**EÖTVÖS LORÁND TUDOMÁNYEGYETEM**



Informatikai Kar

Programozáselmélet és Szoftvertechnológiai Tanszék

**Da Vinci**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Dr. habil. Gregorics Tibor** |  |  |  |  |  |  | **Gyarmati Károly Mátyás** |
| **Tanszékvezető Egyetemi Docens** |  |  |  |  |  |  | **Programtervező Informatikus BSC** |

**Budapest, 2022**

# Tartalomjegyzék

1. Bevezetés 3
2. Felhasználói leírás 4
   1. Játékszabályok 4
   2. Rendszer követelmények 6
   3. Felhasználói felület és program használata 8
3. Fejlesztői leírás 12
   1. Feladat leírás 12
      1. Funkcionális leírás 13
      2. Nem-funkcionális leírás 15
   2. Tervezés 16
      1. Architektúra 17
      2. Modell 19
      3. Nézet tervek 26
      4. Játék megvalósításának terve 30
      5. Gépi ellenfél 32
   3. Megvalósítás 40
      1. Felhasznált technológiák 40
      2. Esetleges eltérések a tervtől 44
   4. Tesztelés TBA
      1. Egység tesztek TBA
      2. Végfelhasználói tesztesetek TBA
   5. Lehetséges jövőbeni fejlesztések TBA
4. Irodalom jegyzék TBA

**1. fejezet**

# **Bevezetés**

Társasjátékokkal mindig jó a kikapcsolódás. Még egyetemi éveim elején ismerkedtem meg egy számomra nagyon élvezetes játékkal, melyben bár szerepet kap a szerencse, nagyban tudja az eredményt befolyásolni, ha figyelmesek vagyunk a legapróbb elszórt információ morzsákra is. Ez a társasjáték a Da Vinci címet viseli és rengeteg órát játszottam vele az egyetemen szervezett Társas-Szerepjáték Hétvégéken. A játék során információkat kell szereznünk, akár a játéktérről, akár korábbi lépésekből, vagy ellenfeleink testbeszédéből.

Szakdolgozatom célja a fent ismertetett társasjáték számítógépre való átültetése. A játékos legyen képes többedmagával különböző eszközökről egymással játszani az interneten keresztül és ezenkívül gépi ellenfelet is lehessen hozzá adni a játékhoz. A gépi ellenfél jelentsen minél nagyobb kihívást és annak működése legyen független a játékos számtól.

A feladatot egy egymástól független szerver-kliens megoldással szeretném megoldani, ahol a szerver futtatja és ellenőrzi a társasjáték lépéseit, a kliensek pedig a felhasználók lépéseit jelenítik meg és közvetítik.

**2. fejezet**

# **Felhasználói leírás**

# **2.1 Játékszabályok**

A képen szöveg látható

Automatikusan generált leírásA játékban egy kódsort állítunk össze számmal és színnel bíró játék elemekből, az alapjáték 0-tól 11-ig tartalmazó játékelemeket tartalmaz két színből, opcionálisan egy kötőjel karakterrel rendelkező elemet is használhatunk színenként. A játék folyamán minden egyéni kódsorban a számoknak növekvő sorrendben kell lenniük, amennyiben két azonos számmal bíró elemünk is van, akkor valamelyik szín mindig precedenciát élvez a másikhoz képest.

**2.1. ábra. Da Vinci társasjáték**

A képen szöveg, padló, beltéri látható

Automatikusan generált leírásA játék elején helyezzük az összes elemet megkeverve és számmal lefordítva az asztal közepére. Ezután játékos számtól függően tetszés szerinti színből 4 vagy 3 elemet húzzon minden játékos magához, fordítsa fel ügyelve, hogy más az értéket ne lássa, és állítsa fel az elemeket az előbbiek szerint növekvő sorrendbe. A kezdő játékos ezután rámutat valamelyik ellenfél rejtett elemére és mond egy számot. Amennyiben eltalálja a mutatott elem valós értékét a birtokos játékos azt felfordítja, a játékos, aki tippelt pedig dönthet, hogy tippel tovább vagy megáll és magához húz az asztal közepéről egy elemet és beilleszti azt a kódsorába a megfelelő helyre úgy, hogy csak ő lássa az értéket. Amennyiben az eredeti tippje a játékosnak nem volt jó, vagy úgy döntött, hogy folytatja a tippelést és bármelyik későbbi tippje helytelen volt akkor szintén egy középről húzott elemet kell beilleszteni a kódsorba, viszont oly módon, hogy annak számlapja minden játékos számára is látható legyen. Ha a játékos abba hagyta a tippelést, akkor a soron következő játékos jön a saját tippjével hasonló módon.

**2.2. ábra. Da Vinci társasjáték játék közben**

Ha egy játékosnak minden kód eleme ki lett találva, akkor kiesik a játékból. A játéknak akkor van vége, ha már csak egy játékosnak van olyan kódsora mely nem teljesen ismert.

# **2.2 Rendszer követelmények**

A játékhoz két komponensre, egy szerver és egy kliensre van szükség. Egy szerverre fizikai kapacitástól függően bárhány klienst és játékot ki tud szolgálni ezért a legtöbb esetben elegendő futtatni egy szervert. A kliens alkalmazás egy weboldal, amelyet a szerverrel azonos gépről kell hosztolni.

**Kliens követelmények**

A játékhoz a kliensnek nincs másra szüksége, csupán egy olyan eszközre, amely képes egy modern böngészőt futtatni (Firefox vagy bármilyen Chromium alapú böngésző). Ezután a játékot könnyedén el lehet érni a szerver IP címén a 4200-as porton. (Például <http://localhost:4200/>)

**Szerver követelmények**

A szerver futtatásához a [futtatandó kódot innen lehet letölteni](https://github.com/Zongi93/da-vinci). A szerver két komponenst kell futtasson, egy frontend komponenst és egy backend komponenst. Mindkét komponens futtatásához [Node.js](https://nodejs.org/en/) környezetre van szükség, a szervernek képesnek kell lenni ennek a környezetnek a támogatására.

*Forráskód:* [*https://github.com/Zongi93/da-vinci*](https://github.com/Zongi93/da-vinci) *Node.js:* [*https://nodejs.org/en/*](https://nodejs.org/en/)

A képen szöveg látható

Automatikusan generált leírás

**2.3. ábra. Helyesen telepített Node.js környezet**

Ha telepítve van *Node.js* környezet és az *npm* csomagkezelő a 2.3-as ábrán szereplő verziókkal (bizonyos esetekben magasabb verziók is megfelelhetnek, de a szoftver a fent mutatott verziókon volt futtatva és tesztelve), akkor a következő lépés két terminál megnyitása a projekt gyökérkönyvtárában és elindítani a szervert az alábbiak szerint. (*npm install* parancsot elegendő első indításnál kiadni)

**Első terminál:**

cd backend/

npm install

npm run serve

**Második terminál:**

cd frontend/

npm install

npm run host

A képen szöveg, monitor, képernyőkép, képernyő látható

Automatikusan generált leírás

**2.4. ábra. Két ablakban helyesen futó szerver**

***Megjegyzés:*** A való világban egy szerver futtatására a fent említett parancsok nem lennének elégségesek, mert bár a dolgunk így egyszerű és praktikus, a fent használt parancsok a fejlesztési és tesztelési folyamat megkönnyítésére valóak inkább. Való világban ily módon futtatni egy éles rendszerben használt szervert felelőtlen és veszélyes lenne.

Ha helyesen csináltunk mindent akkor a 2.4.-es ábrán látható eredményt kapjuk a terminálokban.

# **2.3 Felhasználói felület és program használata**

Ha rendelkezünk egy helyesen elindított szerverrel és egy eszközzel, ami lokális hálózaton eléri a szervert és képes futtatni egy modern böngészőt (pl Chrome), akkor navigáljunk a szerver IP címére és a 4200-as porton megtaláljuk a futó alkalmazásunkat. (Pl <http://192.168.50.159:4200/>)

***Megjegyzés:*** Amennyiben az alkalmazás nem érhető el bár látszólag fut a szerveren, akkor forduljunk rendszer karbantartójához segítségért, valószínűleg egy fordított proxy szolgáltatás fut a szerveren, ahova be kell regisztrálnunk az alkalmazásunkat, hogy engedélyezze annak elérését.

A képen szöveg látható

Automatikusan generált leírásAz alkalmazás sikeres elérése után az alábbi köszöntő képernyőt fogjuk látni, itt mást nem kell tennünk csak meg kell adnunk egy egyedi felhasználó nevet.

**2.5. ábra. Köszöntő oldal**

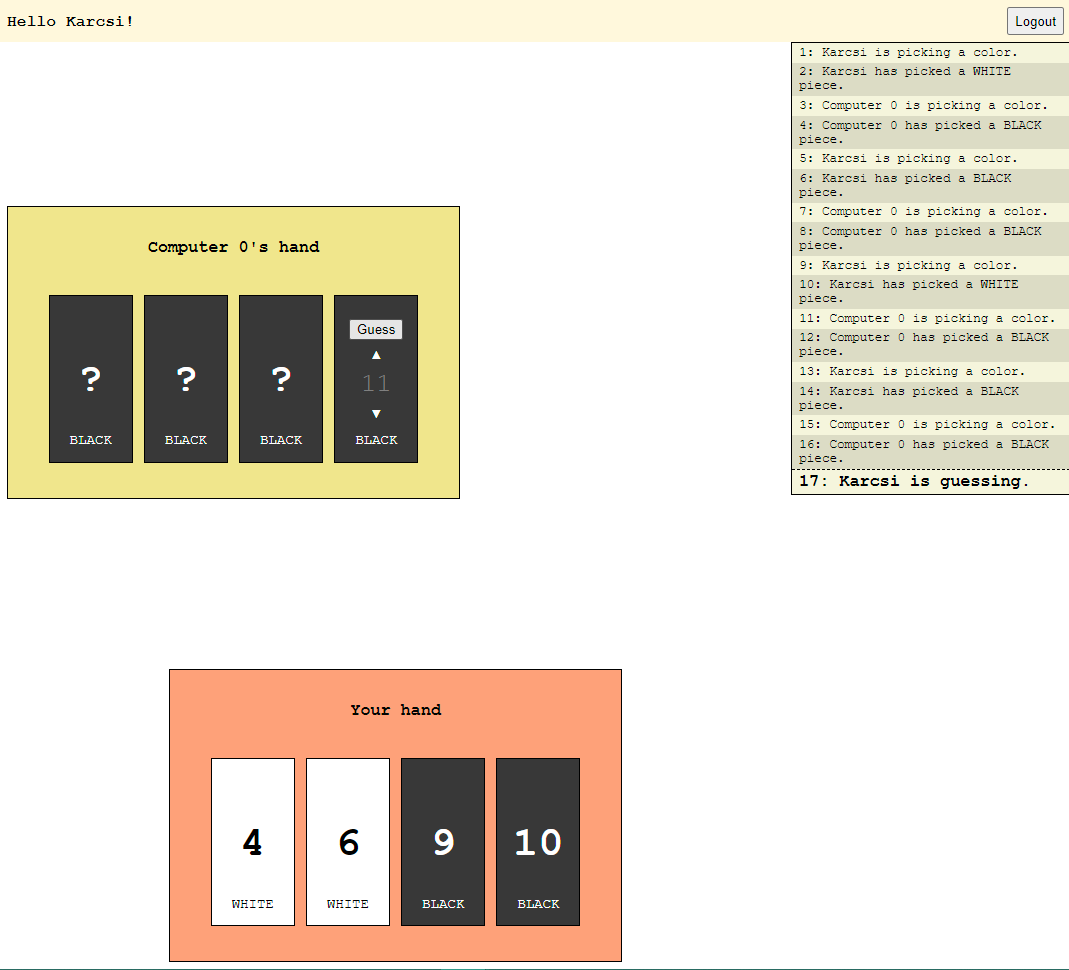
A következő nézetben az összes asztalt látjuk, ahova van lehetőségünk csatlakozni ott ezt egy gombnyomással meg is tehetjük. Ezen kívül, vagy ha nincs ilyen asztal még egy gombot látunk, amivel csatlakozás helyett egy új asztalt kreálunk magunknak.

**2.6. ábra. Asztal választó nézet**



**2.7. ábra. Választott asztal nézet**

Az asztal nézetben azt látjuk hányan ültünk eddig az asztalhoz, név szerint fel sorolva, ezen kívül látjuk hány gép ellenfelet adtak a játékhoz, látjuk a választott játékot is és a játékról pár alap információt. Gépi ellenfelek számát bármelyik játékos tudja módosítani ezen kívül bárki kérheti a játék megkezdését.



**2.8. ábra. Da Vinci játék nézet**

A Da Vinci játék nézetében a képernyő alján látjuk a saját kódsorunkat, jobb oldalt a játék chat-jét látjuk, amiben a korábbi lépések olvashatóak. A képernyő maradék részén pedig ellenfeleink kódsorait láthatjuk. Ha színt vagy extra akciót kell választanunk egy felugró ablakkal megtehetjük azt. Tippet úgy tehetünk, hogy a megfelelő lapkán kiválasztjuk a tippelni való számot és megnyomjuk a kártyán megjelenő gombot.

Ha bármikor megszakadna az internet kapcsolat játék közben, egy oldal frissítéssel/újra töltéssel azonnal visszatérhetünk a játék folytatásához. Játék végén a játék kiírja a nyertest és vissza kerülünk a tábla választó nézetre.

**3. fejezet**

# **Fejlesztői leírás**

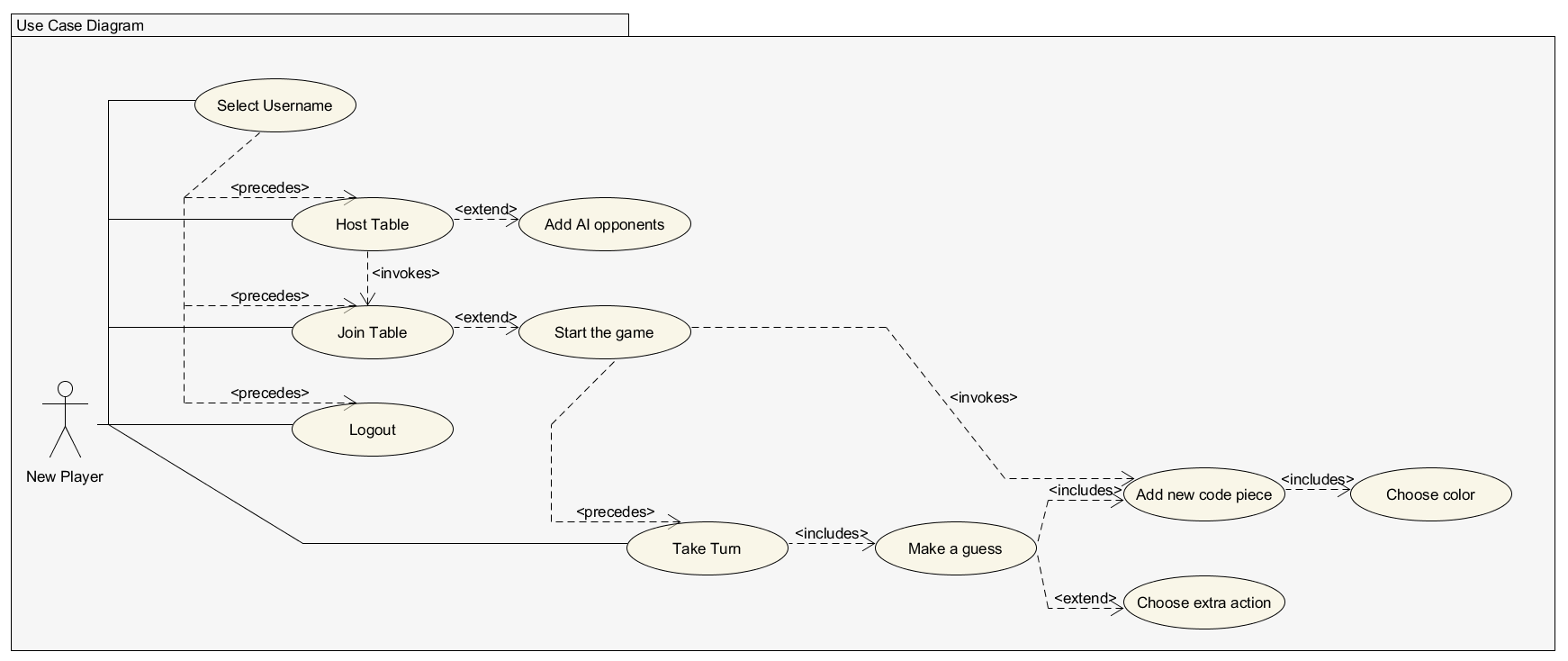
# **3.1. Elemzés**

# **3.1.1. Feladat leírás**

Jelen szakdolgozatom céljai:

* A 2.1-es fejezetben részletezett játékszabályokkal rendelkező társasjáték megvalósítása
* A játékot egyszerűen böngészőből, bármilyen speciális program telepítése nélkül tudjuk játszani.
* A játékot barátainkkal tudjuk játszani a helyi hálózaton különböző kliensekről.
* Legyünk képesek gépi ellenfeleket hozzáadni a játékhoz.
* A gépi ellenfelek nyújtsanak kihívást.

# **3.1.2. Funkcionális leírás**



**3.1. ábra. Felhasználói eset diagram**

# A képen asztal látható Automatikusan generált leírás

**3.2. ábra. Felhasználói történet**

A kész programnak a fenti 3.1-es és 3.2-es ábrákon mutatott funkciókkal kell rendelkeznie a felhasználók számára. Kiemelendő különbség a való élettel szemben, hogy a játék megvalósítása miatt nincs szükség arra, hogy a tippeket az adott játékos ellenőrizze, hanem ezt a szerver végzi mindenki helyett.

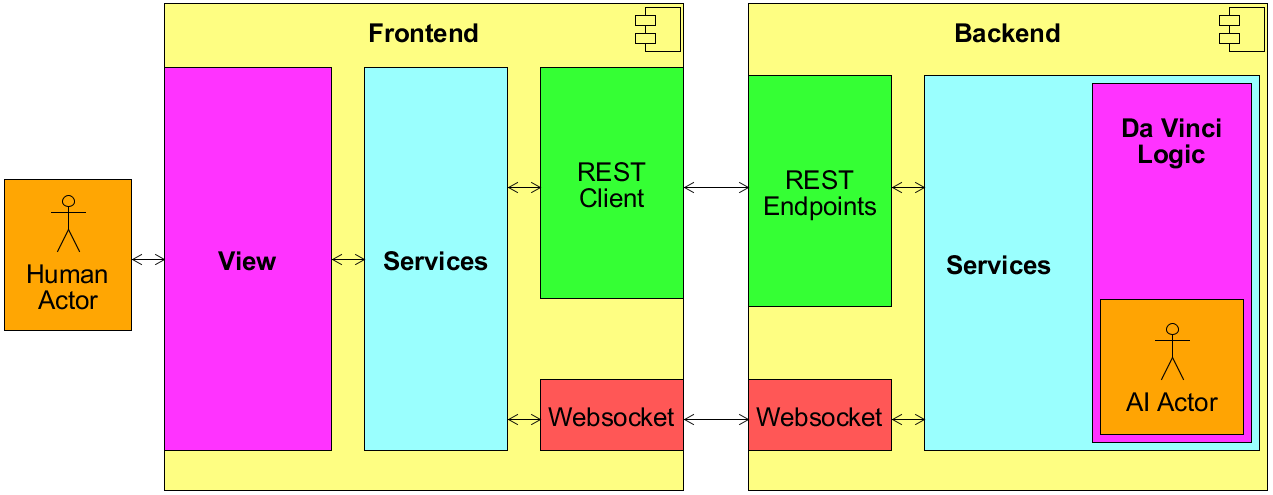
# **3.1.3. Nem-funkcionális leírás**

A tervezés során a következő szempontokat támasztjuk elvárásul a megvalósított program iránt. Ezen szempontokra fordítsunk kiemelt figyelmet.

Főbb szempontok:

* Az alkalmazás ne legyen platformfüggő.
* Az alkalmazás támogasson több képernyő méretet és beviteli módszert.
* Legyen az alkalmazás stabil az internet kimaradással szemben.
* Alapvető kiberbiztonsági megoldásokat valósítsunk meg elvi szinten.
* Célpont, hogy a megoldás könnyen bővíthető legyen más társasjátékokkal.
* Legyen elfedve, hogy a játékos gépi vagy emberi.
* A kliens és szerver közti kommunikációra használjunk REST végpontokat és websocket kapcsolatot.

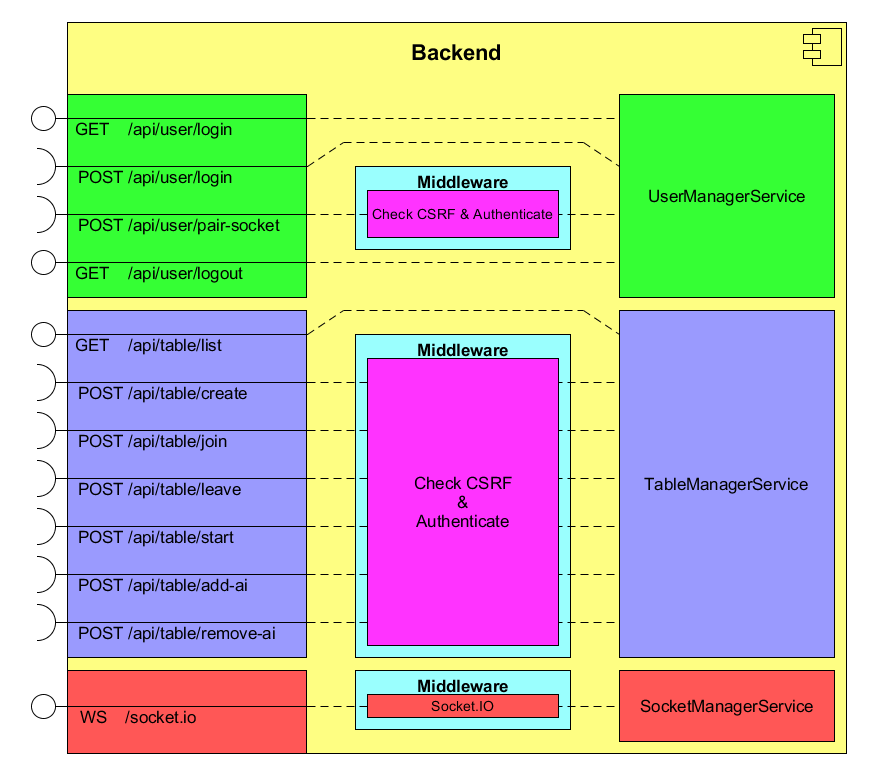
# **3.2. Tervezés**



**3.3. ábra. Magas szintű ábrázolása a teljes architektúrának**

A fenti ábrán látható az architektúra felületes ábrázolása. Mind a két komponensben hasonló struktúrát szeretnénk, külön kommunikációs réteget és külön adattárolás és logikáért felelős réteget. Ezen kívül a frontend biztosítja még a felhasználó felé a felület megjelenítését, és a felhasználói interakciók fogadását.

# **3.2.1. Architektúra**



**3.4. ábra. Magas szintű ábrázolása a backend architektúrának**

**A Backend komponens tervezésekor a filozófia a következő volt:**

* Az alapvető kommunikáció folyjon REST-en és csak értelemszerűen egészítsük ki a funkciókat websocket használatával.
* Minden végpont, amely pusztán csak adatot olvas (GET) legyen hozzáférhető authentikálás nélkül is.
* Használjunk *CSRF-token*-t annak ellenőrzésére, hogy kizárólag a saját frontendünk küldjön nekünk olyan kérést ami adatot is módosít (POST).
* A felhasználó megállapításához írjunk és olvassunk egy *http-only* beállítással rendelkező sütit (kliens csak olvasni képes).
* A megfelelő authentikáció és authorizáció ellenőrzését végezzük middleware komponensekben.
* Értelemszerűen bontsuk szét a feladat körök és felelősségek mentén a végpontokat külön csoportokra, és tegyük azoknak kezelését külön singleton osztályokba.
* Azonos csoportokban lévő végpontoknak legyen egy közös prefix-e az elérési útvonalban.
* A képen asztal látható

  Automatikusan generált leírásSzoruljunk minél kevésbé a frontend által szolgáltatott (ezáltal hamisítható) információkra. Például azt, hogy melyik felhasználó intézi az adott kérést ne egy paraméterként kérjük, hanem használjunk csak a szerver által módosítható sütiket, és használjuk azt a felhasználó azonosítására.

**3.5. ábra. Magas szintű ábrázolása a teljes architektúrának**

***Megjegyzés:****Ahol nincsen várt adat, ott a szerver elegendő információt tud szerezni a regisztrált sütikből, és abból, hogy melyik végpontra érkezett a kérés.*

# **3.2.2. Modell**

**3.6. ábra. A DaVinci játékhoz szükséges komponensek modellje**

***Megjegyzés:****Minden osztály minden publikus adattagja nyelv által támogatott readonly módban legyen, azaz az objektum konstruálásánál beállíthatóak egy értékre, de utána már mindenki csak olvasni tudja a beállított értéket.*

**Game::Table osztály**

Célja egy asztal minden funkcióját mely felelősségi körök mentén jól elkülöníthető megvalósítani. Lehetőséget kell biztosítania játékosok csatlakozására, gépi ellenfelek számának módosítására és a játék elindítására.

* *id : number* - Egyedi azonosítót biztosít az asztalok megkülönböztetésére
* *nOfAiOpponents : number* - Az asztalhoz hozzáadott gépi ellenfelek száma
* *players : User[]* - Az asztalhoz csatlakozott felhasználók listája
* *game : TableGame* - Az asztalnál játszani kívánt játék

A képen asztal látható

Automatikusan generált leírás

**3.7. ábra. Game::Table osztály függvényei**

**Game::User osztály**

Célja információt szolgáltatni a csatlakozott játékosokról.

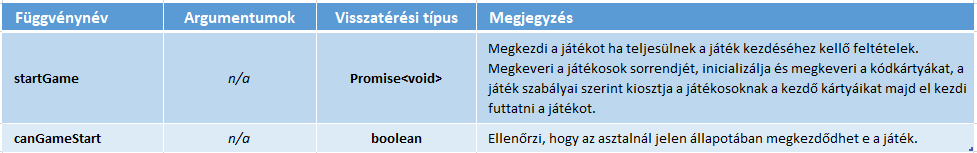
* *id : number* - Egyedi azonosítóval látja el a felhasználókat
* *userName : string* - Adott felhasználó neve



**3.8. ábra. Game::User osztály függvényei**

**Game::TableGame absztrakt osztály**

Célja alapvető információt szolgáltatni a különböző játékokról. Biztosítja annak lehetőségét, hogy a kiválasztott játék pontos ismerete nélkül meg tudjuk kérdezni, hogy elindítható e a játék, és lehetőséget ad a választott játéknak az elindítására is.

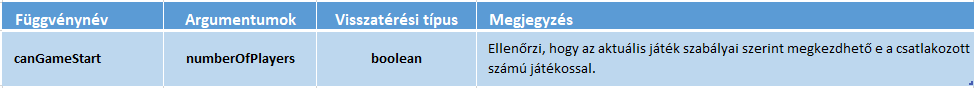


**3.9. ábra. Game::TableGame absztrakt osztály függvényei**

**Game::TableGameInfo osztály**

Információt szolgáltat a kiválasztott játékról, és annak feltételeiről a játék megkezdéséhez.

* *gameTitle : string* - A játék elnevezése
* *minPlayer : number* - A játék szabályai szerinti minimum játékos szám
* *maxPlayer : number* - A játék szabályai szerinti maximum játékos szám
* *aiSupport : boolen* - Támogatja e a játék megvalósítása gépi ellenfelek hozzáadását

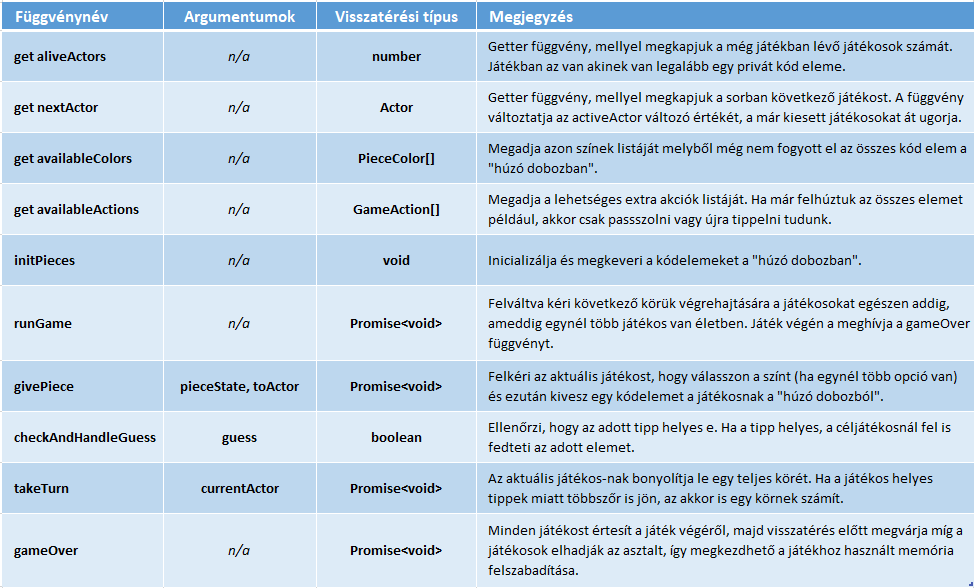


**3.10. ábra. Game::TableGameInfo osztály függvényei**

**DaVinci::TableGameService osztály**

Feladata a teljes Da Vinci játék ellenőrzött szabály szerinti lebonyolítása, az ő felelőssége minden ami ehhez szükséges, tehát például a kód elemek inicializálása, a játékosok sorrendjének megállapítása, a körök lebonyolítása, a játékos lépésének ellenőrzése, nyilvántartani a még életben lévő játékosokat, kódelemek húzása kérésre megfelelő színben és a játék vége állapot észlelése és kezelése.

* *activeActor : number* - Az aktuális játékos indexe az actors listában
* *freePieces : GamePiece[]* - Azon kód elemek halmaza amelyeket még nem húzott fel egyik játékos sem.
* *actors : Actor[]* - A játékban résztvevő játékosok listája

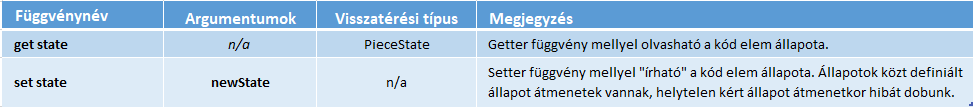


**3.11. ábra. DaVinci::TableGameService osztály függvényei**

**DaVinci::GamePiece osztály**

Feladata tárolni az egy kódelemre vonatkozó minden információt és végrehajtani annak ellenőrzött állapot átmeneteit a kód elem életciklusa alatt.

* *id : number* - Egyedi azonosítója a kódelemnek
* *number : number* - A kódelem szám értéke (Alap szabályok szerint 0 és 11 közötti)
* *color : PieceColor* - A kódelem színé (Alap szabályok szerint fehér és fekete)
* *state : PieceState* - A kódelem állapota (nem ismert, privát, publikus)



**3.12. ábra. DaVinci::GamePiece osztály függvényei**

**DaVinci::Guess osztály**

Feladata az egy tipphez tartozó minden adatot tartalmazni. Külön logikát nem végez.

* *actorId : number* - A cél actor egyedi azonosítója, akinek az elemére tippelünk
* *position : number* - Az kódelem indexe a sorban
* *value : number* - A tippelt érték

**DaVinci::Actor absztrakt osztály**

Felelőssége a kódsor rendben tartása, új elem megfelelő helyre való beszúrása, azon tippek ellenőrzése ahol ezen *Actor* kódelemére vonatkozott a tipp. Továbbá felelősség a játék által kért lépéseknek végrehajttatása. Ezen funkcióra absztrakt függvényeket használunk, melyek különböző játékos típusoknál különféle implementációt kapnak, ezáltal kiszolgálva a játékos típus egyedi igényeit.

* *id : number* - Egyedi azonosító az actoroknak
* *ownedPieces : GamePiece[]* - Az actor által birtokolt kódelemek rendezett tömbje.

A képen asztal látható

Automatikusan generált leírás

**3.13. ábra. DaVinci::Actor absztrakt osztály függvényei**

**DaVinci::Player osztály**

Az *Actor* absztrakt osztály megvalósítása emberi játékos számára. Feladata a kért lépések kommunikálása websocketen keresztül a kliens felé és a válasz lépések továbbítása a *TableGameService* felé. További feladata esetleges kapcsolat bontás esetén szinkronizálni az állapotokat. A tényleges állapotot is ő tartja számon, a kliens egyoldalúan/szabályokat meg kerülve azt nem tudja megváltoztatni. Részletesebb ismertetésre a 3.2.4-es fejezetben kerül sor.

**DaVinci::Computer osztály**

Az *Actor* absztrakt osztály megvalósítása gépi ellenfél számára. A működés részletes ismertetésére egy külön (3.2.5) fejezetben kerül sor.

# **3.2.3. Nézet tervek**

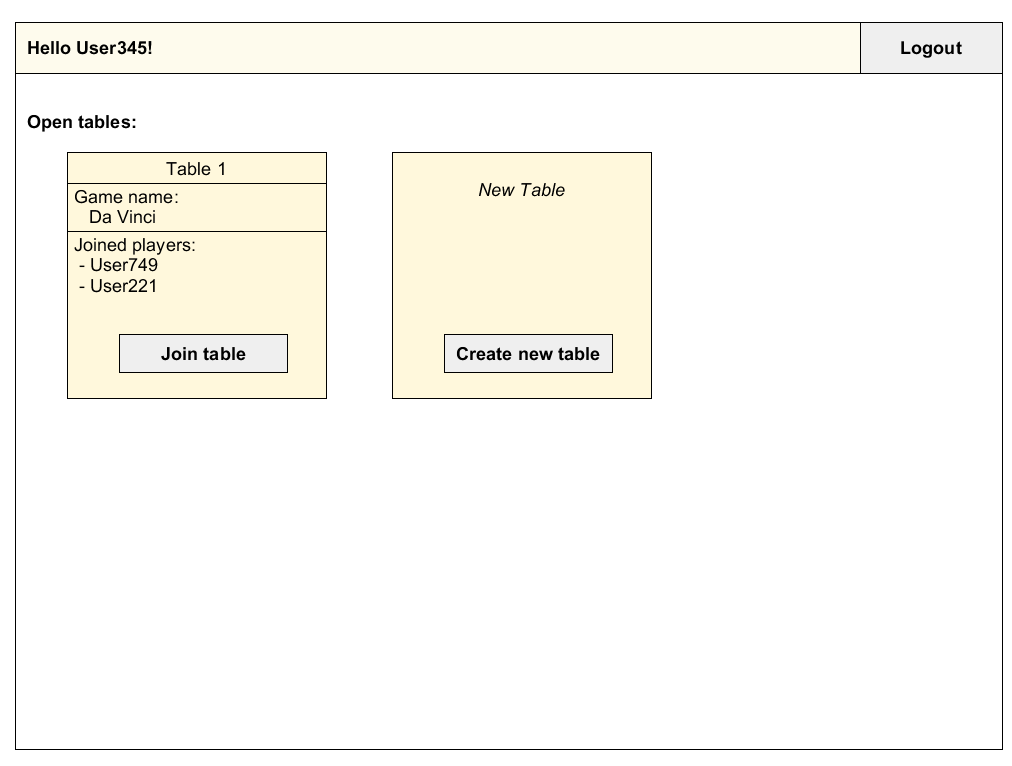
**Köszöntő oldal**

Az oldal első megnyitásakor a köszöntő oldalt fogjuk látni, ennek pusztán annyi lenne a célja, hogy a felhasználóhoz választhasson egy egyedi felhasználó nevet. Belépés után a felhasználó az asztal választó nézetre kerülne automatikusan.



**3.14. ábra. Köszöntő oldal**

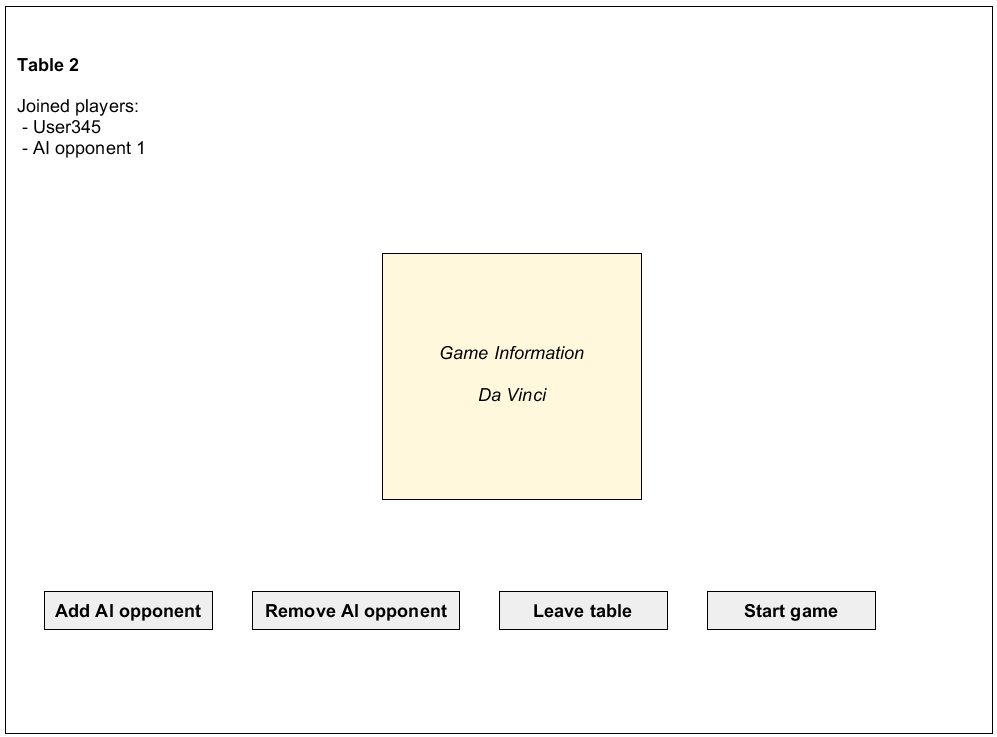
**Asztal választó nézet**

Ezen a nézeten jelenítenénk meg az elérhető asztalokat, sorolnánk fel az azokhoz csatlakozott játékosokat, és adnánk lehetőséget, hogy a jelenlegi felhasználó új asztalt hozhasson létre.

**3.15. ábra. Asztal választó nézet**

**Választott asztal nézet**

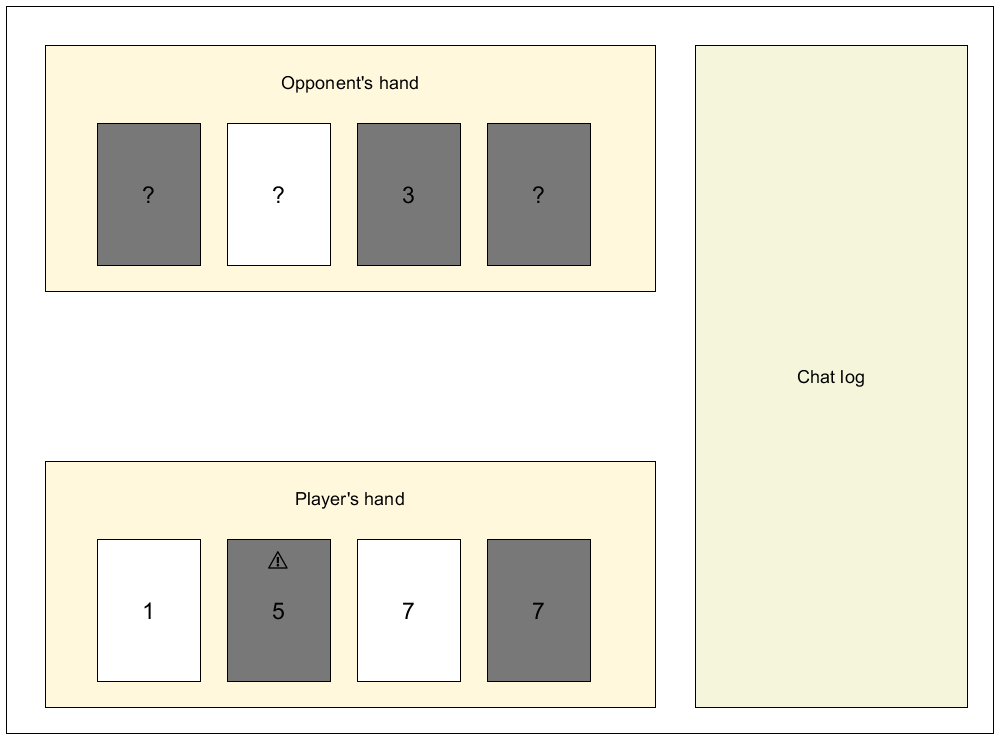
Ezt a nézetet látnánk akkor, ha csatlakozunk egy asztalhoz és várunk arra, hogy az elinduljon. E tudjuk indítani a játékot és tudunk gépi ellenfeleket hozzáadni vagy elvenni az asztaltól.



**3.16. ábra. Választott asztal nézet**

**Da Vinci nézet játék közben**

Alább a Da Vinci játék tervezett nézete látható, a saját kódsorunk a nézet alján, egy chat log a nézet jobb szélén és az ellenfelek kódsorai a maradék rendelkezésre álló helyen látható. Tippeléshez a kódkártyákon elhelyezett extra felületi elemeket használhatjuk majd. Extra akció vagy szín választáshoz egy felugró ablakot fogunk mutatni az aktuális játékosnak.



**3.17. ábra. Da Vinci nézet játék közben**

# **3.2.4. Játék megvalósításának terve**

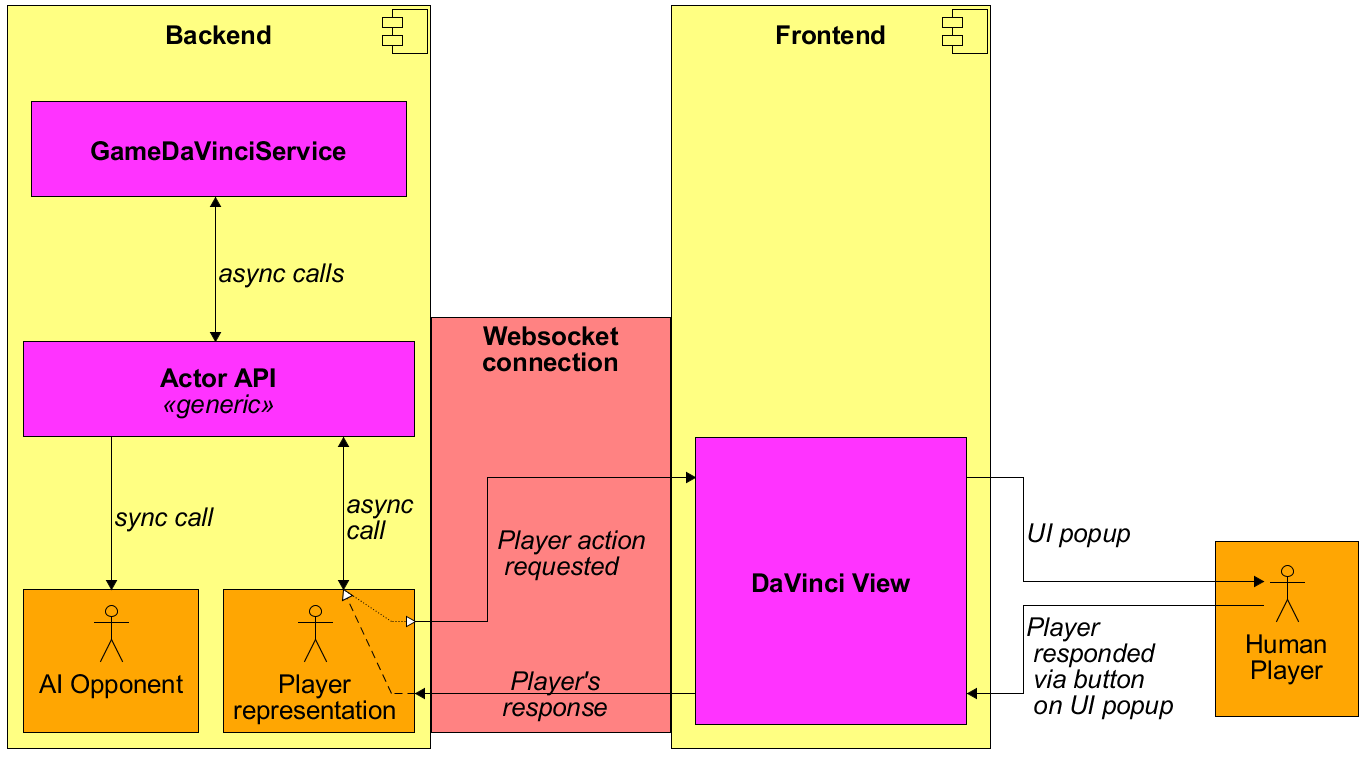
Mivel a játék megköveteli a folyamatos kétirányú kommunikációt, ezért elengedhetetlen a websocket technológia használata. Legyen minden kliens felé egy egyedi websocket kapcsolatunk és legyünk képesek klienseknek különböző üzeneteket küldeni. A rendszerezett működéshez használjunk úgynevezett topik alapú kommunikációt a websocket csatornán, ezzel az érdeklődési körök mentén elvágva jól elkülöníthető eseményekre lehet lebontani a kommunikációt. Topik alapú kommunikációnál általában a websocket kiépítéséért felelős service fogad minden üzenetet és a regisztrált topikok alapján delegálja az üzenetek feldolgozását az arra feliratkozott komponenseknek.

A képen asztal látható

Automatikusan generált leírás

**3.18. ábra. Websocketen folyó topik alapú kommunikáció részletezése**

A 3.18-as ábrán látható táblázatban a teljes websocketen folyó kommunikáció látható. Websocket kapcsolatot használjuk a teljes tábla lista szinkronizálására minden kliensnél, és továbbá használjuk a teljes játék lebonyolítására. Olyan topikok mint *guess*, *pick-color*, *take-extra-action* és *game-over* jól megfeleltethetőek a *DaVinci::Actor* osztály aszinkron függvényeinek, melyek valamilyen interakciót várnak a játékostól. A játék logika által kért lépések ily módon vannak kommunikálva a megfelelő kliens felé és ugyanezen topikok használatával is válaszol ezen kérésekre a kliens.



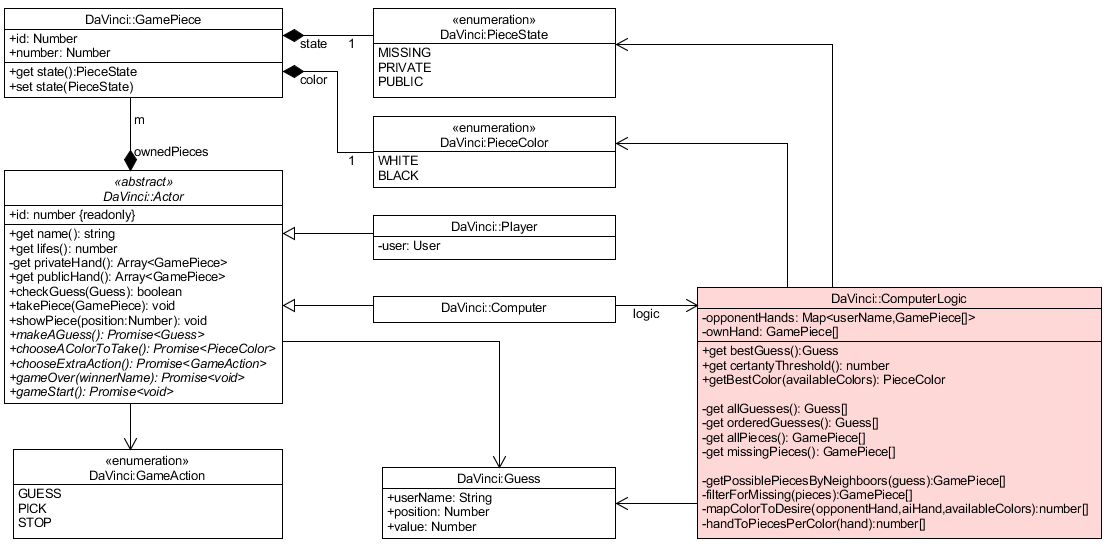
**3.19. ábra. Felhasználót igénylő lépések lebonyolitása**

Felhasználót igénylő interakciók lebonyolítása szemléletesen a fenti ábrán látható. A *GameDaVinciService* meghívja az *Actor* absztrakt osztály egyik függvényét, az alapján, hogy adott *Actor* gépi ellenfél e vagy rendes játékos, annak függvény implementációjában különbség lesz. Gépi ellenfél esetén azonnal megválaszoljuk a kérést, emberi ellenfélnél websocketen a megfelelő topik használatával jelezzük a kliens felé, hogy továbbítsa a kérdést a felhasználó felé. A kliens ekkor a játék felületén valamilyen módon jelzi a kérelmet és lehetőséget biztosít annak megválaszolására is, például egy felugró ablakkal, amely mutatja a választható színeket mint gombok. Ha felhasználó választott, akkor a kliens a kéréssel azonos topikot használva válaszol a kérésre a szervernek melyet a *Player* osztály fogad, és válaszként továbbítja a *GameDaVinciService* felé ami ekkor értékeli a választ és a következő lépéssel folytatja a játék lebonyolítását.

# **3.2.5. Gépi ellenfél**

A gépi ellenfél tervezésekor a filozófia a következő volt:

* Minden döntésnél állítsunk elő egy valószínűségi változót, ezzel el kerülve a determinisztikus működést.
* A valószínűségi változók előállításához használjunk erős heurisztikát, amivel a játék állapotát jellemezni tudjuk.
* A heurisztikák oly módon jellemezzék a helyes lépést, hogy ha minél helyesebbnek gondolunk egy lépést, annál jobban tendáljon a valószínűségi változónk annak a lépésnek irányába.



**3.20. ábra. Az osztály diagram kiegészítése a gépi ellenfél megvalósításához**

A képen szöveg látható

Automatikusan generált leírás

**3.21. ábra. DaVinci::ComputerLogic osztály függvényei**

A Gépi ellenfél logikáját egy külön *DaVinci::ComputerLogic* osztályban valósítottuk meg. A *DaVinci::Computer* osztály minimális felelősséggel rendelkezik szemben a *DaVinci::Player* osztállyal, hiszen itt csupán egy szinkron időben futtatott függvény hívással választ tudunk adni a kérésre míg *DaVinci::Player* osztályban a websocketen való kommunikáció miatt rengeteg extra feladatunk is volt. Például játék vége esetén egy játékostól elvárjuk, hogy erre reagáljon, de ilyen elvárásunk gépi ellenfél felé természetesen nincsen.

A játék szempontjából három esetben van szükségünk a *DaVinci::ComputerLogic* által megvalósított kiértékelő logikára:

* Ha tippet kell tennie
* Ha színt kell választania
* Ha extra akciót kell választania

**A gépi ellenfél tipp tételének ötlete:**

Vegyük az összes kód elemet melyek az ellenfelek kódsoraiban vannak még privát állapotban, a már felfedett kódelemek felhasználásával és a saját kódsorunk felhasználásával számoljuk ki ezen összes elemekre a szabályoknak megfelelő lehetséges értékeket. Vegyük azt az elemet, amelynél a legkevesebb opció van és tegyünk egy tippet az egyik opciót felhasználva.

**Példa:**

* Ellenfelünk 2. kódelemének értékei minden információnk és a szabályok szerint lehet: 4,5,6
* Minden más eleménél három darabnál több lehetséges érték opciót állapítottunk meg.
* Tegyünk egy tippet a 4,5,6 opciók egyikét felhasználva az ellenfelünk 2. kódelemére.

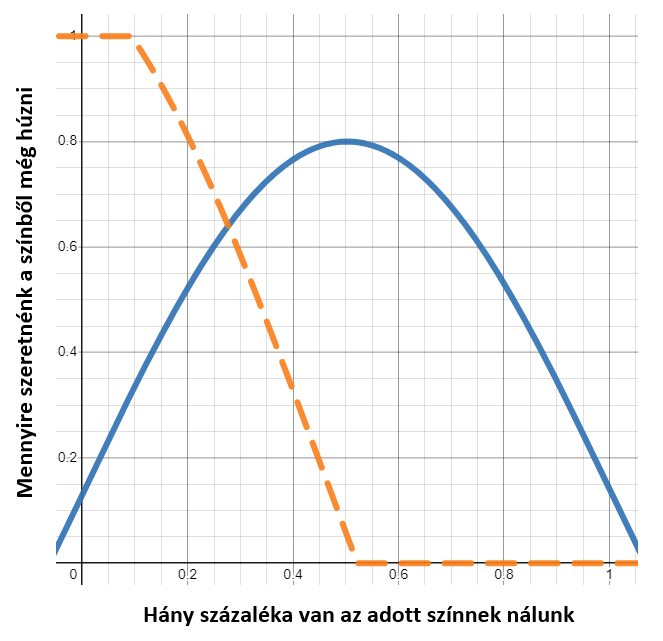
**E megoldás hiányosságai:**

* Nem használ fel információt a játék korábbi lépéseiből ("nincs memória")

**A gépi ellenfél szín választásának ötlete:**

Vegyük a játékosok kódsorait és számoljuk meg melyikük melyik színből mennyi információval rendelkezik. Ezután a *mapColorToDesire* függvény segítségével állapítsuk meg, melyik szín adhat nekünk több információt vagy jelent nagyobb veszélyt. Figyeljünk arra, hogy ha minden elem egy színből fel van húzva csupán két játékos által, akkor ők ketten könnyedén kitalálhatják egymás kódsoruk adott színben lévő elemeiknek értékét.

A *mapColorToDesire* függvényben alkalmazzunk heurisztikát:



**3.22. ábra. Heurisztikát segítő függvények a színválasztáshoz**

* Abban az esetben, ha egy színből még sok elem van a "húzódobozban", akkor azt a színt tartsuk érdekesnek, amiből a legtöbbel rendelkezünk ügyelve arra, hogy viszont ne húzzuk fel az összes elemet a színből. (3.22-es ábrán kék függvény)
* Abban az esetben, ha egy színből már kevés elem van a húzódobozban, akkor ha nekünk nagy a kitettségünk ebből a színből akkor ne akarjuk belőle húzni, viszont ha másik játékos kitettsége nagy és nekünk elenyésző, akkor törekedjünk felhúzni a maradék elemeket a színből, hiszen ha minden elem játékban van egy színből akkor könnyedén megfejthetjük a teljes kódsorát az ellenfelünknek. (3.22-es ábrán narancssárga függvény)

Miután minden színre és játékosra kiszámoltuk a húzási vágy értékét, akkor átlagoljunk, normalizáljunk és a színekből készítsünk egy valószínűségi változót majd annak használatával válasszunk színt.

**Példa:**

* Vegyünk egy két játékos játszmát, ahol színt kell választanunk egy helyes tippünk után.
* Fehér színből nálunk 5 darab van, ellenfelünknél 2, a dobozban még 5.
* Fekete színből nálunk 3 darab van, ellenfelünknél 6, a dobozban még 3.
* Fehér színből mi az elemek 5/12 százalékát birtokoljuk (~0.4116), a dobozban még elegendő elem van tehát a kék függvényt használva a húzási vágyunk kb 77%.
* Fekete színből mi az elemek 3/12 százalékát birtokoljuk (0.25), a dobozban már kevés elem van tehát a narancssárga függvényt használjuk: 70%
* Mivel csak egy ellenfelünk van ezért az átlagoló lépésre ellenfelenként számolt értékekre most nincs szükség.
* Normalizáljuk a kapott értékeket és készítsünk egy valószínűségi változót: 52.3%-ban húzzunk fehéret, 47.7%-ban feketét.
* Dobjunk egy kockával véletlenszerűen generálva egy értéket 0 és 100 között, legyen eredménye x, ha x értéke 52.3% alatt van, akkor fehéret kérjünk, ha nagyobb akkor feketét.

**E megoldás hiányosságai:**

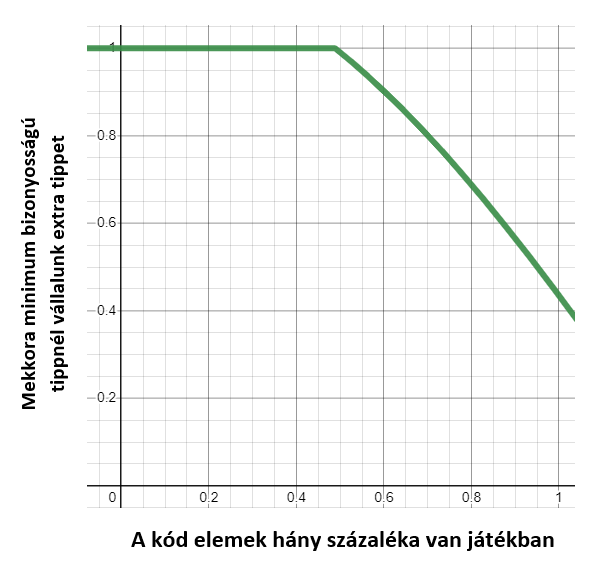
* Nem veszi figyelembe, hogy a húzott kódelem privát vagy publikus lesz-e.
* Talán pontosabb képet kaphatnánk, ha minden húzható elemre kiszámolnánk, hogy tippelésnél melyik húzott kártya birtokában könnyebbülne meg legjobban a dolgunk, ezt színekre összegeznénk és ennek fényében döntenénk a színről.

**A gépi ellenfél extra akció választásának ötlete:**

Számoljuk ki, hogy a jelen helyzetben milyen tippet tennénk, nézzük meg milyen magabiztosak vagyunk ebben a tippben, ha biztosabbak vagyunk a tippben, mint amekkora maximális kockázatot aktuálisan vállalnánk, akkor válasszuk az extra akciók közül a tipp megtételét, ha túl nagy kockázatnak véljük akkor kérjünk kód elemet helyette inkább.

Magabiztosságot számolni a legjobb tippünkhöz egyszerű, csak meg kell nézni hány lehetséges értéket vehet fel a tippelni kívánt elem és vesszük a darabszám reciprokát.

A *certantyThreshold* függvény, avagy kockázatvállaló késség számolásához használt heurisztika:



**3.23. ábra. Heurisztikát segítő függvény a kockázatvállaláshoz**

Ha még a játék elején vagyunk, akkor kevés kockázatot kell vállalnunk, a játék vége felé viszont már nagyobb kockázatot is vállalunk. A kapott értéket még minimálisan eltoljuk annak függvényében, hogy jelenleg hány életünk van, azaz hány olyan elem van a kódsorunkban, amelyek még privát állapotban találhatóak.

**Példa:**

* Sikeresen tettünk egy tippet, választhatunk, hogy extra akcióként tovább tippelünk vagy húzunk egy kód elemet.
* Mivel a játék csak most kezdődött, ezért csak az elemek 30% van játékban. (8 elem a 24-ből)
* A *certantyThreshold* függvényünk 100%ban biztos tippnél vállalná a kockázatot.
* Mivel ilyet a játék elején valószínűleg nem ismerünk, inkább válasszuk a kód elem húzását extra akcióból.

**E megoldás hiányosságai:**

* A gépi ellenfél talán túl konzervatív, nem elég merész.
* A gépi ellenfél csak becsüli azt, hogy veszíthet-e a következő körben, ténylegesen nem "szimulálja le". Valószínűleg ezáltal pontosabb eredményt kaphatnánk pedig.

# **3.3. Megvalósítás**

# **3.3.1.Felhasznált technológiák**

**Backend:**

A backend komponens megvalósításához *Express.js* keretrendszert használtunk, amely a REST szerver megvalósításában játszott nagy szerepet. A websocket szerver kezeléséhez *Socket.IO* szervert használtunk. Ezen kívül érdemes még megemlíteni az *RxJS* könyvtárat mely az adatok "push" alapú elven való folyását tette lehetővé.

A képen szöveg látható

Automatikusan generált leírás

**3.24. ábra. Express.js könyvtár használatából egy részlet**

Szépen látható, hogy a megvalósítás mennyire hasonlít a 3.4-es ábrán mutatott tervekhez képest.

**Push alapú adat mozgás:**

Mind a frontend és a backend komponens is hasonló "push" alapú elvet alkalmaz adatok mozgatására. Ezt az általam nagyon kedvelt *RxJS* könyvtár használatával érjük el. Hogy szemléletes példát adjak, ezen könyvtár használatával olyan getter függvényeket deklarálhatunk, amelyek önmaguktól szólnak, ha változott az értékük. Minden függvényben vagy egyéb helyen, ahol ezen getterek értékeit felhasználjuk, ott továbbra is szemléletesen, automatikusan függőségek regisztrálódnak ezen getterekre. Ezáltal ha a getterek értékeik bizonyos események bekövetkeztekor frissülnek (például REST hívásban regisztrálunk egy új felhasználót), a rájuk függő azaz a belőlük származtatott minden adat is automatikusan frissül.

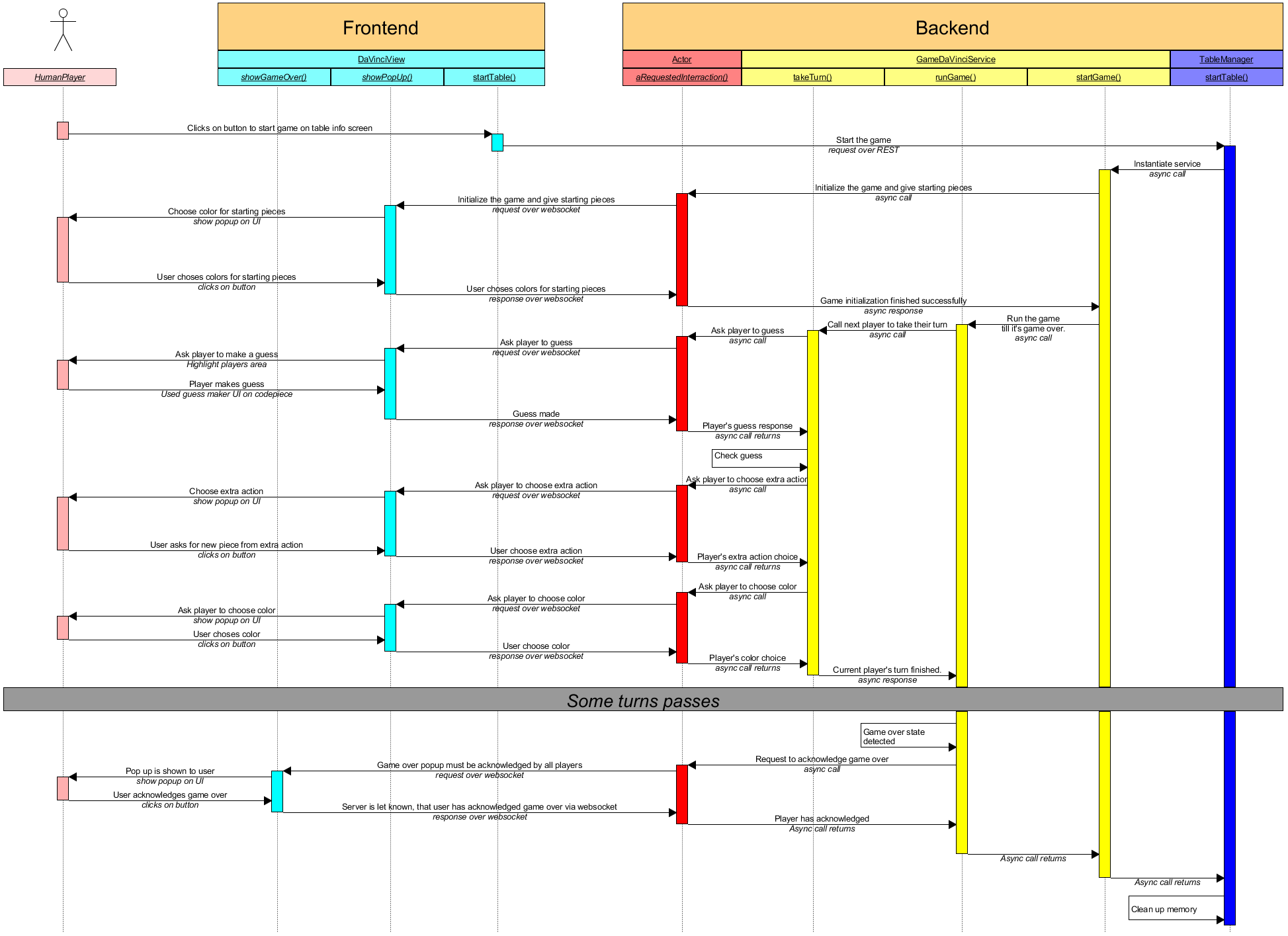
**Aszinkron függvények előnyei:**

A képen szöveg látható

Automatikusan generált leírásA javascript által támogatott *Promise* generikus osztály segítségével könnyedén tudunk aszinkron függvényeket futtatni és az *await* kulcsszóval "callback" függvények regisztrálása nélkül tudjuk azokat helyben kezelni. Aszinkron függvények használata azzal az előnnyel jár, hogy nem kell tudnunk, hogy egy folyamat mennyi ideig tart, mi az eredménnyel ugyanúgy tudunk dolgozni mintha szinkron időben futnának a függvények, viszont amíg várunk a válaszra addig a javascript motor a háttérben tud foglalkozni más feladatokkal, nem blokkoljuk a fő szálat.

**3.25. ábra. Express.js könyvtár használatából egy részlet**

A fenti példán szépen látható a jól strukturált aszinkron függvények használatából fakadó előny. A start függvény elkezdi a játékot futtatni. Ez egy aszinkron függvény mely akkor tér vissza, ha vége van a játéknak. Könnyedén tudunk a már játék vége eseményre is reagálni és szépen felszabadítani a memóriát a befejezett játék után. Az említett *await* kulcsszót a hibakezelés miatt nem használtuk.



**3.26. ábra. A játék lefolyásának részleges folyamatábrája**

**A képen szöveg látható

Automatikusan generált leírás**A 3.26-os ábrán lévő folyamatábrán szépen látható, hogy a játék különböző fázisaiban milyen lépések következnek milyen sorrendben. Jól megfigyelhető, hogy a nyelv által támogatott aszinkronitás miatt milyen könnyű lett a dolgunk több esetben is. Például A játékban egy kört lebonyolító *takeTurn* függvény, ha kérni szeretné az aktuális játékost egy lépésben akkor azt oly módon kezelheti le mintha az szinkron időben hajtódott volna végre, miközben a háttérben kommunikáltunk a klienssel, a felhasználó reagált a kérésre, az vissza érkezett a websocket kapcsolaton, hogy majd feldolgozásra kerüljön sor.

**3.27. ábra. Részlet a GameDaVinciService::takeTurn függvényéből**

A 3.27-es ábra és a 3.26-os folyamatábra együtt azt próbálja bemutatni, hogy a komponensek megfelelő tervezése és együttműködése miatt kódban mégis milyen egyszerűen tudunk dolgozni az emberi felhasználó által szükségszerűen behozott aszinkronitással.

**Frontend:**

A kliens Angular 7-es keretrendszert használva lett megírva, betartva a keretrendszer által elvárt és támogatott konvenciókat, azaz komponens alapú felületet készítettünk, ahol a különböző nézetek melyeket a 3.2.3-as fejezetben részleteztünk mind külön aloldalakként szerepelnek. A backend komponenshez hasonló elven, itt is hangsúlyt kapott a moduláris megoldás, hogy lehetőséget adjunk további társasjáték könnyű hozzáadására. Külön logikát a kliens nem tartalmaz, pusztán csak felületet biztosít a felhasználó számára, hogy a backend által szolgáltatott adatokat megjelenítse, és a felhasználó megfelelő interakcióit továbbítsa a backend felé. A websocket kapcsolat kiépítésére hasonlóan *Socket.IO* könyvtárat használtuk, és az adatok rendezett mozgatására pedig az *RxJS* könyvtárat.

# **3.3.2. Esetleges eltérések a tervtől**

A témabejelentőben ígért-el szemben a gépi ellenfél nem használ megerősítés tanulási technikákat. Döntési logikájában matematikai függvények vannak felhasználva, ezen függvények paramétereit a kívánt hatás irányába könnyedén módosítani lehet, de sajnos ez nem automatikus.

Ezen kívül a megvalósított program a tervtől több helyen nem tér el.

# **3.4. Tesztelés**

# **3.4.1. Egység tesztek**

# **3.4.2. Végfelhasználói tesztesetek**



**3.28. ábra. Végfelhasználói tesztesetek**

Fekete doboz teszteseteket a felhasználói történetek (3.2. ábra) felhasználásával állítottuk elő. Megadott eseménynél részletezzük, hogy milyen eredményt várunk.

# **3.5. Lehetséges jövőbeni fejlesztések**

* Ellenőrizzük le, hogy a pair-socket végpont mindenképpen a legjobb megoldás, amit választhattunk.
* Jelenítsük meg a játékban azon információt, hogy ki milyen elemre tett milyen tippet és helyes vagy helytelen volt e.
* Adjunk lehetőséget az opcionális kötőjel szabály beállítására a játék indításakor.
* Küszöböljük ki a 3.2.5-ös fejezetben taglalt hiányosságait a gépi ellenfél megoldásának.
* További társas játékok implementálása.

**4. fejezet**

# Irodalom jegyzék

A linkek 2022.02 - 2022.06 között voltak elérhetőek

* A társas játékot ma már sajnos nagyon kevés helyen lehet meg kapni, de itt még talán igen: <https://www.aliexpress.com/item/32715285100.html>
* Angular dokumentációja: <https://angular.io/>
* Express.js dokumentációja: <https://expressjs.com/>
* Socket.io dokumentációja: <https://socket.io/>
* RxJS dokumentációja: <https://reactivex.io/>