a class állományokat natív kóddá, így minden rendszeren ugyanaz lesz a végeredmény.
- Interpretált: „Az egyes hardvereken futó natív kódot a virtuális gép futási időben hozza létre, a lefordított tárgykódú állományokból”
- Nagy teljesítményű: Az évek folyamán történt fejlesztéseknek köszönhetően sikerült megfelelő teljesítményűre optimalizálni a Java nyelvet is, kezdetben az interpretált jelleg miatt elég lassú futás keletkezett.
- Többszálú: A Java környezet támogatja a párhuzamos programozást.
- Dinamikus: „Az osztálykönyvtárak szabadon továbbfejleszthetőek, és már a tervezésnél figyelembe vették, hogy egy esetleges későbbi bővítés a már megalkotott programok működését ne akadályozza.”

## 4.1.2 SQLite

„Az SQLite egy kis méretű, beágyazható relációs adatbázis-kezelő rendszer, melyet C forrású programkönyvtárként valósítottak meg, és amely nyelve legjobban az SQL-92 szabvány hasonlít. Részlegesen megvalósítja a triggereket és a legtöbb összetett lekérdezést.”[25]

Az SQLite működése folyamán nem különálló folyamatként működik, ezzel eltér a megszokott gyakorlattól. Tervezésénél fő szempont volt, hogy sokféle nyelven írt programokhoz is könnyen hozzáilleszthető legyen. Az általa kezelt adatbázis egésze egyetlen fájlban kerül eltárolásra.

Egy további fő szempont volt tervezésénél az egyszerűség, ezért más összetettebb működésű adatbázis-kezelőkhöz képest szerény funkcionalitással van ellátva. SQLite esetében például, ha épp valamilyen adatot írnak, akkor az zárolás alá kerül, és mindaddig más

folyamatok azt az adatot csak olvashatják, ha írni próbálnak akkor hibaüzenetet kapnak. A többi adatbázistól eltérően, az SQLite esetében nincs semmilyen belső jogosultsági rendszer. Így nem igényel adminisztrátort se, nem kell felhasználói jogokat kiosztani és az adatbázist külön létrehozni.

Az SQLite nem típusérzékeny, mint általában a többi adatbázis-kezelő. Ha itt a program esetében például egész típusú mező esetében szöveget adunk meg, akkor azt megpróbálja utána egyszerűen egész típusúvá átalakítani. Az adatokat gyengén típusos módon kezeli.[25]

„Az SQLite által használt adattípusok a következők:

- NULL: Az értéke mindig NULL
- INTEGER: Egész típus. Az értéke előjeles egész szám 1, 2, 3, 4, 6 vagy 8 bájton tárolva (az értéktől függ)
- REAL: Valós típus. Az értéke a valós szám 8 bájton tárolva (lebegőpontos ábrázolással)
- TEXT: Szöveg típus. Az értéke szöveg (string), az adatbázis kódolásával tárolva (UTF-8, UTF-16BE vagy UTF-16LE)
- BLOB: Nagyméretű bináris típus. Az értéke blob (bináris adatok gyűjteménye), pontosan olyan módon tárolva, ahogy az a bemenetre érkezett.”

#### 4.1.2.1 Az SQLite felhasználási területei

Egyre több népszerű alkalmazás használja manapság, ilyenek például: Mozilla Firefox, Macintosh számítógép, PHP alapú weboldalak egy része, Skype, iPhone és Android okostelefonok.

Nem alkalmas nagyvállalati felhasználásra, kisebb adatbázis esetében viszont kiválóan alkalmazható, főleg az egyéni vagy mikro vállalatok igényeinek kielégítésére.

„Előnyei:

- Egyszerűen beüzemelhető és kezelhető
- Külön szolgáltatástól, szervertől független
- Adatainkat a helyi rendszeren tudjuk tárolni
- Kis erőforrásigénnyel rendelkezik, de képes a megfelelő szolgáltatások biztosítására
- Egyszerűen hordozható”

Hordozható eszközökre írt szoftver esetén például kiváló választás lehet.[25]

### 4.1.3 Tesseract

A Tesseract egy nyílt forráskódú optikai karakterfelismerő program. A HP kezdte el fejleszteni még az 1980-as években. A 2006-os évek óta a Google fejlesztése alatt áll.

Unicode (UTF-8) támogatással rendelkezik, és több mint 100 nyelven írt szöveg felismerésére képes. Több fajta kimeneti formátumot támogat, többek között txt, PDF, HTML, XML.[26]

### 4.1.4 JDBC

„A Java Database Connectivity, röviden JDBC egy API a Java programozási nyelvhez, amely az adatbázishozzáférést támogatja. A JDBC definiálja az adatbázisok lekérdezéséhez és módosításához szükséges osztályokat és metódusokat. A relációs adatmodellhez igazodik.”[27]

## 4.2 Fejlesztési dokumentáció

Az általam választott pókervariáns, amire az alkalmazásom fejlesztettem, a 9 fős No Limit Texas Hold'em (FullRing) volt. Ez a változat az, amellyel személyesen is a legtöbb tapasztalatom van, ezt ismertem a legjobban.

Fontos még továbbá, hogy a póker esetében minél több játékos van, annál kevesebb lappal optimális a játék, és az agresszivitás mértéke is ellentétesen arányos a játékosok számával. Az fontos volt, hogy lehetőleg olyan játékváltozathoz készüljön el először, ahol a legkevesebb marginális döntéssel kell szembenéznie, mert ezekben a helyzetekben nagyon nehéz lehet sok esetben az optimális döntést meghozni.

Az általam választott tesztalkalmazás, amin képes lesz az általam írt pókerbot játszani, a PokerGenius lett.[28]



19. ábra: PokerGenius

Választásom azért erre az alkalmazásra esett, mert felülete teljesen megegyezik az online pókertermekével. Eléggé hasonlít a korábbiakban már említésre került PokerSnowie-hoz. A fő különbség az, hogy itt nem csak egyfajta bot ellen játszhatunk, hanem nagyon sok különböző féle ellenfél közül válogathatunk. Több csoportra is fel vannak osztva, vannak az igen egyszerű

botok, amik elég rosszul játszanak, sokszor szándékosan is furcsa döntéseket hoznak. Vannak ezen kívül nagyon jól megírt botok is a programban, és azok között is mindenféle játéktípust képviselők. Így ideális tesztkörnyezet az alkalmazáshoz, ugyanis itt nagyon változatos erősségű és stílusú játékos ellen is be lehet vetni az általunk írt programot.

Hasznos még, hogy van saját beépített elemző része, ami a tesztelés során nagyon jól használható arra, hogy megtaláljuk, a játék milyen helyzeteiben hoz rossz döntéseket az általunk készített bot, így miután azt sikeresen meghatároztuk, elkezdhetünk javítani a döntési mechanizmuson.

Az általam választott MI megvalósítási módszer a **tudás-alapú** volt.

A preflop játék folyamán az általam meghatározott szabályok alapján játszik, postflop pedig különböző formulák alapján hozza meg a döntéseket.

#### **Az alkalmazás felhasználói követelményei:**

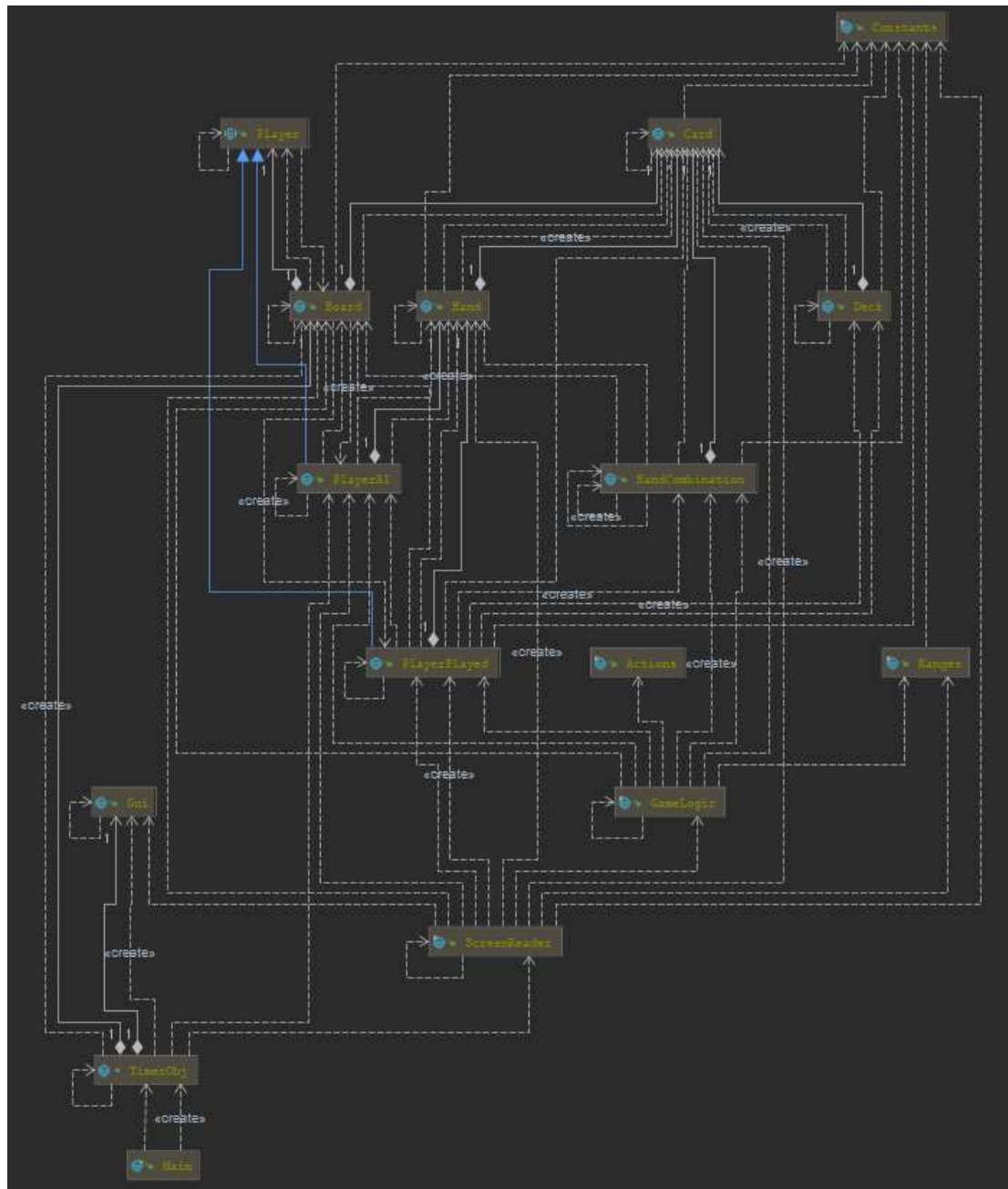
- Egy grafikus felhasználó felület segítségével lehessen működtetni
- Legyen egyszerűen telepíthető és működésbe hozható
- Képes legyen egy amatőr pókerjátékos szintjén játszani
- Automatikusan hajtsa végre az általa hozott döntéseket

#### **Az implementációval szembeni követelmények:**

- Legyen objektumorientált a megvalósítása
- Legyen előtérbe helyezve a későbbi fejlesztési lehetőség, ezért moduláris felépítésű
- Lehetőség szerint a futási idő legyen jól optimalizálva
- Legyen részenként is könnyen tesztelhető
- Rossz input adatok esetén se szakadjon meg a működés

A következő oldalon megtekinthető lesz az elkészült alkalmazás osztálydiagrammja. Így könnyen áttekinthetők lesznek az általam elkészített osztályok, és a köztük lévő kapcsolatok. Ezek a későbbiekben részletesen is bemutatásra kerülnek.





20. ábra: Az elkészült alkalmazás osztálydiagrammja

Jól látható a fenti ábrán, hogy egyes osztályoknak szinte az összes osztállyal valamilyen közvetlen kapcsolatban állnak. Kiemelhető talán a **Card** és a **Board** osztály. Ők rendelkeznek a legtöbb kapcsolattal. A továbbiak folyamán az osztályok egyesével kerülnek bemutatásra.

## 4.2.1 Osztályok

### 4.2.1.1 Constants osztály

Ez az osztály tárolóként funkcionál. Itt vannak eltárolva különféle konstansok, amiket más osztályok felhasználnak. A legtöbb itt tárolt konstans a leolvasáshoz szükséges pixelek koordinátáinak tárolására szolgál, de vannak eltárolva más adatok is itt.

### 4.2.1.2 Ranges osztály

Ez az osztály tárolja a különféle játékos laptartományokat, amelyek a későbbiekben bizonyos események hatására egy-egy játékoshoz lesznek rendelve. Integer tömbökben vannak eltárolva számként. Ezek a számok mögötti logika a **Hand** osztályon belül a **calculateHandId()** metódus ad majd magyarázatot.

Az osztály tartalmaz két metódust is, ezek a laptartományok kialakításában nyújtottak segítséget, szerepük nincs már a program futása során.

#### **rangeHelper() metódus:**

A lapok a **Constants** osztályon belül vannak eltárolva szöveges és prímszamos formátumban. A szöveges formátum esetében egy String tömb került felhasználásra, a lapok itt egymás után vannak eltárolva, a legkisebb 2-es lapoktól az ászokig, egymást követi a 4 szín. A prímszamos változat esetében pedig ugyanezen lapok vannak eltárolva, a legkisebb 52 prímszám segítségével, így mindegyik egyedi módon azonosítható, és a kombinációjuk is mindig egyedi azonosítót fog kapni majd.

A függvény a laptartományok kialakításához nyújtott segítségét, a következő módon:

- Két for ciklus segítségével végig megyünk a prímszámokat tartalmazó tömbön
- Az egyik az elejéről indul, a másik a végéről és mikor találkoznak akkor megáll, így a két ciklus nem tartalmaz ugyanolyan elemet, ezzel elkerülve a duplikációt
- Mivel a lapok sorrendben vannak tárolva, ezért, **ha 4-el kerülnek osztásra megkapjuk a lap értéket, ha pedig maradékos osztást végzünk 4-el akkor a színeket kapjuk meg.**
- A két lap alapján **csoportokra** bonthatjuk a lapokat, minden párból például 6 lehetséges kombináció van, egyszínű kártyákból értelemszerűen 4 fajta kombináció lehet, nem egyszínű nem párt tartalmazó 2 lapos kombinációból pedig 12 féle lehet. Ezeket csoportosítva lementi a **HashMap**-be, ezeket kiírás

után egy txt fájlba mentettem és az értékeket innen tudtam kimásolni a  
laptartományok kidolgozása folyamán.

```
68s ; [5723, 6161, 6901, 7597]
56s ; [2419, 2623, 3149, 3763]
2Qs ; [358, 543, 955, 1351]
6Qo ; [11387, 11269, 10679, 11773, 11651, 10919, 12931, 12127, 11993, 13561, 12851, 12709]
```

## 21. ábra: A rangeHelper() metódus által generált szöveg

A játék laptartományok kidolgozása folyamán nagyjából a jelenleg elfogadott és széleskörben használt meta stratégiák alapján lettek kialakítva.

3 + 1 játékos kategóriát határoztam meg. A játékosok lehetnek lazák, standardok illetve feszesek, ezen kívül van még egy úgynevezett hal (fish) laptartomány, a pókerben így szokás nevezni a gyenge játékosokat, ugyanis a cápák (sharks) a jó játékosok, akik megeszik a halakat.

A tartományok kidolgozásához a PokeStrategy Equilab szoftverét használtam.[29] Nagyon jó vizuális felülete van, és az általunk kialakított tartományok lementhetők benne.

Példának pár kidolgozott tartomány az equilab szoftveréből (balról jobbra):

- Cutoff pozícióból standard nyitóemelés
- Standard cutoff pozíció elleni 3Bet
- Standard button pozíció elleni megadás

## 22. ábra: Kidolgozott laptartományok Equilab-ban.

A kidolgozott laptartományok:

A T karakter a 10-es lapokat tartalmazza, a kombináció végén lévő s karakter pedig, hogy egyszínű lapok.

Nyitóemelő tartományok (pozíció sorrendben):

**Feszés:**

- TT+, AQs+, AKo
- TT+, AQs+, AKo
- TT+, AQs+, AQo+
- 99+, AJs+, AQo+
- 88+, AJs+, KQs, AJo+, KQo
- 22+, A9s+, KTs+, QTs+, JTs, T9s, 98s, 87s, ATo+, KTo+, QTo+, JTo
- 22+, A2s+, K9s+, Q9s+, J8s+, T9s, 98s, 87s, 76s, 65s, 54s, A8o+, KTo+, QTo+, JTo, T9o, 98o
- 77+, A7s+, K9s+, Q9s+, J9s+, T9s, 98s, 87s, ATo+, KTo+, QTo+, JTo

**Standard:**

- 77+, AQs+, AKo
- 77+, AQs+, AKo
- 77+, AJs+, KQs, QJs, JTs, T9s, 98s, AQo+
- 66+, AJs+, KQs, QJs, JTs, T9s, 98s, 87s, AJo+, KQo
- 55+, ATs+, KJs+, QJs, JTs, T9s, 98s, 87s, 76s, ATo+, KQo
- 22+, A7s+, K9s+, Q9s+, J9s+, T8s+, 97s+, 87s, 76s, 65s, 54s, A9o+, KTo+, QTo+, JTo, T9o, 98o, 87o
- 22+, A2s+, K9s+, Q9s+, J8s+, T8s+, 97s+, 86s+, 75s+, 65s, 54s, A2o+, KTo+, QTo+, J9o+, T8o+, 98o, 87o, 76o, 65o
- 22+, A2s+, K9s+, Q9s+, J9s+, T9s, 98s, 87s, 76s, 65s, 54s, ATo+, KTo+, QTo+, JTo

**Laza:**

- 22+, AJs+, KQs, QJs, JTs, T9s, 98s, AQo+
- 22+, AJs+, KQs, QJs, JTs, T9s, 98s, AQo+
- 22+, ATs+, KQs, QJs, JTs, T9s, 98s, 87s, 76s, ATo+, KQo

- 22+, A9s+, KJs+, QTs+, J9s+, T8s+, 97s+, 87s, 76s, 65s, ATo+, KQo
- 22+, A7s+, KTs+, QTs+, J9s+, T8s+, 97s+, 86s+, 76s, 65s, 54s, ATo+, KJo+
- 22+, A2s+, K9s+, Q9s+, J8s+, T8s+, 97s+, 86s+, 75s+, 64s+, 54s, A2o+, KTo+, QTo+, J9o+, T8o+, 98o, 87o, 76o
- 22+, A2s+, K2s+, Q4s+, J7s+, T8s+, 97s+, 86s+, 75s+, 64s+, 54s, A2o+, K7o+, Q8o+, J8o+, T8o+, 97o+, 87o, 76o, 65o, 54o
- 22+, A2s+, K9s+, Q9s+, J9s+, T8s+, 97s+, 86s+, 76s, 65s, 54s, ATo+, KTo+, QTo+, J9o+, T8o+, 98o

Standard tartományok elleni játék, nyitóemelés esetén, pozíció szerint kettesével (3Bet és megadás):

- **UTG1:**
  - KK+
  - AKs, AKo
- **UTG2:**
  - KK+
  - AQs+, AKo
- **MP1:**
  - QQ+, AKs, AKo
  - QQ-TT, AQs+, KQs, QJs, AQo+
- **MP2:**
  - JJ+, AKs, A5s, AKo
  - QQ-TT, AJs+, KQs, QJs, JTs, AQo+
- **MP3:**
  - JJ+, AKs, A5s, AKo
  - QQ-66, AJs+, KQs, QJs, JTs, T9s, 98s, AQo+
- **CO:**
  - JJ+, AKs, A5s-A2s, JTs, AKo
  - QQ-22, ATs+, KTs+, QTs+, JTs, T9s, 98s, 87s, AQo+, KQo
- **BTN:**
  - JJ+, AKs, A5s-A2s, JTs, AKo
  - QQ-22, ATs+, KTs+, QTs+, JTs, T9s, 98s, 87s, AQo+, KQo

- **SB:**
  - QQ+, AKs, AKo
  - QQ-TT, AQs+, KQs, QJs, AQo+

Standard játékos elleni 3Bet megadó tartományok (korai és késői pozíciókból):

- QQ-JJ, AKs, AKo
- JJ-TT, AQs-AJs, KQs, QJs, JTs

Standard játékos elleni 4Bet tartomány:

- KK+

Ellenfelek tartományai sorrendben játékos típus szerint (3Bet korai pozíció, 3Bet késői pozíció, 3Bet megadás, 4Bet):

- **Standard játékos**
  - JJ+, AKs, A5s-A2s, AKo
  - TT+, AQs+, A5s-A2s, JTs, T9s, 98s, 87s, 76s, AQo+
  - TT+, AJs+, KQs, AQo+
  - TT+, ATs+, KTs+, QTs+, J9s+, T8s+, 98s, 87s, 76s, 65s, AJo+, KQo
  - JJ+, AKs, A5s-A2s, JTs, AKo
- **Laza játékos**
  - TT+, AQs+, A5s-A2s, JTs, T9s, 98s, 87s, 76s, AQo+
  - TT+, A2s+, KQs, J9s+, T8s+, 97s+, 87s, 76s, 65s, 54s, AQo+
  - TT+, ATs+, KTs+, QTs+, J9s+, T8s+, 98s, 87s, 76s, 65s, AJo+, KQo
  - TT+, A2s+, KTs+, QTs+, J9s+, T8s+, 97s+, 86s+, 75s+, 65s, AJo+, KQo
  - JJ+, AQs+, ATs, A5s-A2s, QTs, J9s+, T9s, 98s, 87s, 76s, AKo

A feszes játékosnak ilyen tartományai nem lettek kialakítva, ugyanis a saját játékos által használt részletesebb standard játékosok elleni tartományokat használja szintén.

### 4.2.1.3 Card osztály:

Card		
f	name	String
f	value	int
f	suit	String
f	primeValue	int
m	suitDefine(String)	String
m	toString()	String
P	name	String
P	suit	String
P	primeValue	int
P	value	int

23. ábra: Card osztály

Ez az osztály a különböző kártyalapokhoz készült. A kártyák szöveges nevét, színét valamint az egyedi prímszamos értékét tárolja.

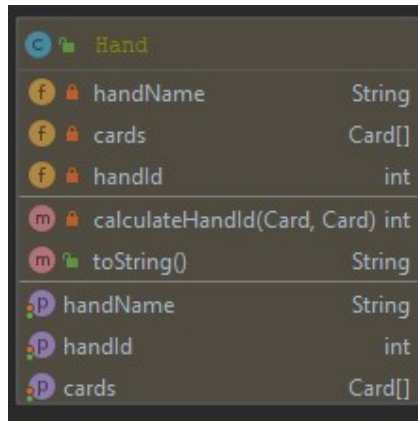
#### 1. kódrészlet: Card osztály konstruktora

```
public Card(String name) {
    name = name.replace("10", "t");
    boolean exists = false;
    for (int i = 0; i < Constants.cardPrimes.length; i++) {
        if (name.toLowerCase().equals(Constants.cards[i])) {
            this.name = name;
            this.value = i;
            this.primeValue = Constants.cardPrimes[i];
            this.suit = suitDefine(name);
            exists = true;
        }
    }
    if (!exists) {
        System.out.println("Error");
        System.out.println(name);
    }
}
```

Fent látható az osztály konstruktora. Egy átalakítás történik, ugyanis a leolvasó a 10-es értékű lapokat a kártyának megfelelő jel szerint látja, de a tárolásuk T formátumban van, ezért ez kicserélésre kerül, ha jelen van az input String-ben. Ezek után megnézi, hogy az adott lap benne van-e az 52 lapot tartalmazó tömbben, ha igen akkor az értékeket beállítja annak megfelelően, ha pedig nem akkor nem jön létre a lap.

**suitDefine(String) metódus:**

Ez a metódus a lap színét állítja be megfelelően, a bejövő String második karaktere alapján, amik az adott szín nevek rövidítései.

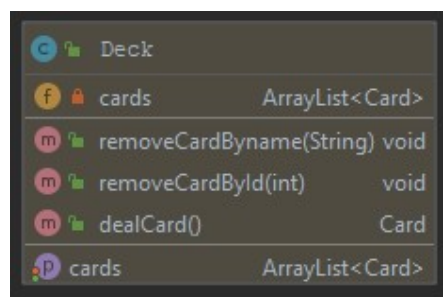
**4.2.1.4 Hand osztály:**

24. ábra: Hand osztály

A Hand osztály a játékos indulókezekhez készült. Két kártyából áll, ezeknek a kombinált nevéből, valamint egy egyedi prímszámú azonosítóból.

**calculateHandId(Card, Card) metódus:**

Ez a metódus a két kártya prímszámú értéke, amiből a kéz áll összeszorozza, így jön létre az egyedi prímszámú azonosító. Ez biztosítja, hogy ha ugyanazt a két lapot más sorrendben is kapjuk azonosító szerint ugyanaz a kéz lesz, ugyanis a sorrendiség nem számít.

**4.2.1.5 Deck osztály**

25. ábra: Deck osztály



Ez az osztály a kártyapaklihoz készült. Az 52 lapot tárolja, ami egy pakliban megtalálható. Ezt a **Constants osztályon** belül található kártyatömbből már a konstruktora segítségével eltárolja a listájában.

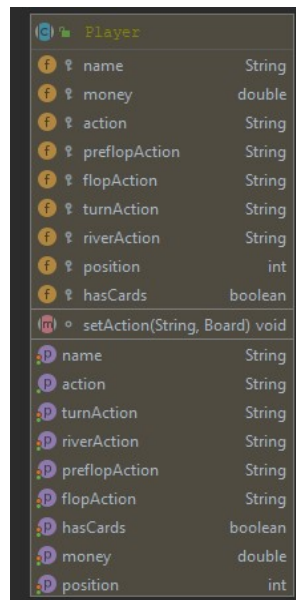
#### **removeCardByname(String) és removeCardById(int) metódus:**

Ez a két metódus arra szolgál, hogy a pakliból 1-1 lap eltávolításra kerüljön. Ez egyik String alapján távolítja el, a másik pedig az egyedi azonosítója alapján.

#### **dealCard() metódus:**

Ez a metódus a kártyák osztására szolgál. A még a pakliban lévő lapok közül véletlenszerűen visszaad egy lapot.

### **4.2.1.6 Player osztály:**

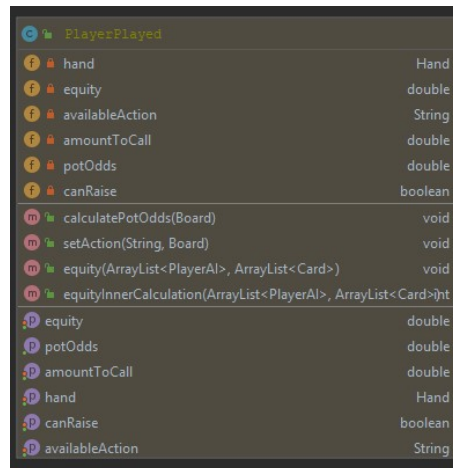


Player	
?	name String
?	money double
?	action String
?	preflopAction String
?	flopAction String
?	turnAction String
?	riverAction String
?	position int
?	hasCards boolean
m	setAction(String, Board) void
P	name String
P	action String
P	turnAction String
P	riverAction String
P	preflopAction String
P	flopAction String
P	hasCards boolean
P	money double
P	position int

**26. ábra: Player osztály**

Ez egy absztrakt osztály. A két fajta játékos típus adatai tárolását szolgálja. Ezek a név, a pénzüsszegük, hogy az adott játékos jelenleg rendelkezik-e lapokkal, tehát aktív-e a körben. A játékosok pozíció, valamint a különböző körökben végrehajtott akcióik.

#### 4.2.1.7 PlayerPlayed osztály:



PlayerPlayed	
hand	Hand
equity	double
availableAction	String
amountToCall	double
potOdds	double
canRaise	boolean
calculatePotOdds(Board)	void
setAction(String, Board)	void
equity(ArrayList<PlayerAI>, ArrayList<Card>)	void
equityInnerCalculation(ArrayList<PlayerAI>, ArrayList<Card>)	int
equity	double
potOdds	double
amountToCall	double
hand	Hand
canRaise	boolean
availableAction	String

27. ábra: PlayerPlayed osztály

Ez az osztály a bot által játszott játékoshoz készült. Tartalmaz egy Hand adattagot, egy equity értéket, ami az aktuálisan kiszámított nyerési esély, egy amountToCall értéket, ami megmondja, hogy éppen mekkora tétet kellene megadni, egy potOdds értéket, ami meghatározza a jelenlegi pot odds-át a játékosnak, valamint egy boolean változót, ami megmutatja, hogy épp lehetséges-e emelést végrehajtani.

##### calculatePotOdds(Board) metódus:

Ez a metódus elosztja azt az összeget, amit éppen meg kellene adni, a pot jelenlegi összegével, ez adja meg a pot odds-ot.

##### equity(ArrayList<PlayerAI>, ArrayList<Card>) metódus:

##### 2. kódrészlet: equity metódus

```
public void equity(ArrayList<PlayerAI> opponent, ArrayList<Card> cards) {
    int wins = 0;
    int hands = Constants.equityHandsCount;
    long timerStart;

    wins = IntStream.range(0, hands).parallel().map(i ->
equityInnerCalculation(opponent, cards)).reduce(0, (x, y) -> x + y);

    double equity = (double) wins / hands;
    this.equity = equity;
}
```

Ez a módszer az **equityInnerCalculation()** módszert hívja meg egymás után a **Constans osztályban** megadott konstans értéknek megfelelően.

Eleinte ez a módszer egy sima for ciklust használt, ám mivel körülbelül 5 másodperc volt a futási idő, így most ezt a műveletet már a Java Stream használatával végzi, így a futási ideje körülbelül a felére csökkent. Így képes többszázon egyszerre végezni a műveletet, az operációs rendszer szabja meg, hogy hány szálat használhat éppen, nem mi végezzük az optimalizálást. Megszámolja, hogy az adott szimulált leosztást hány esetben nyerjük meg mi, ezt a végén elosztja a szimulált leosztások számával, így megkapjuk a nyerési esélyünket.

equityInnerCalculation(ArrayList<PlayerAI>, ArrayList<Card>) metódus:

### 3. kódrészlet: equityInnerCalculation() metódus

```
public int equityInnerCalculation(ArrayList<PlayerAI> opponents,
ArrayList<Card> cards) {
    ArrayList<Card> cardholder = new ArrayList<Card>();

    Deck deck = new Deck();
    deck.removeCardByname(this.hand.getCards()[0].getName());
    deck.removeCardByname(this.hand.getCards()[1].getName());
    int result = 1;

    for (Card card : cards) {
        deck.removeCardByname(card.getName());
    }

    ArrayList<Hand> opponentHands = new ArrayList<Hand>();

    //ellenfél kezek
    for (PlayerAI opponent : opponents) {
        Random random = new Random();
        Hand hand =
opponent.getRange().get(random.nextInt(opponent.getRange().size()));

        opponentHands.add(hand);
        deck.removeCardByname(hand.getCards()[0].getName());
        deck.removeCardByname(hand.getCards()[1].getName());

        cardholder.clear();
        cardholder.addAll(cards);
    }

    if (cardholder.size() == 3) {
        cardholder.add(deck.dealCard());
        cardholder.add(deck.dealCard());
    }
    if (cardholder.size() == 4) {
        cardholder.add(deck.dealCard());
    }
    if (cardholder.size() == 5) {
        int ownHand = new HandCombination(this.hand,
cardholder).getCombinationStrength();
        for (Hand opponentHand : opponentHands) {
            if (ownHand < new HandCombination(opponentHand,
cardholder).getCombinationStrength()) {
                result = 0;
            }
        }
    }
}
```

Ez a metódus paraméterként megkapja az ellenfeleinket, és az asztalon lévő lapokat. Ezt követően létrehoz egy **Deck osztályt**. A kártyák közül kiveszi a lapjainkat, majd az asztal lapjait is kiveszi, hogy ezek ne kerülhessen a későbbiekben kiosztásra. Utána véletlenszerűen az ellenfeleink laptartományából nekik is kioszt egy indulókezet, majd ezeket is kiveszi a

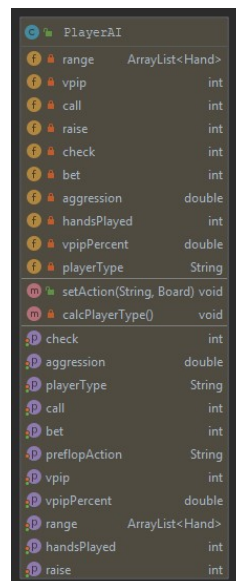
kártyalapok közül. Ezt követően megnézi hány lap van az asztalon, majd addig kisorsol új lapokat ameddig meg nem lesz a szükséges 5 lap.

Végezetül a **HandCombination osztály** segítségével a végső lapkombinációk segítségével összeveti a saját kombinációink erősségét az összes ellenfelével. Ha a miénk a legerősebb 1-es értékkel tér vissza, ha valakinek erősebb kombinációja van akkor pedig 0 értékkel.

#### **setAction(String, Board) metódus:**

Ez a metódus az éppen aktuális akciót állítja be, a **Board osztály** segítségével a játék állapotának megfelelően.

#### **4.2.1.8 PlayerAI osztály:**



**28. ábra: PlayerAI osztály**

Ez az osztály a PokerGenius által használt botok számára készült. Eltárolja a különböző múltbeli cselekvéseiket, ami alapján meghatározható, hogy milyen típusú játékosok. Ezen felül van egy range adattaguk, amiben tárolásra kerül az aktuális laptartományuk.

#### **setAction(String, Board) metódus:**

A metódus első fele megegyezik a **PlayerPlayed osztály** ugyanezen metódusával. A különbség az, hogy a második felében az említett akciótak számontartó változókba is tárolásra kerülnek a megfelelő értékek. A vpip érték az a **setPreflopAction()** metódusban változik, míg a hands változót a **Board osztály** egy metódusa fogja növelni.

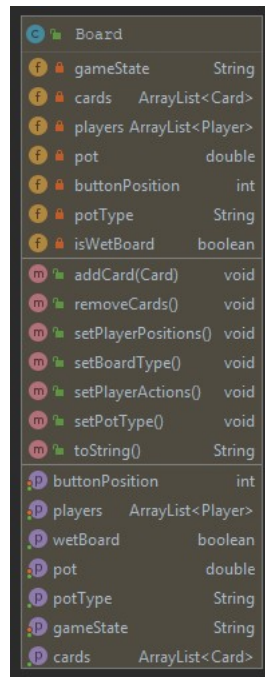
calcPlayerType() metódus:

#### 4. kódrészlet: calcPlayerType() metódus

```
private void calcPlayerType() {  
    this.vpipPercent = this.vpip / (double) this.handsPlayed;  
    this.aggression = (this.raise + this.bet) / (double) this.call;  
  
    if (this.handsPlayed > 50) {  
  
        if (this.vpipPercent < 0.12) {  
            this.playerType = "tight";  
        } else if (this.vpipPercent >= 0.12 && this.vpipPercent < 0.22) {  
            this.playerType = "standard";  
        } else if (this.vpipPercent >= 0.22 && this.vpipPercent < 0.35) {  
            this.playerType = "loose";  
        } else {  
            this.playerType = "fish";  
        }  
    }  
}
```

Ez a metódus végzi az adott játékos típusának a meghatározását. Kiszámolja a vpip százalékot, ami azt mutatja meg a lapok hány százalékával szállt játékba, és az agresszióját a játékosnak. Ha legalább 50 lejátszott leosztás már van az adott játékos esetében akkor meghatározásra kerül a típusa ezen értékek alapján. Ha a lapok kevesebb mint 12%-ával játszik, akkor feszes, ha 12% és 22% között akkor standard, ha ennél több de kevesebb mint 35%, akkor laza, ha pedig 35% felett, akkor hal a játékos típusa. Az agressziófaktor a **GameLogic osztály** esetében játszik majd szerepet, alapvetően a 2.0 agressziónál magasabb játékos lesz agresszív, alatta passzív a játékos.

### 4.2.1.9 Board osztály:



29. ábra: Board osztály

Ez az osztály a játéktérhez készült. Itt vannak eltárolva a játékosok egy listában. A közös lapok szintén egy másik listában, ezen kívül még a pot mérete, az osztógomb pozíciója, valamint a pot az asztal és a játék állapotának a tulajdonságai.

**addCard(Card) metódus:**

#### 5. kódrészlet: addCard() metódus

```
public void addCard(Card card) {
    boolean exists = false;
    for (Card card1 : this.cards) {
        if (card.getPrimeValue() == card1.getPrimeValue()) {
            exists = true;
        }
    }
    if (!exists) {
        cards.add(card);
    }
}
```

A képernyőolvasó által kerül meghívásra. Ellenőrzi, hogy az adott kártya szerepel-e a listában, ha még nem akkor azt hozzáadja.

**removeCards() metódus:**

Törli a listában lévő kártyákat.

**setPlayerPositions metódus:**

Ez a metódus az osztógomb helyzete alapján minden játékosnak kiosztja a saját pozícióját.

**setBoardType() metódus:****6. kódrészlet: setBoardType() metódus**

```
public void setBoardType() {
    int[] ranks = new int[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    // s h d c
    int[] suits = new int[]{0, 0, 0, 0};

    boolean wet = false;

    for (Card card : this.cards) {
        int rankhelper = -1;
        int suithelper = -1;
        for (int j = 0; j < Constants.cards.length; j++) {
            if (card.getName().equalsIgnoreCase(Constants.cards[j])) {
                rankhelper = j / 4;
                suithelper = j % 4;
                ranks[rankhelper]++;
                suits[suithelper]++;
            }
        }
    }

    for (int i = 0; i < ranks.length; i++) {
        if (i != 0) {
            if (ranks[i] >= 1 && ranks[i - 1] >= 1) {
                wet = true;
            }
        }
    }

    for (int suit : suits) {
        if (suit >= 2) {
            wet = true;
        }
    }

    this.isWetBoard = wet;
}
```

Ez a metódus az asztal típusát hivatott meghatározni. A pókerben van egy terminológia, az úgynevezett wet board. A wet board azt jelenti, hogy az asztal lapjai alapján lehetősége van valakinek még 1 lap érkezése esetén sort vagy színt kialakítania.



Ennek ellenőrzése két tömb segítségével történik. Ezek kezdetben ki vannak nullázva. Az első tömb 13 értéket tárol, mert 13 lapfajta van, a második pedig 4 értéket, mert 4 fajta szín van. Mivel az adott lapok sorban vannak tárolva a **Constants osztály** tömbjében, ezért megkeressük az adott lap pontos helyét, majd azt leosztva 4-el megkapjuk a lap típusát, majd maradékosan elosztva 4-el a színét. A kapott értéknek megfelelő helyen az adott tömbök értékeit növeljük 1-el. Így jól látható lesz, hogy melyik lapból illetve színből hány található az asztalon.

Ezt követően ellenőrizzük, hogy van-e két kapcsolódó egymást követő lap, ha van akkor az asztalon találhatók húzók. Ha két lap között található egy kimaradó, az esetben is mondhatnánk, hogy van húzó, de az ilyen, úgynevezett gutshot draw (ahol a sor középső eleme hiányzik) sokkal rosszabb esélyeket ad, mint a nyíltvégű sorhúzók, mert itt csak egy adott lap esetén jön létre a sor, míg a nyíltvégű esetén 2 lapfajta is sort eredményez, ezért ezeket az eseteket nem vettem számításba.

Ezt követően ellenőrzésre kerül, hogy van-e két vagy több ugyanolyan színű lap, ha igen akkor szintén húzós asztalról beszélünk. Ezek után a metódus visszaadja az eredményt.

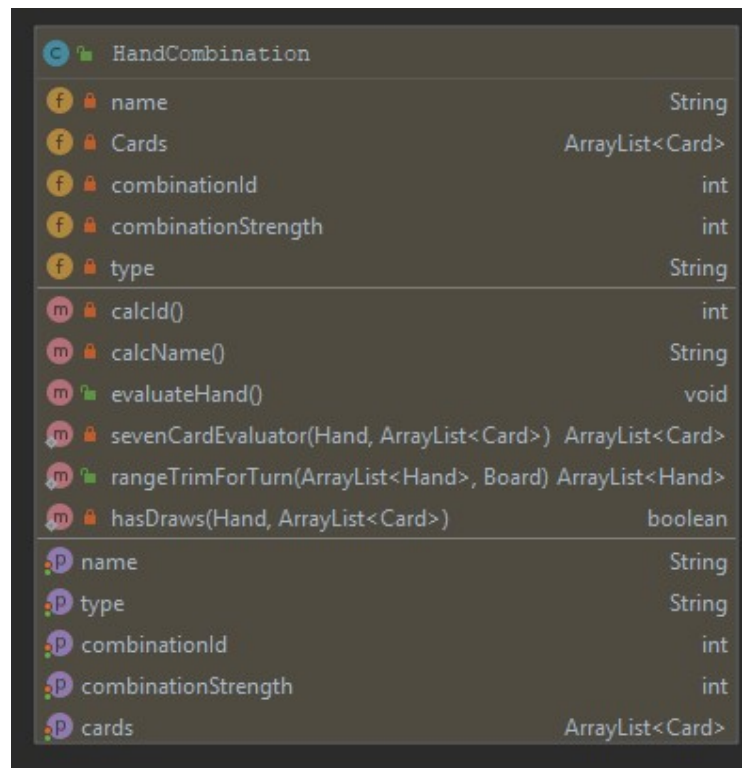
#### **setPlayerActions() metódus:**

Ez a metódus arra az esetre szolgál, ha mi preflop emelést hajtottunk végre, valamelyik ellenfél vagy ellenfelek pedig azt megadták, mivel a programban botok vannak, ezért ez gyakorlatilag kevesebb, mint egy másodperc alatt végrehajtodik, ezért a leolvasónak nincs ideje rögzítenie az ellenfél játékosok megadását. Viszont minden egyes tétkör után nekünk szükséges tudni az ellenfeleink akcióját. Mivel ha mi emeltünk és nem került hozzánk újra az akció, az azt jelenti ha valamely játékosnak lapjai vannak még és az előző tétkörből nincs rögzített akciója még, akkor ő biztosan megadott, ezért ilyenkor automatikusan ezen játékosok előző tétköréhez rögzítve lesz a megadás ténye.

#### **setPotType() metódus:**

A preflop játék utáni döntésekhez, nekünk azt szükséges tudnunk, hogy milyen típusú pot az, amit éppen játszunk. Ezt az emelések számából tudjuk meghatározni, ezt végzi ez a metódus, megszámlolja az emelések számát, ha maximum 1 emelés volt, akkor normál pot, ha 2, akkor 3bet pot, ha több mint kettő, akkor 4bet pot.

#### 4.2.1.10 HandCombination osztály:



HandCombination	
f	name String
f	Cards ArrayList<Card>
f	combinationId int
f	combinationStrength int
f	type String
m	calcId() int
m	calcName() String
m	evaluateHand() void
m	sevenCardEvaluator(Hand, ArrayList<Card>) ArrayList<Card>
m	rangeTrimForTurn(ArrayList<Hand>, Board) ArrayList<Hand>
m	hasDraws(Hand, ArrayList<Card>) boolean
p	name String
p	type String
p	combinationId int
p	combinationStrength int
p	cards ArrayList<Card>

30. ábra: HandCombination osztály

Ez az osztály az 5 lapos pókerkezekhez készült. Ezek azok a végső kész kezek, amik meghatározzák a játék győztesét. Ezeknek a kombinációknak van nevük, amik a lapjaik nevének összefűzése. Típusa, ami a létrejött kombináció neve, mint a sor, két pár, full house stb. Van egy erősségük, amit egy metódus határoz meg, ez alapján lehet egymással őket összehasonlítani.

##### calcId() metódus:

Ez a kombinációk egyedi azonosítója, az őket alkotó lapok azonosítóinak szorzata, mivel azok prímszámok így ez teljesen egyedi mindig, lapok sorrendje mivel nem számít, így sorrendtől függetlenül is ugyanaz lesz.

##### calcName() metódus:

Az 5 lap nevének összefűzésével állítja elő a nevet.

**evaluateHand() metódus:**

Ez a korábban bemutatott **setBoardType() metódushoz** hasonlóan működik, két kezdetben nullákat tartalmazó tömb segítségével állapítja meg, hogy melyik típusú kártyából mennyi van, illetve hogy melyik színből mennyi van. Ezek után ennek segítségével kerül meghatározásra a létrejött lapkombináció erőssége.

A legerősebb kombinációtól haladunk a gyengébb felé, ugyanis ha egy adott lap mondjuk RoyalFlush, akkor az egyben színsor, sor, szín is lenne, de értelemszerűen nekünk a lehető legerősebbre van szükségünk. Amint találtunk egy kombinációtípust tovább nem kell vizsgálódnunk.

RoyalFlush egy módon lehetséges, ha az utolsó 5 lapból van 1 db, illetve az összes egyszínű. Tehát az egyik tömbben az utolsó 5 számjegy 1-es kell, hogy legyen, a színeknél pedig valamelyik 5-ös a többi 0.

Színsor esetén hasonló, ám itt bármelyik egymást követő 5 lap lehet egyszínű, nem csak az utolsó 5. Figyelni kell arra, hogy sor kezdődhet ásztól is, az ász ugyanis lehet a legerősebb és a leggyengébb lap is sor esetében a kettes előtt. Ezért ez külön is ellenőrzésre kerül.

Ez után megvizsgáljuk, hogy póker létrejött-e. Ez úgy lehetséges, hogy egy adott lap típusból 4 ugyanolyan van, tehát az első tömbben kell egy 4-est keresnünk. Ha van, akkor pókerünk van, ez után még fel kell jegyezni az 5. lap erősségét is, ugyanis két póker kombináció között az dönt, hogy kinek magasabb az 5. lapja.

Utána a full vizsgálata történik. Full úgy lehet, ha van az egyik típusú lapból 3 db egy másik fajtából pedig 2 db. Így az első tömbben azt kell vizsgálnunk, hogy legyen egy 3-as és kettes számjegy. Két full közül az az erősebb, amelyiknek a 3-as kombinációja magasabb, ha ez megegyező, akkor a kettes kombináció dönt. Ezért mind a kettő erősségét jegyeznünk kell.

Full után a szín következik, ehhez csak azt kell vizsgálnunk, hogy a második tömbben van-e 5-ös számjegyünk. Ha van, akkor jegyezni kell a szín erősségét, ez az első tömb legmagasabb indexű eleme lesz.

Ez után a sor a következő, itt azt kell néznünk, hogy van-e az első tömbben 5 db egymást követő 1-es számjegy. Ha igen, akkor sorunk van és jegyezzük a legmagasabb tagját. Mint a színsor esetében is külön figyelni kell az ásszal kezdődő sorokra.

Utána a drill következik, ehhez az szükséges, hogy az első tömbben legyen 3-as számjegy. Ha van, akkor feljegyezzük az erősségét, valamint a másik két lap erősségét is, azonosság esetén azok döntenek a győztesről.

Utána következik a pár, két pár és a magas lapok vizsgálata. Ha van 2 db 2-es számjegy az első tömbben, akkor jegyezzük őket, valamint az ötödik lap magasságát is, ez a két pár. Ha csak 1 db 2-es számjegy van, akkor sima párunk van, ezt jegyezzük valamint a maradék 3 lapot is. Ha ez sincs, akkor pedig magas lapunk van, ez esetben szintén jegyezzük az 5 lap értékét.

Az érték meghatározása integer változó segítségével történik. Kellően nagy számokat választottam minden egyes kombinációnak, és ezekhez hozzáadásra kerül még az erősséget befolyásoló tényezők. Ezek úgy lettek tetszőlegesen beállítva, hogy biztos ne legyen köztük átfedés, és a sor értéke sehogyan ne lehessen magasabb, mint egy szín értéke például.

#### **sevenCardEvaluator(Hand, ArrayList<Card>) metódus:**

Mivel a No Limit Texas Hold'em során a legtöbbször 7 lap közül kell kiválasztanunk a legerősebb 5 lapos kombinációt, ezért a 7 lapos kiértékeléshez is szükség volt egy metódusra. Ehhez én egy **brute force** megközelítést alkalmaztam. A 7 lapból végig nézem az összes lehetséges 5 lapos kombinációt és a legerősebbet tartom meg.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad (2)$$

Mivel ezek ismétlés nélküli kombinációk, ahol a nem számít a sorrend, így ez esetben 21 kombináció van. Ezt az összes 21 kombinációt az alábbi dupla for ciklussal állítottam elő:

## 7. kódrészlet: Kombinációk előállítása és értékelése

```

for (int i = 0; i < combinations.size() - 1; i++) {
    for (int j = i + 1; j < combinations.size(); j++) {
        excluded.add(combinations.get(i));
        excluded.add(combinations.get(j));

        //kombináció létrehozása és kiértékelése a két kimaradó lap nélkül
        if (excluded.size() == 2) {
            combination.addAll(combinations);

            for (Card anExcluded : excluded) {
                combination.remove(anExcluded);
            }
            excluded.clear();
        }
        temp = new HandCombination(combination).getCombinationStrenght();

        if (temp >= combinationStrenght) {
            combinationStrenght = temp;
            container.clear();
            container.addAll(combination);
        }

        combination.clear();
    }
}

```

A belső ciklus mindig a külső után 1-el indul és végigmegy a tömbön, utána pedig a belső is halad egyet előre. Így ismétlés nélkül létrejön az összes kombináció. Ez után az összes kiértékelésre kerül, és a legerősebb értékű lesz a végléges 5 lapos kombináció.

### hasDraws(Hand, ArrayList<Card>) metódus:

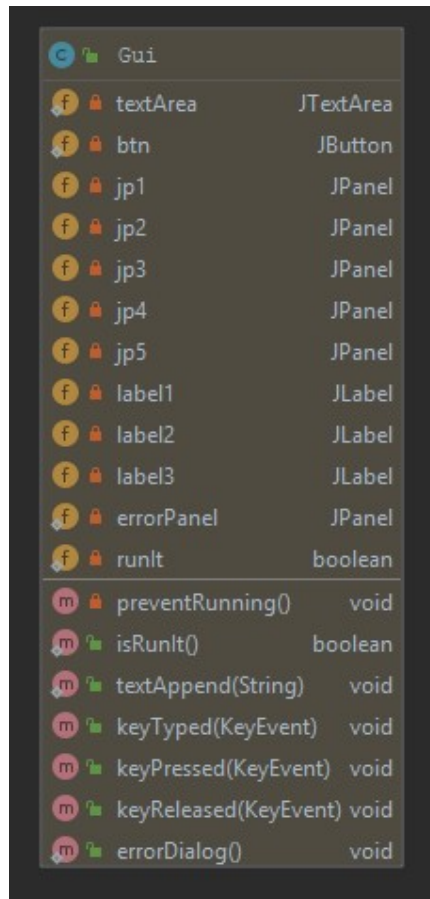
Ez a metódus a **Board osztályon** belül található **isWetBoard()** metódushoz hasonlóan azt nézi, hogy van-e húzó, csak ez adott játékosra vonatkozóan nézi, a **rangeTrimForTurn()** metódus fogja ezt használni.

### rangeTrimForTurn(ArrayList<Hand>, Board) metódus:

Ez a metódus arra szolgál, hogy a turn játékkörbe érkezve bizonyos ellenfeleink laptartományát szűkítsük, hogy a nyerési esély számítása pontosabb értéket adjon. Ezért bizonyos akciók hatására kivesszük azokat a lapokat a tartományukból, amelyekkel az adott akciókat biztosan nem hajtották volna végre. Ezek azok a lapok, amelyek a floppon nem eredményeztek csak magas lapot, tehát még párjuk se volt, illetve semmilyen lehetséges húzójuk se volt a floppon. Végigmegy a tartományunk és egyesével kiértékeli a lapjaikat,

amelyik kéz legalább egy párt vagy húzót tartalmaz az egy új laptartományba kerül eltárolásra, miután pedig az összes lapot megvizsgáltuk ez az új tartomány kerül az adott játékoshoz beállításra.

#### 4.2.1.11 Gui osztály



31. ábra: Gui osztály

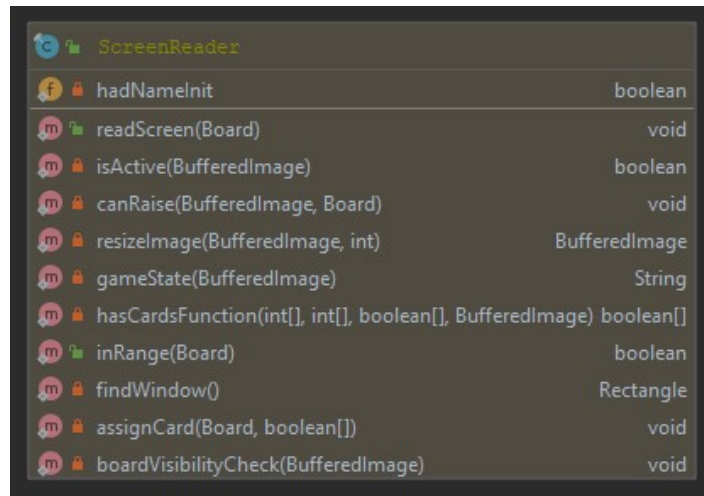
Ez az osztály a grafikus felület megjelenítésére szolgál. Elég egyszerű megvalósítással csak a szükséges dolgok kerültek bele. Van egy ablak egy szöveges ablakkal, ahol alapvető információk jelennek meg a játékról. A **textAppend()** metódussal tudnak ebbe a különböző objektumok írni. Került még bele egy gomb is, aminek segítségével elindítható az alkalmazás, de ez megtehető az F5 billentyű lenyomásával is, az **event listener**ek figyelik az F5 billentyű vagy a gomb megnyomásra. Ha a gomb megnyomásra került, akkor a runIt statikus változó is true állapotba kerül, ezzel elindítva a többi objektum működését.

Került még ide egy hibát jelző ablak is, amely, ha a PokerGenius asztal nem látható, akkor meghívásra kerül és a futást is leállítja.

#### 4.2.1.12 TimerObj osztály

Ez az osztály a Java **TimerTask osztály** leszármazottja. Feladata az alkalmazás folyamatos futtatása. Ha a **Gui osztályban** található **isRunIt** változó **true**, akkor elindítja a leolvasási folyamatot, ami meghívja a többi osztályt is, így kerül megállapításra a szükséges akció végezetül. Itt kerül létrehozásra az adatbázisunk és a táblánk is, ha még nem létezik.

#### 4.2.1.13 ScreenReader osztály



32. ábra: ScreenReader osztály

Ez az osztály felelős az asztalról az összes szükséges adat leolvasásért, majd a további objektumok különböző metódusainak meghívásáért.

**findWindow() metódus:**

#### 8. kódrészlet: findWindow() metódus

```
private static Rectangle findWindow() {
    final Rectangle rect = new Rectangle(0, 0, 0, 0);
    WindowUtils.getAllWindows(true).forEach(desktopWindow -> {
        if (desktopWindow.getTitle().contains("Genius")) {
            rect.setRect(desktopWindow.getLocAndSize());
        }
    });
    return rect;
}
```

Ez a metódus végzi el a PokerGenius ablakánál megtalálását. Ezt egy Windows natív kéréssel keresi, ahol az ablakok címei között megkeresi azt, amelyikben a „Genius” kifejezés

megtalálható. Ennek az ablaknak a paramétereit utána beállítja az általa létrehozott négyzetbe.

Az összes további adat leolvasása, ebből az ablakképből kimetszéssel történik. Az összes elem helyzete mindig fix, az asztal bal felső sarkától nézve ugyanazon a helyen jelennek meg a dolgok az asztalon.

Minden más leolvasása ennek a képnek a sarkához viszonyítva történik, mint az alábbi példa is mutatja:

### 9. kódrészlet: Pot összegét tartalmazó képmetszet készítése

```
BufferedImage screenCapturePot = robot.createScreenCapture(  
    new Rectangle(rect.x + Constants.potTextX, rect.y + Constants.potTextY,  
    Constants.potWidth, Constants.potHeight));  
  
BufferedImage screenCapturePotResized = resizeImage(screenCapturePot, 2);
```

#### **boardVisibilityCheck(BufferedImage) metódus:**

Ez a metódus ellenőrzi néhány előre megadott pixelnek a készített képernyőképen. Ha ezek nem egyeznek meg a hozzájuk tartozó színekkel, akkor nem látható az a része az asztalnak, vagy rosszak a beállítások és meghívja a **Gui osztály** hibaüzenetét.

#### **hasCardsFunction(int[], boolean[], BufferedImage) metódus:**

Ez a metódus végigmegy a játékosokhoz eltárolt lapok pixeleinek helyein. Ez minden játékoshoz a lap sarkának egy fehér pixele. Ha ez megegyezik az adott pixel színekkel, akkor az érték true lesz, tehát a játékos előtt van lap.

#### **assignCard(Board, boolean[]) metódus:**

Ez a metódus az előző metódus által beállított tömbön végighalad, ahol true értéket talál, ahhoz a játékoshoz beállítja, hogy vannak lapjai.



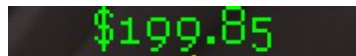
**gameState(BufferedImage) metódus:**

Ez a metódus végig nézi az asztalon lévő közös lapok helyeit, és azt itt talált pixelek alapján eldönti, hogy hány lap található az asztalon. A lapok száma alapján meghatározza a játék jelenlegi állapotát.

**resizeImage(BufferedImage, int) metódus:**

Erre a metódusra azért volt szükség, mert a leolvasó az alkalmazás által használt nem megszokott karakterek olvasásakor sokszor hibás eredményt adott, ezzel a metódus segítségével bizonyos képeket fel lehet nagyítani, aminek hatására az OCR pontosabb eredményt ad vissza.

A legtöbb problémát a pénzösszegek okozták:



**33. ábra: Pénzösszeg példa**

Látható, hogy nem megszokott betűtípust használ, egyes karakterek magassága eltérő, ezért volt szükség a nagyításra, mely esetében a kapott eredmény még mindig nem 100%-os, de sokat javult a nagyítás nélküli állapothoz képest.

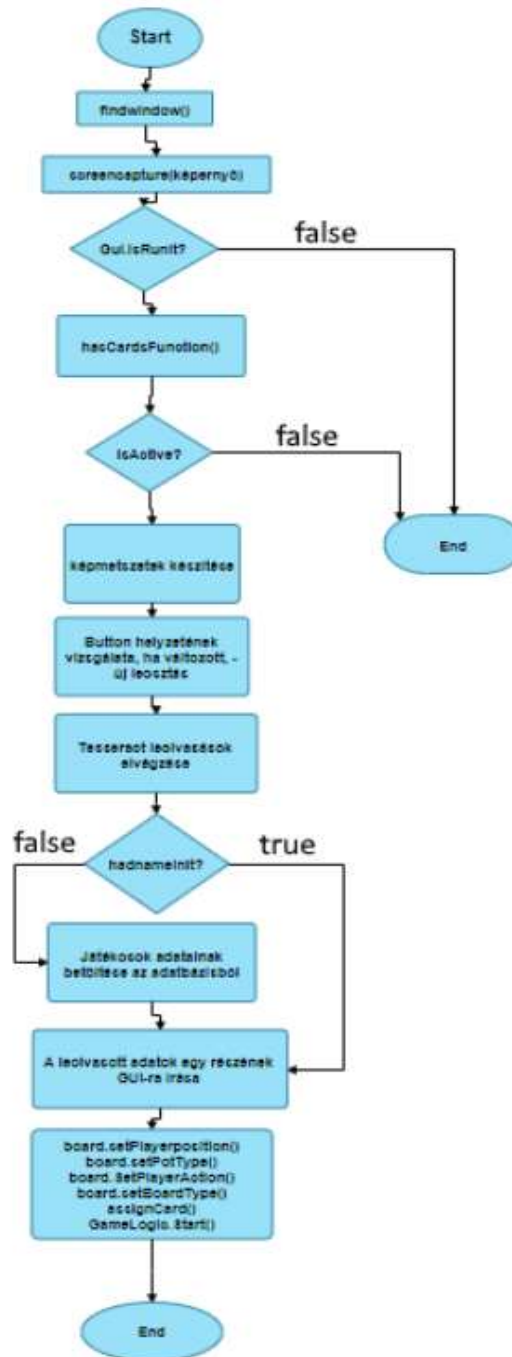
**canRaise(BufferedImage) metódus:**

Ez a metódus a játékos előtt lévő jobb oldali gombot vizsgálja. Ha ez jelen van akkor azt jelenti, hogy tudunk emelni, ha nincs jelen akkor nem lehetséges emelés.

**isActive(BufferedImage) metódus:**

Ez a metódus a játékos előtt található narancssárga gombok pixeleit vizsgálva állapítja meg, hogy van-e épp előtte gomb. Ha van, akkor döntési helyzet előtt állunk, ilyenkor a többi adat is leolvasásra kerül.

**readScreen(Board) metódus:**



**34. ábra: readScreen() metódus folyamatábrája**

Ez egy elég nagy metódus, ahogy az a fenti folyamatábrán is látszik, nagyon sok műveletet végez. Először meghívja a findWindow() metódust, majd az egész képernyőről készül egy kép. Ezt követően megvizsgálja, hogy a Gui-ban található változó szerint kell-e

tovább futnia. Ha igen, akkor az összes adatról elkészíti a kisebb képernyőképeket. Ezt követően újabb pixelek vizsgálatával megkeresi az asztalon lévő osztógomb helyzetét. Ha ezt változott az előzőleg rögzített képest, az új leosztást jelent. A játékosok eddigi cselekvéseit, amit az előző körben hajtottak még végre törli.

Ez után a Tesseract OCR hívásával a kis metszeteket egyesével leolvassa, majd ezeket az adatokat az objektumokba tölti. Külön van a leolvasó paraméterezve a szám adatok leolvasásához, hogy pontosabb legyen az eredmény.

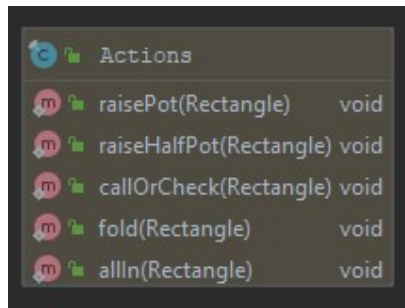
Ezt követően megvizsgálja, hogy a játékosok voltak-e már az adatbázisból betöltve. Ha nem, akkor az adatbázisból lekérdezéseket hajt végre, ha van benne ilyen nevű játékos, akkor betölti az adatait, ha nem volt még ilyen akkor egy új rekordot hoz létre annak a játékosnak.

### 10. kódrészlet: Játékosok betöltése az adatbázisból

```
Connection conn = DriverManager.getConnection("jdbc:sqlite:D:\\players.db");
Statement statement = conn.createStatement();
for (PlayerAI opponentPlayer : opponentPlayers) {
    statement.execute("SELECT * FROM players WHERE name= '" +
        opponentPlayer.getName() + "'");
    ResultSet results = statement.getResultSet();
    if (results.next()) {
        opponentPlayer.setHandsPlayed(results.getInt("hands"));
        opponentPlayer.setVpip(results.getInt("vpip"));
        opponentPlayer.setCall(results.getInt("call"));
        opponentPlayer.setBet(results.getInt("bet"));
        opponentPlayer.setCheck(results.getInt("checks"));
        opponentPlayer.setRaise(results.getInt("raise"));
    } else {
        statement.execute("INSERT INTO players (name, call, raise, bet,
            checks, vpip, hands) " +
            "VALUES ('" + opponentPlayer.getName() + "', " +
            opponentPlayer.getCall() + ", " + opponentPlayer.getRaise() + ", " +
            opponentPlayer.getBet() + ", " + opponentPlayer.getCheck() +
            ", " + opponentPlayer.getVpip() + ", " + opponentPlayer.getHandsPlayed() +
            ")");
    }
}
statement.close();
conn.close();
```

Ezt követően meghívja a **Board objektum** szükséges metódusait, amik az asztalhoz tartozó beállításokat hajtják végre. Végezetül meghívja a **GameLogic osztály metódusát**, ami majd a szükséges akció megtalálásáért felel.

#### 4.2.1.14 Actions osztály:



35. ábra: Actions osztály

Ennek a statikus osztálynak az a feladata, hogy végrehajtsa a PokerGenius asztalon a különböző akciókat. Ezt a java **Robot osztály**a segítségével hajtja végre. Itt találhatók az egérmozgató parancsok.

A metódusok ugyanúgy néznek ki, csak a koordináták változnak, hogy hova kell kattintani az egérrel. A mozgatósi utasítás duplán szerepel, mert ha csak egyszer szerepelt nem mindig kattintott a megfelelő helyre. Valószínűleg a Windows beépített egérgyorsítása okozhatja ezt a problémát. Alább látható egy példa az egyik ilyen metódusról.

#### 11. kódrészlet: raisePot() metódus

```
public static void raisePot(Rectangle rect) throws
AWTException {
    Robot bot = new Robot();
    bot.mouseMove(rect.x + 750, rect.y + 662);
    bot.mouseMove(rect.x + 750, rect.y + 662);
    bot.mousePress(InputEvent.BUTTON1_DOWN_MASK);
    bot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK);
}
```

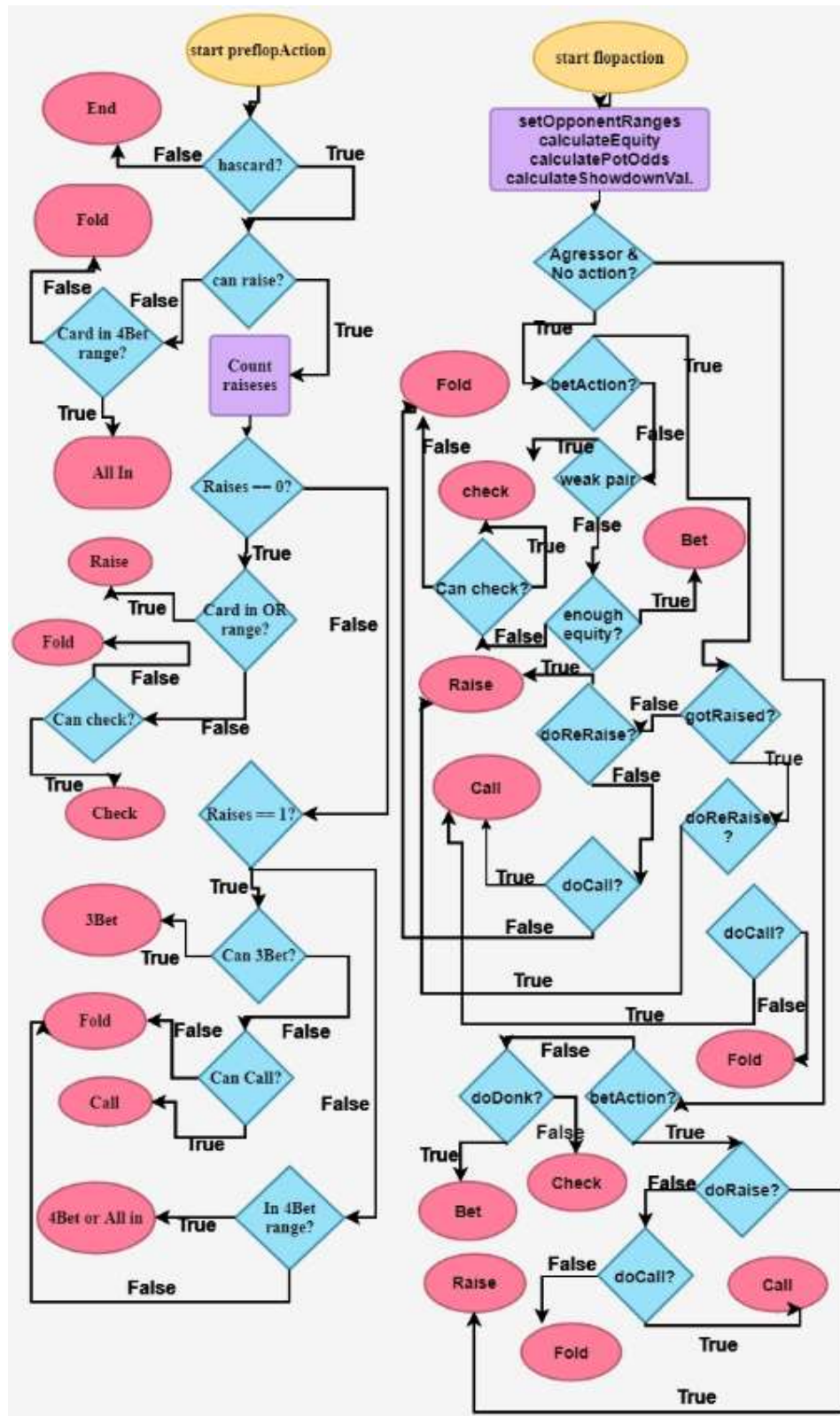
#### 4.2.1.15 GameLogic osztály

Ez a statikus osztály felelős a döntési logika végrehajtásáért. 5 metódusa van, a 4 féle játékkörhöz egy metódus, valamint az ezeket elindító **start()** metódus.

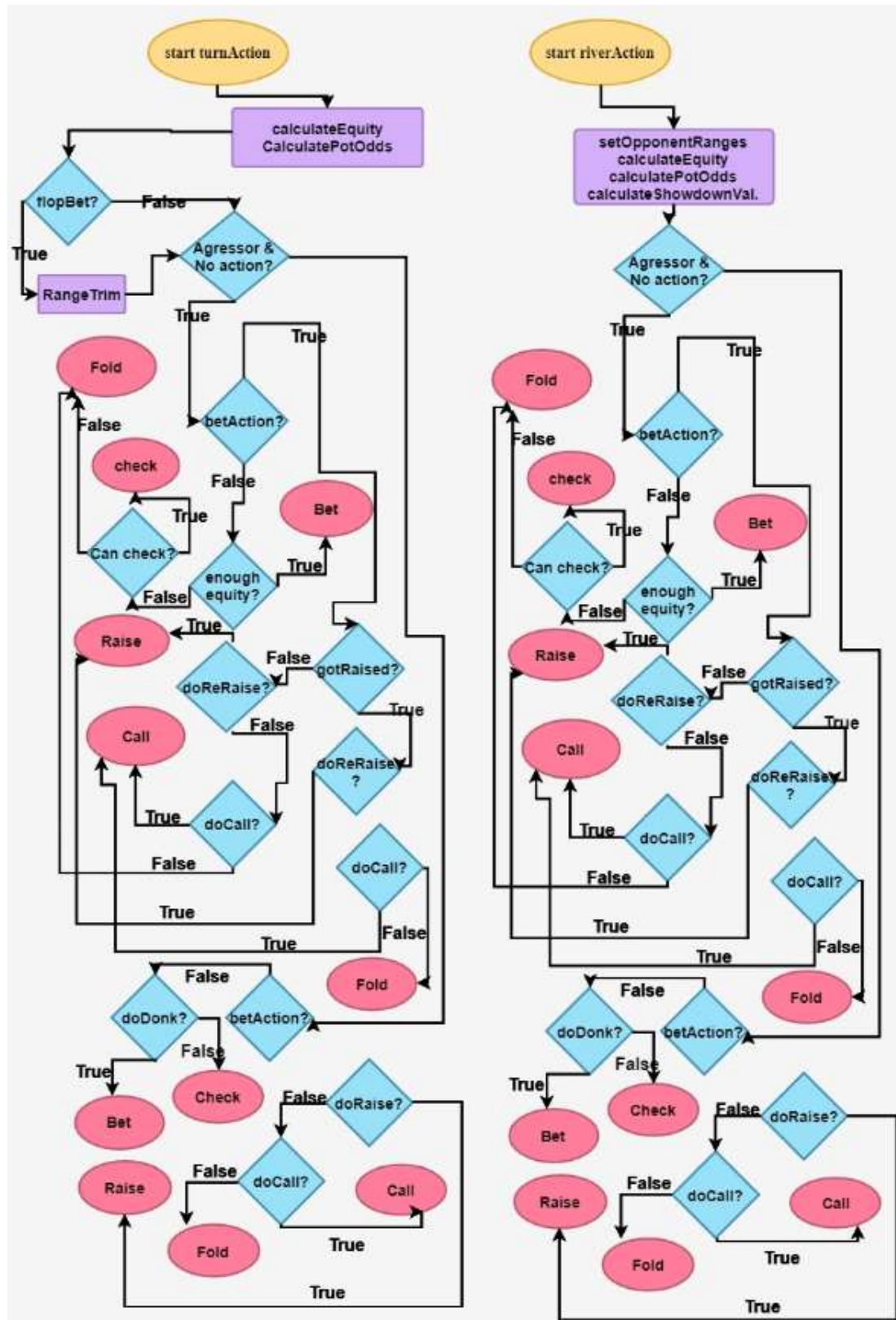
##### **start(Board, Rectangle) metódus:**

Ennek a metódusnak az a szerepe, hogy megállapítsa, épp milyen játékkör van, aztán pedig annak megfelelően meghívni a 4 metódusból az éppen szükségeset.

A másik 4 metódus összetettségük miatt a következő folyamatábrák után lesz részletesen bemutatva:



36. ábra: preflopAction() és flopAction() metódus folyamatára



37. ábra: turnAction() és riverAction() folyamatábra



**preflopAction(Board, Rectangle) metódus:**

Ez a metódus felelős a preflop döntéshozásért. Először megvizsgálja, hogy van-e épp lap a kezünkbe. Ha nincs akkor meg is áll. Ha van lapunk, akkor először megnézi, hogy van-e lehetőségünk emelésre, ha nem tudunk emelni, azt jelenti hogy valaki All In ment, ezért megnézzük, hogy az adott lapunk benne van-e a 4Bet tartományunkba, ha igen akkor megadjuk az emelést.

Ha tudunk emelni, megszámolja, hogy eddig hány emelés történt. Ha eddig nem volt előttünk emelés, akkor megnézi, hogy a lapunk benne van-e az emelő laptartományunkban, ha igen, akkor emelés történik. Ha nincs benne, akkor, ha tud passzol, ha a passzolásra nincs lehetőség, akkor eldobja a lapokat.

Ha volt már előttünk emelés, akkor azt kell megvizsgálni, hogy a 3Bet laptartományunkban benne van-e a lap. Ha benne van, akkor visszaemelünk. Ha nincs benne, akkor utána megnézzük, hogy a megadó tartományunkban benne van-e a lap. Ha igen, akkor megadás történik, ha nem akkor pedig eldobjuk a lapot.

Ha ennél is több emelés volt, akkor pedig megnézzük, hogy a 4Bet tartományba benne van-e a lap, és ha igen akkor All In megyünk, ha pedig nincs, benne akkor a lapunkat eldobjuk.

**flopAction(Board, Rectangle) metódus:**

A metódus először a preflop akciók alapján minden játékoshoz hozzárendel egy laptartományt. Az után kiszámítja a nyerési esélyeinket és a pot odds-unkat. Ezen kívül még a kezünk erejét. Majd megállapítja ki volt a preflop emelő.

Abban az esetben, ha mi voltunk a preflop emelő és még nem cselekedtünk, megnézzük, hogy volt-e már előttünk emelés. Ha nem volt még, akkor, ha gyenge párunk van passzolunk. Ha nem gyenge párunk van, akkor megnézzük mennyire magas a nyerési esélyünk, ha elég magas, akkor további tétet rakunk be, ha nem akkor pedig passzolunk.

Ha mi voltunk a preflop emelő, de már történt előttünk emelés, akkor megnézzük, hogy van-e elég nyerési esélyünk ahhoz, hogy visszaemeljünk. Ha van, akkor visszaemelünk, ha nincs, elég akkor pedig megnézzük, hogy a megadáshoz elég nyerési eséllyel rendelkezünk-e. Ha igen, akkor megadunk, ha nem akkor pedig eldobjuk a lapjainkat.

Ha mi voltunk a preflop emelő és már cselekedtünk a körben, de valaki ránk emelt, akkor megnézzük, hogy van-e elég nyerési esélyünk visszaemelni. Ha van, akkor visszaemelünk, ha



pedig nincs, akkor megnézzük, hogy a megadáshoz van-e elég, ha igen, akkor megadjuk az emelést, ha nincs, akkor bedobjuk a lapjainkat.

Ha nem mi voltunk a preflop emelő és előttünk még senki nem emelt, akkor megnézzük, hogy van-e elég nyerési esélyünk a donkbet-re. Ez a pókeres kifejezés azt jelenti, hogy az előző kör kezdeményezője előtt emelünk. Ha van elég esély, akkor az esetek 40%-ban fogunk emelni, minden más esetben pedig passzolunk.

Ha nem mi voltunk a preflop emelő és előttünk már emeltek, akkor megnézzük, hogy van-e elég nyerési esélyünk egy visszaemeléshez, ha igen, akkor visszaemelünk. Ha nincs elég esélyünk, akkor megnézzük, hogy a megadáshoz elegendő-e, ha igen akkor megadunk máskülönben pedig dobunk.

Az emeléseink mérete mindig az asztal típusától függ, erre külön most nem tértem ki.

#### **turnAction(Board, Rectangle) metódus:**

A metódus először ellenőrzi, hogy a floppon történt-e emelés, ha igen akkor meghívja a játékban lévő ellenfeleinknél a **rangeTrim()** metódust. Ez lecsökkenti az ellenfeleink laptartományait, hogy a nagyon gyenge kezek kikerüljenek, így pontosabb a nyerési esély számítása.

Ezek után a továbbiakban majdnem megegyező a működés az előző metódussal, csupán más-más értékhatárok vannak, illetve a gyenge lap esetén passzolás itt nincs.

#### **riverAction(Board, Rectangle) metódus:**

A metódus megegyezik az előzővel, a **rangeTrim()** végrehajtása nélkül, valamint itt is az értékek mások, amik alapján a cselekvések történnek, de a logikai rész az azonos.

### 4.3 Rövid működési összefoglaló

Igyekeztem az osztályokon keresztül részletesen bemutatni a program működését, de itt még külön röviden összefoglalásra kerül a működési folyamat.

Indításkor a felhasználó számára megjelenik a grafikus felület, valamint ha nem létezik, akkor létrehozásra kerül az adatbázis és a szükséges tábla. Itt a gomb vagy az F5 billentyű lenyomásával lehetősége van a bot elindítására, de ezt csak, ahogy a felület is felhívja rá a figyelmet, csak preflop játékfázisban lehet elindítani.

Indítás után megpróbálja megkeresni a PokerGenius ablakát, ha ez sikertelen, akkor hibaüzenettel jelzi, ha sikeresen megtalálta, akkor elkezd figyelni mikor jelennek meg a gombok, amik azt jelzik, hogy rajtunk van a sor az adott körben.

Mikor ezek megjelennek, akkor a leolvasó elkezd minden szükséges helyről képkivágásokat készíteni, majd miután ez megtörtént, az OCR segítségével a képekről felismerésre kerülnek az adatok. Minden a képernyőről szerzett adat betöltésre kerül a megfelelő osztályokba.

Ezt követően meghívásra kerül a **GameLogic** osztály **start()** metódusa. A metódus megállapítja a szükséges végrehajtandó műveletet, amit automatikus a program el is végez az asztalon. Innentől újra visszakerülünk arra az állapotra, mikor a játékos gombjainak megjelenésére várunk. Ez egészen addig folytatódik, még a felhasználó le nem állítja a futást a gomb segítségével vagy az ablak bezárásával.

## 4.4 Tesztelés

A fejlesztési folyamat során igyekeztem részenként tesztelni a program működését. Minden új metódus létrehozása után, azokat külön-külön leteszteltem.

Az első összetettebb tesztelési folyamat az **equity** kalkulátor esetében volt. Ott az általam megadott adatok alapján számításokat végzett a nyerési esélyre, majd ezeket összevettem a **PokerStrategy Equilab** program számításaival.

A leolvasást végző osztályon is nagyon sok változtatást kellett végezni a tesztelések után, hogy számomra megfelelő eredményeket adjon vissza kellő gyakorisággal. Ezért is kellett létrehozni azt a metódust, ami felnagyítja a képeket.

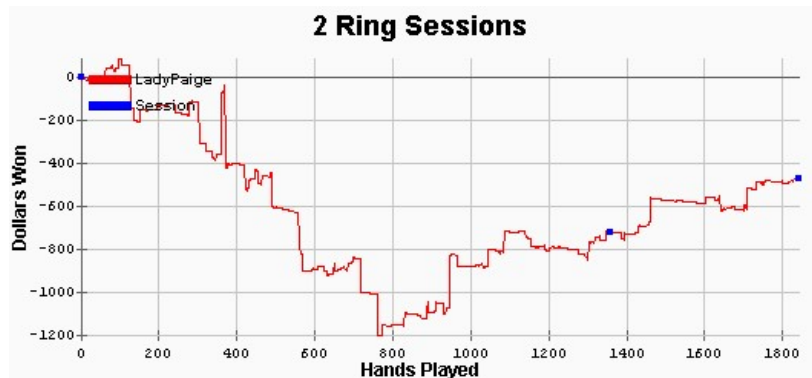
Miután a program teljesen elkészült, a PokerGenius használatával végeztem hosszabb tesztet. Ehhez a PokerGenius által felkínált botok közül, az úgynevezett Xenbot-okat választottam ellenfélnek. Ezek nagyon jó játékot játszó botok, annak idején No Limit bajnoknak tartották őket, mert gyakorlatilag a legjobbnak számító botok voltak, amelyek futtathatók voltak sima PC-ken is.

Igyekeztem ezek a botok közül úgy választani, hogy sok különféle játékstílust játszó ellenfél legyen, az alábbiakban bemutatom a választott ellenfeleket:

- **Al Inne:** Standard No limit stratégiát játszik Darse Billings könyve alapján
- **Mazama:** Agresszív, sokat blöffölő játékos, laza játékstílus.
- **Xengreanu:** Nagyon agresszív játékos, sokat játszik kapcsolódó lapokkal
- **Zeno:** Laza agresszív stratégia
- **Phil Tille:** Agresszív, standard stratégia
- **Olea:** Fesztes játékos
- **Max Bette:** Laza preflop játékos
- **Gus Xensen:** Nagyon agresszív, laza játékos

#### 4.4.1 Eredmények

Körülbelül 2000 leosztás került lejátszásra, amelyen a következő eredmények születtek:



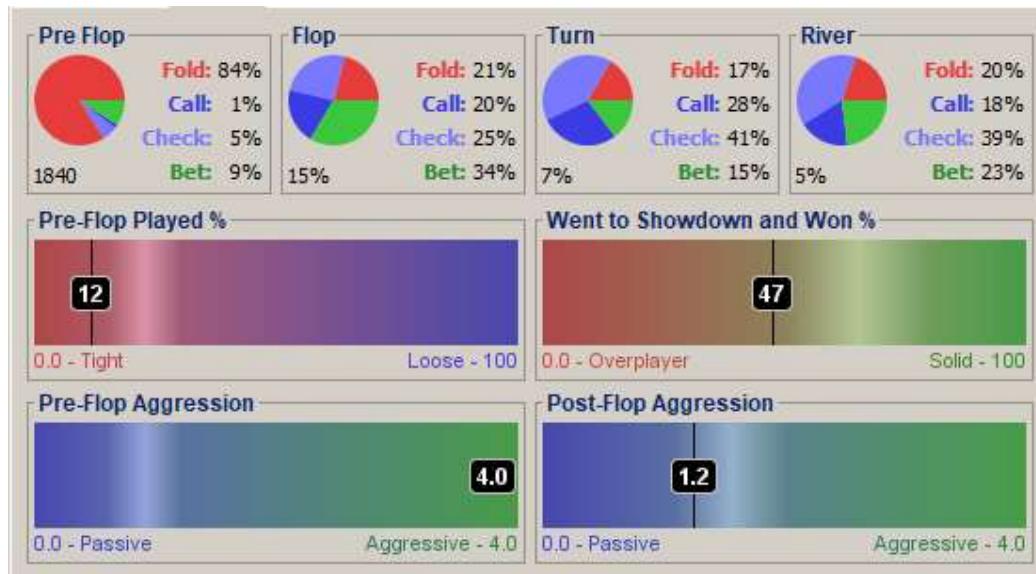
38. ábra: Profitot mutató grafikon

Összességében látható, hogy az általam írt program veszteségesen játszott a Xenbotok ellen ebben a 2000 leosztásban. Azonban a tesztelés folyamán folyamatosan javítottam a logikáján, az pedig látható, hogy az utolsó 1000 leosztást nézve nyereséges volt.

Player	Dollars Won	Hands	sb / Hand	Played %	Win %	Showdown %	Uncontested
Phil Tilt	\$1485.00	1676	+0.89	23.4%	9.3%	2.5%	7.9%
Olea	\$1512.25	1976	+0.77	23.6%	11.5%	2.3%	10.4%
Al Inne	\$228.50	1901	+0.12	25.4%	9.6%	2.5%	8.3%
Mazama	-\$96.00	1946	-0.05	25.8%	10.8%	2.7%	9.6%
Gus Xensen	-\$239.75	1882	-0.13	57.7%	21.3%	6.3%	18.1%
<b>LadyPaige</b>	<b>-\$471.75</b>	<b>1840</b>	<b>-0.26</b>	<b>11.7%</b>	<b>7.4%</b>	<b>3.2%</b>	<b>6.0%</b>
Xegreanu	-\$640.00	1898	-0.34	46.6%	16.5%	4.2%	14.5%
Zeno	-\$747.50	1594	-0.47	28.1%	11.7%	3.0%	10.5%
Max Bette	-\$1030.75	1454	-0.71	34.3%	11.4%	3.0%	10.0%

39. ábra: Különböző botok eredményei

A fenti ábrán látható, hogy a veszteséges játék ellenére is jobban teljesített 3 Xenbotnál, -0.26 kisvak/leosztás volt az eredménye, amellyel én maximálisan elégedett vagyok.



40. ábra: Játék kiértékelését mutató grafikonok

A fenti ábrán látható az összesített értékelés a saját botom játékaról. Amit ezek alapján én gondolok, hogy preflop lehetne egy kicsit nagyobb a megadó tartománya, mert nagyon kevés megadás történt a jelenlegi beállítások mellett. Postflop lehetne kissé agresszívebb, de időközben a teszt folyamán a flop agressziót jelentősen növeltem, így kicsit tovább kellene hozzá tesztelni, hogy látható legyen az milyen értékre áll be.

Amit sikerült megállapítanom, hogy a **rangeTrimForTurn()** metódushoz hasonlóan, kellene egy hasonló arra az esetre, ha az ellenfelek visszaemelnék, mert olyankor szükség lenne a laptartományuk jelentős szűkítésére, mert elég kevés lap esetén szokás visszaemelni, így pontosabbak lennének ott a számítások, a jelenlegi állapotban olyankor sok hibás döntést hozott.

## 5 FELHASZNÁLÓ DOKUMENTÁCIÓ

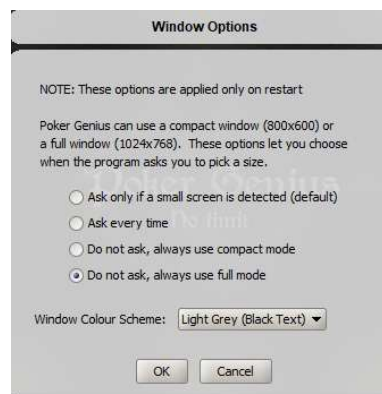
Egy másik számítógépen való tesztelés során kiderült, hogy a PokerGenius ablaka, az 1024x768-as felbontás beállítások ellenére 1280 pixel széles is lehet valami hiba folytán, ilyenkor a program nem képes működni a jelenlegi állapotban.

### 5.1 Rendszerkövetelmények

- Processzor: Intel Core i3 CPU 540 @ 3.07 Gzh x 4
- RAM: 4 GB
- Legalább 1280x720 pixel felbontású kijelző
- Operációs rendszer: Windows 7 64 bit, vagy újabb

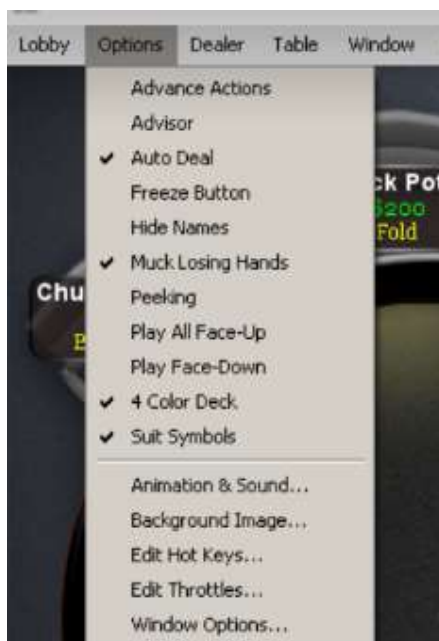
### 5.2 PokerGenius beállítása

- A install.zip fájl kicsomagolása után, a PokerGenius.exe segítségével az utasításokat követve kell telepíteni, **3 napig ingyenesen használható**.
- Telepítést követően a következő beállításokat kell elvégezni:
  - Options/Window Options fülön belül a következőket kell beállítani:



41. ábra: PokerGenius Window fül beállítása

- Options fül alatt a következőket kell kipipálni:



**42. ábra: PokerGenius options fül beállítása**

- Options/Background image fülnél a következő beállítások kellenek:



**43. ábra: PokerGenius Options/Background image beállítása**

- Ez után a Lobby/Ring games alatt a Full Ring Xenbots kiválasztásával elindítható az asztal
- Ezt követően, a jobb sarokban lévő játékost törölni kell, mert a 9 fős online póker asztalokat szimuláljuk, így annak nincs szerepe



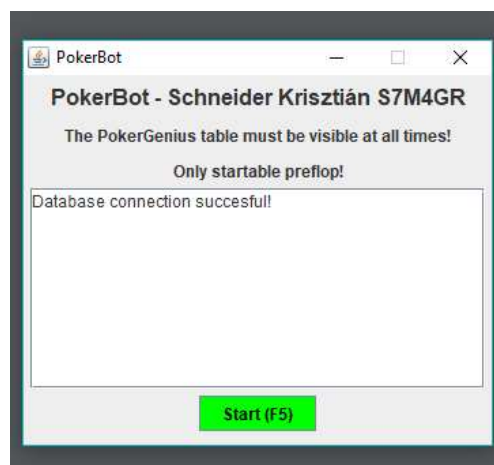
**44. ábra: PokerGenius kitörölendő játékos helye**

- Végezetül a Deal hand gomb megnyomásával elindul a játék

### 5.3 Az alkalmazás elindítása

Az install.zip kicsomagolása után, a PokerBot mappát egy olyan helyre kell másolni, ahol rendelkezik írási jogosultsággal. A mappában található PokerBot.jar fájl segítségével indítható el az alkalmazás.

Ha a PokerGenius ablaka is teljesen látható a megfelelő beállítások mellett, valamint preflop játékhelyzet van és vannak lapjaink, akkor elindítható a bot a Start gomb megnyomásával, vagy az F5 lenyomásával.



**45. ábra: PokerBot gui**



## 6 ÖSSZEFOGLALÁS

A dolgozat során igyekeztem az olvasó számára betekintést nyújtani, hogy miért érdekes terület a póker, mint játék, a mesterséges intelligencia szempontjából. Ismertettem pár híresebb MI projektet a területen, valamint a különféle megvalósítási módszereket.

A saját magam által fejlesztett alkalmazással összességében elégedett voltam. Jól helyt tudott állni a magas szinten játszó botok ellen is, amatőr játékosokat biztos, hogy könnyedén képes lenne legyőzni, így teljesítette az elvárásaimat. Még további leosztások játszásával, majd azok alaposabb kielemezése után, még lennének lehetőségek tovább javítani a bizonyos részeken, így jobb döntéseket lenne képes hozni.

A leolvasást végző részével nem vagyok teljesen megelégedve, sok esetben pontatlan eredményt ad, illetve a PokerGenius választása nem volt a legjobb, mint tesztkörnyezet, ugyanis nagyon rosszul írt alkalmazás, ezért egy hibája miatt máshogy is jelent meg egy másik tesztgépen, mint a sajátomon, így ott az alkalmazásom nem is volt képes működni.

### 6.1 További fejlesztési lehetőségek

A játék logikai részén is sokat lehetne még fejleszteni, az egyik ilyen, ami már említésre is került a dolgozatban, hogy különböző helyzetekben is szűkítse az ellenfelek laptartományait különféle módszerek segítségével. Fejlesztési lehetőség lehetne még az, hogy ő maga állítson fel adatok gyűjtésével játékos laptartományokat, az előre megadott módszer helyett, de ehhez sok adatra van szükség egy adott játékosról.

A grafikus adatok leolvasását végző rendszerre mindenképp kellene egy jobb megoldást találni, esetleg ilyen lehetne például más képfelismerő könyvtár alkalmazása. Érdekes lenne pár online pókerterem asztalaihoz is kifejleszteni az adatok leolvasását, így több helyen is használható lenne, egy listából kiválasztva, hogy épp melyik oldalon kívánjuk használni.

## 7 FELHASZNÁLT IRODALOM

- [1] Poker player research  
<http://pokerplayersresearch.com/>
- [2] Computer poker player  
[https://en.wikipedia.org/wiki/Computer\\_poker\\_player](https://en.wikipedia.org/wiki/Computer_poker_player)
- [3] Libratus  
<https://thegradient.pub/libratus-poker/>
- [4] Claudico Poker bot  
<https://en.wikipedia.org/wiki/Claudico>
- [5] Computer Poker Research Group,  
<http://webdocs.cs.ualberta.ca/~games/poker/>
- [6] Ceepheus Poker bot  
[https://en.wikipedia.org/wiki/Cepheus\\_\(poker\\_bot\)](https://en.wikipedia.org/wiki/Cepheus_(poker_bot))
- [7] DeepStack AI  
<https://www.deepstack.ai/>
- [8] Tóth Bálint: Sztochasztikus folyamatok e-jegyzet  
[https://web.archive.org/web/20100614193133/http://www.math.bme.hu/~balint/oktatas/sztochasztikus\\_folyamatok/jegyzet/](https://web.archive.org/web/20100614193133/http://www.math.bme.hu/~balint/oktatas/sztochasztikus_folyamatok/jegyzet/)
- [9] Darse Billings: Computer Poker MSc Thesis  
<https://poker.cs.ualberta.ca/publications/billings.msc.pdf>
- [10] Kurzweil, Ray, The Singularity Is Near: When Humans Transcend Biology.  
(Penguin Books, 2006).
- [11] "PokerTracker - Online Poker Tracking & Analysis Software Tool ,<http://www.pokertracker.com/>
- [12] D. Sklansky, The Theory of Poker: A Professional Poker Player Teaches You Howto Think Like One, Two Plus Two, 2002.
- [13] Pokersource Poker-Eval  
<http://pokersource.sourceforge.net>
- [14] Foldem Hand Evaluator  
<http://code.google.com/p/foldem/>,
- [15] Poker Stove – Poker Odds Calculator, <http://www.pokerstove.com/pokerstove/features.php>,
- [16] Meerkat API:  
<http://www.poker-academy.com/>
- [17] Pokersnowie  
<https://www.pokersnowie.com/>
- [18] Jonathan Rubin, Ian Watson: Computer Poker: Review  
<http://www.jonathan-rubin.com/files/CPReviewPreprintAIJ.pdf>
- [19] Monte-Carlo módszer  
[http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications\\_files/sim-balance.pdf](http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications_files/sim-balance.pdf)

- [20] A Nash Equilibrium in No-Limit Texas Hold'em  
<http://expertvoices.nsdl.org/cornell-info204/2008/02/29/a-nash-equilibrium-in-no-limit-texas-hold%E2%80%99em/>
- [21] L.F. Teófilo Adapting Strategies to Opponent Models in Incomplete Information Games: A Reinforcement Learning Approach for Poker”, in Autonomous and Intelligent Systems - Third International Conference pp 220–227, 2012
- [22] Horváth G. (szerk.), Altrichter M., Horváth G., Pataki B., Strausz Gy., Takács G., Valyon J.: Neurális Hálózatok, Panem Kiadó 2006, Fejezet 5.5.2
- [23] Case-based Reasoning  
[https://en.wikipedia.org/wiki/Case-based\\_reasoning](https://en.wikipedia.org/wiki/Case-based_reasoning)
- [24] Varjasi Norbert: Programozás III.  
[http://www.sze.hu/~varjasin/publikaciok/Programozas\\_III.pdf](http://www.sze.hu/~varjasin/publikaciok/Programozas_III.pdf)
- [25] Blahota István: SQLite alapok  
[http://zeus.nyf.hu/~blahota/sqlite/SQLite\\_13\\_06\\_03.pdf](http://zeus.nyf.hu/~blahota/sqlite/SQLite_13_06_03.pdf)
- [26] Tesseract  
<https://github.com/tesseract-ocr/tesseract>
- [27] JDBC  
[https://hu.wikipedia.org/wiki/Java\\_Database\\_Connectivity](https://hu.wikipedia.org/wiki/Java_Database_Connectivity)
- [28] PokerGenius  
<https://www.poker-genius.com/>
- [29] PokerStrategy Equilab  
<https://hu.pokerstrategy.com/poker-software-tools/equilab-holdem/>
- [30] Bill Chen: Mathematics of Poker, Conjelco 2006

## 8 Melléklet

[1] `install.zip`

[2] `source_code.zip`