



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

INFORMATIKATUDOMÁNYI INTÉZET

ALGORITMUSOK ÉS ALKALMAZÁSAIK TANSZÉK

# Automatizált felvevőjáték algoritmus a bridzs játékhoz

*Témavezető:*

Dr. Pusztai Kinga Emese  
egyetemi adjunktus, PhD

*Szerző:*

Biborka Ágnes  
programtervező informatikus BSc

*Budapest, 2025*

## SAKADOLGOZAT TÉMABEJELENTŐ

*Figyelem! Nyári záróvizsga esetén a módosítás határideje február 1., őszi záróvizsga esetén augusztus 31.*

### Hallgató adatai:

Név: Biborka Ágnes

Neptun kód: H4JD6B

### Képzési adatok:

Szak: programtervező informatikus, alapképzés (BA/BSc/BProf)

Tagozat : Nappali

Belső témavezetővel rendelkezem

Témavezető neve: Kovácsné dr. Pusztai Kinga Emese

munkahelyének neve, tanszéke: ELTE IK, Algoritmusok és Alkalmazásai Tanszék

munkahelyének címe: 1117, Budapest, Pázmány Péter sétány 1/C.

beosztás és iskolai végzettsége: egyetemi adjunktus, PhD

**A szakdolgozat címe:** Automatizált felvevőjáték algoritmus a bridzs játékhoz

### A szakdolgozat témája:

*(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását )*

A bridzs egy világszerte ismert kártyajáték és sport, amit hobbi szinten vagy versenyszerűen is lehet űzni. A szakdolgozat célja olyan webes alkalmazás létrehozása, ami egy automatizált felvevőjáték algoritmussal önálló ellenjáték gyakorlásra és szabadidős kikapcsolódásra is lehetőséget nyújt.

A bridzs négy játékos játssza, akik a négy égtájon ülnek, és az egymással szemben ülő játékosok kooperáló partnerek. A játék elején egy franciakártya pakli kerül egyenlően kiosztásra a játékosok között. Majd a játékosok információt osztanak meg egymással a lapjaikról a licit során és a licit végére az adott felvétel is meghatározásra kerül. A felvétel azt mondja meg, hogy melyik játékos lesz a felvevő és adott adu mellett hány ütést vállal az összesen megszerezhető tizenháromból. Ezután a lejátszás tizenhárom körből áll, egy körbe mindegyik játékos rak egy lapot a hívóval kezdve és az óramutató járásával megegyező irányba haladva. A kört az viszi, aki a legnagyobb értékű lapot vagy a legnagyobb értékű adut teszi, és ő lesz a következő körben a hívó. A felvevő célja, hogy minél többet üssön és teljesítse a felvételt, az ellenjátékosok célja pedig az, hogy minél többet üssenek és a felvevő ne teljesítse a felvételt.

A játék nagyon változatos, a játékosok különböző stratégiákat követnek egyes lap leosztásoknál és több módszert is bevethetnek, hogy az ütéseiket maximalizálják adott színhosszúság és lapkombináció esetén. A játékosok nem látják egymás lapjait, a felvevő partnerének, az asztalnak a lapjait leszámítva, ezért fontos a játék elején a licit, és ezért fontos a lehetséges lap elosztásokat számba venni és valószínűséget számolni.

A szakdolgozat üzleti logikájának fő komponensét egy játék algoritmus képezi. Az ellenjátékos hívásait a felhasználói interakció határozza meg, de a felvevő és az asztal hívásait egy remélhetőleg ideális hívást kiszámító algoritmus. Ez az algoritmus figyelembe veszi az aktuális adatok alapján az ellenfelek valószínű lap leosztásait és hogy ha azok fennállnak, akkor a felvevő milyen hívássorozata alapján hány pontot lehet várhatóan szerezni. Így határozza meg végül a legkedvezőbbnek tűnő hívást. A hatékonyság növelése érdekében a bridzs játékban alkalmazható heurisztikákat lehet az algoritmusba építeni.

Az alkalmazásnak rendelkeznie kell egy grafikus felülettel, ahol a felhasználó nyomon követheti a játék alakulását, és aktívan alakíthatja a játékot az ellenjátékosok hívásainak meghatározásával.

A webes alkalmazást Spring Boot keretrendszerben, elsődlegesen Java nyelven tervezem implementálni.

Budapest, 2025. 01. 14.

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>3</b>
<b>2. Felhasználói dokumentáció</b>	<b>5</b>
2.1. Az alkalmazás elindítása . . . . .	6
2.1.1. Előkészítés . . . . .	6
2.1.2. Első indítás . . . . .	6
2.1.3. Leállítás és későbbi használat . . . . .	7
2.2. Az alkalmazás használata . . . . .	7
2.2.1. Játékmódok . . . . .	7
2.2.2. Instant játék . . . . .	9
2.2.3. Új játék paraméterekkel . . . . .	11
2.2.4. Navigáció és felhasználó profil . . . . .	12
2.2.5. Súly . . . . .	15
<b>3. Fejlesztői dokumentáció</b>	<b>16</b>
3.1. Elméleti háttér . . . . .	16
3.1.1. Mi alapján játszik ki egy lapot egy emberi játékos . . . . .	16
3.2. Követelményelemzés . . . . .	17
3.2.1. Specifikáció . . . . .	17
3.2.2. Funkcionális követelmények . . . . .	17
3.2.3. Nem funkcionális követelmények . . . . .	18
3.2.4. Megszorítások . . . . .	19
3.2.5. Használati esetek . . . . .	19
3.3. Tervezés . . . . .	19
3.3.1. Architektúra . . . . .	19
3.3.2. Adatbázis . . . . .	20
3.3.3. A bridzs játék modellezése . . . . .	22
3.3.4. A haladó szintű automatizált felvevőjáték algoritmus . . . . .	29

3.3.5.	Felhasználói felület . . . . .	35
3.3.6.	API terv . . . . .	39
3.4.	Megvalósítás . . . . .	39
3.4.1.	Felhasznált eszközök . . . . .	39
3.4.2.	Felhasznált technológiák . . . . .	41
3.4.3.	Lapkijátszás . . . . .	49
3.5.	Tesztelés . . . . .	53
3.5.1.	Egységtesztek . . . . .	54
3.5.2.	Felületi tesztek . . . . .	57
3.5.3.	Az algoritmus tesztelése . . . . .	65
<b>4.</b>	<b>Összegzés</b>	<b>67</b>
4.1.	Kitekintés . . . . .	67
4.2.	Fejlesztési lehetőségek . . . . .	68
	<b>Köszönetnyilvánítás</b>	<b>69</b>
	<b>Irodalomjegyzék</b>	<b>69</b>

# 1. fejezet

## Bevezetés

A bridzs egy világszerte ismert kártyajáték és sport, amit hobbi szinten vagy versenyszerűen is lehet űzni. Jómagam először társasjátékként találkoztam a briddzsel, a nagyapámmal és a két testvéremmel játszottam, majd csatlakoztam egy bridzs szövetséghez, ahol online edzések és pörgős nyári edzőtáborok során elég tapasztalatot gyűjtöttem, hogy versenyekre nevezhessek és végül az U26 női válogatott tagjaként részt vehessek a veldhoveni ifjúsági bridzs Európa bajnokságon. Tapasztalatom szerint a bridzs bármilyen korosztálynak élvezetes lehet, egyrészt közösségi jellege miatt, másrészt mert változatos fejtörők elé állítja a játékosokat.

A szakdolgozat alkalmazásommal ezeket az izgalmas *bridzs fejtörőket* szerettem volna mindenki számára könnyen hozzáférhetővé tenni azzal, hogy egy olyan online felületet készítek, amire belépve bárki egyszerűen gyakorolhatja az ellenjátékot egy automata felvevő ellen, automata partnerrel, vagy elemezhet felvételeket, mindkét ellenjátékos kezeit irányítva egy automata felvevő ellen.

A megoldandó feladat tehát egy webes alkalmazás elkészítése, amit a felhasználó könnyen elérhet laptopról egy modern böngészőből. A webes alkalmazásnak első sorban az *online bridzs játékot* kell lehetővé tennie, és emellett egyéb funkciókkal, például regisztrációval és a játékok eredményeinek listázásával is rendelkeznie kell. Az online bridzs játék során legalább az egyik ellenjátékos hívásait a felhasználó határozza meg, a felvevő hívásait pedig egy *vélhetően ideális hívást kiszámító automatizált felvevőjáték algoritmus*.

Az automata felvevőjáték algoritmusnak úgy kell gondolkodnia, mint egy emberi felve-

vőnek, megjegyeznie a játék során kijátszott lapokat, kiszámítania, hogy milyen lapokat tart legnagyobb valószínűséggel a kezében az ellenjátékos, és hogy ez alapján milyen lap kihívásra a legnagyobb az esélye annak, hogy a legtöbb ütést szerezze.

## 2. fejezet

# Felhasználói dokumentáció

A szakdolgozat alkalmazás fő funkciója az *online bridzs játék automatikus felvevő ellen* és emellett vannak a felhasználói élményt javító funkciói, mint például súgó, és a bejelentkezés, amivel a felhasználó nyomon követheti, hogy eddig melyik leosztásokat játszott le milyen eredménnyel.

A játékelményt nagyban erősíti, ha valaki már jártas a bridzs játékban, de az alkalmazás használata nélkül is könnyen elsajátítható. A felhasználó alapértelmezetten a *keleten ülő ellenjátékos* szerepét tölti be a játékban, vagyis ő hív először. Miután kijátszik egy lapot, az asztal is így tesz, majd alapértelmezetten egy algoritmus is kijátszik egy lapot nyugat helyén. A kör észak lapkijátszásával zárul és az a játékos hívja a következő kör első lapját, aki a legnagyobb értékű lapot, vagy színjáték és kijátszott adu esetén a legnagyobb értékű adut tette.

Lapot kijátszani nagyon egyszerű, ha a játkosnak van olyan színe, mint amit hívtak, akkor abból a színből kell egy tetszőleges lapot kijátszania, egyébként bármit tehet. Olyan lapot választani a színből, amivel a játékos az adott körben ütni tud, vagy olyan lapot választani, ami hosszútávon elég sok ütést biztosít a játékosnak és a partnerének már nehezebb feladat, van hogy alapos megfontolást igényel (lásd: 3.1.1 Mi alapján játszik ki egy lapot egy emberi játékos).

## 2.1. Az alkalmazás elindítása

### 2.1.1. Előkészítés

Az alkalmazás három részből áll, adatbázisból, backendből, és frontendből, amik konténerbe lettek helyezve, hogy a felhasználó könnyebben el tudja indítani az alkalmazást. A felhasználónak rendelkeznie kell a **Docker Desktop** alkalmazással, ami letölthető a következő linken elérhető oldalról: <https://www.docker.com/products/docker-desktop/>. Ha valaki Windowson szeretné használni az alkalmazást, annak *Windows Subsystem for Linux*-al (WSL) is rendelkeznie kell, mert a Docker ezt használja. A WSL a következő lépések követésével telepíthető:

1. *PowerShell* alkalmazást megkeresése a start menüben és futtatása adminisztrátor jogosultsággal.
2. A következő parancs futtatása: `wsl --install -no-distribution`.
3. A számítógép újraindítása, hogy a módosítások végrehajtsódjanak.

A felhasználónak rendelkeznie kell a *szakdolgozat programmal* is, ami a feltöltött ZIP fájl gyökerében található. Az alkalmazás használata előtt ki kell csomagolni a ZIP fájlt.

### 2.1.2. Első indítás

A szakdolgozat alkalmazás elindítása előtt a felhasználónak el kell indítani a számítógépén található *Docker Desktop* alkalmazást. Ezután meg kell nyitnia egy terminált és kicsomagolt szakdolgozat ZIP fájl gyökerében a következő parancsokat kiadnia:

1. `docker compose build` (elkészíti a konténereket)
2. `docker compose up -d` (elindítja a konténereket)



```

=> => exporting config sha256:dbellc668013876beeb9d120e0b66682b2a6107a54e7ea6da66cc0663cd68912 0.0s
=> => exporting attestation manifest sha256:e9027f684e4410147290fe1e5b6383f9dc4168591dabd9dabc021cdf7ffa13af3 0.0s
=> => exporting manifest list sha256:ac9d5aedfd23d7b81cc57e3b097c5ac24c21714dd1965a872240e752bf666917 0.0s
=> => naming to docker.io/library/alkalmazas-frontend:latest 0.0s
=> => unpacking to docker.io/library/alkalmazas-frontend:latest 0.0s
=> [frontend] resolving provenance for metadata file 0.0s
[+] Building 2/2
✓ backend Built 0.0s
✓ frontend Built 0.0s
PS C:\_Agi\Szakdoli\alkalmazas> docker compose up -d
[+] Running 12/12
✓mysql Pulled 14.4s
✓c0468773f197 Pull complete 0.2s
✓26b53dc04f70 Pull complete 7.0s
✓c2eb5d06bfea Pull complete 6.1s
✓a48ef6c0e15d Pull complete 0.6s
✓e0211b440535 Pull complete 11.3s
✓8609c4bc4339 Pull complete 0.6s
✓253ce0d09858 Pull complete 0.6s
✓cc6ed31dcc2a Pull complete 0.6s
✓6c1e029c2325 Pull complete 0.6s
✓fb10e0b9f49b Pull complete 0.7s
✓142bf0fa8655 Pull complete 6.2s
[+] Running 5/5
✓Network alkalmazas_default Created 0.0s
✓Volume "alkalmazas_mysql_data" Created 0.0s
✓Container alkalmazas-mysql-1 Started 0.9s
✓Container alkalmazas-backend-1 Started 0.8s
✓Container alkalmazas-frontend-1 Started 0.9s
PS C:\_Agi\Szakdoli\alkalmazas>

```

2.1. ábra. A konténerek sikeres elindulását jelző szöveg a terminálon.

Ezután az alkalmazás elérhető egy tetszőleges böngészőben a `http://localhost:8084/index.html` URL-en, vagyis egy böngészőt megnyitva ezt az URL-t kell a böngésző tetején található sávba másolni.

### 2.1.3. Leállítás és későbbi használat

Az alkalmazás használatának befejeztével be kell zárni a böngészőablakot, amiben meg volt nyitva és a konténereket is le kell állítani. Ehhez egy terminálon a szakdolgozat ZIP fájl gyökerében a következő parancsot kell kiadni: `docker compose down`.

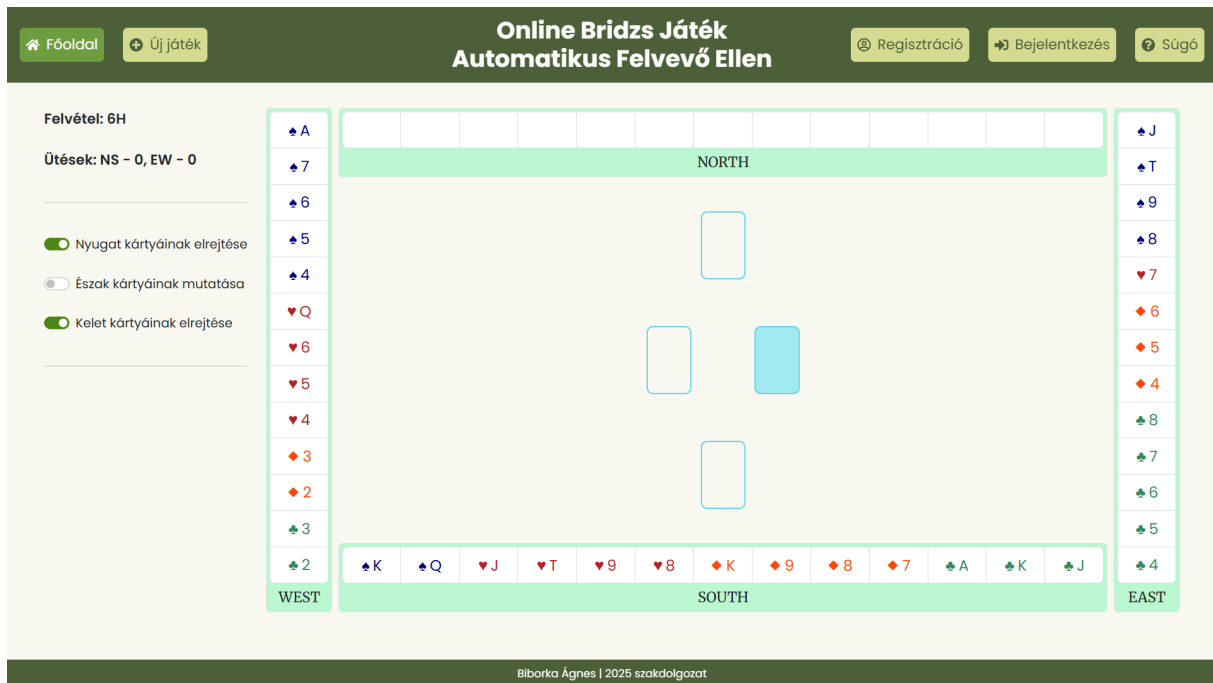
Az alkalmazási későbbi használata előtt ugyanúgy el kell indítani a *Docker Desktop* alkalmazást, de elég csak a következő parancsot kiadni egy terminálon a szakdolgozat ZIP fájl gyökerében: `docker compose up -d`. Ezután a `http://localhost:8084/index.html` URL-en elérhető lesz az alkalmazás a böngészőben.

## 2.2. Az alkalmazás használata

### 2.2.1. Játékmódok

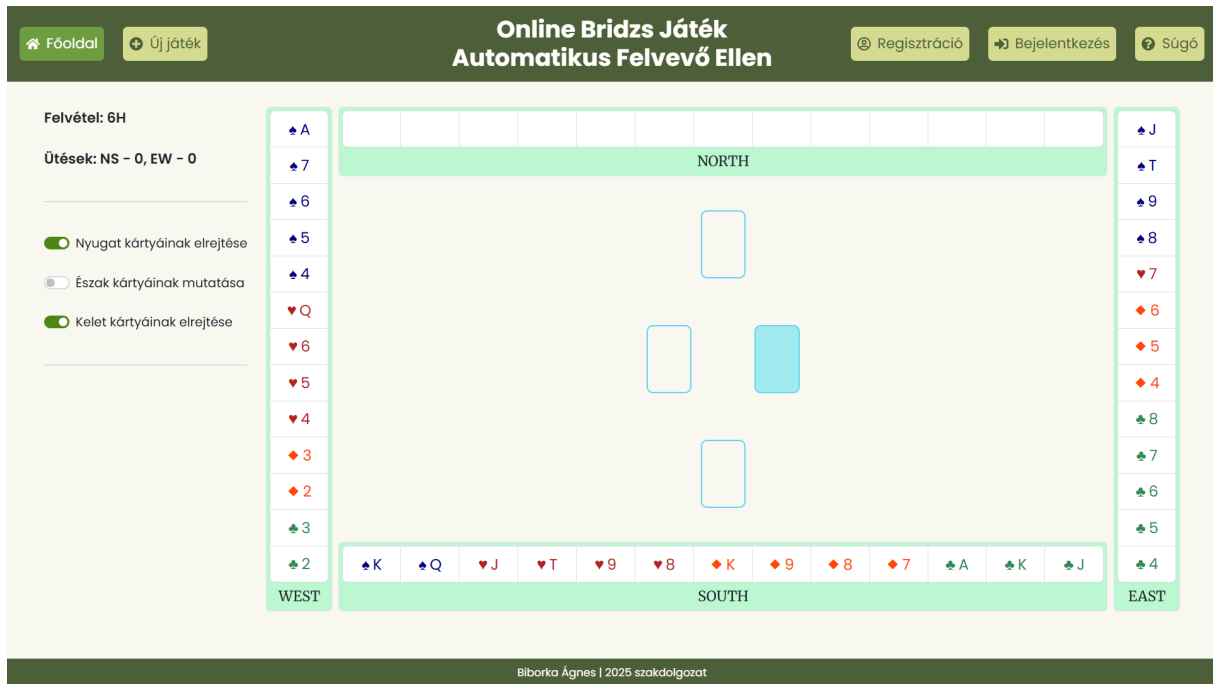
Az alkalmazásban kétféle játékmód érhető el: a *hagyományos ellenjáték* és az *elemző ellenjáték*.

A hagyományos ellenjáték hasonlóan működik ahhoz, ahogy az emberi játékos társakkal: az ellenjátékos a játék során csak a saját és az asztal lapjait látja, ez alapján játssza ki a saját lapjait és a többi játékos magától tesz lapokat. Az alkalmazásban megvalósított ellenjáték abban különbözik az emberi partnerrel játszott ellenjátéktól, hogy az automatikus ellenjátékos partnerrel nem lehet előre egy stratégiát és lapszámjelzési módszereket egyeztetni, de a játék súgójában elérhető, hogy milyen alapelvek alapján dönt az automatikus ellenjátékos. Az alkalmazásban az ellenjátékot az is nehezíti, hogy csak a licit végeredménye ismert, ezáltal nem lehet a licitből információkat kikövetkeztetni a felvevő lapjairól. A hagyományos ellenjáték játékmóddal a felhasználó az ellenjátékot gyakorolhatja. Ez az alapértelmezett játékmód.



2.2. ábra. A hagyományos ellenjáték játékmód.

Az elemző ellenjáték során a felhasználó kelet és nyugat helyében is játszik ki lapokat, tehát az ő kártyáikat ismeri. Az asztalt is ismernie kell, amiből már kiszámolhatja a felvevő kártyát is. (Emiatt ez a játékmód valamelyest a double dummy játékmódhoz hasonlít, azonban itt a felvevő csak a saját és az asztal kártyáit ismeri). Ennek a játékmódnak az a célja, hogy a felhasználó megfigyelhesse, hogy az automatikus felvevő milyen helyzetekben mit játszik ki és ebből tanulhasson.



2.3. ábra. Az elemző ellenjáték játékmód.

### 2.2.2. Instant játék

Ahogy a felhasználó megnyitja az alkalmazás főoldalát azonnal elindul egy "instant játék". Az instant játék a fejezet elején ismertettek szerint a hagyományos játékmódban zajlik, ami azt jelenti, hogy a felhasználó a keleten ülő ellenjátékos szerepét tölti be, a partnere egy algoritmus és a felvevő helyében is egy algoritmus hozza a döntéseket.

Az instant játék nagyban *leegyszerűsíti az alkalmazás fő funkciójának elérését*, mert így a felhasználónak nem kell először egy regisztrációt, bejelentkezést és játék paraméterek beállítását végrehajtania, ha csak egyszerűen játszani szeretne. Az instant játék az *alkalmazás főoldalán* található, és minden információt megjelenít, ami egy bridzs játékhoz szükséges: a felvételt, a vonalak eddig megszerzett pontszámait, a játékosok lapjait és az aktuális körben kijátszott kártyákat. A játéktábla mellett találhatóak a gombok, amikkel egyes játékosok lapjainak láthatósága állítható és a gomb, amivel új, paraméteres játékot lehet indítani. A főoldalról továbbá egy-egy felirattal ellátott gombon keresztül elérhető a súgó és a profillal kapcsolatos műveletek, ha a felhasználó szeretne bejelentkezni vagy egy bejelentkezett felhasználó szeretné megnézni az eredményeit.

A játékosok lapjai a képernyő négy oldalával párhuzamosan a képernyő közepén jelenik meg az ábrán látható módon. A lapokon belül, az asztal közepén jelennek meg az aktuális

körben kijátszott lapok. A felvétel és a vonalak aktuális pontszáma a játékosok kártyái felett jelenik meg.

### Lapkiállítás

Lapot mindig csak a bridzs szabályai szerint soron következő játékos játszhat ki. Hogy a játék könnyebben követhető legyen, az alkalmazás kék színnel kiemeli azt az égtájt, ahova a soron következő játékosnak épp ki kell játszania egy lapot. A felhasználó az általa irányított *ellenjátékos egyik kártyájára kattintva* tudja a kártyát kijátszani, ha az megfelel a fent leírt szabályoknak. Az algoritmus által irányított játékosok lapjai automatikusan kerülnek kijátszásra egy két másodperces késleltetés után.

### Egy égtáj kártyáinak láthatósága

A bridzs szabályai szerint az asztaltól jobbra ülő ellenjátékos kezd. Az alkalmazásban mindig észak a felvevő, dél az asztal és így mindig kelet kezd. Így keleti hívja ki a játékban az első lapot (az indító kijátszást), és az asztal lapjai csak ezután válnak mindenki számára láthatóvá (az asztal leterül). Ennek megfelelően a felhasználó kezdetben csak a saját kezét látja és csak keleti első hívása után ismeri meg az asztal lapjait.

A hagyományos ellenjáték játékmódban a keleten ülő játékos csak a saját és az asztal lapjait látja. Az elemző ellenjáték játékmódban is végig látszik az asztal. Emellett a felhasználó a játék során beállíthatja, hogy melyik ellenjátékos lapjait szeretné látni, és hogy szeretné-e látni észak lapjait (a másik három kéz ismeretében észak lapjai amúgy is kiszámíthatóak).

### Játék vége

Egy leosztás tizenhárom kör után ér véget, amikor a játékosok kijátszották az összes lapjukat. Az alkalmazás a játék végén egy felugró ablakon jeleníti meg a játék eredményét, azt, hogy a felhasználónak sikerült-e megbuktatnia a felvételt, és hogy a felvétel hány plusz ütessel teljesült, vagy hány ütés híján nem teljesült. A felugró ablakot bezárva a felhasználó visszakerül a játék főoldalára, ahol új instant játék indul. Ha be van jelentkezve, az "Eredmények" gombra kattintva az alkalmazás eredményeket mutató oldalára kerül, ahol megtekintheti eddig lejátszott leosztásairól elmentett statisztikákat. A felugró ablakból továbbá új játékot is lehet indítani megadott paraméterekkel.

### 2.2.3. Új játék paraméterekkel

Az új játék indítása funkció egy játék végén felugró ablakból és a főoldalról az "Új játék" gombra kattintva is elérhető. Ekkor a felhasználó a főoldalról az új játék indítása oldalra lesz átirányítva. Itt a felhasználó választhat a játékmódok között, beállíthatja a fellevevő tudásszintjét, aztán kiválaszthatja, hogy melyik leosztást szeretné játszani. A leosztás kiválasztása lehet random, lehet random, azzal a kikötéssel, hogy szín vagy szan játék, és a felhasználó választhat egy leosztás sorszámot is. Ezutóbbi főleg akkor hasznos, ha a felhasználó be van jelentkezve és tudja melyik leosztásokat játszott már milyen eredménnyel. Az új játék a következő gombok egyikére kattintva indul el: "Véletlen játék", "Paraméteres játék", "Leosztás lejátszása".



2.4. ábra. Az alkalmazás új játék indításának oldala.



2.5. ábra. Az alkalmazás új játék indításának oldala paraméterekkel.

#### 2.2.4. Navigáció és felhasználó profil

A navigáció az alkalmazásban egyszerű. Minden oldal tetején elérhető a menü, amiben gombok segítségével bármelyik oldalra el lehet jutni: főoldal, új játék, súgó és a profillal kapcsolatos oldalak.

A felhasználó profillal kapcsolatos műveletek aszerint különbözöek, hogy a felhasználó be van-e jelentkezve vagy sem. Egy nem bejelentkezett felhasználó a következő menüpontokat látja, amit egy bejelentkezett felhasználó nem: "Regisztráció" és "Bejelentkezés". Egy bejelentkezett felhasználó a "Kijelentkezés" és az "Eredmények" menüpontjokat látja ezek helyett.



2.6. ábra. A menü, amit egy nem bejelentkezett felhasználó lát.



2.7. ábra. A menü, amit egy bejelentkezett felhasználó lát.

## Regisztráció

Bármelyik felhasználó tud új profilt regisztrálni azzal, hogy a profil menüben a "Regisztráció" pontot választja. Ekkor átirányításra kerül az alkalmazás regisztráció oldalára. Itt a szöveges beviteli mezőben kötelező megadnia egy felhasználónevet és egy jelszót. Az E-mail cím megadása opcionális. Regisztrálni a "Regisztráció" gombra kattintva lehet. A regisztráció sikeres, ha a megadott felhasználónév egyedi. Sikeres regisztráció után egy még nem bejelentkezett felhasználó bármikor be tud lépni a regisztrált felhasználónevét és jelszavát használva.

Felhasználónév\*

Felhasználó

Kötelező mező, egyedi, 3-20 karakter nem lehet ékezetes vagy speciális.

E-mail cím

E-mail

Jelszó\*

\*\*\*\*\*

Kötelező mező, legalább 8 karakter.

Regisztráció

Van már fókod? [Bejelentkezés](#)

Biborka Ágnes | 2025 szakkolgozat

2.8. ábra. Az alkalmazás regisztráció oldala.

## Bejelentkezés és kijelentkezés

Egy felhasználó a profil menü "Bejelentkezés" pontjából elérhető bejelentkezés oldalon tud bejelentkezni. A bejelentkezéshez meg kell adnia a felhasználónevét és a hozzá tartozó jelszót. Ha a felhasználónév létezik, és a megadott jelszó is helyes, akkor sikeres a bejelentkezés. Egy bejelentkezett felhasználó a profil menü "Kijelentkezés" pontjára kattintva tud kijelentkezni.

Főoldal Új játék Bejelentkezés Regisztráció Súlyó

Felhasználónév  
Felhasználó

Jelszó  
\*\*\*\*\*

Bejelentkezés

Nincs még fókod? [Regisztráció](#)

Biborka Ágnes | 2025 szakkolgozat

2.9. ábra. Az alkalmazás bejelentkezés oldala.

## Eredmények

Egy bejelentkezett felhasználó meg tudja tekinteni az eredményeit a profil menü "Eredmények" pontjára kattintva. Ekkor átirányításra kerül az alkalmazás eredmények oldalára. Az eredmények oldalon egy rövid magyarázat alatt egy táblázatban megjelennek a felhasználó eddigi játékának eredményei. A táblázat első oszlopában a leosztás sorszáma, a második oszlopában a leosztás eredménye.

Főoldal Új játék Eredmények Kijelentkezés Súlyó

Üdv, hammy!

Ezen az oldalon látszanak az eredményeid. Ha egy adott leosztást szeretnél lejátszani az adott sorban szereplő paraméterekkel, kattints a sor végén lévő ikonra.

Jelmagyarázat: ezt a meccset még nem játszottad le sikeresen megbuktattad a felvételt nem sikerült megbuktatnod a felvételt

Leosztás száma	Felvétel	Eredmény	Felvevő ütészülönbsége	Ellenjáték módja	Felvevő szintje	Utolsó játék vége	Játék
1	3NT			hagyományos	kezdő		
1	3NT			hagyományos	haladó		
1	3NT			elemző	kezdő		
1	3NT		-1	elemző	haladó	04.25. 09:48	
2	6H			hagyományos	kezdő		
2	6H			hagyományos	haladó		
2	6H			elemző	kezdő		
2	6H		1	elemző	haladó	04.27. 20:58	

Biborka Ágnes | 2025 szakkolgozat

2.10. ábra. Az alkalmazás eredmények oldala.



### 2.2.5. Sútó

A fűoldalón a jobb felsű sarokban található "Sútó" gombra kattintva a felhasználó át lesz irányítva az alkalmazás sútó oldalára. Ez az oldal információval szolgál a bridzs játék szabályairól és arról is, hogy hogyan lehet az alkalmazást használni.



2.11. ábra. Az alkalmazás sútó oldala.

## 3. fejezet

# Fejlesztői dokumentáció

### 3.1. Elméleti háttér

#### 3.1.1. Mi alapján játszik ki egy lapot egy emberi játékos

A felvevőjáték és az ellenjáték különbözik egymástól, mind a kettőhöz más-más stratégia kell. A stratégia változik aszerint is, hogy milyen típusú mérkőzésen veszünk részt: páros mérkőzésen, csapat mérkőzésen, IMP vagy MP pontozás szerint. A bridzs játékosok éveken keresztül tökéletesítik a tudásukat, sokat gyakorolnak, elemzik a lejátszott partikat, bridzs közvetítéseket néznek és bridzs könyveket olvasnak.

A szakdolgozat csak a felvevőjátékkal foglalkozik, mert ennek a lejátszásához nem kell kooperáció és nem kell a partner jelzéseit figyelni, mint ellenjáték esetében, de itt is jól látszik egy jellegzetesség, aminek nagy szerepe van abban, hogy egy adott helyzetben milyen lapot játszik ki egy játékos: ez pedig a *kijátszott lapok megjegyzése* (bridzs nyelven számolása) és ennek alapján a *lehetséges lap eloszlások valószínűségének a kiszámítása*.

Néha egyértelmű, hogy mit kell tenni egy adott színből, vannak bevett fordulatok is (informatikai szakszóval heurisztikák), de gyakran összetett számításokkal lehet csak megindokolni, hogy egy hívás miért jó hívás. Míg egy emberi felvevő játékos a játék elején *játéktervet készít*, vagyis átgondolja, hogy melyik színből hány ütést tud szerezni, milyen valószínűséggel, figyel a közlekedésre és számításba veszi azt is, hogy mi történik, ha nem válnak be a számításai, a szakdolgozatban megvalósított felvevő egy egyszerű heurisztika

alapján kiválaszt egy szint (amelyikről a legtöbb információja van) és azt a lapot hívja, amelyikről úgy gondolja, hogy az *adott színben a legtöbb ütéshez vezet*.

A szakdolgozatban megvalósított felvevő nem figyel egyszerre mind a négy színre, sem a közlekedésre, de sok bridzs tankönyv tanítja úgy a felvevőjátékot, hogy egy-egy szint hoz fel csak példaként és azt mutatja be, hogy az adott színben hogy lehet a legtöbbet ütni, ezért ez az alkalmazás is alkalmas kezdők számára gyakorlásra.

## 3.2. Követelményelemzés

### 3.2.1. Specifikáció

A feladat egy webes alkalmazás elkészítése, amit a felhasználó könnyen elérhet laptopról egy modern böngészőből. Az alkalmazásnak lehetővé kell tennie az online bridzs játékot, és emellett a regisztrációt, bejelentkezést, bejelentkezés esetén az eredmények listázását is biztosítani kell. Az online bridzs játéknak rendelkeznie kell egy intuitívan kezelhető grafikus felülettel, amin megjelennek a játékosok lapjai és az éppen kijátszott kártyák. Az ellenjátékosok játékát a felhasználónak kell tudnia irányítani a kijátszani kívánt lapra kattintva. A felvevő és adott esetben a felhasználó partnerét helyettesítő automatikus ellenjátékos hívásait egy algoritmusnak kell meghatároznia.

### 3.2.2. Funkcionális követelmények

- **Játék betöltés**

- *Instant játék*: A játéknak be kell töltnie, amikor a felhasználó először megnyitja az alkalmazás főoldalát a böngészőben.
- *Random játék indítása*: Egy véletlenszerűen kiválasztott játéknak be kell töltnie, amikor a felhasználó a "Véletlen játék" gombra kattint.
- *Paraméteres játék indítása*: egy véletlenszerűen kiválasztott és a paramétereknek megfelelően egy szín vagy egy szan játéknak be kell töltnie, amikor a felhasználó a "Paraméteres játék" gombra kattint.
- *Kiválasztott játék indítása*: A megadott azonosítóval rendelkező játéknak be kell töltnie, amikor a felhasználó a "Leosztás lejárta" gombra kattint.
- *Nehézségi szint beállítása*: A felhasználó be kell tudja állítani a játék nehézségi szintjét az automatikus felvevő tudásszintének meghatározásával.

- **A játék**

- *Ellenjátékos lépése*: A felhasználó a soron következő ellenjátékos egy kártyájára kattintva ki kell tudjon játszani egy lapot a bridzs szabályai szerint (színre szint kell tenni).
- *Felvevő lépése*: Az alkalmazásnak ki kell tudnia számítani, hogy a felvevő vagy az asztal kezéből melyik lapját érdemes kijátszania a legtöbb ütés megszerzése érdekében és ezt a lapot ki is kell tudja játszani.

- **Felhasználói profil**

- *Regisztráció*: A felhasználónak létre kell tudnia hozni egy új profilt egy egyedi felhasználónév és egy jelszó megadásával. Ezek mellett opcionálisan megadhat egy e-mail címet is.
- *Bejelentkezés*: A felhasználónak be kell tudnia lépni a felhasználónevének és a jelszavának a megadásával.
- *Kijelentkezés*: A bejelentkezett felhasználónak ki kell tudnia jelentkezni, amikor a "Kijelentkezés" menüpontra kattint.

- **Információk**

- *Súgó*: A felhasználónak meg kell tudnia nyitni egy súgó oldalt, amikor a "Súgó" menüpontra kattint.
- *Eredmények*: Egy bejelentkezett felhasználónak meg kell tudnia nézni a játékban elért eredményeit, amikor az "Eredmények" menüpontra kattint.

### 3.2.3. Nem funkcionális követelmények

- **Adattárolás**

- A rendszernek perzisztensen tárolnia kell a regisztrált felhasználókat.
- A rendszernek perzisztensen tárolnia kell egy regisztrált felhasználó játék eredményeit, amiket minden játék végén frissítenie kell.
- A rendszernek perzisztensen tárolnia kell bridzs leosztások reprezentációját.

- **Modellezés**

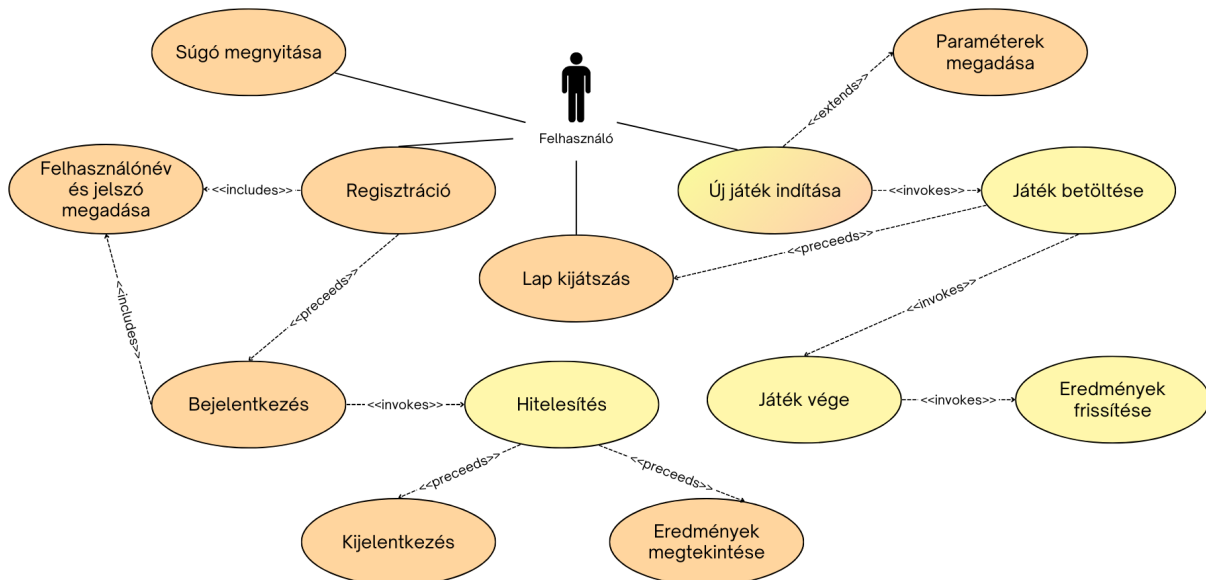
- Az alkalmazásnak objektum-orientáltan modelleznie kell egy bridzs játék lejátzását.
- A modellezett lejátzás során a felvevő hívásait egy automatikus felvevő játék algoritmusnak kell meghatároznia.
- A modellezett lejátzás során a felhasználó hívásait a felhasználónak kell tudnia

meghatározni.

### 3.2.4. Megszorítások

- Az alkalmazás webes alkalmazásként kell működjön, amit a felhasználó az alábbi böngészők egyikéből tud elérni: Chrome, FireFox, Microsoft Edge.
- Az alkalmazás backendjét SpringBoot keretrendszerben, Java nyelven kell implementálni.
- Az alkalmazásnak rendelkeznie kell egy grafikus felülettel.

### 3.2.5. Használati esetek



3.1. ábra. Használati eset diagram. A narancssárga háttérű használati eseteket a felhasználó hajtja végre, a sárga háttérű használati eseteket a rendszer hajtja végre a felhasználó interakcióinak hatására.

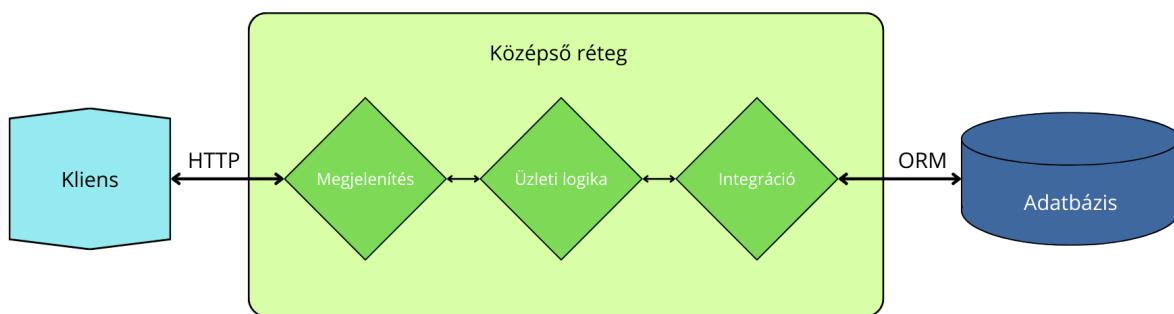
## 3.3. Tervezés

### 3.3.1. Architektúra

A háromrétegű architektúra egy sztenderd modell dinamikus, webes alkalmazások megvalósítására [1], ezért egy ilyen architektúra képi az automatikus felvevőjáték alkalmazás alapját azzal a kiegészítéssel, hogy a középső réteg üzleti logikája fel van osztva további három rétegre.

- Kliens réteg (client tier): a felhasználói felület megjelenítéséért felel.
- Középső réteg (middle tier):
  - Megjelenítési réteg (presentation layer): kezeli a HTTP kéréseket (webszerver).
  - Üzleti logika (Business logic layer): az üzleti logikát tartalmazza (alkalmazás-szerver).
  - Integrációs réteg (integration layer): egy ORM framework segítségével kezeli az interakciót az adatbázissal.
- Adatbázis réteg (resource tier): egy relációs adatbázis kezelő rendszer a perzisztens adatkezelésre.

A középső réteg további felosztását az indokolja, hogy a rétegek összetettsége tovább csökken és az alkalmazáson belül a különböző folyamatok jobban elkülönülnek egymástól, ezzel pedig csökken a hiba lehetőségek száma is [1].



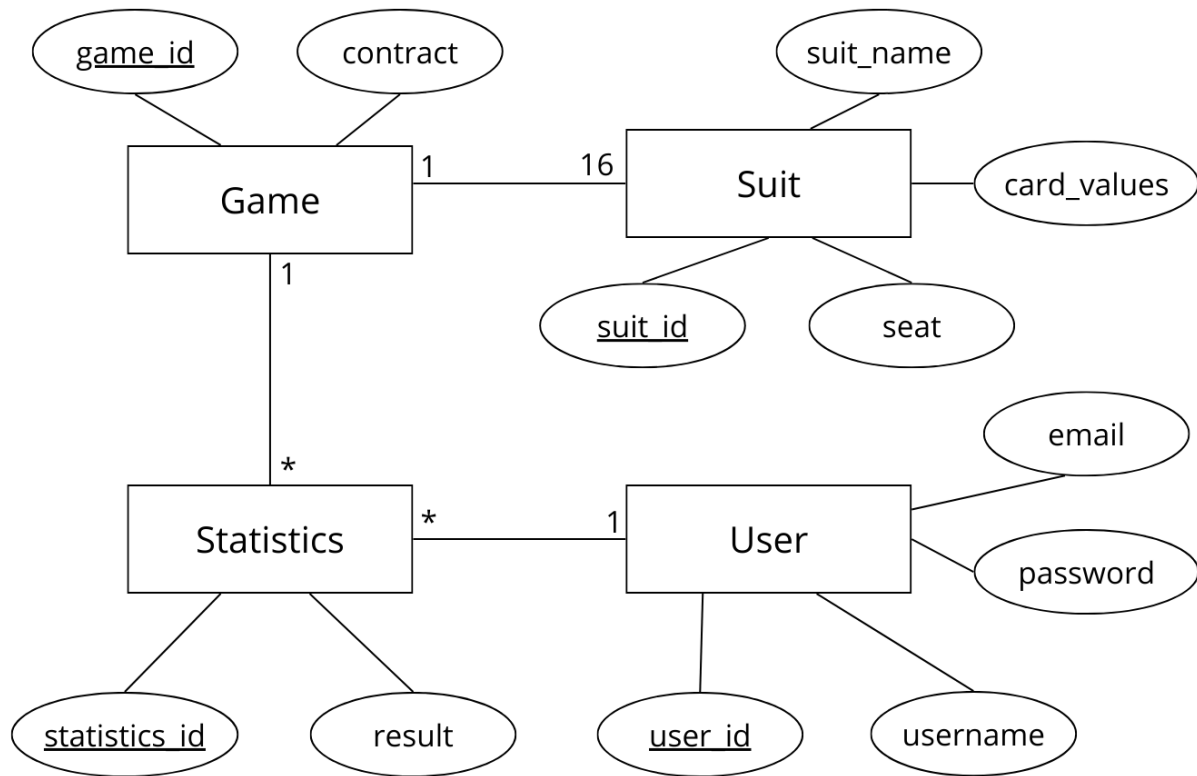
3.2. ábra. Az automatizált felvevőjáték rendszerének architektúrája.

### 3.3.2. Adatbázis

A relációs adatbázis séma megtervezésének egyik módja az, hogy a feladateleírás alapján először egy egyed-kapcsolat modellt vázolunk fel, majd azt alakítjuk át relációs adatbázis sémává [2]. Így ügyes tervezéssel elkerülhető a redundancia, ami inkonzisztenciához vezethet. Az egyed-kapcsolat diagramot úgy lehet átalakítani relációsémává, hogy az egyedhalmazokat relációvá alakítjuk, amiknek az attribútuma az egyedhalmaz attribútumai és a kapcsolatokat is relációkká alakítjuk, amiknek az attribútumai az összekapcsolt egyedhalmazok kulcs-attribútumai [3].

**Feladateleírás:** Az adatbázisban egyrészt egy bridzs leosztást (game) kell eltárolni. Az alkalmazásban minden leosztáshoz előre meg van határozva a hozzátartozó felvétel (contract) is. Egy leosztás során az ötvenkét lapos franciakártya pakli kerül szétosztásra egyen-

lően az égtájak (seat) között. Egy égtáj lapjait a lapok színe szerint lehet csoportosítani (suit\_name), az egy színhez tartozó kártyákat (card\_values) pedig egy legfeljebb tizenhárom hosszú karaktersorozat reprezentálja. Az adatbázisban felhasználókat is el kell tárolni. Minden felhasználónak van egy egyedi felhasználóneve és egy jelszava, amit titkosítva kell tárolni. A felhasználóknak opcionálisan lehet e-mail címe is. A felhasználókhoz tartoznak statisztikák, amik a játékok eredményeit (result) tárolják.



3.3. ábra. Az adatbázisséma egyed-kapcsolat diagramja.

A 3.3 ábra alapján a relációs adatbázisséma:

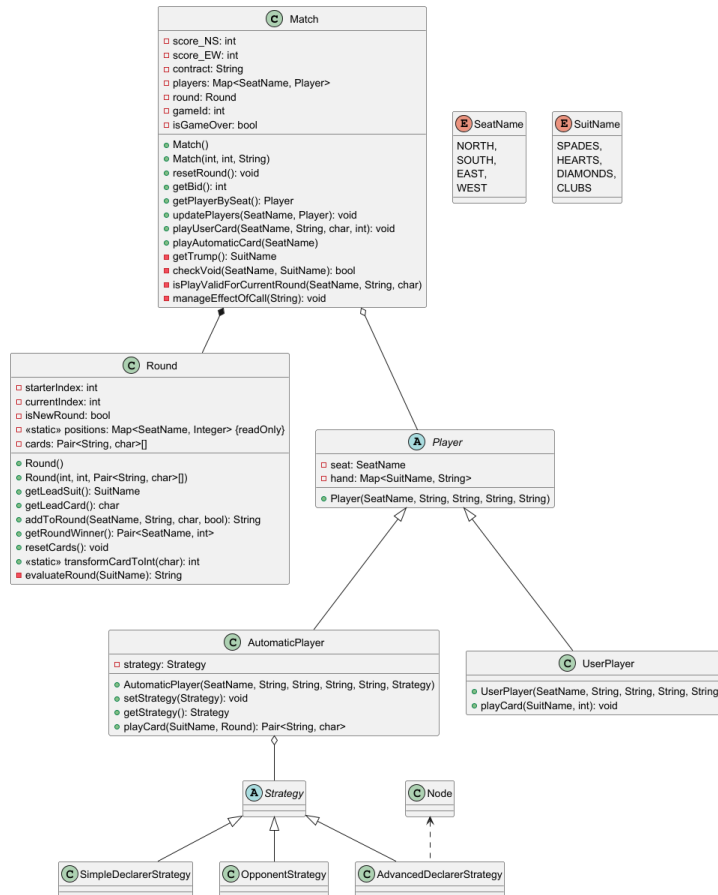
- Game(game\_id, contract)
- Suit(suit\_id, game\_id, seat, suit\_name, card\_values)
- User(user\_id, email, password, username)
- Statistics(statistics\_id, user\_id, result, game\_id)

Perzisztens adatelérésre két okból is szükség van. Ha nem lennének adatbázisban eltárolva a leosztások, hanem minden leosztást random generálna az alkalmazás, akkor szükség lenne egy licit funkcióra is a felvétel megállapítására. Ráadásul sok tanító célú bridge feladvány tartalmaz előre meghatározott leosztásokat, amik tanulságos, gyakran előfor-

duló vagy éppen trükkös eseteket mutatnak be, ezért hasznosak az ilyen előre megadott leosztások.

### 3.3.3. A bridzs játék modellezése

Ahhoz, hogy a bridzs játékot, pontosabban meccsek sorozatát modellezni tudjunk, modelleznünk kell a meccshez tartozó leosztást és a négy játékost, akiknek releváns tulajdonságuk, hogy melyik égtájon ülnek és hogy a leosztásban milyen lapokat kaptak. A meccsnek információt kell tárolnia a felvételtől, arról, hogy eddig melyik vonal hány ütést szerzett, a játékosokról, és arról, hogy egy adott körbe melyik játékos milyen lapot rakott. A szakdolgozatban a felvelőt és az ellenvonal játékosait egy kis eltéréssel kell modellezni, mert az ellenjátékos hívásait a felhasználó határozza meg, a felvevő hívásait pedig egy algoritmus. Az algoritmusnak két változata is van, amit aszerint kell megválasztani, hogy a felhasználó egy kezdő vagy egy haladó szintű felvevő ellen szeretne játszani. Ezeket a szempontokat figyelembe véve az 3.4 és a 3.5 ábrán szemléltetett osztálydiagrammot rajzolhatjuk fel.



3.4. ábra. Az osztálydiagram felső része.



Az alábbiakban az üzleti logika osztályai kerülnek részletesen ismertetésre, majd egy fontos tervezési döntés, a stratégia tervezési minta.

## A Match osztály

A modellezett bridzs játékban a `Match` osztálynak központi szerepe van. Ez az osztály *reprezentál egy meccset* és így tartalmazza az összes releváns információt az aktuális meccsről az alábbi adattagok formájában:

- A *felvétel* adatai:
  - `score_NS` és `score_EW`: az észak-dél és a kelet-nyugat vonal vonal aktuális pontszáma, amit az `updateScores` metódus frissít minden kör végén.
  - `contract`: a felvétel sztring formátumban, pl. "3NT", "4H", "6D".
- A *játékosok* adatai - `players`: map, ahol az égtáj a kulcs, és a hozzátartozó érték egy `Player` típusú objektumként.
- Az *aktuális kör* adatai - `Round`: `Round` típusú objektum.
- A *meccs állapota* - `isGameOver`: egy igaz-hamis érték, ami addig igaz, amíg tart a játék.

A `Match` osztály legfontosabb publikus metódusai a `playUserCard` és a `playAutomaticCard` metódus. Ezek ellenőrzik, hogy egy lap kijátszás helyes-e: az a játékos játszik-e ki lapot, aki épp soron van, és színre színt rak-e, ha tud. Ha minden feltétel teljesül meghívja a `Player` objektumra annak `playCard` metódusát, és frissíti a meccs objektum adattagjait: az éppen aktuális kört, és a vonalak megszerzett ütésszámát.

- `playUserCard`: az `isPlayValidForCurrentRound` privát segédmetódust használva ellenőrzi, hogy a paraméterben megadott játékos a bridzs szabályai szerint kijátszhatja-e a paraméterben meghatározott kártyát. Ha igen, az `isPlayValidForCurrentRound` metódus megváltoztatja a meccs aktuális körében lévő lapokat. A játékos kezében lévő lapok a `UserPlayer` altípusú játékos `playCard` metódusának meghívásával változik meg. Szabályos lapkijátszás esetén a `manageEffectOfCall` metódus frissíti a pontszámokat, és azt, hogy véget ért-e a játék.
- `playAutomaticCard`: a `playUserCard` metódushoz hasonlóan működik, de mivel itt egy algoritmus választja ki, hogy a játékos milyen lapot játszik ki, ezeket nem várja

paraméterben, hanem a `AutomaticPlayer` altípusú játékos `playCard` metódusának visszatérése határozza meg.

Az osztály privát `getTrump` metódusa az adott meccsben érvényes adu színnel tér vissza, hasznos információval szolgálva az automatikus felvevőnek annak `playCard` metódusában. Emellett az osztálynak van egy privát `checkVoid` metódusa is, ami egy igaz-hamis értékkel tér vissza aszerint, hogy a paraméterben megadott égtájon ülő játékosnak van-e a paraméterben megadott színéből. Ezt a metódust az `isPlayValidForCurrentRound` metódus használja segédmetódusként.

## A Round osztály

A Round osztály *reprezentálja az éppen aktuális kört* a következő adattagokkal:

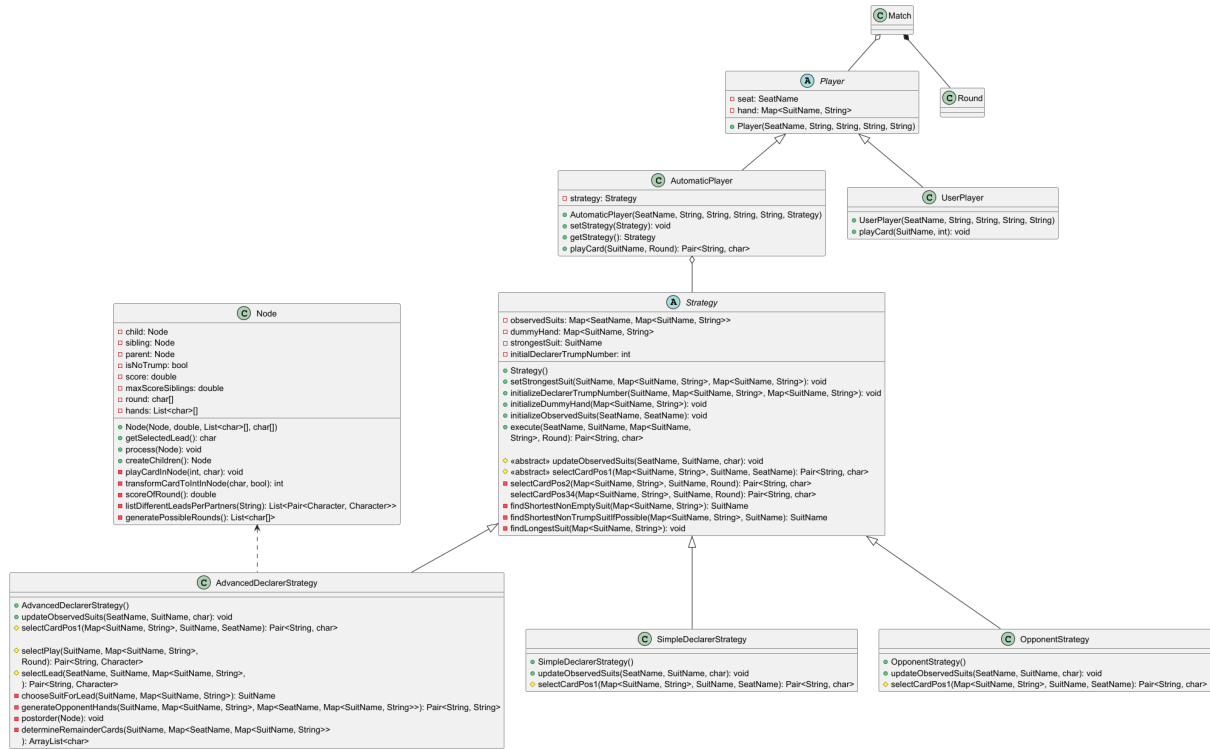
- *Információ az égtájakról:*
  - `positions`: égtáj-pozíció kulcs-érték hozzárendelések egy statikus és nem módosítható mapben. Mindegyik égtájra megadja, hogy az melyik pozíción ül. `{"EAST" => 0, "SOUTH" => 1, "WEST" => 2, "NORTH" => 3}`
  - `starterPosition`: megmutatja, hogy az égtájakhoz rendelt pozíciók közül melyik pozíció játszott/játszja ki az első lapot a körben. Pl. ha `starterPosition = 1`, akkor dél hívott elsőként.
  - `currentIndex`: megmutatja, hogy az égtájakhoz rendelt pozíciók közül melyik pozíció van éppen soron. Pl. ha `currentIndex = 2`, akkor épp nyugat van soron.
- Az *eddig kijátszott lapok* - `cards`: lapok négy hosszú tömbje, ahol egy lapot a színének sztring reprezentációja és értékének karakter reprezentációja jelöl és azokon az indexeken, ahol az ahhoz a pozícióhoz tartozó játékos még nem játszott ki semmit, ott üres sztring és space karakter található. Pl. A `[("spades", 'A'), ("spades", '5'), ("", ' '), ("", ' ')]` jelentése: az első játékos egy pikk ászt hívott, majd a következő játékos egy pikk ötöst rakott. A maradék két játékos pedig még nem tett semmit (most a harmadik játékos van soron).
- Az információ, hogy *épp új kör kezdődik-e* - `isNewRound`: igaz-hamis érték, ami igaz, ha még egy játékos sem játszott ki egy lapot sem.

A Round osztály legfontosabb metódusa a publikus `addToCards` metódusa, amit a meccs `isPlayValidForCurrentRound` metódusa hív a meccs osztály `round` adattagjának álla-

potának megváltoztatására. A metódus az égtájak információi alapján ellenőrzi, hogy az az égtáj van-e soron, aki lapot szeretne kijátszani és jó színből szeretne-e lapot kijátszani. Ha ezek teljesülnek, akkor frissíti a frissíti a `cards` tömböt, a megfelelő pozícióhoz adva a kijátszott kártyát és frissíti a soron következő pozíciót is. Ha ezzel a kiinduló pozícióhoz jutott vissza, akkor az `evaluateRound` privát segédmetódus meghívásával kiértékeli a kört és visszatér annak visszatérési értékével. A lap kijátszási próbálkozás kimenetelétől függően más-más üzenet sztringgel tér vissza: egy megerősítő üzenettel vagy egy hibüzenettel tér vissza.

A `evaluateRound` metódus kiértékeli az adott kört. Feltételes maximumkereséssel megállapítja, hogy ki vitte az adott kört az alapján, hogy mi volt a hívás színe, és ki tette az adott színből a legnagyobb lapot a meccsben érvényes adut is figyelembe véve. A keresés után alapértékre (" ", ' ') állítja a `cards` tömb elemeit, igazra állítja az `isNewRound` adattag értékét, és beállítja a következő hívó (`starterPosition`) és egyben a következő játékos (`currentPosition`) pozíciójának értékét is a nyertes indexére. Végül visszatér a megállapított nyertes vonalával: "EW" vagy "NS".

A `transformCardToInt` metódus minden kártyához, a kettéstől az ásig növekvő sorrendben rendre egy egész szám értékét rendeli kettőtől tizennégyig növekvő sorrendben. Ez a hozzárendelés tükrözi kártyák egymáshoz viszonyított értékét a bridzs szabályai szerint, tehát megállapítható vele, hogy egy kártya nagyobb-e egy másiknál. Az `evaluateRound` felhasználja ezt a tulajdonságot a maximumkeresés során.



3.5. ábra. Az osztálydiagram alsó része.

## A Player absztrakt osztály és leszármazottai

A Player absztrakt osztály *egy játékost reprezentál*, egy valós bridzs játékos két releváns tulajdonsága alapján: hogy melyik égtájon ül, és hogy milyen kártyákat tart a kezében.

A Player absztrakt osztály tehát két adattaggal rendelkezik:

- A játékos *égtája* - **seat**: az égtáj neve.
- A játékos *kezében tartott kártyái* - **suits**: szín-kártyaérték kulcs-érték párok halmaza. Pl. {"SPADES" => "AKT", "HEARTS" => "QJ6543", "DIAMONS" => "AJ6", "CLUBS" => "7"}

A szakdolgozat alkalmazásban *kétféle játékos van modellezés szempontjából*: egy olyan, akinek a lap kijátszásait a felhasználó határozza meg, és egy olyan, akinek a lapkijátszásait az alkalmazás algoritmusa. Ez alapján a Player osztálynak két leszármazott osztálya van, az AutomaticPlayer és a UserPlayer. A kétféle alosztály azért is hasznos, mert a meccs mind a négy égtájon Player típusú objektumokat tárol, és futásközben dőlhet el, hogy attól függően, hogy a felhasználó milyen játékmódot választ a nyugaton ülő játékos AutomaticPlayer vagy UserPlayer altípusú legyen. Ez az egyik példája a **futásidejű polimorfizmusnak**.

Mivel a játékosok objektumai a `Match` osztályban vannak, ott a `round` információiból megállapítható a meccshez minden egyéb fontos játékoshoz kapcsolódó tulajdonság, pl. hogy épp melyik játékos van soron.

A `UserPlayer` osztálynak az `ő`osztály adattagjain kívül nincs más adattagja. Egyetlen, fontos publikus metódusa a `playCard` metódus, amit a `Match` osztály `playUserCard` metódusa hív, a megfelelő paraméterekkel: egy szín nevével és a színben belüli kártyák közül az egyiknek az indexével. A `playCard` eltávolítja a játékos kezéből a megadott indexen lévő kártyát.

A `AutomaticPlayer` osztálynak van egy lényeges adattagja, ami a `UserPlayer`-ben nincs meg: ez a stratégia. Mivel az `AutomaticPlayer` saját maga dönti el, hogy milyen lapot játsszon ki, szüksége van egy stratégiára. Ahelyett, hogy az osztályon belül többféle stratégiának megfelelő lapkijátszó metódus lenne, a stratégia egy objektum, aminek altípusai határoznak meg egy-egy stratégiát. Így az `AutomaticPlayer` osztály logikája egyszerűbb lesz, a fejlesztés során könnyebb új stratégiákat hozzáadni és ez a megoldás lehetővé teszi azt is, hogy futási időben a felhasználó választásának megfelelően más-más stratégiájú automatikus játékosok működhessenek. Ez az előnyös tervezési döntés a **stratégia tervezési minta** egyik példája.

Az `AutomaticPlayer` osztálynak a `playUserCard` osztályhoz hasonlóan csak egyetlen publikus metódusa van, ez pedig a `playCard` metódus, ami a meccs során érvényes adu szint, és az aktuális körben kijátszott lapokat várja paraméterül és a `strategy` adattagra meghívott `execute` metódus visszatérése alapján meghatározza, hogy melyik színből milyen értékű lapot érdemes kijátszania a játékosnak. Ezt a választott lapot eltávolítja a játékos kezéből, majd visszatér a lappal.

#### A Strategy absztrakt osztály és leszármazottai

A szakdolgozat alkalmazásban *háromféle automatikus játék stratégiát kell modellezni*:

- egy "haladó" felvevőt, aki valószínűségeket és lehetséges hívássorozatokat figyelembe véve választ egy lapot (`AdvancedDeclarerStrategy`),
- egy "kezdő" felvevőt, aki egyszerű heurisztikákat követve tesz lapot (`SimpleDeclarerStrategy`),
- és egy ellenjátékost, aki a felhasználó partnere lehet nyugat helyén (`OpponentStrategy`).

Ezekhez a stratégiákhoz mind hasznos a kiment lapok megjegyzése, ezért az osztály rendelkezik egy `observedSuits` adattaggal, ahol egy mapben a játékosokat és a tőlük kiment lapokat szín-értékek párokként figyeli. Az osztálynak továbbá rendelkeznie kell egy `execute` metódussal is, ami a kijátszásra kiválasztott lappal tér vissza. Aszerint, hogy a ő hív-e első helyen a körben, vagy sem, két különböző segédmetódus egyikét hívja a lap kiválasztására: `selectLead` vagy `selectPlay`. Ezek a segédmetódusok a `Strategy` osztályban absztraktok, mindegyik leszármazott az általa képviselt stratégia szerint valósítja meg. Emellett az osztály olyan privát és védett láthatóságú metódusokat tartalmaz, ami az összes megvalósító osztály hasznára válik: az `encodeAllHands` átalakítja a kezeket a stratégia által könnyen feldolgozható formátumúra és az `updateObservedSuits` hozzáveszi az újonnan látott kiment lapokat az eddig megjegyzettekhez.

A leszármazott osztályok a következő bridzsos ökölszabályokat (avagy heurisztikákat) követve választják ki, hogy milyen lapot hívjának ki.

#### Automatikus hívás

- Szín játéknál
  - hosszú színt, ha hosszú az adu, rövid színt, ha rövid
  - ha nincs figura az adott színből, akkor 3. vagy 5. legnagyobb
  - ha előző körben egy színből ütést szereztünk, akkor azt érdemes folytatni magasan
- Szín játéknál - felvevő specifikus
  - leaduzik, ha nem kell később lopnia
- Szan játéknál
  - leghosszabb szín
  - ha nincs figurakombináció, akkor alacsonyan
  - kidolgozandó színt folytatni

#### Automatikus lapkijátszás

- Döntés adott helyen, ha van az első helyen hívott színből
  - 2. helyen: ha látunk figurát, fedjük, egyébként kicsit
  - 3. helyen: legnagyobbat, hacsak nem a partner vagy az előző viszi
  - 4. helyen: ütni, ha lehet
- Ha nincs a színből és dobni kell

- rövid színből kell dobni, ha az nem a kidolgozás alatt álló szín

Az `AdvancedDeclarerStrategy` döntési algoritmus a lapkihívásra nagyságrenddel bonyolultabb a fent leírtakénál, ezért ez az algoritmus és az ezt támogató `Node` adatszerkezet a következő részben kerül részletes bemutatásra.

### 3.3.4. A haladó szintű automatizált felvevőjáték algoritmus

A követelményelemzés során megállapítottuk, hogy a bridzs játékban nagyon fontos az eddig összesen kijátszott lapok figyelése és megjegyzése. Ezeknek, valamint a kezünkben és az asztalon látott lapoknak az ismeretében tudjuk megtippelni, hogy az ellenfélnél *milyen elosztásban vannak a még nem látott lapok*. Továbbá azt is megjegyeztük, hogy a szakdolgozatban implementált felvevő kiválaszt egy színt, és abból próbál a lehető legtöbbet ütni (lásd: 3.1.1 Mi alapján játszik ki egy lapot egy emberi játékos).

Tehát a haladó felvevőjáték algoritmusnak a meccs során megfigyelt lapok alapján meg kell tippelnie, hogy milyen elosztásban vannak az ellenfél kezében a még nem látott lapok. Ezután, *mind a négy kezet "ismerve" az adott színből a felvevő le tudja generálni az összes lehetséges állást*, ami a négy kéz egy tetszőleges lap kijátszásából kialakulhat. Az adott állásban kijátszott lapokat ezután figyelmen kívül hagyva a lépés megismételhető az összes lehetséges állásra, így azokra is legenerálva az összes lehetséges következő állást. Ez az ötlet adja az automatizált felvevőjáték algoritmus alapját.

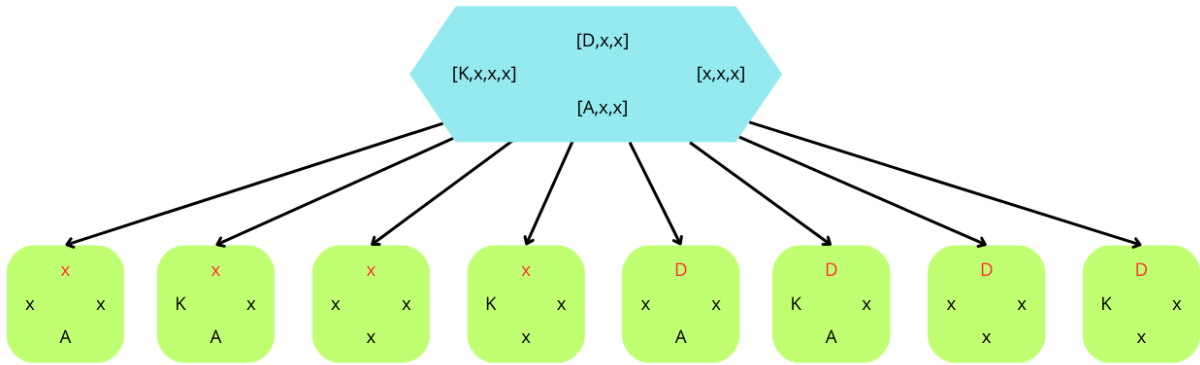
**Példa.** *Ellenfél kezeinek megtippelése.* Legyen a kiválasztott szín a pikk, amiből még egy lapot sem játszott ki senki korábban. Ha a felvevő kezében a következő lapok vannak pikk színből: D, 7, 4 és az asztalon a következő lapok vannak a színből: A, 6, 3, akkor tudjuk, hogy az ellenvonalon a következő lapok vannak: K, J, T, 9, 8, 5, 2. Tegyük fel, hogy ezek a lapok a következőféleképp oszlanak el az ellenfelek kezeiben: K, T, 8, 2 és J, 9, 5. Ezekből az adatokból szeretnénk képezni az összes releváns, lehetséges első kört. Ehhez elég, ha csak a figurákat és a tízest tüntetjük ki, és a kisebb lapokat "x"-el jelöljük, ahogy azt sok bridzs tankönyv is teszi.

Tehát a kezek: Észak: D,x,x. Dél: A,x,x. Kelet: J,x,x. Nyugat: K,T,x,x.

**Példa.** *Az első állások meghatározása.* Egy állás négy lapot tartalmaz, mindegyiket az adott égtájon ülő játékos játssza ki a kezében található lapok közül. Az összes első állás meghatározásához ki kell választanunk egy lapot az összes játékos kezéből és az összes

lehetőséget felírni (lásd 3.6 ábra). Ha északnak  $n$  különböző lapja van, délnek  $m$  különböző lapja van, keletnek  $l$  és nyugatnak  $k$  különböző lapja van az adott színből, akkor a lehetséges első körök száma:

$$n * m * l * k, \text{ ahol } n, m, l, k \in \mathbb{N}, 0 \leq n, m, l, k < 13). \quad (3.1)$$



3.6. ábra. A négy kéz ismeretében (kék hatszög) az összes releváns első állás meghatározása (zöld lekerekített négyzetek).

Minden állást összekötve azokkal a belőle következő állásokkal, amiket belőle generáltunk le, olyan gráfokat kapunk, aminek mindegyik csúcsához egy állás van rendelve. Az így kapott gráf minimálisan összefüggő és maximálisan körmentes, ezért egy fa. Annyi fát kapunk, ahány elsőkörös állásunk volt. Az elsőkörös állásoknak a fákban kitüntetett szerepe van, ez a fa gyökere. Ahhoz, hogy megállapítsuk melyik lapot kell hívni, vagyis melyik elsőkörös hívás vezethet a legkedvezőbb kimenethez, *meg kell vizsgálni a benne gyökerező fát, vagyis be kell járni azt*. Nevezzük ezeket a fákat **játékfáknak**.

Egy fa vizsgálata során szeretnénk mindegyik csúcsra kideríteni, hogy ha olyan hívássorozatot követünk, hogy oda jussunk, akkor hány ütést szerzünk a hívássorozat során. Ehhez mindegyik csúcsra meg kell állapítani, hogy ott a felvevő vagy az ellenvonal szerez-e ütést, vagy nem lehet megmondani, mert a reprezentációból nem derül ki egyértelműen, hogy ki tette az adott állásban a legnagyobb lapot. És ezeket az információkat összegezni. Ezt, az adott csúcsban az odavezető úton gyűjtött ütések várható értékét nevezzük egy csúcs **rátermettségi értékének**.



A rátermettségi függvény, amivel a rátermettségi értéket lehet meghatározni:

$$fitness(node) = \begin{cases} \text{ha a csúcshoz tartozó állást} & \\ \text{a felvevő vitte} & 1 + fitness(node.parent), \\ \text{ha a csúcshoz tartozó állásról} & \\ \text{nem lehet megállapítani, hogy ki vitte} & 0 + fitness(node.parent), \\ \text{ha a csúcshoz tartozó állást} & \\ \text{az ellenjátkos vitte} & -1 + fitness(node.parent). \end{cases}$$

, ahol a  $node.parent$  a  $node$  szülője és  $fitness(null) = 0$ .

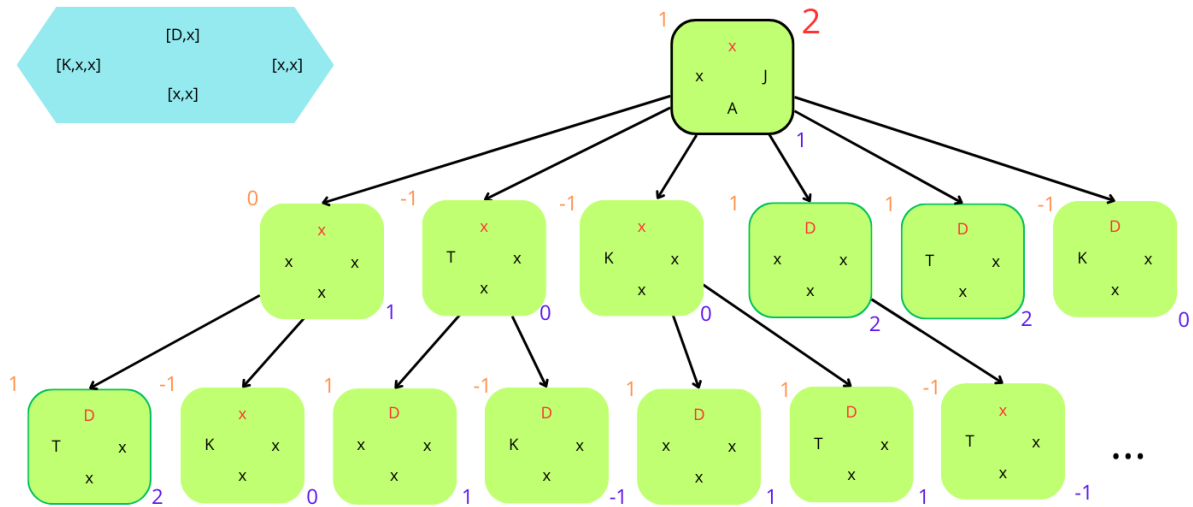
Szeretnénk az összes fában megtalálni azt a csúcst, aminek a legnagyobb a rátermettségi értéke, majd a fák közül kiválasztani azt, ahol az össze fa közül a legnagyobb rátermettségi értékkel rendelkező csúcs található. Így tudni fogjuk, hogy mi legyen a felvevő első hívása egy remélhetőleg maximális ütésszám eléréséhez. A legnagyobb rátermettségi értékű csúcsot a fa bejárásával lehet megtalálni, és ahhoz, hogy az összes fa között ezek az értékek könnyen összehasonlíthatóak legyenek, ezt az értéket fel kell terjeszteni a fa gyökerébe. Egy érték felterjesztése úgy működik, hogy egy szülő csúcsnak vannak olyan gyerekei, akiknek az értéke nagyobb, mint az övé, akkor a gyerekek közül a maximális értékű értéke felkerül a szülő csúcsba. Így a maximális érték a fa bármely csúcsából eljut a gyökérbe.

**Példa.** *Játékfa felépítése és a legnagyobb értékű csúcs meghatározása.* Az előző példát folytatva tegyük fel, hogy a négy kiszámolt kéz: Észak: D,x,x. Dél: A,x,x. Kelet: J,x,x. Nyugat: K,T,x,x. Vegyük az [x,J,A,x] állást, mint gyökeret.

A négy kezlet az állásban szereplő lapok kijátszása után is ismerjük, az eredeti négy kézből úgy kapjuk őket, hogy elvesszük belőlük ezeket a lapokat. Így Észak: D,x. Dél: A,x. Kelet: x,x. Nyugat: K,T,x. Ekkor az első állások meghatározásához hasonlóan meg tudjuk határozni a gyökeret követő lehetséges állásokat is, ezek lesznek a fa első szintjén: a második állások. Minden második állás után is ismerjük a négy kezlet, az aktuális kezekből úgy kapjuk meg ezeket, hogy elvesszük az adott második állásban kijátszott lapokat. Így meg tudjuk határozni a második állásokat követő harmadik állásokat is, amik a fa második szintjén lesznek. A példában ekkor elfogynak a felvevő lapjai, ezért a játékfa építés végére értünk (lásd: 3.7 ábra).

A játékfa építésével párhuzamosan az egyes csúcsok fitness értéke is meghatározható. A fitness érték két komponensből tevődik össze: hogy a csúcshoz tartozó állásnak mi a ki-

menetele, és, hogy a szülőnek mi a fitness értéke. A csúcsához tartozó állásról bármilyen plusz információ nélkül megállapítható, hogy a felvevő vitte-e, az ellenvonal vitte-e vagy mindenki "x"-et rakott, így nem eldönthető, hogy ki vitte. (Ezek az értékek narancssárgával vannak feltüntetve a 3.7 ábrán.) Ha a gyökércsúcsban eltároljuk annak fitness értékét, majd minden további csúcsban is eltároljuk az arra kiszámolt fitness értéket, akkor megkapjuk az összes csúcs fitness értékét. (Ezek az értékek lilával vannak jelölve a 3.7 ábrán.) A 3.7 ábrán a gyökércsúcs mellett pirossal van feltüntetve a játékfában található csúcsok közül a legnagyobb rátermettségi értékű értéke.



3.7. ábra. Az Észak: D,x; Dél: A,x; Kelet: x,x; Nyugat: K,T,x; kezekhez és a [x,J,A,x] állású gyökérhez tartozó játékfá. Az egyek állásokig tartó úton a várható ütés értéke a jobb alsó sarkukban (lila).

### A Node osztály és a haladó felvevőjáték algoritmus

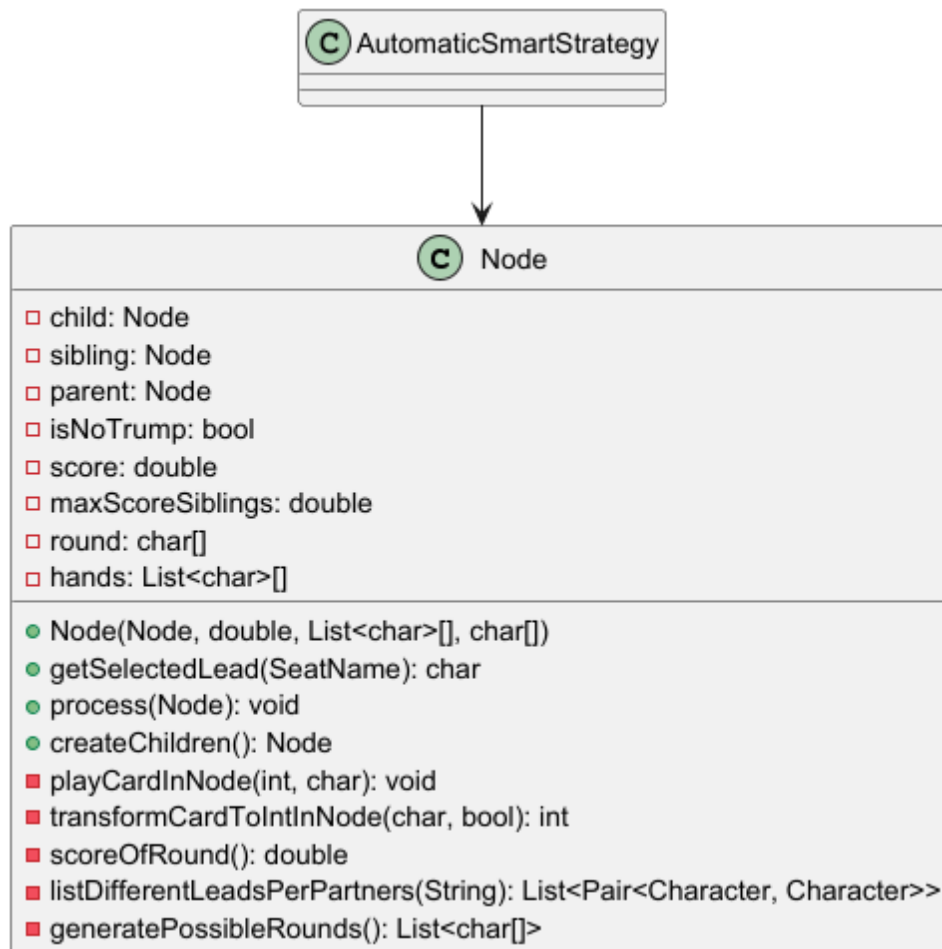
Az előző részt összegezve megkapjuk a haladó automatizált felvevőjáték algoritmus vázlatát:

1. Az ellenfél kezeinek **legenerálása**, hogy "ismerjük" mind a négy kezét.
2. Mind a négy kéz ismeretében **legeneráljuk a kiinduló köröket**, és a bennük **gyökerező játékfákat**.
3. Az összes gyökérre egy **fabejárással megállapítjuk az adott gyökér értékét**.
4. A gyökerek közül maximum kiválasztással **kiválasztjuk azt, amelyiknek a legnagyobb a rátermettségi függvény értéke**.

Ahhoz, hogy a vázlatból egy megvalósítható algoritmus tervet tudjunk készíteni szükség-

günk van egy a *játékfát reprezentáló adatszerkezet*. Az algoritmus vázlat alapján látszik, hogy ennek az adatszerkezetnek képesnek kell lennie állásokat eltárolni egy-egy csúcsban, továbbá az állások rátermettségi értékét is tárolnia kell. Emellett fontos, hogy létre lehessen vele hozni és be is lehessen járni vele egy fa struktúrát. Mindemellett, memória-takarékossági megfontolásból legyen minél egyszerűbb.

Ezek alapján az *általános fa bináris láncolt reprezentációjában szereplő Node adatszerkezet* [4] kiegészítve néhány adattaggal ideális erre a feladatra.



3.8. ábra. A Node osztály.

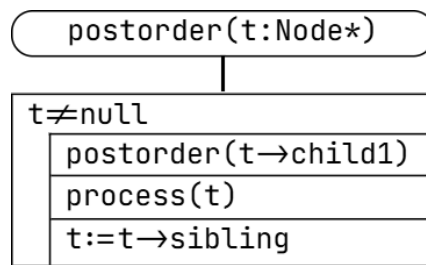
A Node osztály adattagjai:

- Az általános fa adatszerkezetében is meglévő adattagok, amik lehetővé teszik a *fa-bejárást* is:
  - `child`: az első gyerek, mint `Node`, `null`, ha nincs a csúcsnak gyereke.

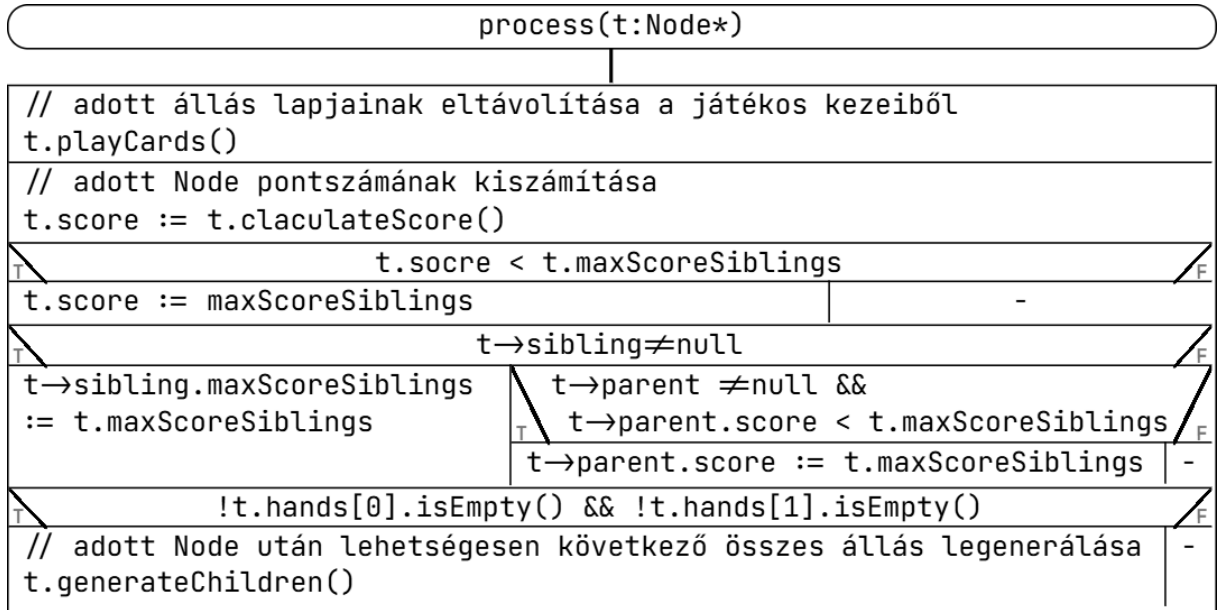
- **sibling**: a csúcs jobb oldali testvére, mint `Node`, *null*, ha nincs a csúcsnak jobb oldali testvére.
- **parent**: a csúcs szülője, mint `Node`, *null*, ha nincs a csúcsnak szülője.
- Speciális, a *felvevő lapválasztási algoritmusát lehetővé tevő adattagok*:
  - **isNoTrump**: logikai érték, igaz, ha szanzadu játék van.
  - **score**: a csúcs fitness értéke.
  - **round**: a csúcsához tartozó állás lapjai egy karaktertömbként reprezentálva.
  - **hands**: a csúcsához tartozó négy kéznek a lapjai, amiknek segítségével a csúcsot követő állásokat lehet legenerálni, egy négy hosszú tömbként reprezentálva, ahol a tömb minden eleme egy karakterlista.

A `Node` osztály metódusai a játékfa legenerálást, és a játékfa kiértékelést segítik:

1. A publikus `createChildren` metódus generálja le egy csúcsnak a gyerekeit, megfelelően inicializálva a pontszámukat és a hozzájuk tartozó állást. A gyerekcsúcsokhoz tartozó állásokat a privát `generatePossibleRounds` metódus hívásával kapja.
2. A publikus `process` metódus dolgoz fel egy csúcsot azzal, hogy végrehajtja rá a lapkijátszást, ezzel aktualizálva a **hands** adattagot, majd kiszámítja a csúcs fitness értékét és a testvérei fitness értékének maximumát továbbterjeszti a testvérének, ha van, különben felterjeszti a szülőbe. Végül legenerálja a lehetséges gyerekeit a `createChildren` metódust hívva.
3. A publikus `getSelectedLead` metódus megadja az adott csúcsra, hogy a paraméterben meghatározott égtájon mi a felvevő hívása.



3.9. ábra. Egy általános fa postorder bejárásának algoritmusá [4].



3.10. ábra. A játékfa egy csúcsának feldolgozása a postorder bejárás során, a Node osztály process metódusa.

A haladó automatizált felvevőjáték algoritmus az `AdvancedDeclarerStrategy` osztály `selectLead` metódusában került megvalósításra. Először választ egy szint, aztán az algoritmus vázlat szerint először legenerálja az ellenfél kezeit a `generateopponenthands` metódust meghívva. Ezután legenerálja az elsőkörös állásokat, majd a postorder fabejárást alkalmazva bejárja a játékfát. Végül egy maximumkiválasztással megkeresi a legnagyobb fitness értéket tartalmazó fát, és annak elsőkörös állására meghívva a `Node` osztály `getSelectedLead` metódusát megkapja, hogy mit érdemes hívnia a lehető legtöbb ütés megszerzésének reményében.

### 3.3.5. Felhasználói felület

A követelményelemzésnek megfelelően a felhasználó felületnek az alábbi oldalakkal kell rendelkeznie:

- *Főoldal*: ahol az "instant játék" és a paraméterek alapján indított játék található.
- *Új játék*: ahol a felhasználó új, alapértelmezett, vagy paraméteres játékot tud indítani.
- *Regisztráció*: ahol a felhasználó egy felhasználónevet és jelszót megadva regisztrálni tud.
- *Bejelentkezés*: ahol a felhasználó be tud lépni már létező profiljának felhasználónevét és jelszavát megadva.

- *Eredmények*: ahol a bejelentkezett felhasználó egy táblázatban láthatja a játékban elért eredményeit.
- *Súgó*: a bridzs szabályait és az alkalmazás működését ismertető oldal.

#### Menü és navigálás

A menünek könnyű hozzáférést kell biztosítania a felsorolt oldalakhoz, ezért az alábbi pontokat kell tartalmaznia:

- Mindig látható pontok: Főoldal, Új játék, Súgó
- Csak nem bejelentkezett felhasználó számára látható pontok: Regisztráció, Bejelentkezés
- Csak bejelentkezett felhasználó számára látható pontok: Eredmények, Kijelentkezés

A menü az összes oldal tetején megjelenik és segítségével bármelyik oldalról át lehet navigálni bármelyikre.

#### Képernyőtervek és fontosabb események

A **főoldalon** az alábbi elemeknek kell megjelenennie:

- A menünek és a főoldal címének.
- Az aktuális meccs adatainak: felvétel és a vonalak megszerzett ütéseinek a száma.
- A játékosok kártyáinak, a játékmódnak megfelelő láthatósággal.
- Elemző játékmód esetén három gombnak, ami nyugat, dél és kelet kártyáinak láthatóságát állítja.
- A játéktábla közepén az éppen aktuálisan kijátszott lapoknak.

A fontosabb események, a menüben való navigálást leszámítva:

- A soron következő ellenjátékos egy szabályosan kijátszható lapjára kattintva, az a lap eltűnik az ellenjátékos kezéből és megjelenik az aktuális állás megfelelő helyén.
- Elemző játékmód esetén, ha a felhasználó az "Észak kártyáinak mutatása" gombra kattint, észak kártyái láthatóvá válnak. (Nyugat és kelet lapjaira hasonlóan.)
- Elemző játékmód esetén, ha a felhasználó az "Észak kártyáinak elrejtésre" gombra kattint, észak kártyái helyett csak fehér téglalapok jelennek meg. (Nyugat és kelet lapjaira hasonlóan.)

Az **új játék oldalon** az alábbi elemeknek kell megjelennie:

- A menünek és az oldal címének.
- Központi helyen, nagy méretben megjelenő "játék indítása" gomb.
- Központi helyen, a "játék indítása" gombnál kisebb méretű, "paraméterek megadása" gomb.
- Rövid magyarázat az alapértelmezett játékindításról, vagy a paraméterekről aszerint, hogy a "paraméterek megadása gomb" ki lett-e választva.
- Ha a "paraméterek megadása" gomb ki lett választva: meg kell jelennie a játékmódot, felvevő szintjét és a szan- vagy színfelvétel kiválasztását biztosító formnak.

A fontosabb események, a menüben való navigálást leszámítva:

- Ha paraméterek megadása nélkül a felhasználó a "játék indítása" gombra kattint, átirányítják a főoldalra, és ott elkezdődik egy játék az alapértelmezett paraméterekkel.
- Paraméterek megadása után, ha a felhasználó a "játék indítása" gombra kattint, átirányítják a főoldalra, és ott elkezdődik egy játék a megadott paraméterekkel.

Az **regisztráció oldalon** az alábbi elemeknek kell megjelennie:

- A menünek és az oldal címének.
- Központi helyen, jól olvasható betűmérettel egy formnak, amin felhasználónév, e-mail cím és jelszó megadására van beviteli mező.
- A beviteli mezők alatt magyarázat a kötelező mezőkről és a rájuk tett kikötésekről.
- A regisztráció sikerességét vagy sikertelenségét visszaigazoló felugró üzenet.

A fontosabb események, a menüben való navigálást leszámítva:

- Ha a felhasználónév vagy a jelszó helytelen megadása után a felhasználó a "regisztráció" gombra kattint, megjelenik egy sikert visszaigazoló üzenet és a felhasználó átirányításra kerül a bejelentkezés oldalára.
- Ha a felhasználónév és a jelszó helyes megadása után a felhasználó a "regisztráció" gombra kattint, megjelenik egy sikert visszaigazoló üzenet és a felhasználó átirányításra kerül a bejelentkezés oldalára.

Az **bejelentkezés oldalon** az alábbi elemeknek kell megjelennie:

- A menünek és az oldal címének.
- Központi helyen, jól olvasható betűmérettel egy formnak, amin felhasználónév, és jelszó megadására van beviteli mező.
- A bejelentkezés sikerességét vagy sikertelenségét visszaigazoló felugró üzenet.

A fontosabb események, a menüben való navigálást leszámítva:

- Ha a felhasználónév vagy a jelszó helytelen megadása után a felhasználó a "bejelentkezés" gombra kattint, megjelenik egy sikert visszaigazoló üzenet és a felhasználó átirányításra kerül a bejelentkezés oldalára.
- Ha a felhasználónév és a jelszó helyes megadása után a felhasználó a "bejelentkezés" gombra kattint, megjelenik egy sikert visszaigazoló üzenet és a felhasználó átirányításra kerül a főoldalra.

Az **eredmények oldalon** az alábbi elemeknek kell megjelenennie:

- A menünek és az oldal címének.
- Egy rövid magyarázatnak, hogy milyen eredményeket mutat be az oldal.
- Egy táblázat, ami jól olvasható betűmérettel összefoglalja az eredményeket, magába foglalva a leosztás számát, a felvételt, a játék eredményét, a felvevő ütéskülönbségét, az ellenjáték módját, a felvevő szintjét és az utolsó lejátékozás végét.
- A táblázat utolsó oszlopában gomboknak, amikkel az adott sorhoz tartozó leosztás játéka indítható.
- Egy jelmagyarázatnak a táblázatban található ikonokhoz.

A fontosabb események, a menüben való navigálást leszámítva:

- Ha a felhasználó a táblázat utolsó sorában található gombra kattint, átirányításra kerül a főoldalra, és elindul a gombhoz tartozó sorban meghatározott paraméterekkel a játék.

A **súgó oldalon** az alábbi elemeknek kell megjelenennie:

- A menünek és az oldal címének.
- Egy bekezdésnek jól olvasható betű mérettel, ami a bridzs szabályaihoz ad útmutatást.
- Egy bekezdésnek jól olvasható betű mérettel, ami a bridzs szabályaihoz ad útmutatást.



### 3.3.6. API terv

A kliens HTTP kérésekkel, az alábbi táblázatban összefoglalt végpontokat használva kommunikál a középső réteg megjelenítési szintjével. Az API határozza meg a kérésekhez a válasz struktúráját.

Oldal	Metódus	Végpont	Elérhetőség
Főoldal	GET	game/start/params	mindenki
	GET	game/play/card	mindenki
	GET	game/play/automatic	mindenki
	GET	game/over	mindenki
	GET	game/over/auth	csak bejelentkezett felhasználó
Regisztráció	POST	/api/users	mindenki
Eredmények	GET	/api/results	csak bejelentkezett felhasználó
Bejelentkezés	GET	/api/token	mindenki
	GET	/api/token/validate	csak bejelentkezett felhasználó

## 3.4. Megvalósítás

### 3.4.1. Felhasznált eszközök

#### IntelliJ

A JETBARINS IntelliJ IDEA praktikus fejlesztői környezet Spring framework alkalmazások fejlesztésére, többek között olyan *webes Spring Boot alkalmazások fejlesztésére*, mint amilyen a szakdolgozat alkalmazás is. A beépített fejlesztői eszközökkel könnyű futtatni és teszteket készíteni és futtatni az alkalmazáshoz. A kód megírását gyorsítják és megkönnyítik a kód kiegészítő és a kód generáló (pl. osztály getter és setter metódusai) funkciók. Az IntelliJ IDEA letölthető az [itt linkelt oldalról](#). Én a fejlesztéshez az IntelliJ IDEA Community Editiont használtam. Az IntelliJ IDEA dokumentációja a Spring Boothoz elérhető [erre a linkre kattintva](#).

## Visual Studio Code

A Visual Studio Code egy elterjedt fejlesztői környezet, ami integrált terminállal, webes támogatással és könnyű kód futtatási lehetőséggel sok nyelvet és könyvtárat, köztük a szakdolgozat frontendjéhez használt *TypeScriptet és a React könyvtárat is támogatja*. A Visual Studio Code szövegszerkesztője támogatja a könnyű kódnavigációt és a React.js IntelliSense-et, ami lehetővé teszi az automatikus kódkiegészítést. A Visual Studio Code ingyenesen letölthető az [itt linkelt oldalról](#). [Erre a linkre kattintva](#) pedig elérhető a Visual Studio Code React tutorialja.

## MySQL Workbench

A MySQL Workbench egy hasznos eszköz vizuális adatmodellezésre és átfogó adminisztrációs eszköz szerver konfigurációra. A szakdolgozat alkalmazás *adatbázisának kezelését* végeztem ezen a platformon, kihasználva a MySQL szerver kapcsolat létrehozó varázsló (connection wizard) és a vizuális adat szerkesztő (leosztások adatainak bevitele) funkcióit. A MySQL Workbench letölthető az [itt linkelt oldalról](#). Én a fejlesztéshez a nyílt forráskódú és ingyenesen letölthető MySQL Workbench 8.0 Community Editiont használtam. [Ezen az oldalon](#) lehet részletesebben olvasni a MySQL Workbench funkcióiról.

## Postman

Postman platform olyan átfogó eszközöket foglal magába, amik felgyorsítják az APIk fejlesztését. Többek között segítik az API tervezést, tesztelést, dokumentációt. A szakdolgozat alkalmazás fontos alkotóeleme a 3.3.6 szekcióban bemutatott *API, aminek a tesztelésére kiválóan alkalmas a Postman*. A backend API endpointjain történő függvényhívásokat érdemes először Postmannel tesztelni, hogy kizárhassuk a frontend hibáit a fejlesztés korai szakaszában. A Postman API kliense az autentikáció tesztelésére is hasznos eszköz. A Postman ingyenesen letölthető az [itt linkelt oldalról](#). Az [itt linkelt oldalon](#) pedig a Postman API fejlesztő eszközeiről lehet bővebben olvasni.

## Webes böngésző

Az alkalmazás fejlesztése során egy webes böngészőben ellenőriztem a grafikus felhatalmított felület megjelenését és a felhasználó interakciók hatását. Egyes *változók értékeinek követésére, hibakeresésre és a hálózati kommunikáció ellenőrzésére* hasznos a modern bö-

gészőkbbe épített Developer Tools, ami megtalálható a Microsoft Edge, Google Chrome és a Firefox böngészőkben is.

### 3.4.2. Felhasznált technológiák

A szakdolgozat alkalmazás megvalósításához olyan platformokat, keretrendszereket és könyvtárakat választottam, amikkel hatékonyan lehet implementálni a 3.3.1 szekcióban vázolt architektúra tervet és emellett széles körben elterjedtek, így megbízhatóak és használatuk jól dokumentált.



3.11. ábra. Az alkalmazás architektúráját megvalósító technológiák.

(A 3.11. ábrán látható logók forrásai: [React](#), [Spring Boot](#), [MySQL](#).)

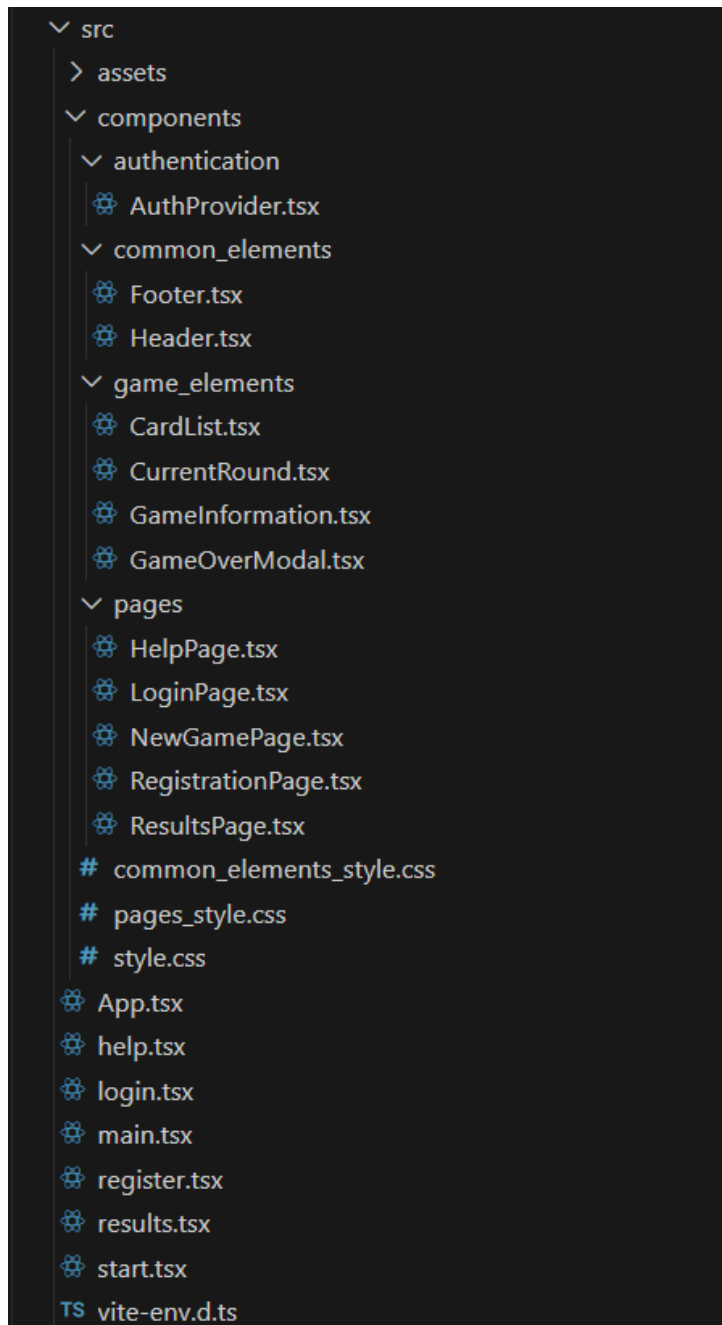
#### React

Az alkalmazás felhasználói felülete HTML oldalakból áll. Az oldalak interaktivitását TypeScript nyelvvel, React könyvtárat használva biztosítottam. Az oldalak formázásához Bootstrapet, Tailwind CSS formázást és saját css fájlokat használtam. A TypeScript egy statikusan típusozott nyelv, ami Javascriptre fordul. nagyban hasonlít a Javascriptre, de itt vannak típusok, amik biztonságosabbá teszik a fejlesztést. A React egy UI könyvtár, ami lehetővé teszi, hogy JavaScript logikával lehessen HTML nézeteket módosítani.

A React alkalmazások általában egyetlen oldalból állnak, míg a szakdolgozat alkalmazás frontendjén mindegyik oldalhoz tartozik egy fő React komponens. Azért ezt a megvalósítást választottam, mert ezzel elkerülhető a bonyolult beállítások és könyvtárak használata, mint például a React router használata.

Az *src* mappa gyökerében találhatóak a fő React komponensek, amik az egyes HTML oldalakhoz vannak rendelve és így összekötik azokat a többi React komponenssel, amik a *components* mappában találhatóak. Ezek négy almappába vannak csoportosítva a funkciójuk szerint (lásd 3.12. ábra):

- *authentication*: a bejelentkezésért felelős React komponens helye.
- *common\_elements*: a fejléc komponens, benne a menüvel és lábléc komponens, amik minden oldalon megjelennek.
- *pages*: az egyes oldalak alkotó komponensei, a főoldal kivételével.
- *game\_elements*: a főoldalon látható elemek komponensei: a játékosok kártyái, az éppen aktuális kör, a játék információ és a játék vége modul.

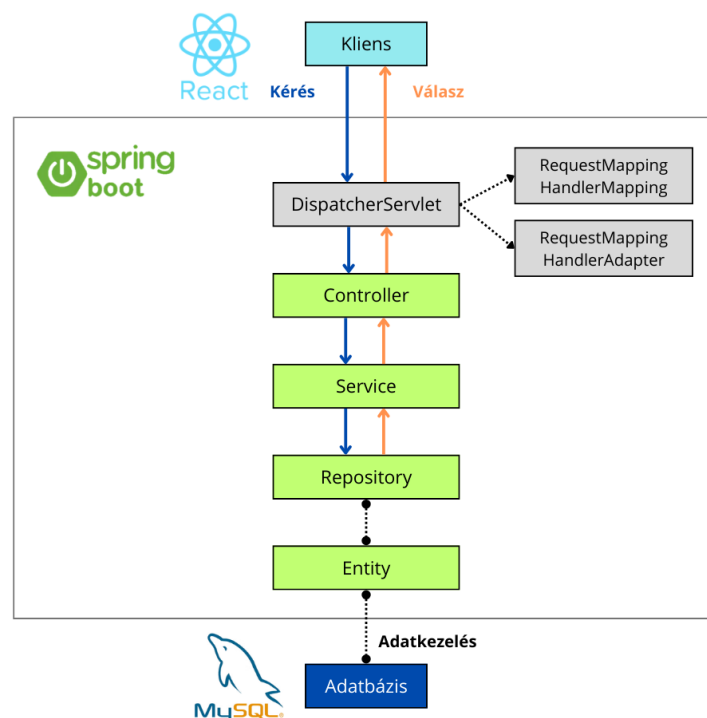


3.12. ábra. A felhasználói felület TypeScript forrásfájljai.

## Spring Boot

Az alkalmazás backendjét Spring Boot keretrendszerben valósítottam meg. "A Spring Framework egy Java platform, ami átfogó programozási és konfigurációs modellként szolgál modern Java alapú alkalmazások számára. A Spring kezeli a szoftver infrastruktúráját, így a programozónak csak a saját alkalmazásának különlegességeit kell megvalósítania. A Spring funkciói modulokba vannak rendezve. A funkciók többek között: Inversion of Control (IoC) container dependency injectionnel, adatelérés, tesztelés, webes keretrendszer. A Spring Boot a Spring Framework továbbfejlesztésének tekinthető. A Spring Framework moduljait használja és előnye, hogy gyorsan lehet benne modulokat konfigurálni. Meghatározó tulajdonsága, hogy beágyazott web szerveret tartalmaz, amikor csak lehetséges automatikusan konfigurálja a Spring és 3rd party könyvtárakat, és annotációk vannak benne" [1].

A rétegek közti kommunikáció és a backend architektúrájának szintjei a 3.13 ábrán láthatóak. A következő szekciókban a kommunikáció módja és az egyes szintek kerülnek részletesen bemutatásra.



3.13. ábra. Kommunikáció a rétegek között és a Spring Bootban megvalósított középső réteg szintjei.

## Megjelenítési szint

A kontroller osztályok a középső réteg megjelenítési szintjén (presentation layer) helyezkednek el, és a Spring Bootban `@Controller` annotáció jelöli őket. "A Spring támogatja a RESTnek<sup>1</sup> megfelelő (RESTful) webes alkalmazások létrehozását. A RESTful kontrollerek *objektumokat inicializálnak és visszaadnak HTTP-n keresztül, JSON-ként szerializálva*. Ezzel szemben a Spring MVC kontrollerek nézetekkel térnek vissza. A szakdolgozat alkalmazásban RESTful kontrollereket használtam, amiket `@RestController` annotáció jelöl az osztályban. Ez az annotáció magába foglalja a `@Controller` és a `@ResponseBody` annotációkat.

"A kérések fogadásához és megválaszolásához egy kontrollernek meg kell valósítania a kérésleképezést (Request Mapping) és a kérések kezelését (Handler Methods). Az előbbiért a metódusok fölé írt `@RequestMapping` annotáció felel, aminek attribútumai határozzák meg, hogy milyen URLhez, HTTP metódushoz, kérés paraméterekhez és headerhöz tartozik. A handler metódusoknak rugalmas szignatúrája van, többféle argumentumuk és visszatérési értékük is lehet" [5].

A 3.1 kódrészlet a GameController két handler metódusát és azok jellegzetes annotációt szemlélteti. Az annotációk magyarázata:

- `@RestController`: az osztály metódusai által visszaadott értékek egyenesen a response bodyba kerülnek
- `@CrossOrigin`: meghatározza honnan fogad kéréseket
- `@RequestMapping`: HTTP kéréseket rendel a kontroller handler metódusaihoz
- `@GetMapping`: egy HTTP GET metódust rendel az egyik handler methodhoz
- `@RequestParam`: a kérés paraméterek
- `ResponseEntity`: egy HTTP választ reprezentál, beleértve a headert, bodyt és státusz kódot

```
1 // ... a szükséges függőségek importálása
2 @RestController
3 @CrossOrigin(origins = "*")
4 @RequestMapping("/game")
5 public class GameController {
```

---

<sup>1</sup>A reprezentációs állapot átvitel (Representational State Transfer, REST) egy szoftver architektúrai stílus osztott hipermédia rendszerek számára, mint amilyen a világháló is. Architektúrai megszorításokat tartalmaz, mint például, a komponens interakciók skálázhatóságát, az interfészek általánosságát és a komponensek önálló deployolhatóságát" [5].

```
6
7     @Autowired
8     private MainService mainService;
9
10    @GetMapping("/start/params")
11    public ResponseEntity<?> loadGameWithParams(@RequestParam String
        level, @RequestParam String mode, @RequestParam String trumpType,
        @RequestParam int id){
12        return new ResponseEntity<Match>(mainService.
            loadParameterizedMatch(level, mode, trumpType, id),
            HttpStatus.OK);
13    }
14
15    @GetMapping("/gameOver")
16    public ResponseEntity<?> readResult(){
17        return new ResponseEntity<Pair<MatchResult, Integer>>(
            mainService.manageGameOverNotAuthenticated(), HttpStatus.OK);
18    }
19
20    // ... mas handler metodusok
21 }
```

3.1. forráskód. Kódrészlet a GameController osztályból.

## Üzleti logika

Az üzleti logika osztályaiból már sok bemutatásra került a 3.3 szekcióban. Az ott bemutatott osztályok a bridgez játék modellezését valósítják meg. Emellett az üzleti logikának vannak olyan, `@Service` annotációval jelölt osztályai is, amik közvetítésre szolgálnak a kontrollerek és a repositoryk között (vagyis a perzisztencia szint és a megjelenítési szint között). A `@Service` annotációnak köszönhetően ezek az osztályok könnyen felhasználhatóak a controller osztályokban, ugyanis az annotáció miatt automatikusan példányosodnak és a `@Autowired` kulcsszót használva könnyen be lehet fecskendezni őket a controller osztályba. Erre szolgál példaként a 3.1 kódrészlet 7. és 8. sora. A következő szekcióban a perzisztencia szint osztályainál használt `@Repository` annotáció hasonlóan lehetővé teszi, hogy ezeket a `@Service` osztályok a `@Autowired` annotációval használni tudják.

"A **függőség befecskendezés** (Dependency Injection) kezelése a komponensekre vonatkozóan annotációkkal a Spring Boot egyik előnye. A dependency injection egy tervezési

minta és elkülöníti a befecskendezett objektum létrehozását és használatát" [1].

```
1 // ... a szukseges fuggosegek importalasa
2 @Service
3 public class StatisticsService {
4
5     @Autowired
6     private StatisticsRepository statisticsRepository;
7
8     public Statistics saveStatistics(Statistics newStatistics) {
9         return statisticsRepository.save(newStatistics);
10    }
11
12    public Iterable<Statistics> findAllStatistics() {
13        return statisticsRepository.findAll();
14    }
15
16    public Statistics findByProperties(int gameId, String username,
17        GameMode mode, DeclalerLevel level) {
18        return statisticsRepository.findByProperties(gameId, username,
19            mode, level).get(0);
20    }
21
22    // ... mas metodusok
23 }
```

3.2. forráskód. Kódrészlet a StatisticsService osztályból.

## Perzisztencia szint

A perzisztencia szint használata mögötti motiváció az, hogy az alkalmazás üzleti logikájának el kell érnie a perzisztensen tárolt adatokat, úgy, hogy az adatbázis tranzakciók ACID<sup>2</sup> tranzakciók legyenek és az üzleti logika független legyen az adat elérés formájától. Egy ilyen megoldás az *Object-Relational Mapping* (ORM) [1].

A Jakarta Persistence egy sztenderd API-t definiál, amivel relációs adatbázis sémában tárolt adatokat lehet kezelni Java környezetben. A JPA egy specifikáció, amit több keretrendszer, például az EclipseLink, Hibernate és OpenJPA is megvalósít. Ezek közül

---

<sup>2</sup>Atomicity, Consistency, Isolation, Durability



a keretrendszerek közül a *Hibernate*-et választottam, mert könnyen integrálható volt az alkalmazásomba.

A *perzisztens entitások* (Persistent Entities) lehetővé teszik, hogy a fejlesztő objektum orientáltan érje el az adatbázisban tárolt adatokat. Az entitások a JPA specifikációt ki-elégítő Java osztályok, amik `@Entity` annotációval vannak ellátva. Az entitás osztályok a relációs adatbázis egy-egy táblájához vannak rendelve. A tábla egyes sorai az entitás osztályok egyes példányai, amik addig léteznek, amíg az adatbázis létezik [6].

```
1 // ... a szükséges függőségek importálása
2 @Entity
3 @JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class
4     , property = "id")
5 public class Game {
6     @Id
7     @GeneratedValue(strategy = GenerationType.IDENTITY)
8     private int id;
9
10    @Size(min=2, max=3, message="A contract is between 2 and 3
11        characters long")
12    private String contract;
13
14    @OneToMany(mappedBy = "game", cascade = CascadeType.ALL)
15    private List<Hand> hands;
16
17    public Game(){}
18
19    public Game(String contract, List<Hand> hands) {
20        this.contract = contract;
21        this.hands = hands;
22    }
23
24    // ... getters and setters for each field
25 }
```

3.3. forráskód. Game entitás. Kódrészlet a Game.java fájlból.

A *Spring Data repositoryk* lehetővé teszik az entitások létrehozását, olvasását, módosítását és törlését. A repositoryk Java osztályok, amik `@Repository` annotációval vannak ellátva és lehet bennük *Java Persistence Query Language* (JPQL) lekérdezéseket írni. A JPQL

az SQLre hasonlít és egyszerű a szintakszisa, ezért választottam ezt más bonyolultabb lekérdezési formákkal (Criteria API, Quersdsl) szemben.

```
1 // ... a szükséges függősegek importálása
2 @Repository
3 public interface GameRepository extends JpaRepository<Game, Integer> {
4     @Query("SELECT g FROM Game g")
5     public List<Game> findAllGames();
6 }
```

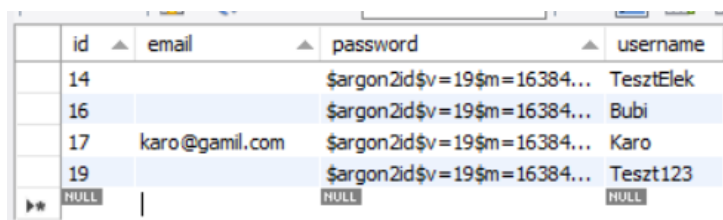
3.4. forráskód. Game repository és a repositoryban található JPQL lekérdezés.

Kódrészlet a GameRepository.java fájlból.

## MySQL

Az alkalmazás adatbázisának egy MySQL adatbázist választottam, mert a MySQL széles körben elterjedt és könnyű integrálni a Spring Boottal. A MySQL egy nyílt forráskódú relációs adatbázis kezelőrendszer.

A 3.3.2. szekcióban felvázolt adatbázissémát a Spring Boot hozza létre a perzisztencia szint (3.4.2. szekció) entitásai alapján. A **user táblába** a felhasználók adatai az alkalmazás használatával, a regisztrációval kerülnek be. Biztonsági megfontolásokból a felhasználók jelszavai nem egyszerű szöveggént, hanem a Spring Argon2PasswordEncoderével kódolva, titkosítva kerülnek be az adatbázisba (lásd: 3.14 ábra). Amikor egy felhasználó regisztrál, a backend legenerálja hozzá az alapértelmezett eredményeket is, ezek a **statistics táblába** kerülnek (lásd: 3.15 ábra).



id	email	password	username
14		\$argon2id\$v=19\$m=16384...	TesztElek
16		\$argon2id\$v=19\$m=16384...	Bubi
17	karo@gamil.com	\$argon2id\$v=19\$m=16384...	Karo
19		\$argon2id\$v=19\$m=16384...	Teszt123
	NULL	NULL	NULL

3.14. ábra. A user tábla.

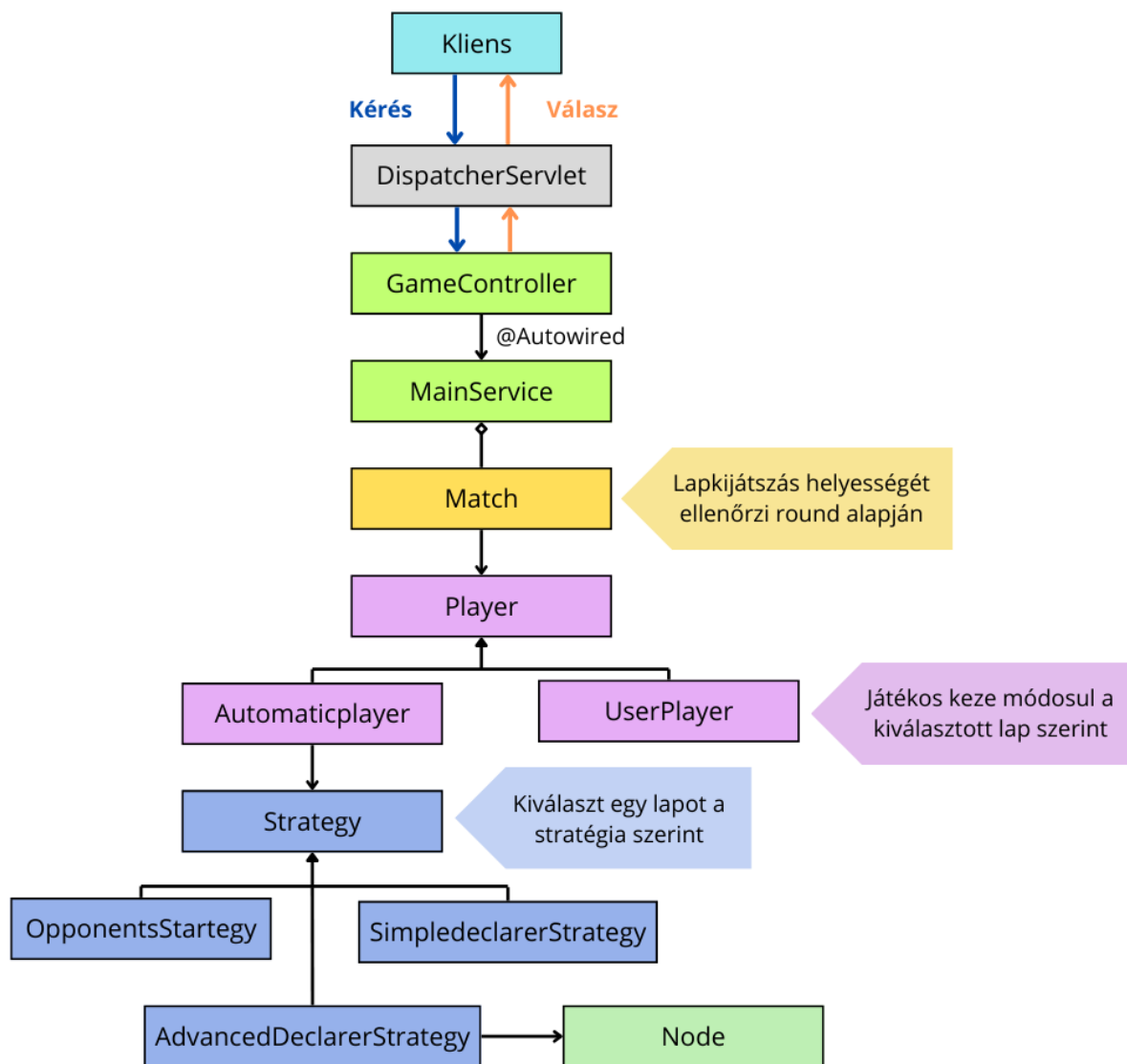
id	user_id	result	game_id	difference	ended_at	declarer_level	game_mode
61	18	LOST	2	1	04.16. 11:30	MEDIUM	DEFENSE
62	18	NO_DATA	2	0		BEGINNER	ANALYSIS
63	18	NO_DATA	2	0		MEDIUM	ANALYSIS
64	19	WON	1	-3	04.15. 15:30	BEGINNER	DEFENSE
65	19	NO_DATA	1	0		MEDIUM	DEFENSE
66	19	LOST	1	1	04.16. 14:01	BEGINNER	ANALYSIS
67	19	LOST	1	0	04.16. 14:22	MEDIUM	ANALYSIS
68	19	NO_DATA	2	0		BEGINNER	DEFENSE
69	19	WON	2	-1	04.17. 8:50	MEDIUM	DEFENSE
70	19	NO_DATA	2	0		BEGINNER	ANALYSIS
71	19	WON	2	-2	04.17. 8:30	MEDIUM	ANALYSIS
72	19	NO_DATA	3	0		BEGINNER	DEFENSE
73	19	NO_DATA	3	0		MEDIUM	DEFENSE
74	19	NO_DATA	3	0		BEGINNER	ANALYSIS
75	19	LOST	3	2	04.11. 17:32	MEDIUM	ANALYSIS
76	19	LOST	4	0	04.09. 11:43	BEGINNER	DEFENSE
77	19	NO_DATA	4	0		MEDIUM	DEFENSE
78	19	NO_DATA	4	0		BEGINNER	ANALYSIS
79	19	WON	4	-3	04.17. 8:41	MEDIUM	ANALYSIS
80	20	NO_DATA	1	0		BEGINNER	DEFENSE

3.15. ábra. Részlet a statistics táblából.

A **game** és a **suit táblák** adatait "kézzel", SQL utasításokkal szűrtam be. Olyan leosztásokat rögzítettem, amik felvevőjáték demonstrálásra alkalmasak, vagy ellenjáték feladványt jelentenek. A példákat a [Bridge Base Online](#) felületről és Hugh W. Kelsey *Gyilkos ellenjáték* könyvéből válogattam.

### 3.4.3. Lapkijátszás

A 3.3 Tervezés szekcióban bemutatásra kerültek az üzleti logika azon osztályai, amik a felhasználó lakijátszásáért és az automatikus felvevő algoritmus lapkijátszásában közreműködnek. Ezek az osztályok: **Match**, **Player** és leszármazott osztályai, **Strategy** és leszármazottai, valamint a **Node** osztály. Ezek mellett a lapkijátszás megvalósításában más osztályok is részt vesznek, és ezek egy láncolatot alkotnak. Mindegyik osztálynak megvan a maga felelőssége a lapkijátszás során, amit a 3.16-es kép illusztrál. Az osztályok emellett más funkciókat is betöltenek és a középső réteg különböző szintjein helyezkednek el. A bemutatott folyamat a **Chain of Responsibility** design pattern egy példája.



3.16. ábra. Lapkijátszásért felelős osztályok láncolata.

Az alábbiakban a lapkijátszás folyamata kerül ismertetésre a kienstől kezdve a középső réteg szintjein keresztül. A felhasználó hívását egy kattintás váltja ki, az automatikus játékos hívása pedig a felhasználó hívása, vagy egy másik automatikus játékos hívása után következik.

A `game/playCard` és a `game/playAutomatic` endpointok `GET` módszerát a `GameController` kontrollor fogadja. Ekkor meghívja a `MainService` osztály `updateUserHand`, illetve `updateAutomaticHand` módszerát. Mindig egy "OK" HTTP státusszal tér vissza, és a `mainService match` objektumával, ami változatlan marad, ha nem szabályos a lapkijátszási kísérlet.

```
1 // ... a szükséges függősegek importálása
```

```
2 @RestController
3 @CrossOrigin(origins = "*")
4 @RequestMapping("/game")
5 public class GameController {
6
7     @Autowired
8     private MainService mainService;
9
10    //... mas metodusok
11
12    @GetMapping("/playCard")
13    public ResponseEntity<?> updateUserHand(@RequestParam String seat,
14        @RequestParam String suit, @RequestParam String card,
15        @RequestParam int ind){
16        return new ResponseEntity<Match>(mainService.updateUserHand(seat
17            , suit, card.charAt(0), ind), HttpStatus.OK);
18    }
19
20    @GetMapping("/playAutomatic")
21    public ResponseEntity<?> updateAutomaticHand(@RequestParam String
22        seat){
23        return new ResponseEntity<Match>(mainService.updateAutomaticHand
24            (seat), HttpStatus.OK);
25    }
26 }
```

3.5. forráskód. A GameController kontroller lapkijátszásért felelős kódrészlete.

A MainService osztály ellenőrzi, hogy a seat paraméter helyesen lett-e átadva, és ha a match adattagja nem null, meghívja rá annak playUserCard illetve playAutomaticCard metódusát. Ezek a metódusok módosítják a match objektum állapotát, ha a lapkijátszás érvényes, így a MainService-beli metódusokat hívó kontrollerhez, majd végső soron a klienshez egy frissített match objektum jut el.

```
1 // ... a szukseges függosegek importalasa
2 @Service
3 public class MainService {
4
5     private Match match;
6
7     //... mas adattagok es metodusok
```

```
8
9 public Match updateUserHand(String seat, String suit, char card, int
10     ind) {
11     if(Objects.equals(seat, SeatName.EAST.name()) || Objects.equals(
12         seat, SeatName.WEST.name())) {
13         if (match != null){
14             match.playUserCard(SeatName.valueOf(seat), suit, card,
15                 ind);
16         }
17     }
18     return null;
19 }
20
21 public Match updateAutomaticHand(String seat) {
22     if(Objects.equals(seat, SeatName.NORTH.name()) || Objects.equals
23         (seat, SeatName.SOUTH.name()) || Objects.equals(seat,
24             SeatName.WEST.name())) {
25         if (match != null){
26             match.playAutomaticCard(SeatName.valueOf(seat));
27         }
28     }
29     return match;
30 }
31 }
```

3.6. forráskód. A MainService osztály lapkijátszásért felelős kódrészlete.

A Match osztály playUserCard és playAutomaticCard metódusa és segédmetódusaik már ismertek a tervezés szekcióból (lásd: 3.3.3 A Match osztály).

```
1 // ... a szukseges fuggosegek importalasa
2 public class Match {
3
4     private Map<SeatName, Player> players = new HashMap<>();
5     private Round round = new Round();
6
7     //... mas adattagok es metodusok
8
9     public void playUserCard(SeatName seat, String suitname, char card,
10         int ind) {
11         String state = isPlayValidForCurrentRound(seat, suitname, card);
```

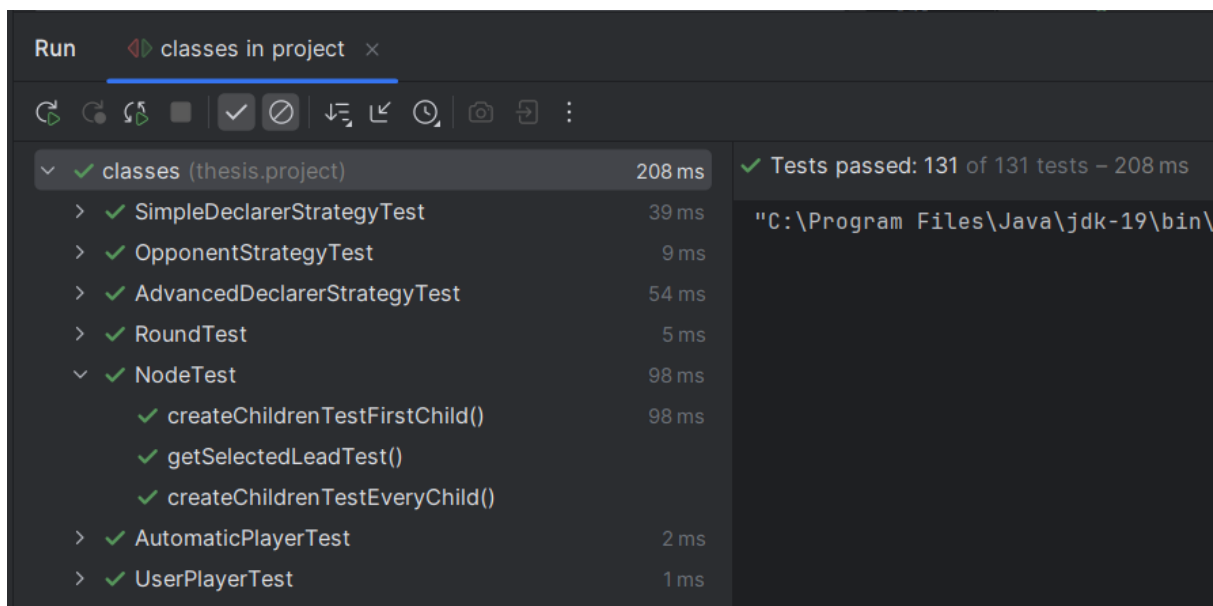
```
11         if(!Objects.equals(state, "Invalid")){
12             UserPlayer player = (UserPlayer) players.get(seat);
13             player.playCard(SuitName.valueOf(suitname), ind);
14             manageEffectOfCall(state);
15         }
16     }
17
18     public void playAutomaticCard(SeatName seat) {
19         AutomaticPlayer player = (AutomaticPlayer) players.get(seat);
20         if(round.getPositions().get(seat) == round.getCurrentPosition())
21         {
22             Pair<String, Character> cardToPlay = player.playCard(
23                 getTrump(), this.round);
24             String suit = cardToPlay.getFirst();
25             char card = cardToPlay.getSecond();
26             String state = isPlayValidForCurrentRound(seat, suit, card);
27             if (!Objects.equals(state, "Invalid")) {
28                 manageEffectOfCall(state);
29             }
30         }
31     }
32 }
```

3.7. forráskód. A Match osztály lapkijátszásért felelős kódrészlete.

## 3.5. Tesztelés

A tesztelés során meggyőződtem róla, hogy az alkalmazás úgy működik, ahogy az a követelményelemzés alapján elvárható. Az üzleti logika metódusainak működését egységteszttekkel (3.5.1) ellenőriztem, majd felületi tesztekkel (3.5.2) ellenőriztem, hogy a felhasználói felületen helyesen jelennek-e meg az elemek és adott felhasználói interakcióra az történik-e, amit elvárunk. A következőkben az egyes módszerek tesztéseit részletezem és a tesztelés eredményét.

### 3.5.1. Egységtesztek



3.17. ábra. Az üzleti logika tesztjeinek eredménye: az összes teszt megfelelt.

#### Strategy osztály és alosztályai

A **Strategy** absztrakt osztályt nem lehet példányosítani, ezért a publikus metódusait az összes alosztályának példányára teszteltem. A következő teszteket hajtottam így végre: Az **execute** metódus tesztelése a **SimpleDeclarerStrategy** alosztályban (feltételek és elvárt viselkedés):

1. **setStrongestSuit** metódus - megfelelően állítja-e be a legerősebb színt (a vonalon a leghosszabb szín az adu után), ha van adu és ha nincs adu
2. **setInitialDeclarerTrumpNumber** metódus - megfelelően állítja-e be a felvevő vonalán a meccs kezdetén található aduk számát, ha van adu
3. **setInitialDeclarerTrumpNumber** metódus - meghagyja-e nullának felvevő vonalán a meccs kezdetén található aduk számát, ha nincs adu
4. **initializeDummyHand** metódus - a paramétereknek megfelelően állítja-e be asztal kezét, mint adattagot
5. **initializeObservedSuits** metódus - beállítja-e kezdeti üres értékre a megfigyelt kezeket
6. **executeTestAtPos2HasStarterSuit1** metódus - második helyen, ha az induló figurát tett, a játékosnál pedig van az induló színből az indulásnál 1 figura: fed<sup>3</sup>

<sup>3</sup>bridzsos kifejezés, jelentése: nagyobb értékű lapot tesz



7. `executeTestAtPos2HasStarterSuit2` metódus - második helyen, ha az induló figurát tett, a játékosnál pedig van az induló színből az indulásnál több nagyobb figura: legkisebbel fed
8. `executeTestAtPos2HasStarterSuit3` metódus - második helyen, ha az induló figurát tett, a játékosnál pedig nincs az induló színből az indulásnál nagyobb figura: alacsonyat<sup>4</sup> tesz
9. `executeTestAtPos2HasStarterSuit4` metódus - második helyen, ha az induló nem figurát tett, a játékosnál pedig van az induló színből lap: alacsonyat tesz
10. `executeTestAtPos2HasNoStarterSuitHasTrump1` metódus - második helyen, ha az induló figurát tett, a játékosnál pedig nincs az induló színből lap, de van nála adu: alacsonyan lop<sup>5</sup>
11. `executeTestAtPos2HasNoStarterSuitDispose1` metódus - második helyen, ha az induló nem figurát tett, a játékosnál pedig nincs az induló színből lap, és nincs nála adu: dob legrövidebb (nem adu) színből
12. `executeTestAtPos2HasNoStarterSuitDispose1` metódus - második helyen, ha az induló figurát tett, a játékosnál pedig nincs az induló színből lap, és szanzadu van: dob legrövidebb színből
13. `executeTestAtPos2HasNoStarterSuitDispose3` metódus - második helyen, ha az induló figurát tett, a játékosnál pedig nincs az induló színből lap, és nincs nála adu: dob legrövidebb színből
14. `executeTestAtPos3HasStarterSuit1` metódus - harmadik helyen, ha a játékosnál van az induló színből és eddig a partner viszi kört: kicsit tesz
15. `executeTestAtPos3HasStarterSuit2` metódus - harmadik helyen, ha a játékosnál van az induló színből és eddig az ellenfél viszi a kört, de a játékosnak van annál nagyobb lapja: legalacsonyabb figurával fed
16. `executeTestAtPos3HasStarterSuit3` metódus - harmadik helyen, ha a játékosnál van az induló színből és eddig az ellenfél viszi a kört, és a játékosnak nincs nagyobb lapja se aduja szanzadu játékban: kicsit tesz
17. `executeTestAtPos3HasNoStarterSuitHasTrump1` metódus - harmadik helyen, ha a játékosnál nincs az induló színből, van aduja, és eddig a partner viszi a kört: legrövidebb nem aduszínből dob
18. `executeTestAtPos3HasNoStarterSuitHasTrump2` metódus - harmadik helyen, ha

---

<sup>4</sup>bridzsos kifejezés, jelentése: alacsony értékű lapot, pl. legkisebbet

<sup>5</sup>bridzsos kifejezés, jelentése: aduval (próbál) ütni

a játékosnál nincs az induló színből, van aduja, és eddig az ellenfél viszi a kört: alacsonyan lop

19. `executeTestAtPos3HasNoStarterSuitDispose1` metódus - harmadik helyen, ha az induló nem figurát tett, a játékosnál pedig nincs az induló színből lap, és nincs nála adu: dob legrövidebb színből
20. `executeTestAtPos3HasNoStarterSuitDispose2` metódus - harmadik helyen, ha az induló figurát tett, a játékosnál pedig nincs az induló színből lap, és szanzadu van: dob legrövidebb színből
21. `executeTestAtPos3HasNoStarterSuitDispose3` metódus - harmadik helyen, ha az induló figurát tett, a játékosnál pedig nincs az induló színből lap, és nincs nála adu: dob legrövidebb színből

A 14.-től 21.-ig felsorolt tesztesetek a negyedik helyre is érvényesek, ezeket is teszteltem. Az összes teszt megtalálható a `SimpleDeclarerStrategy`, `AdvancedDeclarerStrategy` és a `OpponentStrategy` osztályokban.

### Player osztály és alosztályai

A `Player` absztrakt osztályt nem rendelkezik saját publikus metódusokkal, de alosztályainak van egy-egy publikus metódusa, a lapkijátszásért felelős metódus. Ez a metódus a `UserPlayer` osztályban paraméterek alapján játszik ki lapot. Az `AutomaticPlayer` osztály pedig meghívja `Strategy` típusú adattagjának `execute` metódusát. (Ennek a metódusnak a helyes működéséről már meggyőződünk a `Strategy` osztály alosztályainak egység tesztelésénél.)

A `UserPlayer` tesztesetei, amik az `UserPlayerTest` osztályban vannak.

1. `playCardTestValid1` metódus - tetszőleges nem üres színből (pl. pikkből) az első pozíción lévő kártya kijátszása: kártya értéke valóban eltűnik a játékos kezéből, többi szín kártyáinak értéke változatlan
2. `playCardTestValid2` metódus - tetszőleges nem üres színből (pl. kőrből) tetszőleges pozíción (pl. harmadikon) lévő kártya kijátszása: kártya értéke valóban eltűnik a játékos kezéből, többi szín kártyáinak értéke változatlan
3. `playCardTestValid3` metódus - tetszőleges nem üres színből (pl. káróból) tetszőleges pozíción (pl. negyediken) lévő kártya kijátszása: kártya értéke valóban eltűnik a játékos kezéből, többi szín kártyáinak értéke változatlan

4. `playCardTestValid4` metódus - tetszőleges nem üres színből (pl. treffből) utolsó pozíción lévő kártya kijátszása: kártya értéke valóban eltűnik a játékos kezéből, többi szín kártyáinak értéke változatlan
5. `playCardTestFromVoid` metódus - üres színből (pl. káróból vagy pikkből) próbál meg lapot kijátszani: nem változnak a kártyák értékei
6. `playCardTestInvalidPos1` metódus - túl nagy, a sztring hosszánál nagyobb pozíción próbál meg lapot kijátszani (pl. 4. kártyát 1 kártyából): nem változnak a kártyák értékei
7. `playCardTestInvalidPos2` metódus - túl nagy, a sztring hosszánál nagyobb pozíción próbál meg lapot kijátszani (pl. 5. kártyát 3 kártyából): nem változnak a kártyák értékei

Az `AutomaticPlayer` tesztjei az `AutomaticPlayerTest` osztályban vannak.

### Node osztály és alosztályai

A `Node` osztály tesztjei a `NodeTest` osztályban vannak:

1. `getSelectedLeadTest` metódus - ellenőrzi, hogy jó-e a `selectedLead` a `round` alapján
2. `createChildrenTestFirstChild` metódus - ellenőrzi, hogy az első gyereket megfelelően hozta-e létre (adattagok helyesek-e)
3. `createChildrenTestEveryChild` metódus - ellenőrzi, hogy az összes gyereket megfelelően hozta-e létre (adattagjaik helyesek-e)

### Round osztály

A `Round` tesztjei az `RoundTest` osztályban vannak.

## 3.5.2. Felületi tesztek

Az összes oldalra ellenőriztem, hogy a 3.3.5 Felhasználó felület szekcióban az adott oldalakhoz tartozó elemek valóban megjelennek. A szekcióban ismertetett eseményeket manuális teszttel ellenőriztem, és őket eredményeikkel együtt az alábbi táblázatokban foglaltam össze.

## Menü

A menü az összes oldal tetején megjelenik, tartalmazza az összes oldal gombját, amik közül az éppen aktuális oldal gombja a többinél sötétebb színnel van kiemelve.

Teszteset	Elvárt eredmény	Kapott eredmény
"Főoldal" gomb megnyomása.	A felhasználó átirányításra kerül a főoldalra.	Megfelelt az elvártnak.
"Új játék" gomb megnyomása.	A felhasználó átirányításra kerül az új játék oldalra.	Megfelelt az elvártnak.
"Regisztráció" gomb megnyomása.	A felhasználó átirányításra kerül a regisztráció oldalra.	Megfelelt az elvártnak.
"Bejelentkezés" gomb megnyomása.	A felhasználó átirányításra kerül a bejelentkezés oldalra.	Megfelelt az elvártnak.
"Eredmények" gomb megnyomása.	A felhasználó átirányításra kerül az eredmények oldalra.	Megfelelt az elvártnak.
"Kijelentkezés" gomb megnyomása.	A menüből eltűnik az "eredmények" és a "kijelentkezés" gomb, helyette újra megjelenik a "regisztráció" és a "bejelentkezés" gomb.	Megfelelt az elvártnak.
"Súgó" gomb megnyomása.	A felhasználó átirányításra kerül a súgó oldalra.	Megfelelt az elvártnak.

## Főoldal

A főoldal összes, a 3.3.5 Felhasználó felület szekcióban felsorolt eleme megfelelően megjelenik.

Teszteset	Elvárt eredmény	Kapott eredmény
-----------	-----------------	-----------------

Kattintás a soron következő, látható lapokkal rendelkező ellenjátékos egy szabályosan játszható lapjára.	A lap eltűnik az ellenjátékos kezéből, és megjelenik az aktuális állásban az ellenjátékos lapjaihoz legközelebb eső helyen. Ha ezzel a lapkijátszással véget ért a kör és az ellenvonal vitte az ütést, az ellenvonal ütéseinek száma eggyel nő.	Megfelelt az elvártnak.
Kattintás a soron következő, látható lapokkal rendelkező ellenjátékos egy szabályosan játszható lapjára.	A lap eltűnik az ellenjátékos kezéből, és megjelenik az aktuális állásban az ellenjátékos lapjaihoz legközelebb eső helyen. Ha ezzel a lapkijátszással véget ért a kör és a felvevő vitte az ütést, a felvevővonal ütéseinek száma eggyel nő.	Megfelelt az elvártnak.
Kattintás a soron következő, látható lapokkal rendelkező ellenjátékos egy szabályosan játszható lapjára.	A lap eltűnik az ellenjátékos kezéből, és megjelenik az aktuális állásban az ellenjátékos lapjaihoz legközelebb eső helyen. Ha ezzel a lapkijátszással még nem ért véget a kör, két másodperc késleltetés után az automatikus felvevő játékos is játszik egy lapot.	Megfelelt az elvártnak.
A soron következő automata játékos játszik egy lapot.	A lap eltűnik a kezéből, és megjelenik az aktuális állásban. Ha ezzel a lapkijátszással véget ért a kör és az ellenvonal vitte az ütést, az ellenvonal ütéseinek száma eggyel nő.	Megfelelt az elvártnak.

A soron következő automata játékos kijátszik egy lapot.	A lap eltűnik a kezéből, és megjelenik az aktuális állásban. Ha ezzel a lapkijátszással véget ért a kör és a felvevő vitte az ütést, a felvevővonal ütéseinek száma eggyel nő.	Megfelelt az elvártnak.
A soron következő automata játékos kijátszik egy lapot.	A lap eltűnik a kezéből, és megjelenik az aktuális állásban. Ha ezzel a lapkijátszással nem ért véget a kör és a soron következő játékos automata játékos, akkor az két másodperc késleltetéssel kijátszik egy lapot.	Megfelelt az elvártnak.
Elemző játékmód esetén, az "Észak kártyáinak mutatása" gomb megnyomása.	Észak kártyái láthatóvá válnak.	Megfelelt az elvártnak.
Elemző játékmód esetén, az "Nyugat kártyáinak mutatása" gomb megnyomása.	Nyugat kártyái láthatóvá válnak.	Megfelelt az elvártnak.
Elemző játékmód esetén, az "Kelet kártyáinak mutatása" gomb megnyomása.	Kelet kártyái láthatóvá válnak.	Megfelelt az elvártnak.
Elemző játékmód esetén az "Észak kártyáinak elrejtésre" gomb megnyomása.	Észak kártyái helyett csak fehér téglalapok látszanak.	Megfelelt az elvártnak.
Elemző játékmód esetén az "Nyugat kártyáinak elrejtésre" gomb megnyomása.	Nyugat kártyái helyett csak fehér téglalapok látszanak.	Megfelelt az elvártnak.
Elemző játékmód esetén az "Kelet kártyáinak elrejtésre" gomb megnyomása.	Kelet kártyái helyett csak fehér téglalapok látszanak.	Megfelelt az elvártnak.

## Új játék

Az új játék oldal összes, a 3.3.5 Felhasználó felület szekcióban felsorolt eleme megfelelően megjelenik.

Teszteset	Elvárt eredmény	Kapott eredmény
Ha paraméterek megadása nélkül a "játék indítása" gomb megnyomása.	A felhasználó átirányításra kerül a főoldalra, ahol elindul egy véletlenszerűen kisorsolt leosztás hagyományos ellenjáték módban kezdő szintű felvevő ellen.	Megfelelt az elvártnak.
A "paraméterek megadása" gomb megnyomása, úgy, hogy eddig nem látszott a paraméter form.	Megjelenik a paraméterek beviteli formja.	Megfelelt az elvártnak.
A "paraméterek megadása" gomb megnyomása, úgy, hogy eddig látszott a paraméter form.	A paraméterek beviteli formja eltűnik.	Megfelelt az elvártnak.
A paraméterek beviteli formján található Játékmódok közül a "Hagyományos ellenjáték" kiválasztása kattintással, majd a "játék indítása" gomb megnyomása.	A felhasználó átirányításra kerül a főoldalra, ahol elindul egy véletlenszerűen kisorsolt leosztás hagyományos ellenjáték módban.	Megfelelt az elvártnak.
A paraméterek beviteli formján található Játékmódok közül az "Elemzés" kiválasztása kattintással, majd a "játék indítása" gomb megnyomása.	A felhasználó átirányításra kerül a főoldalra, ahol elindul egy véletlenszerűen kisorsolt leosztás elemző ellenjáték módban.	Megfelelt az elvártnak.

A paraméterek beviteli formjában található Játékmódok közül a "Hagyományos ellenjáték" kiválasztása kattintással, majd a "játék indítása" gomb megnyomása.	A felhasználó átirányításra kerül a főoldalra, ahol elindul egy véletlenszerűen kisorsolt leosztás hagyományos ellenjáték módban.	Megfelelt az elvártnak.
A paraméterek beviteli formjában található Játékmódok közül az "Elemzés" kiválasztása kattintással, majd a "játék indítása" gomb megnyomása.	A felhasználó átirányításra kerül a főoldalra, ahol elindul egy véletlenszerűen kisorsolt leosztás elemző ellenjáték módban.	Megfelelt az elvártnak.
A paraméterek beviteli formjában található Felvevő szintek közül a "Kezdő" kiválasztása kattintással, majd a "játék indítása" gomb megnyomása.	A felhasználó átirányításra kerül a főoldalra, ahol elindul egy véletlenszerűen kisorsolt leosztás kezdő szintű felvevő ellen.	Megfelelt az elvártnak.
A paraméterek beviteli formjában található Felvevő szintek közül a "Haladó" kiválasztása kattintással, majd a "játék indítása" gomb megnyomása.	A felhasználó átirányításra kerül a főoldalra, ahol elindul egy véletlenszerűen kisorsolt leosztás haladó szintű felvevő ellen.	Megfelelt az elvártnak.
A paraméterek beviteli formjában található Szan- vagy színfelvétel opciókból a "Szan" kiválasztása kattintással, majd a "játék indítása" gomb megnyomása.	A felhasználó átirányításra kerül a főoldalra, ahol elindul egy véletlenszerűen kisorsolt szanzadu felvétel.	Megfelelt az elvártnak.
A paraméterek beviteli formjában található Szan- vagy színfelvétel opciókból a "Szín" kiválasztása kattintással, majd a "játék indítása" gomb megnyomása.	A felhasználó átirányításra kerül a főoldalra, ahol elindul egy véletlenszerűen kisorsolt leosztás, amiben színjáték van.	Megfelelt az elvártnak.



**Regisztráció**

A regisztráció oldal összes, a 3.3.5 Felhasználó felület szekcióban felsorolt eleme megfelelően megjelenik.

Teszt eset	Elvárt eredmény	Kapott eredmény
A beviteli mezőkben egy 3-20 karakter hosszú, speciális és ékezetes karaktert nem tartalmazó felhasználónév és legalább 8 karakter hosszú jelszó megadása, majd a "bejelentkezés" gomb megnyomása.	Felugrik egy zöld, sikert visszaigazoló üzenet és a felhasználó átirányításra kerül a bejelentkezés oldalra.	Megfelelt az elvártnak.
A beviteli mezőkben egy rövidebb, mint 3 karakter hosszú felhasználónév és tetszőleges jelszó megadása, majd a "bejelentkezés" gomb megnyomása.	Felugrik egy piros, sikertelen bejelentkezést visszaigazoló üzenet.	Megfelelt az elvártnak.
A beviteli mezőkben egy hosszabb, mint 20 karakter hosszú felhasználónév és tetszőleges jelszó megadása, majd a "bejelentkezés" gomb megnyomása.	Felugrik egy piros, sikertelen bejelentkezést visszaigazoló üzenet.	Megfelelt az elvártnak.
A beviteli mezőkben egy tetszőleges felhasználónév és kevesebb, mint 8 karakter hosszú jelszó megadása, majd a "bejelentkezés" gomb megnyomása.	Felugrik egy piros, sikertelen bejelentkezést visszaigazoló üzenet.	Megfelelt az elvártnak.

## Bejelentkezés

A bejelentkezés oldal összes, a 3.3.5 Felhasználó felület szekcióban felsorolt eleme megfelelően megjelenik.

Teszteset	Elvárt eredmény	Kapott eredmény
A beviteli mezőkben egy létező felhasználónév és a hozzá tartozó jelszó megadása, majd a "bejelentkezés" gomb megnyomása.	Felugrik egy zöld, sikert visszaigazoló üzenet és a felhasználó átirányításra kerül a főoldalra.	Megfelelt az elvártnak.
A beviteli mezőkben egy nem létező felhasználónév és tetszőleges jelszó megadása, majd a "bejelentkezés" gomb megnyomása.	Felugrik egy piros, sikertelen bejelentkezést visszaigazoló üzenet.	Megfelelt az elvártnak.
A beviteli mezőkben egy létező felhasználónév de nem a hozzá tartozó jelszó megadása, majd a "bejelentkezés" gomb megnyomása.	Felugrik egy piros, sikertelen bejelentkezést visszaigazoló üzenet.	Megfelelt az elvártnak.

## Eredmények

Az eredmények oldal összes, a 3.3.5 Felhasználó felület szekcióban felsorolt eleme megfelelően megjelenik.

Teszteset	Elvárt eredmény	Kapott eredmény
Általánosan: a táblázat egy tetszőleges sorában található gomb megnyomása.	Átirányítás a főoldalra, a főoldalon a táblázat sorában szereplő paraméterekkel játék indítása.	Megfelelt az elvártnak.

A táblázat egy olyan sorában található gomb megnyomása, ahol a felvétel 3NT.	Átírányítás a főoldalra, a főoldalon egy olyan játék töltődik be, ahol a felvétel 3NT.	Megfelelt az elvártnak.
A táblázat egy olyan sorában található gomb megnyomása, ahol a felvétel 6H.	Átírányítás a főoldalra, a főoldalon egy olyan játék töltődik be, ahol a felvétel 6H.	Megfelelt az elvártnak.
A táblázat egy olyan sorában található gomb megnyomása, ahol a játékmód hagyományos ellenjáték.	Átírányítás a főoldalra, a főoldalon a játék hagyományos ellenjáték módban töltődik be.	Megfelelt az elvártnak.
A táblázat egy olyan sorában található gomb megnyomása, ahol a játékmód elemzés.	Átírányítás a főoldalra, a főoldalon a játék elemző játékmódban töltődik be.	Megfelelt az elvártnak.

### 3.5.3. Az algoritmus tesztelése

A haladó szintű felvevőjáték algoritmus teljesítménye nem mérhető jól egységtesztekkel, ezért felkértem Biborka Dánielt<sup>6</sup>, hogy segítsen az algoritmus értékelésében a Bridzs Barátok Klubja Egyesület *Felvevőjáték, nem csak kezdőknek* című könyv első fejezete alapján. A fejezet címe *Ütésszerzés egy adott színben*, ami az automatikus felvevő fő feladata. Ez a fejezet a különböző felvevőjáték fogásokat ismertet és a tesztekkel azt mértük, hogy a haladó automatikus felvevőjáték algoritmus tudja-e alkalmazni ezeket a fogásokat. Illetve összehasonlítottuk a kezdő és a haladó szintű algoritmust.

#### Hosszú szín magasítása

A kezdő felvevő általában megpróbált a hosszú színeiben ütésekot kovácsolni, viszont volt rá példa, hogy a magasítás helyett ütésekot lehívása mellett döntött, amivel ütésekot veszí-

<sup>6</sup>Biborka Dániel napi szinten foglalkozik a bridzsszel, a magyar U21 válogatott tagja. Számos országos versenyen vett részt és ért el kiemelkedő eredményeket, mint az országos csapatbajnokság 1B osztály 3. helye, Ifjúsági párosbajnokság 2. helye, Országos Páros Bajnokság B kategória 1. helye. A 2024-es Ifjúsági Európabajnokságon 6. helyezést ért el.

tett. A haladó felvevő elég jó arányban választotta megfelelően a magas színét magasítás céljából, viszont itt is volt rá példa, hogy a magasítandó színe helyett egy rövid színéből hívott, ami miatt elbukott a felvétel.

### Figurák erejének a kihasználása

- **Szöktetés:** Nem volt rá példa a tesztelés során.
- **Impassz:** A haladó felvevőjátékos sikeresen felismerte, hogyan kell impasszt adni, hogy ütést szerezzen a színben.

### Közlekedés, deblokkolás

A tesztek során úgy tűnt, hogy a felvevő többször jól döntött azzal kapcsolatban, hogy amikor mindkét kézzel üthetett, akkor a megfelelőt választva biztosította, hogy később hozzájusson a hosszú színében elérhető ütésekhez.

### Dobás

A tesztek során úgy tűnt, hogy az automatikus lapkijátszó algoritmus, amikor döntenie kellett, hogy mit dobjon el, akkor sokszor nem a megfelelő döntést hozta. Mind ellenjátékosként, mind felvevőjátékosként eldobta az ütésnek számító lapját.

### Ütésszerzés

Ellenjátékos részéről volt rá példa, hogy nem ütött negyedik helyen, de olyan is, hogy a partnere ütését ütötte át.

### Összegzés

A tesztelés eredményeképpen elmondhatjuk, hogy a haladó felvevőjáték algoritmus jobban teljesít, mint a kezdő. A haladó felvevőjáték algoritmus az esetek **60-70%-ában hoz jó döntéseket** (egy versenyszerűen bridzselő játékos szempontjából).

A tesztelés alapján úgy tűnik, hogy egy kezdő bridzstudás szintű felhasználó tanulhat mind a kezdő, mind a haladó szintű automatikus felvevőtől, míg egy közepes bridzstudás szintű felhasználónak a hagyományos ellenjáték mód (főleg, ha nyugat hibázik) jelenthet kihívást, de az elemző módban nem esik neheze nyernie.

## 4. fejezet

# Összegzés

A szakdolgozat alkalmazásban megvalósítottam egy *online bridzs játékot*, amiben kiemelt szerepe van az automatikus felvevőjáték algoritmusnak. Az alkalmazás kétféle játék módot tesz lehetővé kétféle játék szinten. A felhasználók tudnak egy profilt regisztrálni, amivel bejelentkezve az *eredményeik mentésre kerülnek az adatbázisban* és felhasználók meg tudják nézni az eredményeiket, sőt ezek alapján tudnak adott beállításokkal egy adott leosztást indítani.

Az automatikus felvevő játék algoritmust magam dolgoztam ki, felhasználva bridzs tapasztalataimat és azt, amit az egyetemen az *általános fákról és a kétszemélyes játékokról* tanultam.

Az alkalmazás megtervezésében nagy szerepe volt az *objektum orientált programozásnak és a tervezési mintáknak*. Az alkalmazás megvalósítására kiválasztott technológiák közül a Spring Boot keretrendszer használatát a Technische Universität Münchenen tanultam meg, a React könyvtárak használatával pedig nagyrészt önállóan ismerkedtem meg, de jó alapokat adott hozzá az ELTE webfejlesztés és webprogramozás tárgya.

### 4.1. Kitekintés

Léteznek más online bridzs platformok is. A [Bridge Base Online](#) (BBO) az egyik legnépszerűbb ilyen alkalmazás, ahol az ember online versenyekre tud nevezni, tud más felhasználók vagy robotok ellen játszani és előre meghatározott bridzs példákkal (Bridge Master) tudja gyakorolni a felvevőjátékot.

## 4.2. Fejlesztési lehetőségek

Az alkalmazás az előrelátó tervezési döntések miatt könnyen bővíthető, például könnyen be lehet vezetni *más tudás szintű felvevő játék algoritmusokat*, újabb Strategy alosztályokként. Ugyanígy lehet több, személyre *szabott ellenjáték startégiát* bevezetni, hogy a játékos olyn startégiával gyakorolhasson, amit előben a partnerével is játszik.

Könnyen bővíteni lehet az alkalmazást admin jogú felhasználókkal is, a SpringSecurity és a user entity getAuthorities metódusa már elő is készítette a *különböző jogosultságok használatát*. Az adminok a Game repositoryk és a Game RestControllerének segítségével hozzá tudnának adni leosztásokat az adatbázishoz, így nem egy fejlesztőnek kéne a leosztásokat az adatbázisba bevinni.

Egy további érdekes fejlesztési lehetőség az alkalmazás többjátékossá tétele, ahol a felhasználók ellenjátékos partnerekként tudnának egymással valós időben játszani.

# Köszönetnyilvánítás

Szeretnék köszönetet mondani mindenkinek, aki közvetlenül vagy közvetetten segítette a szakdolgozatom létrejöttét. Hálás köszönet illeti a témavezetőmet, Pusztai Kingát, akiről úgy éreztem mindig támogatott és hasznos visszajelzéseket adott a munkám során.

Szeretnék köszönetet mondani az oktatóimnak és kollégáimnak is, akiktől egyetemi éveim során felbecsülhetetlen értékű tudást szereztem.

# Irodalomjegyzék

- [1] Prof. Florian Matthes. *Software Engineering for Business Applications - Bachelor's Course: 5. Technical foundations of Business Information Systems*. Wintersemester 24/25.
- [2] Szalai-Gindl János Márk. *Adatbázisok 1. Egyed-kapcsolat modell*. 23/24 tavaszi félév.
- [3] Szalai-Gindl János Márk. *Adatbázisok 1. Egyed-kapcsolat modell - 4. rész*. 23/24 tavaszi félév.
- [4] Ásványi Tibor. *Algoritmusok és adatszerkezetek II. előadásjegyzet: Fák*. Utolsó elérés: 2023. szeptember 11. URL: <http://aszt.inf.elte.hu/~asvanyi/ad/ad2jegyzet/ad2jegyzetFak.pdf>.
- [5] Prof. Florian Matthes. *Software Engineering for Business Applications - Bachelor's Course: 7. Architecture of Distributed Information Systems*. Wintersemester 24/25.
- [6] Prof. Florian Matthes. *Software Engineering for Business Applications - Bachelor's Course: 6. Persistent Data Management*. Wintersemester 24/25.