**Szolnoki Szakképzési Centrum**

**Rózsa Imre Technikum**

**Az 54 213 05 számú Szoftverfejlesztő szakképzés záródolgozata**

**Match 3**

**Készítette:**

**Csontos Krisztián**

**2021.**

Tartalomjegyzék

[Match 3 1](#_Toc73358736)

[A feladat leírása 1](#_Toc73358737)

[Adatbázis 1](#_Toc73358738)

[Az adatmodell 1](#_Toc73358739)

[Az adatmodell ismertetése 2](#_Toc73358740)

[Továbbfejlesztési lehetőségek 3](#_Toc73358741)

[Teszt adatok készítése 3](#_Toc73358742)

[Backend 4](#_Toc73358743)

[A választott technológia 4](#_Toc73358744)

[A backend működése, szerkezete 4](#_Toc73358745)

[A backend tesztelésének leírása 5](#_Toc73358746)

[Frontend 6](#_Toc73358747)

[A választott technológia 6](#_Toc73358748)

[A fejlesztő környezet ismertetése 6](#_Toc73358749)

[Szerkezeti leírás 7](#_Toc73358750)

[Fő funkciók 7](#_Toc73358751)

[Menü szerkezet 8](#_Toc73358752)

[Képernyő képek 9](#_Toc73358753)

[Az alkalmazás felépítése 12](#_Toc73358754)

[Kapcsolata backend-el 13](#_Toc73358755)

[A játék 14](#_Toc73358756)

[Előkészületek és a két játékos szerepe 14](#_Toc73358757)

[Működése, szerkezete 15](#_Toc73358758)

[Továbbfejlesztési lehetőségek 18](#_Toc73358759)

[Állapotleírás 18](#_Toc73358760)

[Hiányosság lista 18](#_Toc73358761)

[Fejlesztési irányok 19](#_Toc73358762)

[Mellékletek 20](#_Toc73358763)

# Match 3

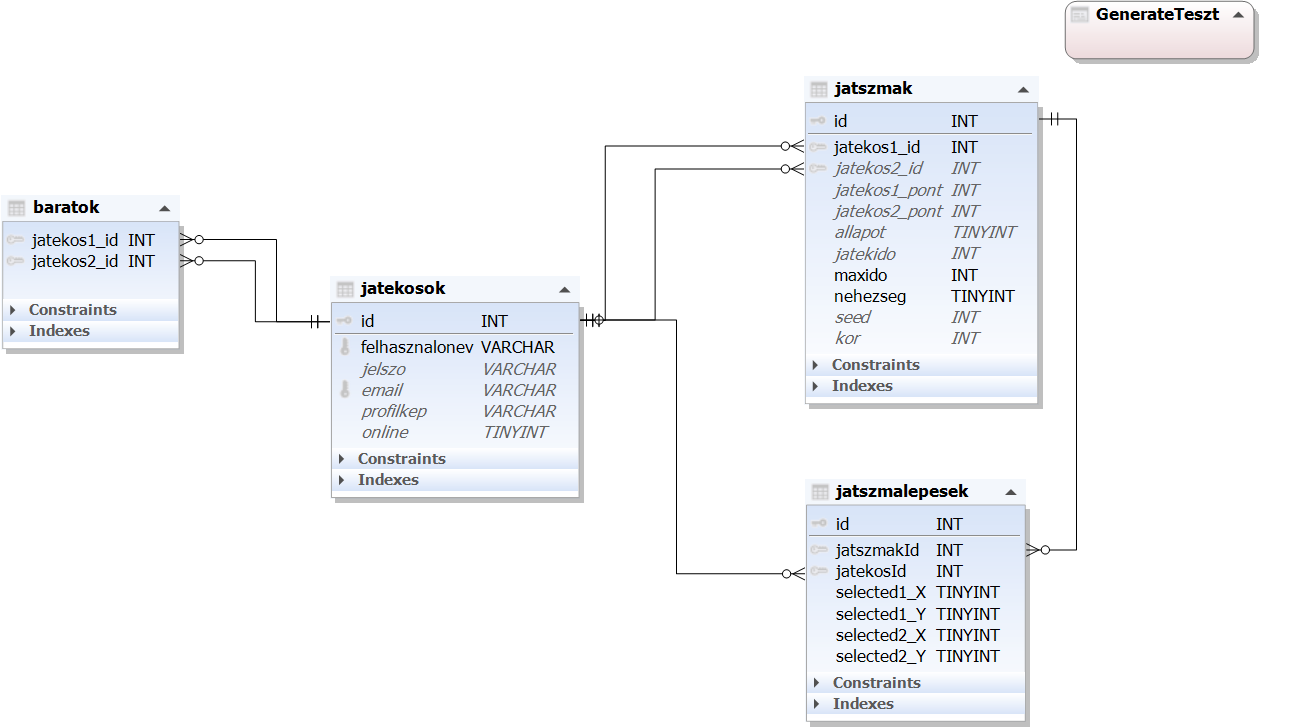
A nevem Csontos Krisztián, érdeklődési köreim közé tartozik a játékfejlesztés, a mesterséges intelligencia és Data Science. Szeretnék tovább tanulni ebben az irányban és minimum elvégezni az alapképzést és utána lehet megpróbálkoznék a mesterképzéssel is.

## A feladat leírása

A feladat egy webalkalmazás elkészítésé, ahol regisztrált játkosok online játszhatnak egy „Match 3” nevű párosító játékot egymás ellen. Az eredményeiket eltároljuk egy adatbázisban, és azok alapján rangsoroljuk őket.

# Adatbázis

## Az adatmodell



. ábra: "match3" adatbázis táblái

## Az adatmodell ismertetése

|  |  |  |
| --- | --- | --- |
| **Tábla neve: *jatekosok*** | | |
| Játékos azonosító | id | INT |
| Játékos felhasználóneve | felhasznalonev | VARCHAR |
| Játékos jelszava (titkosítva) | jelszo | VARCHAR |
| Játékos email címe | email | VARCHAR |
| Játékos profilképe | profilkep | VARCHAR |
| Be van-e jelentkezve a játékos | online | TINYINT |

|  |  |  |
| --- | --- | --- |
| **Tábla neve: *baratok*** | | |
| Hivatkozás a játékos azonosítójára | jatekos1\_id | INT |
| Hivatkozás a játékos azonosítójára | jatekos2\_id | INT |

|  |  |  |
| --- | --- | --- |
| **Tábla neve: *jatszmak*** | | |
| Játszma azonosítója | id | INT |
| Hivatkozás az első játékos azonosítójára | jatekos1\_id | INT |
| Hivatkozás az második játékos azonosítójára | jatekos2\_id | INT |
| Első játékos pontszáma | jatekos1\_pont | INT |
| Második játékos pontszáma | jatekos2\_pont | INT |
| A játszma állapota  (1 – folyamatban, 0 – befejezett) | allapot | TINYINT |
| Mennyi idő maradt a játszmából | jatekido | INT |
| A játszma időtartalma | maxido | INT |
| Nehézségiszint  (1 – könnyű, 2 – Normál, 3 – Nehéz) | nehezseg | TINYINT |
| A random szám generáláshoz felhasznált érték | seed | INT |
| Melyik játékosnak van éppen a köre | kor | INT |

|  |  |  |
| --- | --- | --- |
| **Tábla neve: *jatszmalepesek*** | | |
| Azonosító | id | INT |
| Hivatkozás a játszma azonosítójára | jatszmakId | INT |
| Hivatkozás annak a játékos azonosítójára, aki ezt a lépést tette | jatekosId | INT |
| Az első kiválasztott kocka X koordinátája | selected1\_X | TINYINT |
| Az első kiválasztott kocka Y koordinátája | selected1\_Y | TINYINT |
| A második kiválasztott kocka X koordinátája | selected2\_X | TINYINT |
| A második kiválasztott kocka Y koordinátája | selected2\_Y | TINYINT |

## Továbbfejlesztési lehetőségek

* Mező nevek rendszerezése
* Egy „ertesitesek” tábla bevezetése, ahol a különböző értesítéseket tárolhatjuk
* Egy „ertesitesFajtak” tábla bevezetése, ahol az értesítés fajtáját tárolhatjuk
  + Pl.: Barátnak jelölés, Meghívás egy játékra stb.
* „ertesitesek” és „jatekosok” tábla összekötése

## Teszt adatok készítése

Teszt adataimat nagyrészt véletlenszerűen állítottam elő Microsoft Excel-ben, majd az adott „csv” kiterjesztésű fájlt beimportáltam a megfelelő táblába.

A „jatekosok” tábla adatai a kivétel ez alól, ugyanis itt az osztálytársaim Discord felhasználóneveit használtam fel. A jelszó a gyors tesztelés érdekében az, hogy „jelszo” mindegyik felhasználónál. Az email cím és a profilkép a felhasználónév alapján van elnevezve. Pl. a felhasználónév az „Lysander13”, akkor az email cím az „Lyander13@example.com” és a profilkép neve pedig „Lysander13.png”.

A „baratok” tábla adatai viszont már véletlenszerűen vannak legenerálva. Valamint beleraktam egy ismétlődést, ezzel tesztelve, hogy ez milyen hatással lenne rá.

|  |  |
| --- | --- |
| ***jatekos1\_id*** | ***jatekos2\_id*** |
| 1 | 3 |
| 3 | 1 |

Egy ilyen eset általában nem fordul elő, hisz ehhez az kell, hogy mindkét fél egyszerre, egyidőben legyen az adott oldalon, hogy egymást barátnak adják hozzá. Nem életszerű, de ettől függetlenül előfordulhat. Ezt a problémát úgy oldottam meg, hogy mikor lekérdezzük a táblát, akkor az SQL utasítás kiszűri az ismétlődéseket, tehát egyik oldalon sem fog megjelenni kétszer egy barát. Ha pedig törlünk, akkor mindkét esetet törölni fogja az alábbi utasítás miatt:



. ábra: Rekord törlése ismétlődés esetén is

# Backend

## A választott technológia

A backend-hez Apache webszervert és MySQL adatbázisszervert használtam. Az adatbázissal PHP programozási nyelv segítségével kommunikáltam.

## A backend működése, szerkezete

A belépési pontunk az „index.php” fájl, ami először létrehoz vagy folytat egy meglévő session-t, majd betölti az adatbázis csatlakozásához szükséges konfigurációkat. A backend képes dinamikusan betölteni osztályokat, mikor használjuk azokat, tehát így tölti be, és végül példányosítja az App osztályt, aminek meghívja a „start” függvényét.

A függvény ellenőrzi a megadott paraméterekt a $\_GET szuperglobális tömbből. Ha a „query” paraméterünk nem szerepel a „$services” tömbben, akkor paraméter hibát dobunk vissza. Ellenkező esetben példányosítjuk a megfelelő osztályt a megadott paramétereinket tovább adva neki, és elindítjuk a „render” függvényét.

Az adott osztály „render” függvényében először kapcsolódunk az adatbázishoz, hiba esetén a „status” változónkban „Connecton Error” lesz. Kapcsolódás után az osztály SQL utasítását ellenőrizzük, hiba esetén logolunk és „Querry Error” lesz a „status”-ban. Ha minden helyes, akkor az SQL utasításhoz fűzzük a megadott paramétereket, így megakadályozva egy „SQL Injection” fajta támadást. Ezek után lefuttatjuk a parancsot, majd az eredményt eltároljuk a „rows” változóban, ha szükség van rá. A „status” változónkban „Ok”, vagy más ehhez hasonló üzenet lesz.

Végül példányosítjuk a „SendDataJson” osztályunkat és meghívjuk a „sendData” függvényét, ami visszaküldi a fontend-re a paramétereinket, a „status”-t, az osztály „title”-jét, valamint a „rows” és „columns” tömböt.

Hogy ne legyen ismétlődés a kódunkban, az osztályainknak különböző szülő osztályaik vannak, attól függően, hogy mit csinál:

**ParentTabla**: Egyszerű lekérdezés esetén használjuk ezt.

**ParentRekordById**: Ha specifikusan keresünk rekordokat egy érték alapján.

**ParentInsert**: Ha egy új rekordot szeretnénk felvinni az adatbázisba.

**ParentUpdate**: Ha frissíteni szeretnénk egy rekordot.

**ParentDelete**: Ha törölni szeretnénk egy rekordot.

Ezek alól is vannak kivételek, ilyenek például a „LoginUser”, „LogoutUser” és „GetUser” osztályok. Ezek speciális eseteket kezelnek, ezért kódjaik nem ismétlődnek, mint ahogy a „ParentTable”-é ismétlődne, amit arra használunk lekérdezzünk egy tábla minden adatát.

## A backend tesztelésének leírása

A teszteléshez egy „Postman” nevű alkalmazást használtam, aminek a segítségével nagyon egyszerűen lehet „GET” és „POST” kéréseket küldeni a backend-re. A paramétereket is megadhatunk, amik közül kiválaszthatjuk miket használunk, és miket nem, így megvizsgálhatjuk a válaszokat a különböző paraméterekre.

Használata nagyon egyszerű. Adjuk meg a backend URL-jét, majd válaszuk, hogy milyen kérést szeretnénk küldeni, és adjuk meg a paramétereket. Ezek után csak a „Send” gombra kell kattintani és küldés után nemsokkal megkapjuk a backend válaszát.

# Frontend

## A választott technológia

A frontend elkészítéséhez egy Vue nevű keretrendszert használtam és annak dizájnolásához Bootstrap-et. Valamint a backend-el való kommunikáláshoz Ajax-ot.

### A fejlesztő környezet ismertetése

A projekt létrehozásához Vue CLI-t alkalmaztam, ami egy globálisan telepített „npm” csomag. Segítségével egy paranccsal létrehozhattam a frontend vázát. Ezen kívül még a tesztelésben is segíttet, hisz képes a webalkalmazásomat futtatni egy lokális fejlesztésre szánt szerveren. Ráadásul képes előállítani egy kész végterméket is, ha a fejlesztést befejeztük.

A különböző oldalak közötti navigáláshoz Vue Router-t használtam, ami annak a látszatát kelti, hogy több oldal között ugrálunk miközben végig az index.html fájlban vagyunk. Ez úgy oldható meg, hogy a weboldalunkon a fejléctől a láblécig minden egy Vue komponens. A fejléc, menü és lábléc mindig jelen van, és a weboldal közepén található „router-view” az útvonaltól függően jeleníti meg a komponenst. Például ha az útvonalunk „/”, akkor a „Home” komponenst fogja használni, ha pedig azt írjuk be a böngésző címsorába a weboldal címe után, hogy „/bejelentkezes”, akkor a „Bejelentkezes” komponenst fogja használni. Ezeket az útvonal-komponens hozzárendeléseket a „routes.js” fájlban adjuk meg a „routes” tömbben.

Hogy illetéktelenek ne tudjanak hozzáférni a weboldal bizonyos részeihez, ezért a Vue Router rendelkezik egy úgy nevezett „Navigation Guard”-al, ami megakadályozza az átirányítást, ha nincs jogosultságunk látni az adott oldalt.

Valamint a Vue Router segítségével hiba oldalakra is átirányíthatunk. Ilyen például a 404-es hiba oldal, ha olyan helyre akar menni a felhasználó, ami nem létezik, vagy az 500-as hiba oldal, amit akkor láthatunk, ha az egyik Ajax kérésünk hibával jön vissza.

## Szerkezeti leírás

### Fő funkciók

Az oldal képes regisztráció által felhasználókat felvinni az adatbázisba. Majd ezeket a felhasználókat bejelentkeztetni. A nem bejelentkezetett látogatók ezeken a funkciókon kívül még elérhetik a főoldalt.

A bejelentkezett felhasználók megnézhetik a profiljukat, szerkeszthetik vagy akár törölhetik is azt. Barátnak adhatnak más felhasználókat és törölhetik is őket a barátaik közül. Megnézhetik a toplistát, valamint a legutóbbi saját játszmáikat végeredménnyel együtt.

Nem utolsó sorban elindíthatnak egy játszmát egy véletlenszerűen választott játékos ellen, általuk megadott ideig és nehézségi fokon. Esetleg csatlakozhatnak egy már elindított játékhoz. Ha a rendszer nem talál ellenfelet egy percen belül vagy nincs játék, amihez csatlakozni lehetne, akkor azt közli a felhasználóval.

A játékon belül, miután legenerálódott véletlenszerűen a tábla, a felhasználó kicserélheti a szomszédos kockákat, és ha párt alkot, akkor a pár elemi eltűnnek, a fölöttük lévő kockák pedig leesnek a helyükre. A legtetején keletkezett üres hely pedig feltöltődik új véletlenszerűen generált kockákkal.

A játék számon tartja az összes lehetséges lépést, ami párt alkothat, és ha már nincs ilyen, akkor a tábla újra rendeződik, ezzel biztosítva, hogy a játék folyamatosan menjen, míg le nem telik az idő.

A páralkotás közben szerzett pontokat a rendszer eltárolja az adatbázisban és ezek a pontok alapján rangsorolja a felhasználókat.

### Menü szerkezet

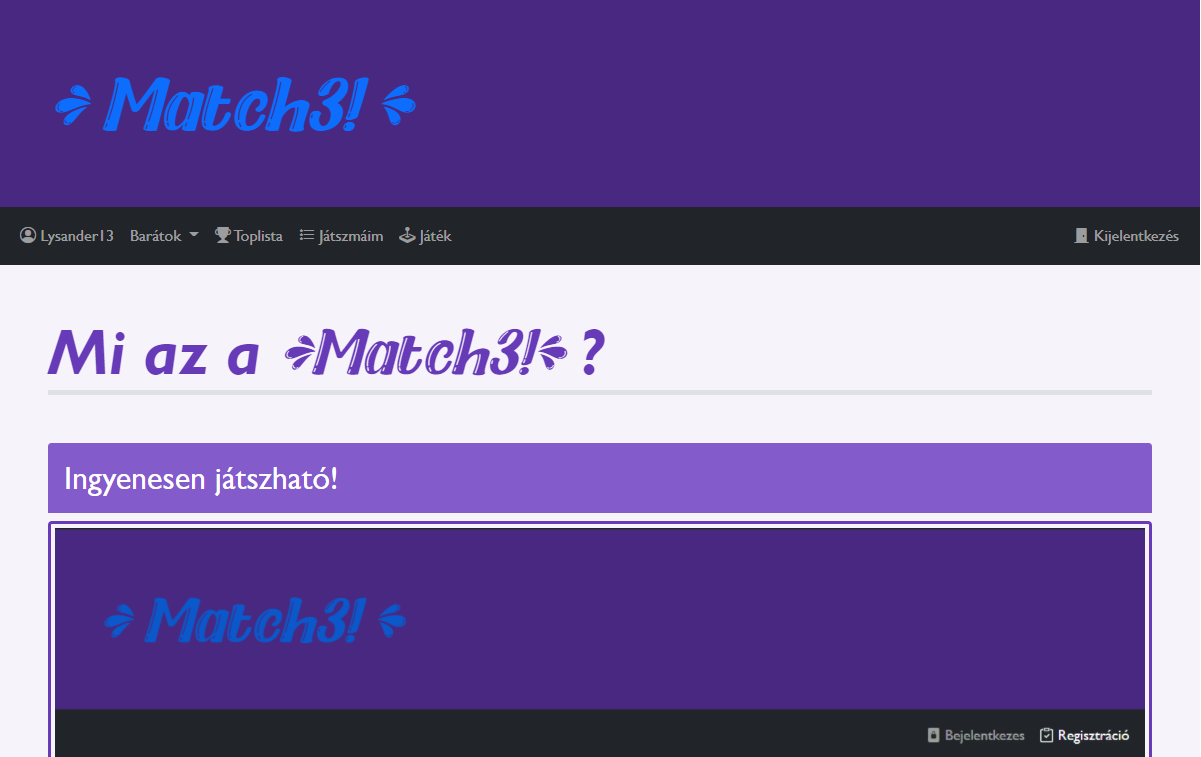
Nem bejelentkezett látogatók számára:

* Főoldal
* Bejelentkezés
* Regisztráció

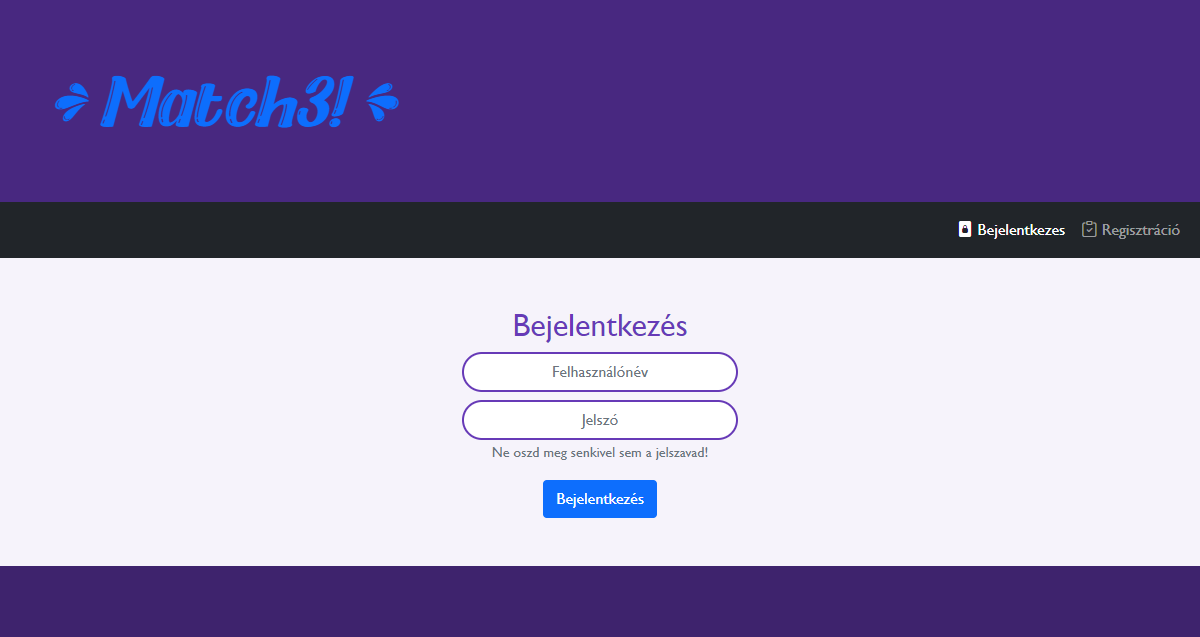
Bejelentkezett látogatók számára:

* Főoldal
* Profil (Felhasználónév)
* Barátok
  + Barátaim
  + Barátok keresése
* Toplista
* Játszmáim
* Játék
* Kijelentkezés

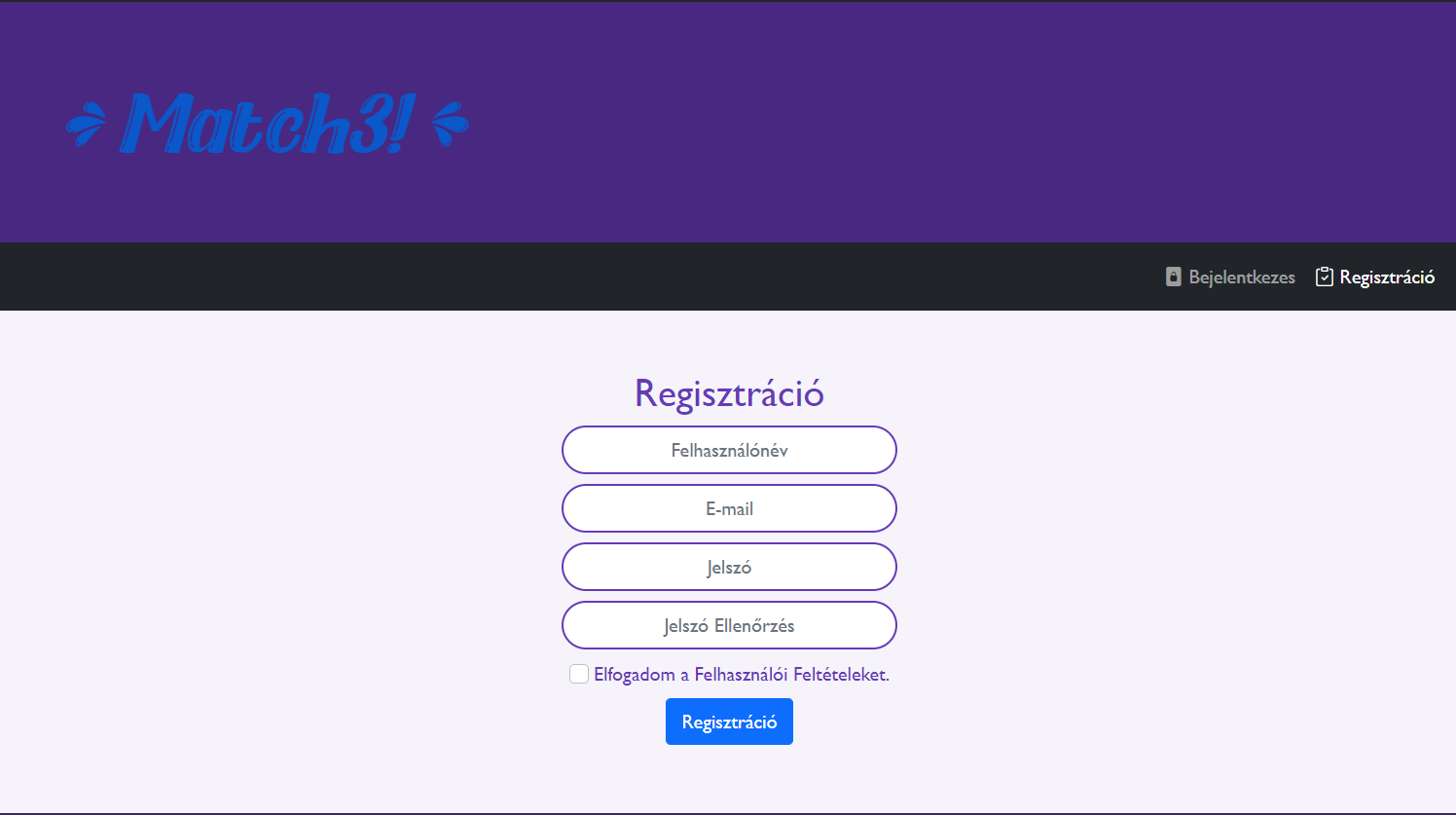
### Képernyő képek



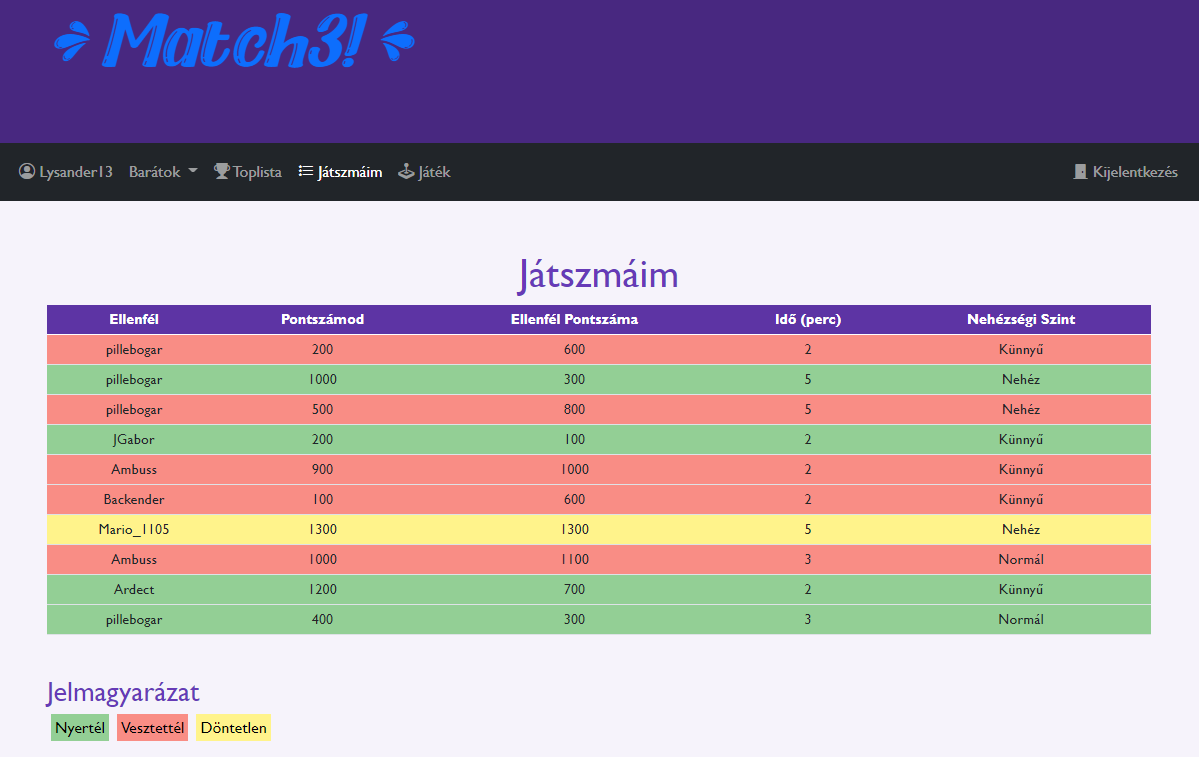
. ábra: Főmenü



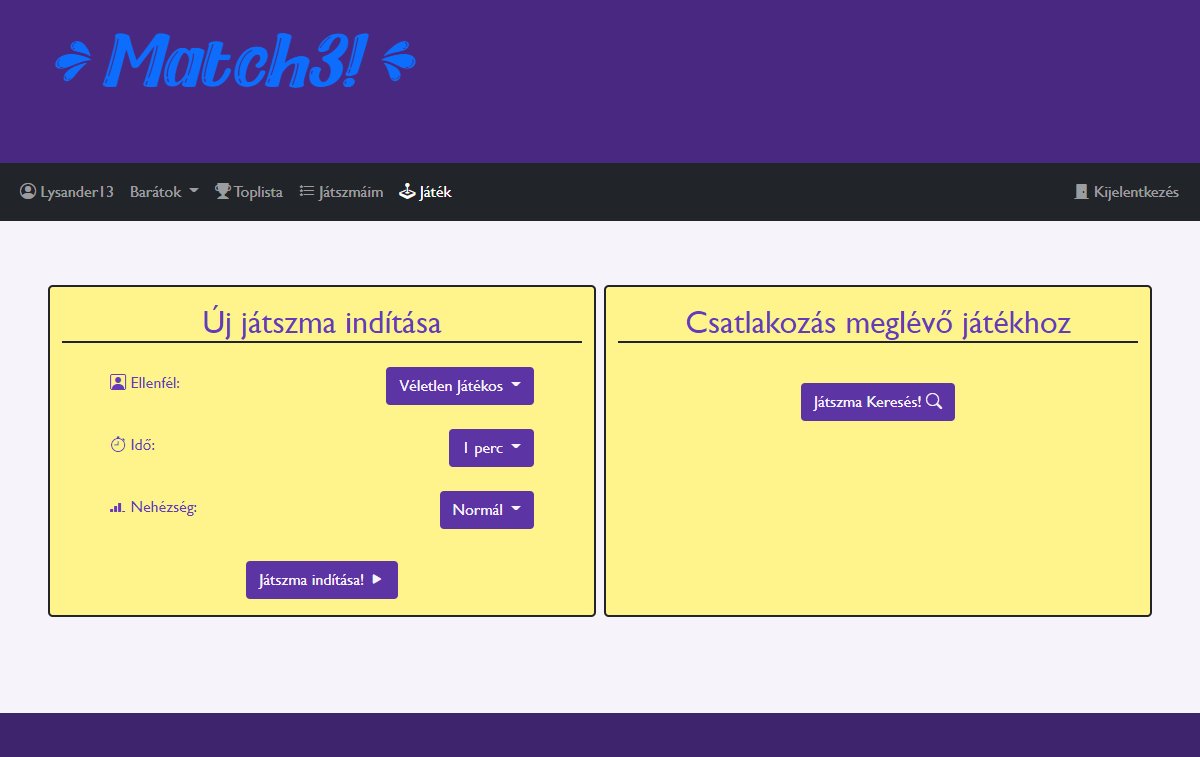
. ábra: Bejelentkezés



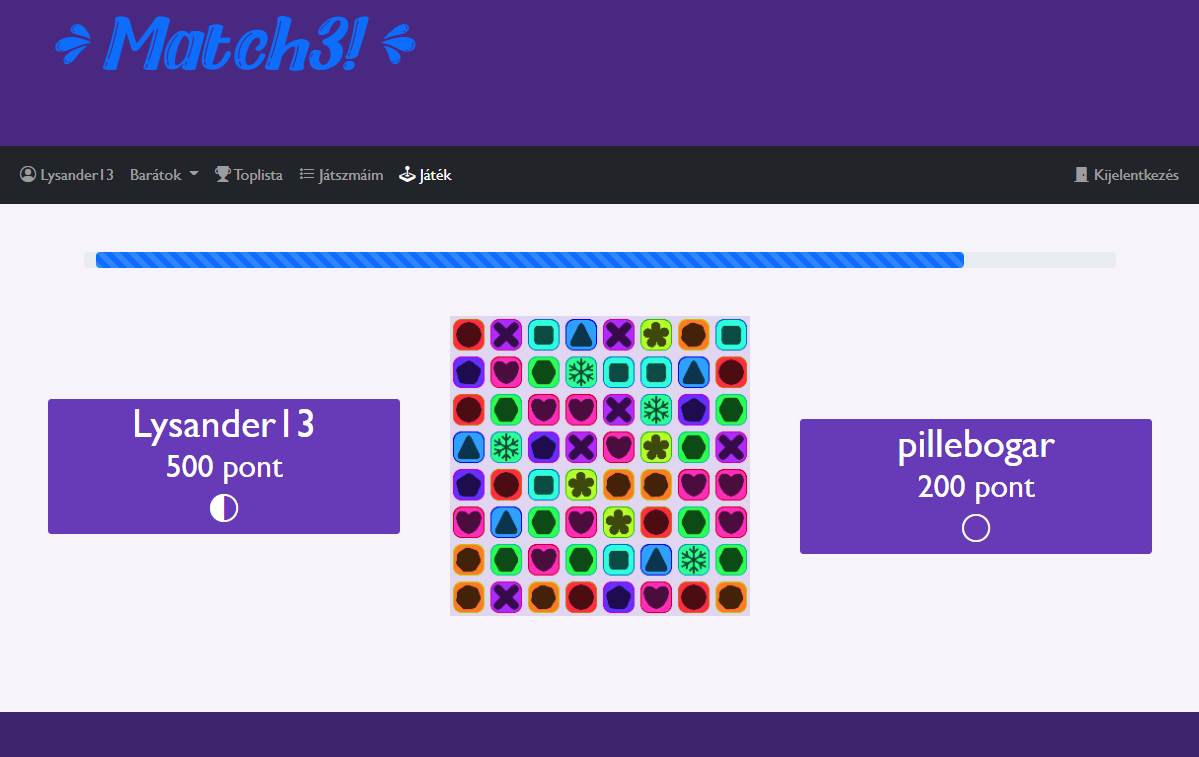
. ábra: Regisztráció



. ábra: Legutóbb játszott játszmák



7. ábra: Új játék indítása vagy csatlakozás egy meglevőhöz



8. ábra: Folyamatban lévő játék

### Az alkalmazás felépítése

Az alkalmazás belépési pontja a „main.js” fájl. Itt importáljuk a Vue-t, az „App.vue” komponenst és a Vue Router-t, valamint a Bootstrap-et. Itt konfiguráljuk a routert és a Vue Mixin segítségével továbbítjuk a backend URL-jét minden komponensnek. Végül példányosítjuk a Vue-t és az „index.html”-ben található „app” azonosítóval rendelkező „div”-hez kötjük.

Az oldal, amit látunk valójában az „App.vue” nevű komponens, amiben elhelyezkedik a „Header”, a „Menu” és a „Footer”. Valamint egy „router-view”, ami mindig az útvonalhoz társított komponenst fogja megjeleníteni.

A „Menu.vue” komponensben találhatóak a „router-link”-jeink, amikkel navigálunk a különböző oldalak között.

A „router-view” által megjeleníthető komponenseink a „routes” mappában találhatóak. Ezek és az összes többi Vue komponens három részből áll: „template”, „script”, „style”. Ezek megfelelnek a jól ismert „html”, „js” és „css” hármasnak. A „script”-en belül található egy „default” objektum is, amit exportálunk. Ennek az objektumnak különböző tulajdonságai és metódusai vannak, mint pl.:

* name – A komponens neve
* components – Használt komponensek
* props – Kívülről megadott adatok
* data() – Itt definiáljuk azokat a változókat, amiket később a „template”-ben vagy a „methods”-ban használunk.
* computed – Olyan függvény, ami egy konstans változóként funkcionál. Ha az egyik változó megváltozott, amit a függvényben használtunk, akkor a végeredmény is frissül vele együtt.
* methods – Itt hozzuk létre a függvényeinket.

A „script” legelején általában a beimportált komponenseink és könyvtáraink találhatóak, ilyen például az „axios” is, amivel Ajax kéréseket intézünk.

### Kapcsolata backend-el

A frontend a backend-el Ajax kéréseken keresztül tartja a kapcsolatot. Ehhez egy „axios” nevű könyvtárat használunk, aminek két fő függvénye van: „get” és „post”. A „get” függvényt adatok lekérésére használjuk, míg a „post”-ot adatok frissítésére. Tehát törlés, frissítés és új adat felvitele esetén „post” kérést alkalmazunk, minden másnál „get” kérést.

Mindkét függvény esetén 2 paraméter van: az URL és egy objektum, ami tartalmazza a paramétereinket. A mi alkalmazásunkban a „query” tulajdonság mindig jelen van a paraméterek között. A többi paraméter a backend oldalon lévő query igényeitől függ.

Ezek a függvények aszinkronikusak, ígéretek, amik egyszer teljesülnek. Tehát miután meghívtuk őket a programunk tovább megy, nem várja meg a választ. Emiatt a „get”, illetve „post” függvény után szükséges egy „then”-t is meghívni. Ebben a függvényben definiálunk egy névtelen függvényt egy „res” paraméterrel, ami akkor fog lefutni, mikor az ígéret teljesült.

Ha valamilyen oknál fogva nem teljesül az ígéret, akkor a „catch” függvényben definiált függvény fut le. Ez minden esetben átirányít egy hiba oldalra, ahol kiírja a hibaüzenetet, de semmi többet. Pl.: Ha nem tudott csatlakozni a backend-hez, akkor a „Error: Network Error” üzenetet fogja kiírni.

## A játék

### Előkészületek és a két játékos szerepe

Mielőtt elindítanánk a játékot, előtte több dologról is meg kell bizonyosodnunk, és minden lehetőséget kezelnünk kell.

Lényeges megemlíteni, hogy a „Jatek.vue” komponensünknek több kinézete is van, attól függően, hogy mi a megjelenítési módunk:

„NewGame” – Az amit az oldal betöltésekor is látunk. Itt csinálhatunk egy új játékot vagy csatlakozhatunk egy meglévőhöz.

„Load” – Ebben a módban csak egy üzenet látszódik, hogy keressük a felhasználó ellenfelét, és zárójelben a visszaszámlálásból maradt másodpercek láthatóak.

„Game” – Ebben az állapotban a játékot láthatjuk, a két oldalán a játékosok neveit és azoknak pontszámaikat, valamint, hogy mennyit léphetnek még.

„OutCome” – Itt látható a játék után, hogy nyertünk, vesztettünk vagy döntetlen-e a játszma. Az is látható itt, hogy mennyi pontot nyertünk vagy vesztettünk.

Mikor megnyomjuk az oldal két gombjai közül az egyiket, akkor előkészületek folynak le, még mielőtt példányosítanánk a „Match3” nevű osztályunkat, ezzel elindítva a játékot. Attól függően, hogy mi indítjuk-e a játékot, vagy csatlakozunk-e egy meglevőhöz, különböző függvények hívódnak meg.

Ha mi hozzuk létre az új játékot, akkor a „createNewGame” függvény hívódik meg, ami eltárolja a jelenlegi idő milliszekundum részét, hogy később fel tudjuk használni a véletlen szám generátorhoz. Ezután meghívja az „insertJatszmak” függvény, ami feltölt egy új játszmát az adatbázisunkba. Jelenleg a „jatekos2\_id” mező még üres. Elindul egy visszaszámláló, ami 60 másodperc után megszakítja a második játkos utáni való keresését. Végül a megjelenítési módunk „Load” lesz. Ekkor lefutattjuk a „getJatszmak” függvényt, ami 100 milliszekundumonként lekérdezi az backend-et, hogy van-e „jatekos2”, tehát az érték nem null. Ha nem csatlakozik senki a megadott időn belül, akkor töröljük az adatbázisból a játszmát és értesítjük a felhasználót erről. A megjelenítési mód ismét „NewGame” lesz. Ha viszont van, akkor többet nem kérdezgessük az backend-et. Megkérdezzük a backend-et mi a felhasználóneve a második játékosnak, majd a megjelenítési módot „Game”-re állítjuk, végül meghívjuk a „startGame” függvényt. Példányosítjuk a „Match3” osztályt és beállítjuk a hozzá tartozó eseményeket, mint például páralkotás esetén fusson le az „insertJatszmaLepesek”, és ha a következő játékos köre jön, akkor pedig az „updateJatszmak”-at hívja meg. Végül beállítjuk a játék időzítőjét, ami ha lejár, akkor frissíti a „Jatszmak” táblát, és törli a „Match3” példányunkat, valamint átállítja a módot „OutCome”-ra.

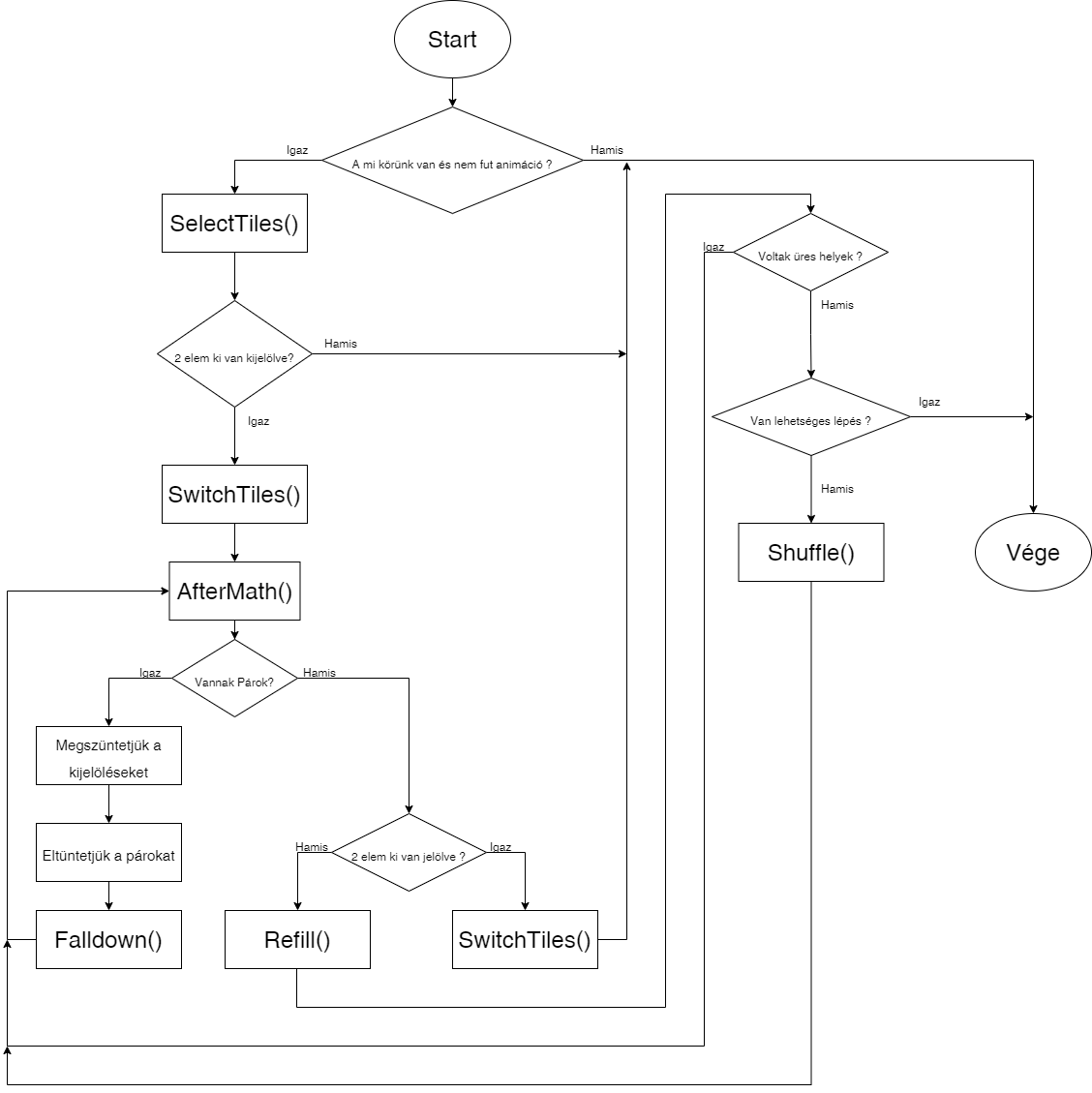
Ha csatlakozni szeretnénk egy játszmához, akkor a „findGame” függvény fut le, ami át állítja a megjelenítési módot „Load”-ra és meghívja a „findJatszmak” függvényt, ami Ajax kérésen keresztül megkérdezi a backendet, hogy van olyan játszma, ami fut (allapot = 1) és nincs második játékosa (jatekos2\_id = null), akkor megpróbálunk csatlakozni hozzá az „updateJatzsmak” segítségével, ami megpróbálja a mi azonosítónkat eltárolni a „jatekos2\_id” mezőben. Ha ez nem sikerült vagy eleve nem találtunk ilyen játszmát, akkor erről értesítjük a felhasználót és megváltoztatjuk a megjelenítési módot „NewGame”-re. Sikeres frissítés esetén viszont átáll a mód „Game”-re és lekérdezzük az ellenfelünk felhasználónevét. Elindítjuk a játékot a „startGame” függvénnyel, de ez mellet meghívjuk a „watchLepesek”-et is, ami megnézi kinek a köre van. Mivel mindig az első játkos kezd, ebből kifolyólag nem a mi körünk van. Tehát nekünk az a dolgunk, hogy félmásodpercenként megkérdezzük az adatbázist, hogy lépett-e már az ellenfelünk a „getJatszmaLepesek” függvény segítségével. Ha lépett, akkor lefuttatjuk a „deletJatszmaLepesek” és „OnOppentMove” függvényket. Az első törli a lépést az adatbázisból, ezzel jelezve, hogy értesültünk róla és a mi képernyőnkön is meg fog jelenni ez, az utóbbi függvény segítségével. Miután lefutott a „deleteJatszmaLepesek” ismét futtatjuk a „watchLepesek”-et.

Ha egyszerűsíteni szeretnénk, akkor fogjuk fel úgy, hogy van egy aktív és passzív játékos. Az aktív cselekszik és ezt továbbítja, míg a passzív figyeli a cselekvéseket és megbizonyosodik, hogy az ő felén is az van, ami a másik tábláján. Ezek a szerepek körönként felcserélődnek, egy kör pedig két lépésből áll.

### Működése, szerkezete

Az osztályunk példányosításakor lefut egy „Init” nevű függvény is, ami többek között definiálja a függvényeket, amik lefutnak az ellenfél lépésekor, és mikor sikeresen feltöltöttük a lépésünket az adatbázisba. Példányosítjuk a random szám generátorunkat tovább adva neki a „seed”-et, amit még az előkészítés alatt definiáltunk. Hozzárendeljük az egér „onmousemove” eseményéhez a függvényt, ami követi a „canvas”-hoz képest a relatív pozícióját. Itt generálódik le a tábla véletlen generált kockákkal. Valamint elindít egy „interval”-t is, ami négy milliszekundumonként (régebbi böngészőkön tíz milliszekundum) meghívja a „Render” és „Update” függvényeket. Ezek a függvények felelősek a felrajzolásért és az animációkért, ezért szükséges őket a lehető legkevesebb időközönként meghívni.

Kattintásra vagy az ellenfél lépése esetén a következő algoritmus fut le:



9. ábra: Kattintásra lefutó algoritmus

A „SelectTiles” függvény a legelső dolog, ami meghívódik. Ez a függvény felelős a kockák kijelöléséért, és ha már 2 elem ki van jelölve, akkor felcseréli őket, meghívja a függvényt, ami feltölti az adatbázisba, és lefuttatja az „AfterMath”-ot.

Az aftermath következményt, utóhatást jelent. Ez remekül leírja mit csinál a függvényünk, ugyanis itt dől el, hogy mi a következő lépés az algoritmusunkban. Ha a lépés nem alkot párt, akkor visszacseréli a két kockát, és ha igen, akkor láthatatlanná teszi őket és pontoz a pároktól függően, végül meghívja a „Falldown” függvényt. Viszont ha nincsenek párok és nincsen semmi kijelölve, akkor a „Refill” fog letufni.

A „Falldown” függvény számláló ciklussal végig megy a tömbön, ami tartalmazza a kockákat és az összes láthatatlant a tábla legtetejére helyezi, míg a helyükbe egy látható lép. Ez adja a leesés érzetet minden párosítás után.

A „Refill” pedig arra van, hogy feltöltse ezeket az „üres helyeket”. Valójában annyi történik, hogy a láthatatlan kockákat a látóhatár fölé viszi és újra generálja őket. Végül egy animációval olyan, mintha a semmiből esne le. Ha volt láthatatlan kocka, akkor az „AfterMath” függvény hívódik meg újra, mert itt nincs megszabva, hogy úgy generálódjanak a kockák, hogy ne alkossanak párokat. Ha nem voltak láthatatlan elemek és a lehetséges lépések száma nulla (FindPossibleMoves().length == 0), akkor a „Shuffle” függvényt hívja meg.

A „Shuffle” pedig véletlenszerűen újra rendezi a tábla elemeit, ügyelve, hogy közben ne alakuljanak ki párok. Újra rendezés után ismét meghívódik az „AfterMath” függvény.

# Továbbfejlesztési lehetőségek

## Állapotleírás

A webalkalmazás működőképes, minden funkciója probléma nélkül használható, akár egyszerre több felhasználó által is. Ennek ellenére még nem befejezett. Hiányoznak még alapvető dolog, amik szükségesek ahhoz publikálni lehessen a projektet.

## Hiányosság lista

* Jelenleg semmi nem állít meg, hogy a játékban folyamatosan a te köröd legyen azáltal, hogy nem lépsz
* Ha játék közben lépünk át egy másik oldalra, vagy bezárjuk a böngészőt, attól a játék még nem áll fejeződik be
* Ha mindkét fél kilép idő előtt, akkor a játék nem fejeződik be
* Nem lehet profilképet cserélni

## Fejlesztési irányok

*Weboldal*

* Szebb dizájn kialakítása.
* Más profiljának megtekintése
* Állapot megváltoztatása ha bezárjuk a böngészőt
* "inGame" állapot bevezetése az "online" és "offline" állapotok mellé
* Barátok vagy online lévő felhasználók meghívása egy játszmára
* Értesítések:
  + Barátnak jelöléskor értesítést kap a másik felhasználó, hogy elfogadja-e a jelölést vagy visszautasítja azt
  + Ha játszmára hívnak azt szintén elfogadhatja vagy elutasíthatja
* Ha játékban vagyunk, akkor az oldal figyelmeztessen, hogy ha elhagyjuk az oldalt, akkor automatikusan elveszítjük azt a játszmát
* Szűrő a "Csatlakozás meglévő játékhoz" részhez, hogy a felhasználónak legyen beleszólása, hogy milyen hosszú és milyen nehéz

*Játék*

* A játékos köre bizonyos idő múlva lejárjon és az ellenfél jöjjön
* Újra rendezés animálása
* Különböző speciális kockák
  + Pl.: Négyes pár esetén egy olyan speciális kockát kapunk, amit ha párba állítunk, akkor az egész oszlopot/sort törli
* Ha ötös párt alkotunk, akkor kapunk egy extra lépést
* Körváltáskor a játék értesítse a játkost, hogy kinek a köre jön

# Mellékletek

* [git hub publikáció](https://github.com/csontoskrisztian/zarovizsga_2021)
* Adatbázis dokumentáció
* backend, frontend forráskódok
* Dokumentációk:
  + Felhasználói Dokumentáció
  + Fejlesztői Dokumentáció