

# Four in a row

## Specifikáció

Másik nevén *Connect 4*, magyarul *Négy a nyerő*

### A program célja:

A feladat egy olyan program készítése mely „leszimulálja” a jól ismert táblás társasjátékot. A program futása során a felhasználó a játékot egy másik emberrel, vagy egy általa kiválasztott nehézségi fokozatnak megfelelő robottal játszik. A program addig fut ameddig a játékos ki nem lép. Eredményét rögzítheti a dicsőséglistában.

### Játéktér:

A játék egy  $7 \times 4$ -s függőleges kék táblán játszódik.

### Játékmenet:

A játékosok felülről csúsztatják bele a táblába a saját színeiket melyek piros és sárga korongok, ilyen szempontból a jelrakások lehetőségei korlátozottak.

Az a játékos nyer, amelyik függőlegesen, vízszintesen, vagy átlósan kirakott négyet a saját színéből. Ha senki sem képes erre, akkor döntetlennel zárul a mérkőzés.

### Játékszabályok:

- Minden oszlopban csak a legalsó helyre lehet korongot tenni.
- Teli oszlopba nem lehet tenni.
- Egy lépés megtételére nincs időkorlát.
- Egy meccs nyertesének eldöntésére a Best of 3 megoldást választom, az nyer aki először nyer 2 game-et.
- Mindig a piros szín kezd.
- Ki kezd pirosként és ki sárgaként?
  - Ember vs ember:
    - A felek egymással megegyezhetnek, ez alapján kiválaszthatják a színeiket.
    - Vagy kiválaszthatják véletlen a színüket, ekkor a program kisorsol egy-egy szint a játékosoknak.
  - Ember vs bot:
    - Az ember kiválaszthatja a színét.
    - Vagy itt is alkalmazhatja a random generált színválasztást.

Ezek a szabályok biztosítják, hogy a lehető leg fair-ebb legyen a játék.

### Nehézségi szintek:

1	Beginner	Véletlenszerű lépéseket tesz, így a játékos könnyebben győzhet.
2	Intermediate	Már több lépéssel előre gondolkodik, és megpróbálja meggátolni a játékost, ha az közel áll a győzelemhez.

A fenti nehézségi szinteket a bot elleni meccs előtt lehet kiválasztani.

## A program használata

A program indítása után, a main menu-ben 5 opciója lesz a felhasználónak

- 1) Play (Ember vs bot) – Ezt követően ki kell választani a nehézségi szintet.
  - a) Beginner
  - b) Intermediate

A nehézségi szint kiválasztása után, ki kell választani, hogyan szeretnék választani a színt

- a) Random color
- b) Pick color
  - i) Red
  - ii) Yellow

Ezután kezdetét veszi a játék.

- 2) Two Players (Ember vs ember)
  - a) Random color
  - b) Pick color (Ilyenkor az 1. játékos választ, a másikat megkapja a 2. játékos)
    - i) Red
    - ii) Yellow

Ezután kezdetét veszi a játék.

- 3) Load
  - a) Itt megjelennek az elmentett játékállások. Ezek az adatok txt szövegfájlokba lettek elmentve. A felhasználó kiválaszthatja, hogy melyiket tölti be.
- 4) Leaderboard – Itt meg lehet tekinteni a dicsőséglistát.
- 5) Quit - Programból való kilépés

## A játék vége

A játék 3 féleképp érhet véget:

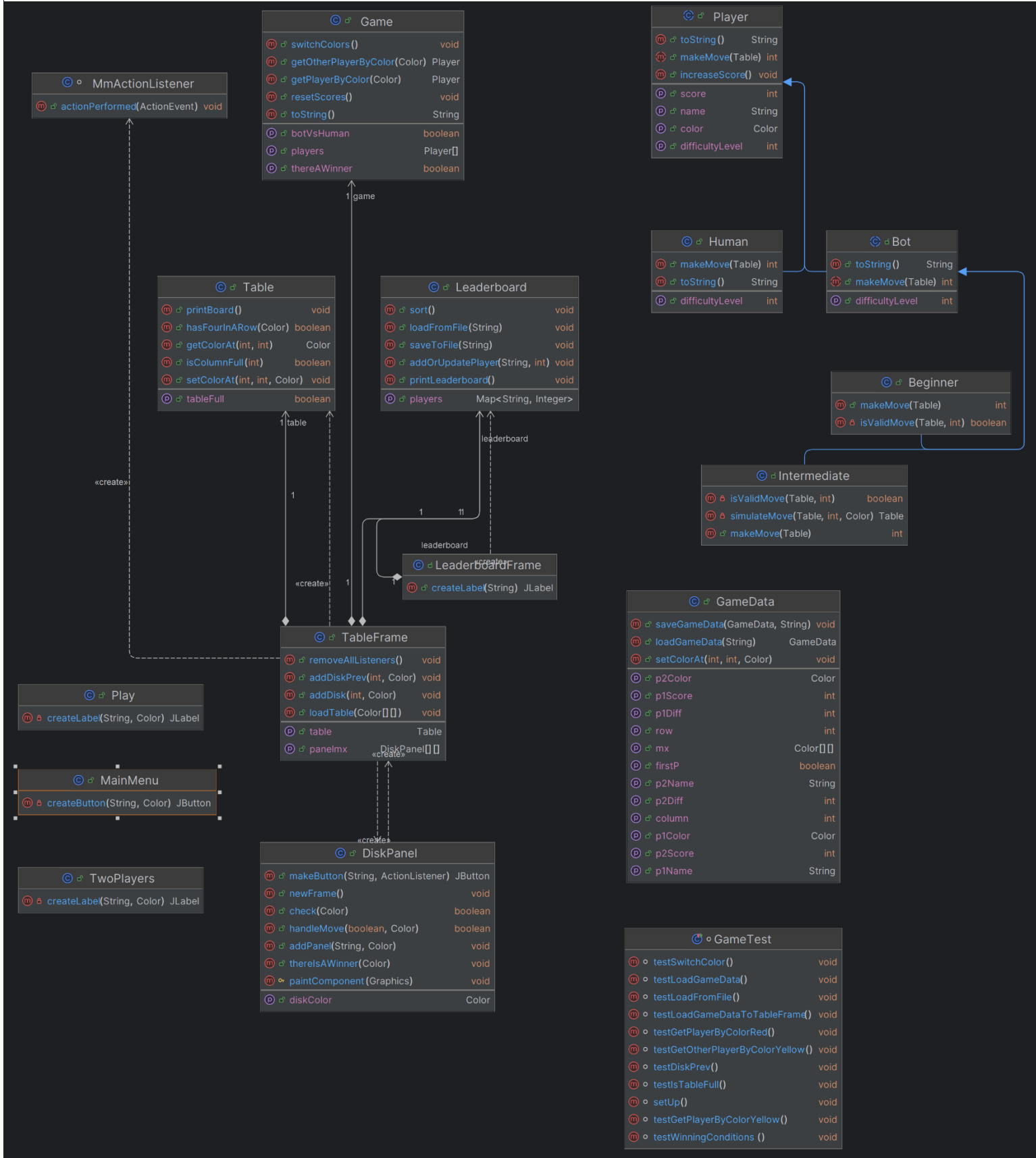
- 1) Az egyik játékos nyer
  - a) megnyer 2 game-etEzután az alábbi opciók jönnek elő:
  - a) Quit to main menu – Kilép a fő menübe.
  - b) Rematch – Újra kezdődik a játék
  - c) Save to leaderboard
- 2) Döntetlen
  - a) Tele lesz a tábla
- 3) Az egyik játékos kilép
  - a) Ez előtt van lehetőség elmenteni a játékot

Programnak úgy van vége, ha a felhasználó a main menu-ből is kilép.

### Megoldáshoz használni tervezett technológiák:

- **Dat** fájlok: A játékállások és a dicsőséglista egyszerű mentésére és betöltésére fogom használni.
- Java szerializáció (**Serializable**): Az osztályok objektumait bináris formátumban fogom elmenteni és vissza olvasni, egyszerűsítve a játékállás kezelését.
- **Swing GUI**: A felhasználói felület megjelenítéséhez és interakcióhoz Swing komponenseket és Graphics osztályt fogok használni.
- Gyűjtemények keretrendszer (**Collections**): Sok jól használható kollekciót foglal magában, amik közül használni fogok párat, például a dicsőséglista tárolásához és rendezéséhez LinkedHashMap-et tervezek használni.
- **JUnit** tesztelési keretrendszer: Az osztályok és metódusok teszteléséhez ezt fogom alkalmazni.

1. ábra Osztálydiagram



## Components

### Table osztály

- Az mx attribútuma tárolja a játékasztalon lévő színeket. Alapértelmezetten fehér mindegyik cella, tehát üres.
- A row, column attribútumai eltárolják a méreteit.

Metódusok (getter, setter, printer nélkül):

hasFourInARow(Color player):

- Ellenőrzi, hogy az adott player színű játékosnak van-e négy egymás melletti korongja bármelyik sorban, oszlopban vagy átlón.
- **Paraméterek:** player – a szín, amelyet ellenőrizni szeretnénk.
- **Visszatérési érték:** boolean – true, ha a játékosnak van négy egy sorban, különben false.

isColumnFull(int col):

- Ellenőrzi, hogy egy adott oszlop tele van-e korongokkal. Az oszlopot akkor tekinthetjük telinek, ha az első sorában már van korong.
- **Paraméterek:** col – az ellenőrizni kívánt oszlop indexe.
- **Visszatérési érték:** boolean – true, ha az oszlop tele van, különben false.

isTableFull():

- Ellenőrzi, hogy az egész asztal tele van-e, azaz minden oszlop tetején van már korong.
- **Visszatérési érték:** boolean – true, ha az asztal tele van, különben false.

## Game

Játékban szereplő objektumok.

### Player

- Absztrakt osztály ebből származnak le a példányosítható játékosok pl.: Human
- Eltároljuk a játékos nevét, színét, eredményét.
- Metódusai a getter, setter, printereken kívül mind absztraktak.

### Bot

- Továbbra is absztrakt osztály, de már van difficulty-a.
- ebből származik le a példányosítható Beginner és Intermediate.

### Human

- Már példányosítható osztály.

Metódusok:

getDifficultyLevel():

- Az emberi játékosok nem rendelkeznek nehézségi szinttel, ezért ez a metódus mindig 0 értéket ad vissza. Ez egyfajta típuslekérdezést valósít meg.

makeMove(Table table):

- Ez a metódus nem használatos az emberi játékosok esetében, mivel az ő lépéseiket nem egy algoritmus vezérli. Az osztályban üresen van implementálva.

## Beginner

- Már példányosítható Bot.
- A nehézsége mindig egy, bár ezt itt nem korlátozom, mindig 1 nehézséggel példányosítom.

Metódusok:

**makeMove(Table table):**

- Ez a metódus meghatározza a bot következő lépését. A bot véletlenszerűen választ egy oszlopot, majd ellenőrzi, hogy az oszlop érvényes-e (van-e hely a felső sorban).
- Ha a választott oszlopban van hely, a bot azt választja; ha nincs, újra próbálkozik egy másik oszloppal.
- **Visszatérési érték:** int – A kiválasztott oszlop indexe.

**isValidMove(Table table, int col):**

- Segédmetódus, amely ellenőrzi, hogy az adott oszlopban van-e még hely, azaz, hogy a legfelső sorban nem foglalt-e már a hely.
- **Visszatérési érték:** boolean – Igaz, ha az oszlopban még van hely.

## Intermediate

- Már példányosítható Bot.
- A nehézsége mindig 4, bár ezt itt nem korlátozom, mindig 4 nehézséggel példányosítom.

Metódusok:

**makeMove(Table table):**

- Ez a metódus végzi el a bot lépését. Prioritás:
  1. **Nyerő lépés:** A bot végigellenőrzi az oszlopokat, hogy talál-e nyerő lépést. Ha igen, azt választja.
  2. **Blokkolás:** Ha az ellenfél képes nyerni a következő lépésével, akkor azt a lépést választja, hogy blokkolja az ellenfél nyerését.
  3. **Véletlenszerű lépés:** Ha nem talált sem nyerő, sem blokkoló lépést, akkor egy véletlenszerű lépést választ, amit a Beginner osztály makeMove metódusával hajt végre.
- **Visszatérési érték:** int – A kiválasztott oszlop indexe, amelybe a bot leteszi a korongot.

**simulateMove(Table table, int col, Color color):**

- Segédmetódus, amely lemásolja a jelenlegi táblát, és egy adott oszlopban szimulálja a lépést, anélkül, hogy ténylegesen módosítaná a játék aktuális állapotát.
- **Visszatérési érték:** Table – A táblát, amely a szimulált lépést tartalmazza.

## Leaderboard

- Játékosok elmentett adatait tárolja. Név és Score formájában.
- Választott adatszerkezet erre a LinkedHashMap. Biztosítja, hogy a játékosok sorrendje megmaradjon, ahogyan azok hozzáadódnak (beszúrás sorrendje), és hogy gyors hozzáférést biztosítson a nevekhez.

Metódusok:

#### **loadFromFile(String filename):**

Ez a metódus egy fájlból olvas be adatokat. Az `ObjectInputStream` segítségével tölti be az objektumokat (jelen esetben a `Map`-et), majd azt a `players` térképre másolja.

- Ha a fájl nem létezik vagy a beolvasás során hiba történik, akkor egy hibaüzenet jelenik meg.
- A fájl olvasása után a `players` `map` tartalmazza a játékosokat és azok pontjait.

#### **saveToFile(String filename):**

Ez a metódus a játékosok adatait egy fájlba menti el.

- Az `ObjectOutputStream` segítségével a `players` `map`-et írásra kerül a fájlba.
- Ha valami hiba történik a mentés közben, akkor egy hibaüzenet jelenik meg.

#### **addOrUpdatePlayer(String playerName, int score):**

Ez a metódus egy új játékost ad hozzá a leaderboard-hoz vagy frissíti egy meglévő játékos pontszámát. Ha a játékos már létezik, a pontszámot növeli a meglévő értékkel, ha nem, akkor hozzáadja a játékost az új pontszámmal.

- Miután a játékos hozzáadódott vagy frissült, a `sort()` metódus rendezni fogja a leaderboard-et, hogy mindig csökkenő sorrendben jelenjenek meg a játékosok a pontszámok alapján.

#### **sort():**

A `sort()` metódus egy rendezést végez a játékosok pontszáma szerint csökkenő sorrendben. Az `ArrayList`-be másolja a `map` elemeit, és a `List.sort()` metódust használja, amely a játékosok pontszámait (Integer értékeket) csökkenő sorrendbe rendezi. Miután a lista rendezve van, a `players` `map`-t újratöltődik a rendezett értékekkel.

### **GameData**

- A `GameData` osztály egy játék összes fontos adatát tárolja, amelyek szükségesek a játék későbbi betöltéséhez és folytatásához. Az osztály célja, hogy a játék különböző paramétereit (pl. játékosok nevei, pontszámai, színei, játékállás) egy fájlba mentse és onnan visszaolvassa.

Metódusok:

#### **saveGameData(GameData gameData, String filename):**

Ez a metódus a játék adatainak mentésére szolgál. Az `ObjectOutputStream` segítségével az `GameData` objektumot egy fájlba menti.

- Az objektumot a megadott fájlneve alapján írja ki a rendszer.
- Ha a mentés sikeres, akkor egy sikerüzenet jelenik meg. Ha hiba történik, akkor hibajelentés kerül a konzolra.

#### **c. loadGameData(String filename):**

Ez a metódus egy fájlból tölti be a játék állapotát. Az `ObjectInputStream` segítségével beolvassa a fájl tartalmát, és visszaadja a `GameData` objektumot.

- Ha sikerül a betöltés, akkor egy sikerüzenet jelenik meg, és a betöltött adatokat visszaadja.



- Ha valami hiba történik (pl. a fájl nem létezik vagy a formátum helytelen), akkor hibajelentés kerül a konzolra.

ui

## MainMenu

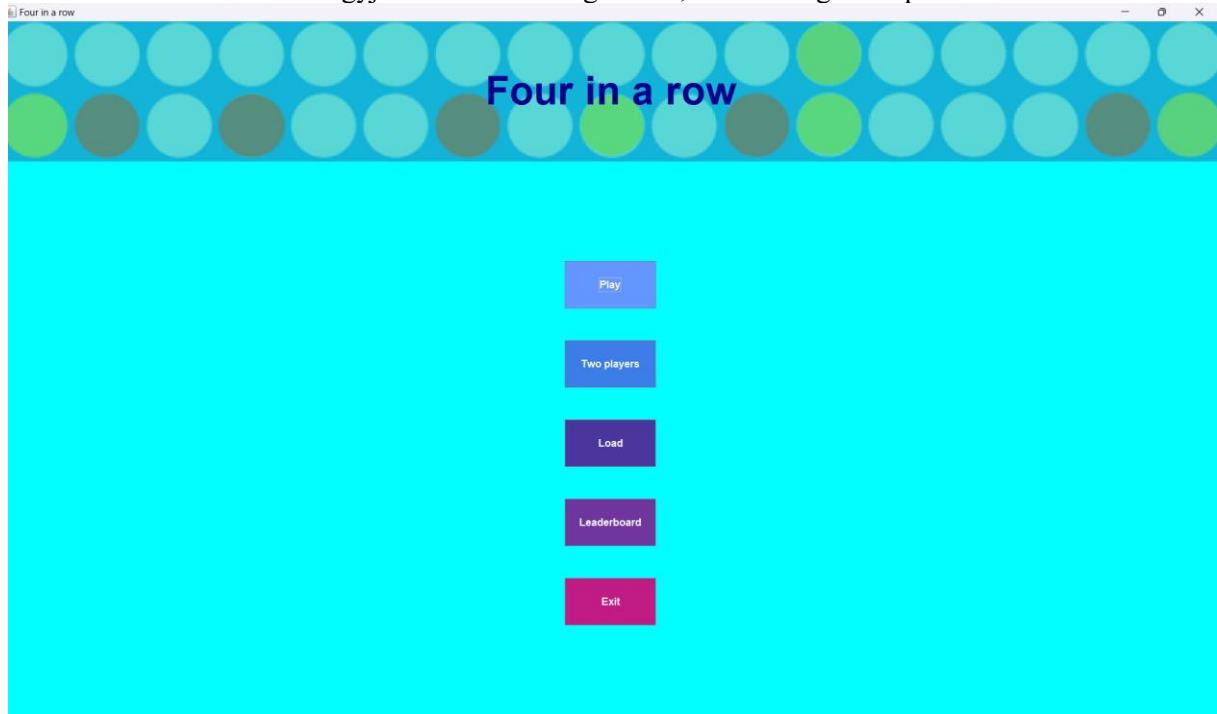
- Kezdőképernyő, innen indul a játék.

UI felépítő elemek:

- **JPanel:**
  - mainP: Középen helyezkedik el, rajta vannak a gombok,
- **JLabel:** A cím megjelenítésére szolgál, amely középen található. Háttérét egy képpel töltöttem ki.
- **JButton:** Az interaktív gombok, amelyek mindegyikéhez egy-egy ActionListener van rendelve.
  - **Play:** Elindítja a játékot, és elnavigál a játékelületre. - PActionListener
  - **Two players:** Kétjátékos módot indít el. - TPAActionListener
  - **Load:** Betölti a korábban mentett játékállást egy fájl kiválasztásával. - LoadActionListener
  - **Leaderboard:** Megjeleníti a játékosok eredményeit (leaderboard). - LActionListener
  - **Exit:** Bezárja az alkalmazást. – lambda kifejezés
- A gombok elhelyezését a GridBagConstraints-el valósítottam meg.

Segédmetódus:

- **createButton:** Mivel nagyjából hasonlóak a gombok, ezért ez segít a duplikáció ellen.





## Play

- A Play osztály a "Four in a row" játék egy képernyője, amely lehetővé teszi, hogy a felhasználó beállítsa a játék paramétereit, például a játékos nevét, a választott bot típusát és nehézségi szintjét, a játékos színét, valamint a játéktábla méretét. Az osztály a grafikus felhasználói felületen (GUI) keresztül interakciókat biztosít a felhasználó számára. Megfelelő akciókat végrehajtja a felhasználó inputokat ad. Ezek alapján jön létre a játék.

UI felépítő elemek:

- **JPanel**ek:
  - topPanel: Magában foglalja a topMenuBar-t és az mm gombot. Ezek különböző navigálási opciókat tesznek elérhetővé:
    - mm (< gomb): vissza megy a főmenübe
    - topMenuBar->menu (options)->menui(Main menu): vissza a menübe.
  - mainP: Középen helyezkedik el, JLabelokat, JTextFieldeket és Comboboxokat tartalmaz.
- **JLabel** jelzik, hogy milyen adatokat hova várunk.
  - playerName
  - chooseBot
    - Intermediate
    - Beginner
  - chooseColor
    - Yellow
    - Red
    - Random Color
  - tableSize
  - x – Csak esztétikai szerepe van
  - Play: megadott inputokkal elindítja a játékot.
    - **PBActionListener** osztály felelős az akció feldolgozásáért, beleértve a játék inicializálását és a hibaüzenet megjelenítését, ha szükséges.
- **JTextfield**: Lehetővé teszik a felhasználó számára, hogy beállítsa a játéktábla sorainak és oszlopainak számát.
  - player
  - row
  - column
- Az objektumok elhelyezését a GridBagConstraints-el valósítottam meg.

Segédmetódus:

- **createJLabel**: Mivel nagyjából hasonlóak a gombok, ezért ez segít a duplikáció ellen.

Four in a row

Options

Player name:

Choose bot:

Beginner

Choose Color:

Red

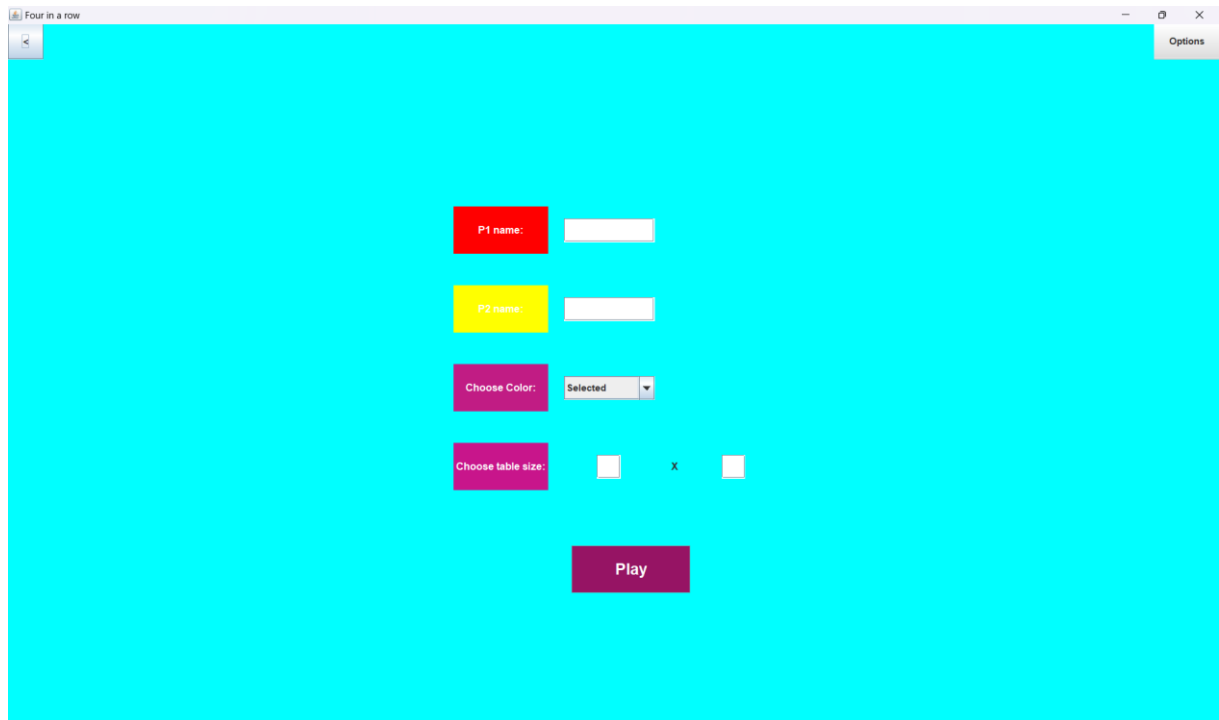
Choose table size:

X

Play

## Two players

Nagyon hasonló mint a Play osztály frame-je. Csak itt két játékos nevét kell megadni



The screenshot shows a web browser window with the title "Four in a row". The page has a light blue background. In the center, there is a form for setting up a game. The form consists of several colored boxes and input fields:

- A red box labeled "P1 name:" followed by a white text input field.
- A yellow box labeled "P2 name:" followed by a white text input field.
- A purple box labeled "Choose Color:" followed by a dropdown menu showing "Selected".
- A purple box labeled "Choose table size:" followed by two small white square input fields, an "X" symbol, and another small white square input field.
- A large purple button labeled "Play" at the bottom.

There is a small "Options" button in the top right corner of the page.

## TableFrame

A) játék grafikus felületét valósítja meg.

### Főbb jellemzők:

#### 1. UI elemek:

- Több panel (topPanel, centerPanel, bottomPanel) van használva a különböző UI elemek elrendezésére.
- Tartalmaz egy **vissza gombot** (<), amely lehetővé teszi a felhasználó számára, hogy visszatérjen az előző képernyőre.
- Az options-ből innen is el lehet jutni a Mian menu-be.
- Megjeleníti a játékosok nevét, színét és pontjait.
- A játéktáblát (táblát) egy **GridLayout** segítségével jeleníti meg, ahol minden cella egy-egy DiskPanel komponens.

#### 2. Játék inicializálása:

- A TableFrame konstruktor a megadott sorok és oszlopok számával, valamint a játék adatainak betöltésével inicializálja az ablakot.
- A játék betöltésénél az előző mentett állapotot a loadTable metódus segítségével tölti be.

#### 3. Korong elhelyezése:

- A játéktábla egy 2D rácsként van ábrázolva, ahol minden cella egy-egy DiskPanel objektumot tartalmaz, amely megjeleníti a színes korongokat.
- Az addDisk metódus felelős azért, hogy egy korongot elhelyezzen a kiválasztott oszlopban úgy, hogy az alulról felfelé keresi meg a legelső üres helyet.

#### 4. Játék logika és mentés:

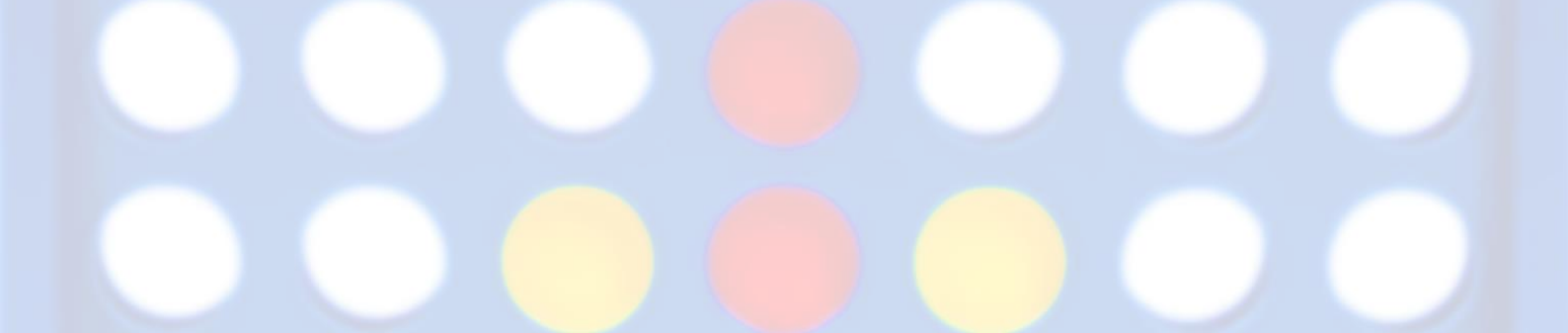
- A játék során a játékosok felváltva dobják le a korongjaikat, és minden lépés után ellenőrzi a rendszer, hogy van-e győztes vagy döntetlen.
- A **Mentés** funkció lehetővé teszi a játék aktuális állapotának mentését fájlba a GameData osztály segítségével.
- Az SActionListener a mentés gomb lenyomásakor figyeli az eseményt, és lehetőséget ad a felhasználónak a fájl mentési helyének kiválasztására.

#### 5. Felhasználói interakció kezelése:

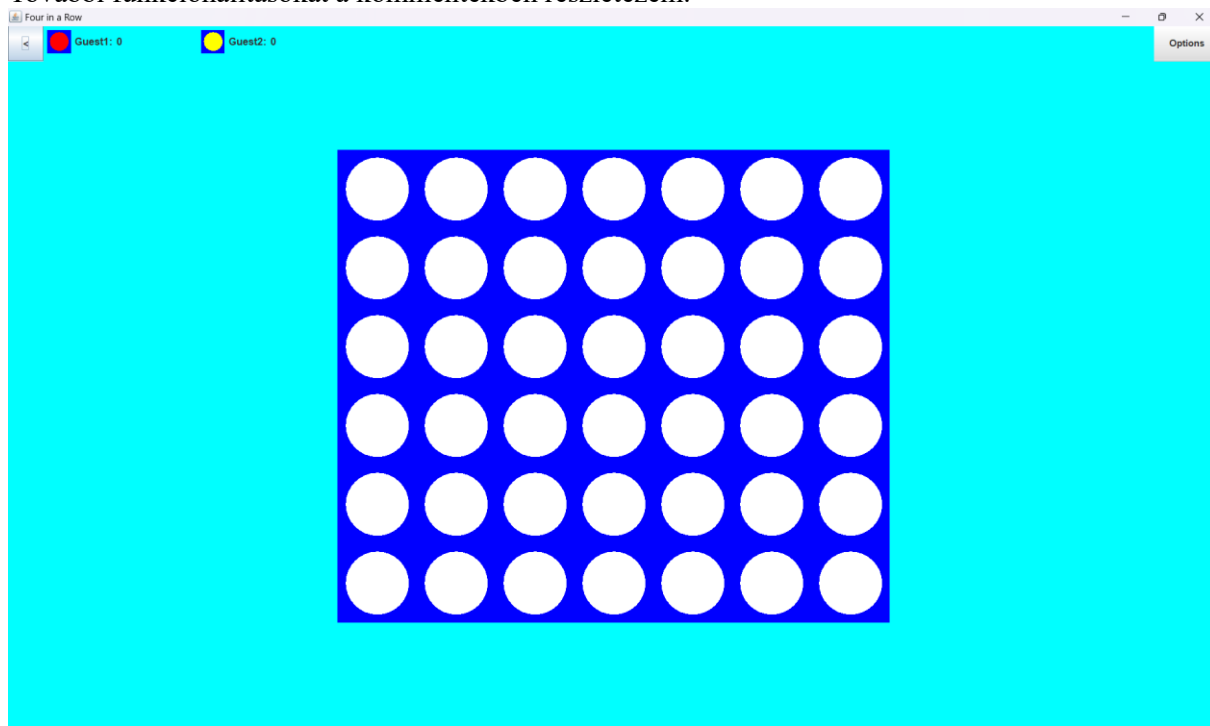
- Amikor egy győztes vagy döntetlen állapotot érnek, a táblát "lezárják", vagyis már nem lehet több lépést tenni. Ezt a removeAllListeners metódus biztosítja, amely eltávolítja az összes MouseListener-t a tábláról.

#### 6. Egyedi DiskPanel osztály:

- Az egyes táblasejteket a DiskPanel osztály reprezentálja, amely felelős a korongok színének megjelenítéséért és a felhasználói interakciók kezeléséért.



További funkcionálisokat a kommentekben részletezem.



## DiskPanel

A DiskPanel osztály játéktérnek egy-egy celláját reprezentálja, amely tartalmazhat egy korongot. Az osztály az egyes cellák interakcióit kezeli, beleértve a korongok elhelyezését, a színek váltását, és a játékosok lépéseit.

### Főbb funkciók:

#### 1. Játékcellák megjelenítése:

- Az osztály egy-egy cellát (oszlopot) reprezentál, ahol a korongok elhelyezhetők.
- A cella kezdetben üres (`diskColor = Color.WHITE`), majd a játék során a korongok színét (`Color.RED` vagy `Color.YELLOW`) tárolja.

#### 2. Játéklogika és eseménykezelés:

- Az osztály kezelni tudja az egér eseményeket, hogy a felhasználó meg tudja helyezni a korongot egy adott oszlopba.
- **mouseEntered**: Ha az egér belép egy cellába, akkor annak az oszlopnak a színét megjeleníti a `mouseEntered` eseménykezelő segítségével.
- **mouseExited**: Ha az egér elhagyja a cellát, akkor eltávolítja az előnézeti színt.
- **mousePressed**: Ez az esemény az oszlop kiválasztásakor szürke előnézeti színt ad a cellának.
- **mouseReleased**: Ha a felhasználó elengedi az egér gombját, akkor a korong a kiválasztott oszlop legalsó üres helyére kerül.

#### 3. Korong elhelyezése és győzelem ellenőrzése:

- **setDiskColor**: Beállítja a cella színét, amely azt jelenti, hogy ott egy korong van, és újrarajzolja a panelt.
- **getDiskColor**: Visszaadja a cellában lévő korong színét.
- **check**: Ellenőrzi, hogy van-e győzelem a kiválasztott színű koronggal. Ha van, akkor aktiválja a győztes értesítést, és növeli a pontszámot. Ha döntetlen van, akkor annak megfelelően jelenít meg üzenetet.
- A győztes állapot vagy döntetlen esetén új gombokat ad hozzá az alján, például "Rematch", "Save to leaderboard", és "Quit to main menu".

#### 4. Bot lépései:

- A bot lépéseit a **handleMove** metódus kezeli. Ha a játékosok felváltva játszanak és a bot következik, akkor a bot automatikusan végrehajtja a következő legjobb lépést.

#### 5. Új játék és rematch:

- Az **newFrame** metódus létrehozza az új játék felületet, és elindít egy új játékot, ha a felhasználó a "New Game" vagy "Rematch" gombra kattint.
- A **NGActionListener** és **RActionListener** osztályok felelősek a gombok eseménykezeléséért, amelyek új játékot indítanak vagy a rematch-et kezelik.

#### 6. Leaderboard kezelés:

- Az **LActionListener** lehetővé teszi a felhasználók számára, hogy elmentsék a győztes játékokat a leaderboard-ra, és megjelenítsék a rangsort.

#### Fontosabb metódusok:

- **addPanel(String text, Color actionC):** Hozzáad egy szöveges címkét (pl. "Győztes", "Döntetlen") a játék központi paneljéhez, amely megjeleníti az aktuális játék állapotát.
- **makeButton(String text, ActionListener ac):** Egy gombot hoz létre a megadott szöveggel és eseménykezelővel. Használják a "Rematch", "Leaderboard" és "Quit to main menu" gombok esetén.
- **thereIsAWinner(Color actionC):** Ez a metódus kezeli a győztes állapotot. Ha győztest találtak, akkor egy üzenetet ad a képernyőhöz, növeli a játékos pontszámát és hozzáadja a győztest a leaderboardhoz. A győzelem után új gombokat is megjelenít.
- **check(Color actionC):** Ellenőrzi, hogy van-e győztes a megadott színű koronggal. Ha van győzelem vagy döntetlen, akkor frissíti a játék állapotát és eltávolítja az eseménykezelőket, így már nem lehet több lépést tenni.
- **handleMove(boolean hasFourInARow, Color actionC):** A bot lépéseit kezeli, ha a játékos nem nyert, akkor a bot végrehajtja a következő legjobb lépést.
- **paintComponent(Graphics g):** A panel kinézetét kezeli, először kitölti kék színnel a hátteret, majd ha van korong, akkor a megfelelő színű kört rajzol a panel közepére



## LeaderboardFrame

A **Leaderboard** ablakát implementálja. A Main menu-ből lehet elérni. A felhasználók itt láthatják a legjobb játékosokat és azok pontszámait, amelyeket a játék során értek el. Az alkalmazás Java Swing keretrendszert használ a grafikus felhasználói felület (UI) kialakításához.

### Főbb funkciók:

#### 1. Leaderboard (Vezetői táblázat) megjelenítése:

- Az alkalmazás a **Leaderboard** osztály használatával betölti a vezetői táblázatot, amely tartalmazza a játékosok nevét és azok pontszámát.
- A táblázat tartalma egy dinamikusan generált lista, amely az adatokat a leaderboard.dat fájlból olvassa be.

#### 2. Felhasználói felület (UI):

- Az ablak maximalizálva van, és a különböző komponensek jól elrendezett módon jelennek meg.
- A **Cím panel** tartalmazza a játék neve "Leaderboard" szöveget, valamint egy vissza gombot, amely visszavezeti a felhasználót a főmenübe.
- A **táblázat panel** az adatokat egy **GridBagLayout** elrendezésben jeleníti meg.

#### 3. Interakció:

- A **Vissza gomb** segítségével a felhasználó visszatérhet a főmenübe. A gombhoz egy MmActionListener van rendelve, amely egy eseménykezelőt biztosít.

### Főbb komponensek:

#### 1. Leaderboard betöltése és megjelenítése:

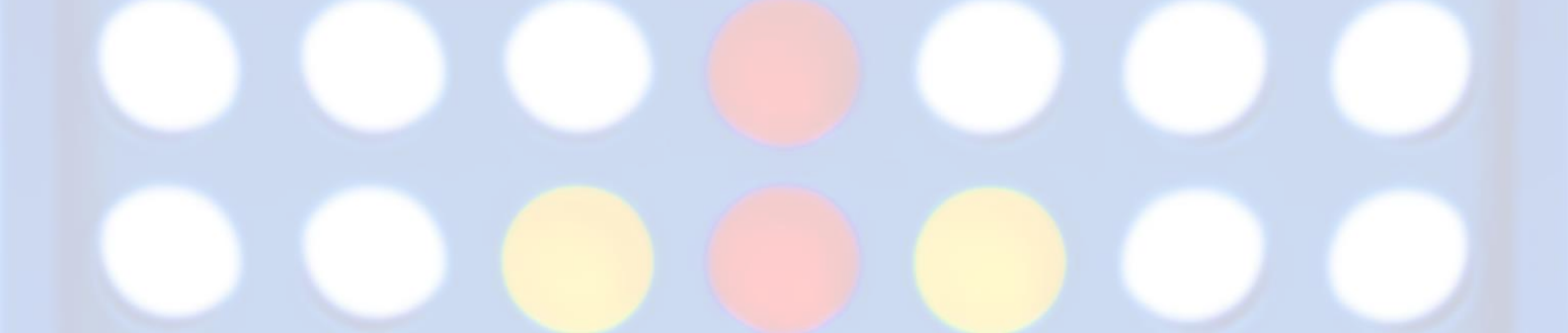
- A **Leaderboard** osztály felelős a játékosok és pontszámok tárolásáért. A leaderboard.loadFromFile("leaderboard.dat") függvény betölti a fájlból az adatokat.
- A getPlayer() metódus visszaadja a játékosok nevét és pontszámát egy Map<String, Integer> formátumban.

#### 2. Felhasználói felület:

- A **Cím panel** (topPanel) tartalmazza a **Leaderboard** feliratot és egy vissza gombot, amely visszavezet a főmenübe.
- A **Táblázat panel** (mainPanel) a **GridBagLayout** segítségével rendezi el a táblázatot, ahol minden játékos neve, helyezése és pontszáma megjelenik.

#### 3. Játékosok és pontok megjelenítése:

- A **LeaderboardFrame** osztály iterál a **leaderboard** objektum játékosain és azok pontszámán, és egy-egy **JLabel** elem segítségével megjeleníti őket a táblázatban.
- Minden játékos adatai (helyezés, név, pontszám) egy-egy új sorban jelennek meg, és a megfelelő helyeken szerepelnek a táblázatban.



Four in a Row

✕

✕

Leaderboard

Rank	Player name	Score
1	Guest	12
2	Soma	8
3	Csabi	3
4	Ábel	1
5	Guest1	0



test

## GameTest

- A tesztelést a JUnit keretrendszerrel valósítom meg, assert-ek segítségével.
- A tervezői döntéseket a kommentekben indoklom.

main

## App

- Létrehoz egy Main Menu-t és láthatóvá teszi, elindul a program.