



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Informatikai Kar

Programozáselmélet és Szoftvertechnológiai Tanszék

Da Vinci

Dr. habil. Gregorics Tibor
Tanszékvezető Egyetemi Docens

Gyarmati Károly Mátyás
Programtervező
Informatikus BSC

Budapest, 2022

Tartalomjegyzék

1. Bevezetés	3
2. Felhasználói leírás	4
2.1. Játékszabályok	4
2.2. Rendszer követelmények	6
2.3. Felhasználói felület és program használata	8
3. Fejlesztői leírás	12
3.1. Feladat leírás	12
3.1.1. Funkcionális leírás	13
3.1.2. Nem-funkcionális leírás	15
3.2. Tervezés	16
3.2.1. Architektúra	17
3.2.2. Modell	19
3.2.3. Nézet tervek	26
3.2.4. Hálózati kapcsolatok terve	30
3.2.5. Gépi ellenfél	32
3.3. Megvalósítás	40
3.3.1. Felhasznált technológiák	40
3.3.2. Esetleges eltérések a tervtől	44
3.4. Tesztelés	45
3.4.1. Egység tesztek	45
3.4.2. Végfelhasználói tesztesetek	49
3.5. Lehetséges jövőbeni fejlesztések	50
4. Irodalom jegyzék	51

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 játéktérrel, 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 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 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 arra, 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 asztalt 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 [forráskódot innen lehet letölteni](#). A szerver két komponenst kell futtasson, egy frontend komponenst és egy backend komponenst. Mindkét komponens futtatásához [Node.js](#) környezetre van szükség, a szervernek képesnek kell lenni ennek a környezetnek a támogatására.

Forráskód: <https://github.com/Zongi93/da-vinci>

Node.js: <https://nodejs.org/en/>

```
dearo@DESKTOP-BJET5DH MINGW64 ~/work/da-vinci (master)
$ node -v
v12.18.3

dearo@DESKTOP-BJET5DH MINGW64 ~/work/da-vinci (master)
$ npm -v
6.14.8
```

2.3. ábra. Helyesen telepített Node.js környezet

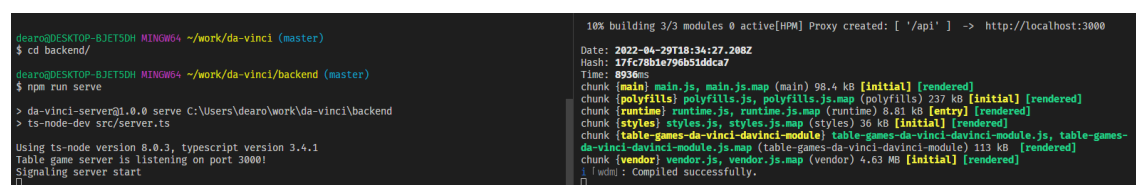
Ha telepítve van *Node.js* környezet és az *npm* csomagkezelő a 2.3-as ábrán szereplő verziókkal (bizonyos esetekben magasabb verziók is megfelelhetnek, de a szoftver a fent mutatott verziókon volt futtatva és tesztelve), akkor a következő lépés két terminál megnyitása a projekt gyökérkönyvtárban és a szerver az alábbi instrukciók alapján való elindítása. (*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
```



```
dearo@DESKTOP-BJET5DH MINGW64 ~/work/da-vinci (master)
$ cd backend/
dearo@DESKTOP-BJET5DH MINGW64 ~/work/da-vinci/backend (master)
$ npm run serve
> da-vinci-server@1.0.0 serve C:\Users\dearo\work\da-vinci\backend
> ts-node-dev src/server.ts

Using ts-node version 8.0.3, typescript version 3.4.1
Table game server is listening on port 3000!
Signaling server start
[]

10% building 3/3 modules 0 active[HPM] Proxy created: [ '/api' ] -> http://localhost:3000
Date: 2022-04-29T18:34:27.208Z
Hash: 17fc78b1e796b51ddca7
Time: 893ms
chunk [main] main.js, main.js.map (main) 98.4 kB [initial] [rendered]
chunk [polyfills] polyfills.js, polyfills.js.map (polyfills) 237 kB [initial] [rendered]
chunk [runtime] runtime.js, runtime.js.map (runtime) 8.81 kB [entry] [rendered]
chunk [styles] styles.js, styles.js.map (styles) 36 kB [initial] [rendered]
chunk [table-games-da-vinci-davinci-module] table-games-da-vinci-davinci-module.js, table-games-da-vinci-davinci-module.js.map (table-games-da-vinci-davinci-module) 113 kB [rendered]
chunk [vendor] vendor.js, vendor.js.map (vendor) 4.63 MB [initial] [rendered]
[ ] [webpack]: Compiled successfully.
```

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 valók 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, annak ellenére hogy 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.

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

Please enter your username:

Login

Please log in to join tables or to create a new one.

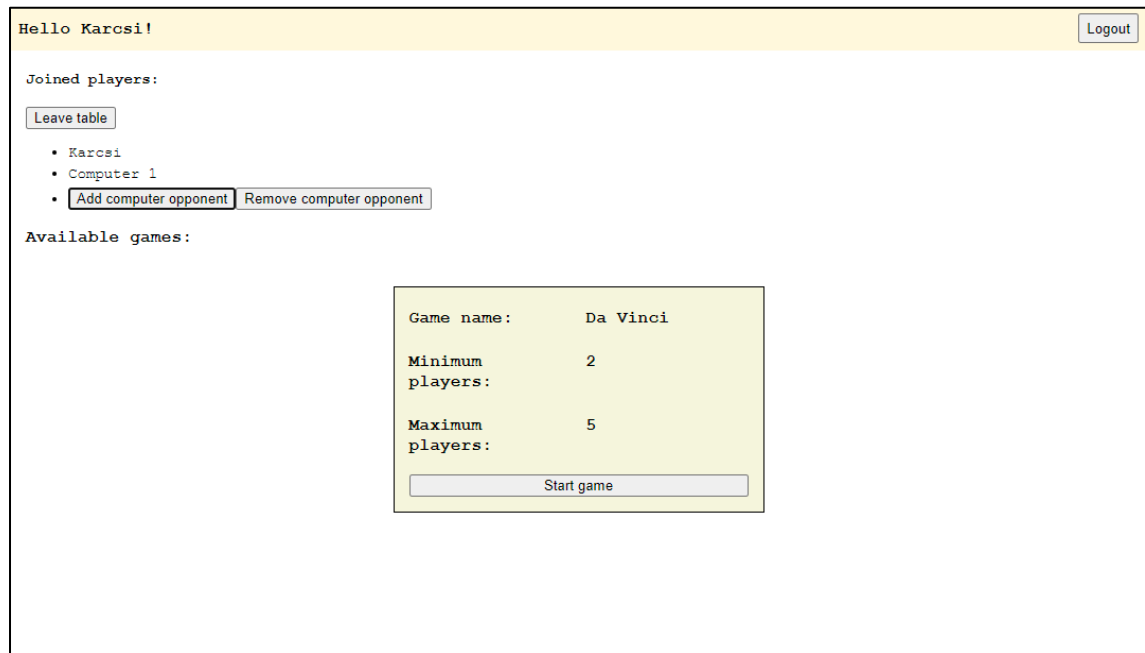
There are no open tables at the moment.

2.5. ábra. Köszöntő oldal



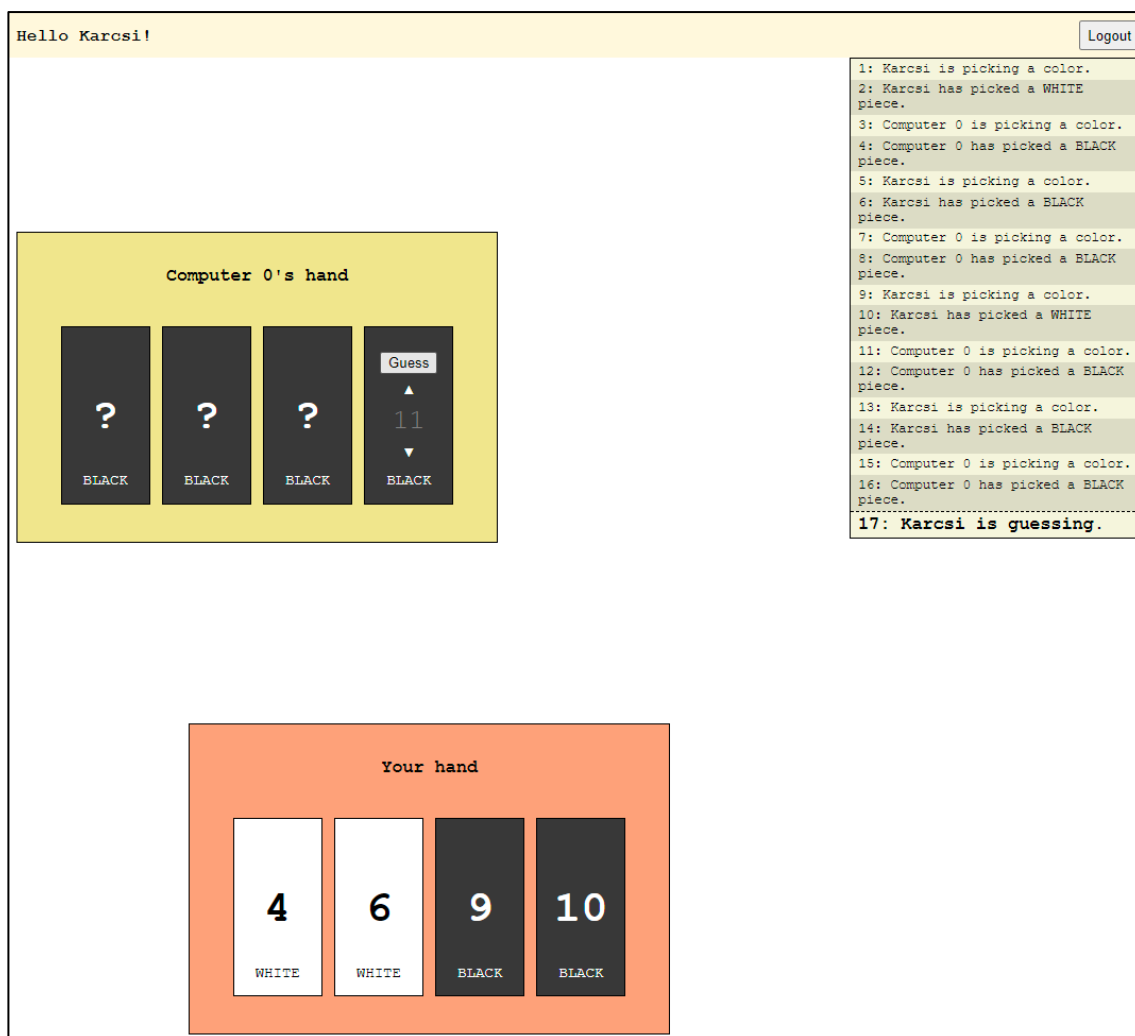
2.6. ábra. Asztal választó nézet

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.



2.7. ábra. Választott asztal nézet

Az asztal nézetben azt látjuk hányan ültünk eddig az asztalhoz, név szerint felsorolva, 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 szint 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

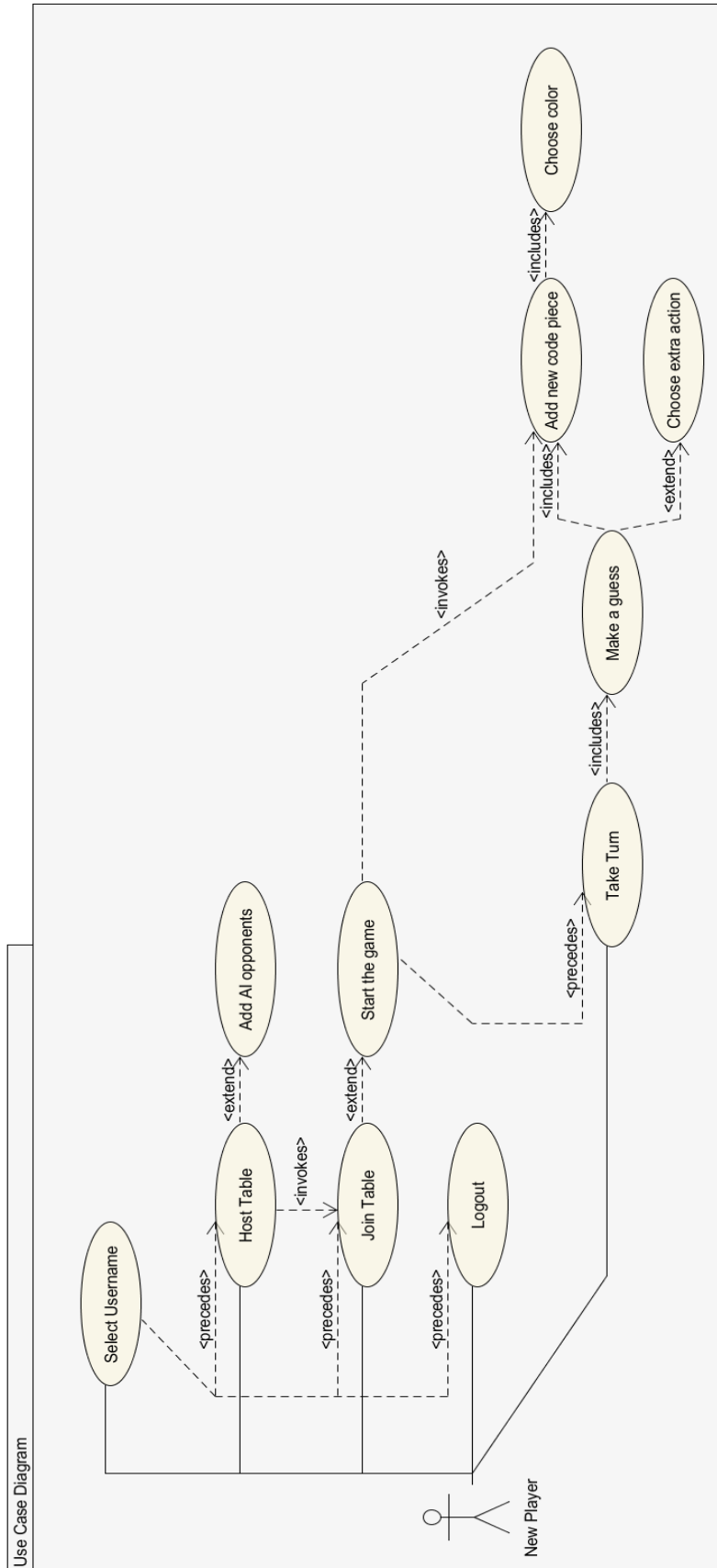
3.1. Elemzés

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

	Felhasználói eset	Leírás
1	Weboldal elérése	GIVEN Van telepítve támogatott böngésző
		WHEN A felhasználó megnyitja az alkalmazás weboldalát
		THEN Az alkalmazás megnyílik
2	Bejelentkezés	GIVEN Az alkalmazás meg van nyitva
		WHEN A felhasználó nincs bejelentkezve
		THEN A felhasználó a bejelentkezés képernyőt látja
3	Kijelentkezés	GIVEN Az alkalmazás meg van nyitva és a felhasználó be van jelentkezve
		WHEN A felhasználó a kijelentkezés gombra kattint
		THEN A felhasználót kijelentkeztetjük az alkalmazásból és a bejelentkező képernyőre irányítjuk
4	Asztalhoz csatlakozás	GIVEN A felhasználó be van lépve és egy másik játékos készített asztalt
		WHEN A felhasználó csatlakozik a másik játékos asztalához
		THEN A felhasználó az asztal specifikus képernyőt látja
5	Új asztal létrehozása	GIVEN A felhasználó be van lépve de még nincs asztalhoz csatlakozva
		WHEN A felhasználó rá kattint az új asztal kreálása gombra
		THEN A felhasználó a saját asztal specifikus képernyőjét látja
6	Gépi ellenfél hozzáadása	GIVEN A felhasználó létrehozott egy új saját asztalt és a választott játék támogatja a gép ellenfeleket
		WHEN A felhasználó hozzáad gépi ellenfelet a játékhoz
		THEN A játékos listában megjelenik egy új gépi ellenfél
7	A játék kezdése	GIVEN Megfelelő számú játékos ül az asztalnál
		WHEN Az asztal létrehozója megkezdi a játékot
		THEN Minden csatlakozott játékos képernyője a játék asztalra vált
8	Da Vinci	GIVEN Da Vinci játék van folyamatban
		WHEN A felhasználó lépése jön
		THEN A felhasználó a játék szabályainak megfelelően lép
9	Lépés a játékban	GIVEN Egy játék folyamatban van és a játékos köre van éppen
		WHEN A játékos tesz egy tippet valamelyik ellenfelének még nem felfedett kód elemére.
		THEN A szerver ellenőrzi a tipp helyességét, annak helyességének megfelelően a játék folytatódik.
10	Extra akció	GIVEN A játékos egy helyes tippel felfedte ellenfelének egy kód elemét.
		WHEN A játékos dönt, hogy kíván e új tippet végezni, vagy beilleszt egy felfedetlen elemet a kódsorába.
		THEN A választott akciónak megfelelően folytatódik tovább a játék
11	Helytelen tipp	GIVEN A játékos köre van és tesz egy tippet
		WHEN A játékos tippje helytelen
		THEN A játékosnak fel kell húznia egy elemet és be kell illesztenie azt a kódsorába felfedve.
12	Játék vége	GIVEN Da Vinci játék van folyamatban
		WHEN Már csak egy játékos van életben
		THEN A játék véget ér az utolsó életben maradt játékoskal nyertesként.

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.

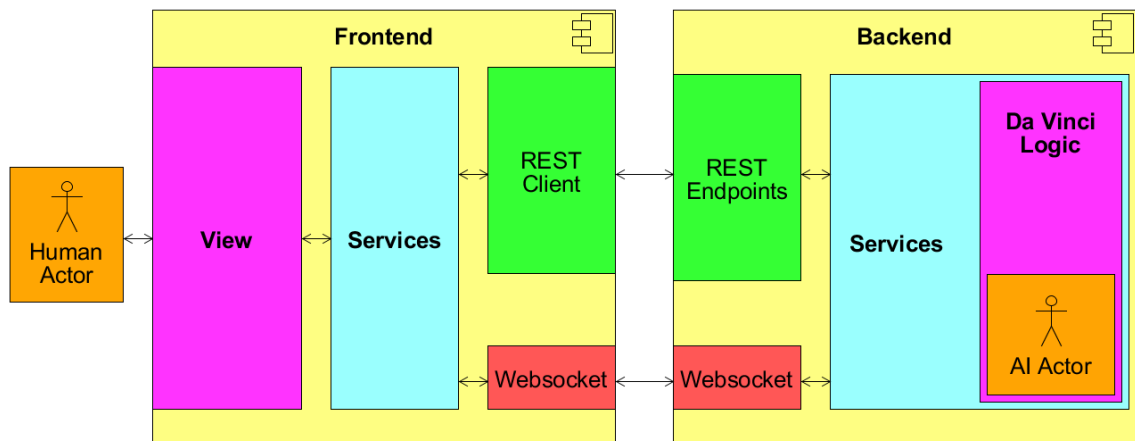
3.1.3. Nem-funkcionális leírás

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.

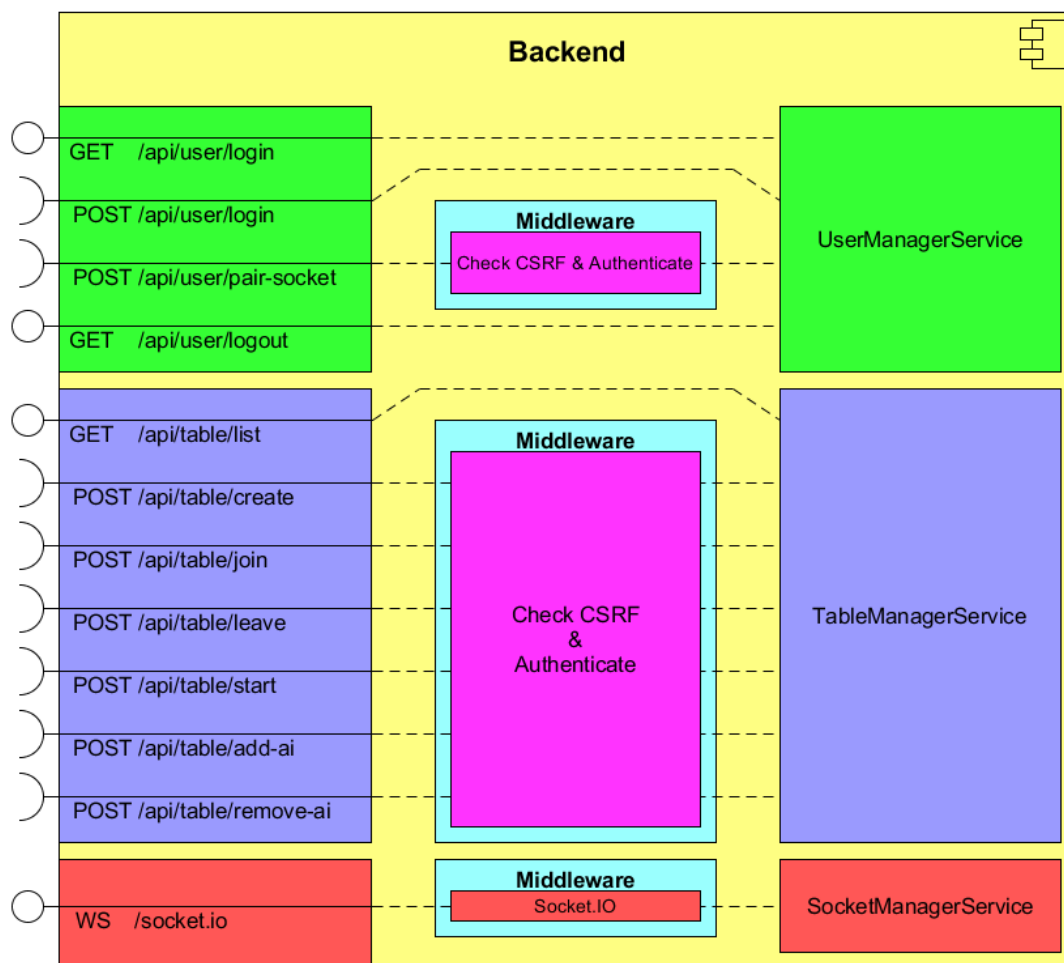
3.2. Tervezés



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 autentikálás 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).

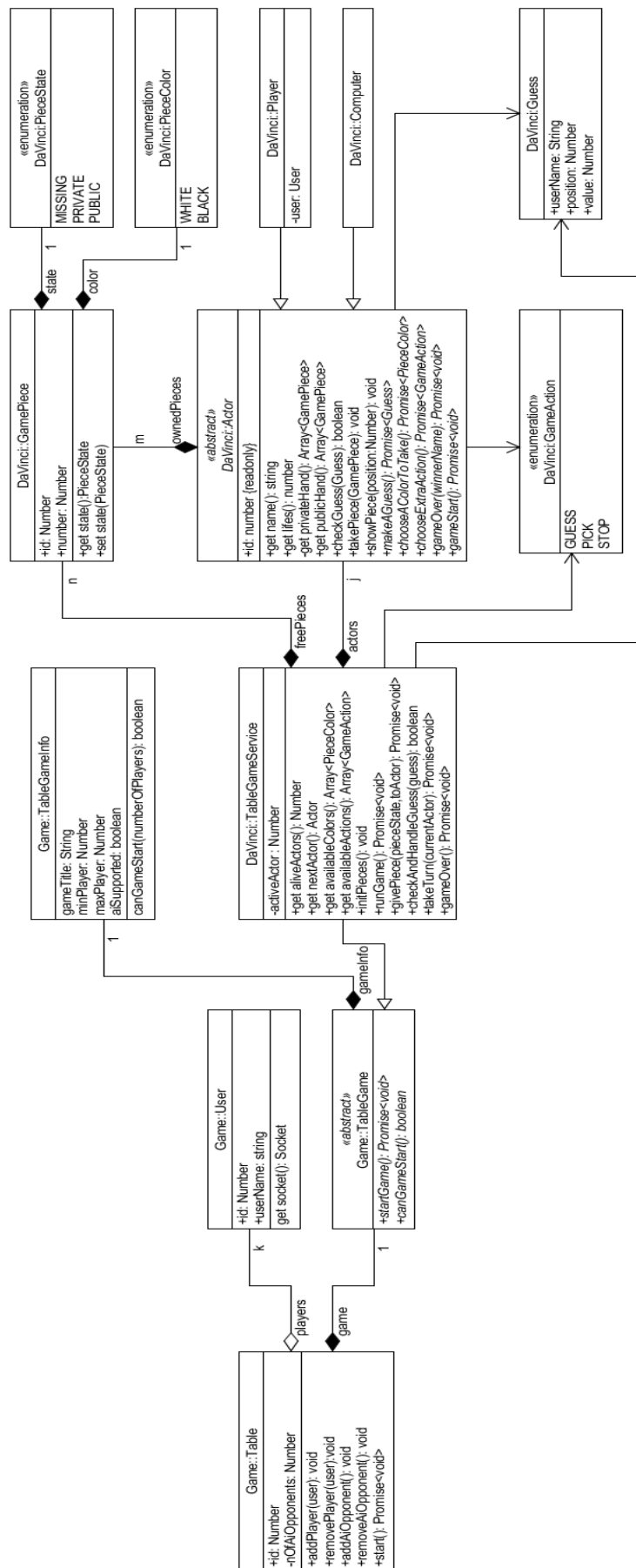
- A megfelelő autentikáció és autorizáció ellenőrzését végezzük *middleware* komponensekben.
- Értelmszerűen bontsuk szét a feladatkörök és felelősségek mentén a végpontokat külön csoportokra, és tegyük azoknak kezelését külön singleton osztályokba.
- Azonos csoportokban lévő végpontoknak legyen egy közös prefix-e az elérési útvonalban.
- Szoruljunk 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.

Végpont	Rest metódus	Várt adat	Válasz adat	Megjegyzések
/api/user/login	GET	n/a	Aktuális felhasználói név	Tulajdonképpen a kliens felteszi azt a kérdést, hogy "be vagyok e jelentkezve?"
/api/user/login	POST	Választott felhasználó név	Aktuális felhasználói név	Itt kerül beállításra a SESSIONID és a CSRF süti is.
/api/user/pair-socket	POST	n/a	n/a	Két sütit kiolvasva össze párosítjuk a felhasználót és a regisztrált socketet.
/api/user/logout	GET	n/a	n/a	Törli a SESSIONID sütit.
/api/table/list	GET	n/a	Jelenleg létező táblák és azok minden kliens számára érdekes adata.	Listában tartalmazza minden futó asztalról a választott játék címét, csatlakozott játékos neveit, az asztal azonosítóját, hozzá adott gépi ellenfelek számát és hogy lehet e csatlakozni még az asztalhoz.
/api/table/create	POST	n/a	n/a	Adott felhasználó vezetésével hozunk létre egy asztalt. Az asztalok állapotának változásáról websocketen értesítünk minden csatlakozott klienst.
/api/table/join	POST	A cél asztal azonosítója.	n/a	Az asztalok állapotának változásáról websocketen értesítünk minden csatlakozott klienst.
/api/table/leave	POST	n/a	n/a	Az asztalok állapotának változásáról websocketen értesítünk minden csatlakozott klienst.
/api/table/start	POST	Játszani kívánt játék megnevezése	n/a	Az asztalok állapotának változásáról websocketen értesítünk minden csatlakozott klienst.
/api/table/add-ai	POST	n/a	n/a	Az asztalok állapotának változásáról websocketen értesítünk minden csatlakozott klienst.
/api/table/remove-ai	POST	n/a	n/a	Az asztalok állapotának változásáról websocketen értesítünk minden csatlakozott klienst.
/socket.io	WS	n/a	n/a	A végpont-ra érkezett kérés helyes eredménye egy kiépült websocket kapcsolat.

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 létrehozásánál 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 biztosítani 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

Függvénynév	Argumentumok	Visszatérési típus	Megjegyzés
addPlayer	user	void	Hozzá ad egy játékost a players tömbhöz.
removePlayer	user	void	Kivesz egy játékost a players tömbből.
addAiOpponent	n/a	void	Növeli a hozzáadott gépi ellenfelek számát eggyel.
removeAiOpponent	n/a	void	Csökkenti a hozzáadott gépi ellenfelek számát eggyel.

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

Függvénynév	Argumentumok	Visszatérési típus	Megjegyzés
get socket	n/a	Socket	Megadja az adott felhasználóhoz tartozó websocket kapcsolatot.

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.

Függvénynév	Argumentumok	Visszatérési típus	Megjegyzés
startGame	<i>n/a</i>	Promise<void>	Megkezdi a játékot ha teljesülnek a játék kezdéséhez kellő feltételek. Megkeveri a játékosok sorrendjét, inicializálja és megkeveri a kártyákat, a játék szabályai szerint kiosztja a játékosoknak a kezdő kártyáikat majd el kezdi futtatni a játékot.
canGameStart	<i>n/a</i>	boolean	Ellenőrzi, hogy az asztalnál jelen állapotában megkezdődhet-e a játék.

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* : *boolean* - Támogatja-e a játék megvalósítása gépi ellenfelek hozzáadását

Függvénynév	Argumentumok	Visszatérési típus	Megjegyzés
canGameStart	numberOfPlayers	boolean	Ellenőrzi, hogy az aktuális játék szabályai szerint megkezdhető-e a csatlakozott számú játékosal.

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

Függvénytípus	Argumentumok	Visszatérési típus	Megjegyzés
get aliveActors	<i>n/a</i>	number	Getter függvény, mellyel megkapjuk a még játékban lévő játékosok számát. Játékban az van akinek van legalább egy privát kód eleme.
get nextActor	<i>n/a</i>	Actor	Getter függvény, mellyel megkapjuk a sorban következő játékost. A függvény változtatja az activeActor változó értékét, a már kiesett játékosokat át ugorja.
get availableColors	<i>n/a</i>	PieceColor[]	Megadja azon színek listáját melyből még nem fogyott el az összes kód elem a "húzó dobozban".
get availableActions	<i>n/a</i>	GameAction[]	Megadja a lehetséges extra akciók listáját. Ha már felhúztuk az összes elemet például, akkor csak passzolni vagy újra tippelni tudunk.
initPieces	<i>n/a</i>	void	Inicializálja és megkeveri a kódelemeket a "húzó dobozban".
runGame	<i>n/a</i>	Promise<void>	Felváltva kéri következő körök végrehajtására a játékosokat egészen addig, ameddig egynél több játékos van életben. Játék végén a meghívja a gameOver függvényt.
givePiece	pieceState, toActor	Promise<void>	Felkéri az aktuális játékost, hogy válasszon a színt (ha egynél több opció van) és ezután kivesz egy kódelemet a játékosnak a "húzó dobozból".
checkAndHandleGuess	guess	boolean	Ellenőrzi, hogy az adott tipp helyes e. Ha a tipp helyes, a céljátékosnál fel is fedteti az adott elemet.
takeTurn	currentActor	Promise<void>	Az aktuális játékos-nak bonyolítja le egy teljes körét. Ha a játékos helyes tippel miatt többször is jön, az akkor is egy körnek számít.
gameOver	<i>n/a</i>	Promise<void>	Minden játékost értesít a játék végétől, majd visszatérés előtt megvárja míg a játékosok elhadják az asztalt, így megkezdhető a játékhoz használt memória felszabadítása.

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)

Függvénynév	Argumentumok	Visszatérési típus	Megjegyzés
get state	n/a	PieceState	Getter függvény mellyel olvasható a kód elem állapota.
set state	newState	n/a	Setter függvény mellyel "írható" a kód elem állapota. Állapotok közt definiált állapot átmenetek vannak, helytelen kért állapot átmenetkor hibát dobunk.

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.

Függvénynév	Argumentumok	Visszatérési típus	Megjegyzés
get name	<i>n/a</i>	string	Getter függvény a játékos nevének lekérdezésére.
get lifes	<i>n/a</i>	number	Getter függvény a játékos privát állapotban lévő kód elemeinek darabszámának lekérdezéséhez.
get privateHand	<i>n/a</i>	GamePiece[]	Privát getter függvény a játékos kódelem listájának rendezett listájának olvasásához oly módon, hogy a privát elemek is olvashatóak. Erre főleg tipp ellenőrzésre van szükség.
get publicHand	<i>n/a</i>	GamePiece[]	Getter függvény a játékos kódelem listájának olvasásához oly módon, hogy a privát elemek értékeit elrejtjük.
checkGuess	guess	boolean	Ellenőrzi és visszaadja egy tipp helyességét.
takePiece	newPiece	void	Rendezetten beilleszt egy kód elemet a játékos által birtokolt kódsor-ba.
showPiece	position	void	Felfed egy játékos által birtokolt privát kód elemet.
makeAGuess	<i>n/a</i>	Promise<Guess>	Felkéri a játékost, hogy adjon egy tippet.
chooseAColorToTake	state, availableColors	Promise<PieceColor>	Felkéri a játékost, hogy válasszon egy színt az adott állapotú elemnek. Példa: Válasszunk színt egy új privát elemhez.
chooseExtraAction	availableActions	Promise<GameAction>	Felkéri a játékost, hogy válasszon a lehetséges extra akciók közül
gameOver	winnerName	Promise<void>	Felkéri a játékost, hogy reagáljon a játék végére.
gameStart	<i>n/a</i>	Promise<void>	Jelzi a játékosok számára, hogy a játék elindult.

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.

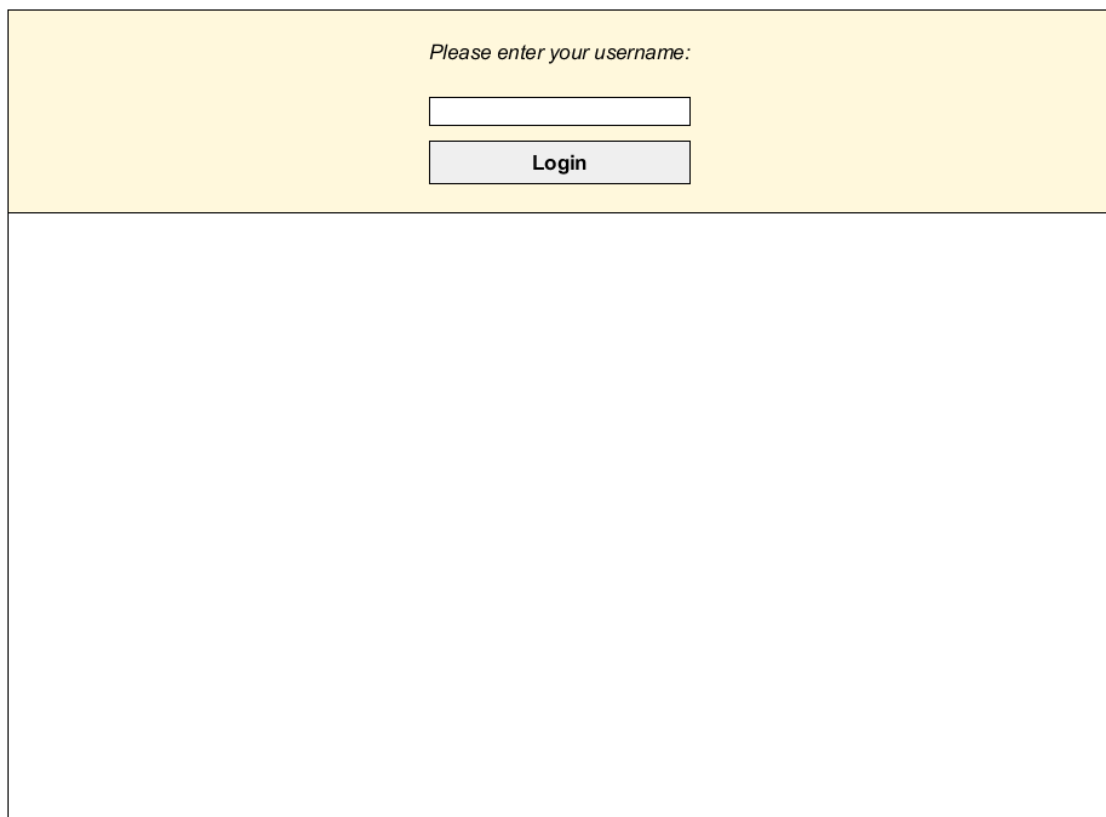
DaVinci::Computer osztály

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.



The image shows a wireframe of a login page. The top section is a yellow header containing the text "Please enter your username:" in italics. Below this text is a white rectangular input field. Underneath the input field is a grey rectangular button with the word "Login" in white text. The bottom section of the page is a large, empty white rectangle.

3.14. ábra. Köszöntő oldal

Asztal választó nézet

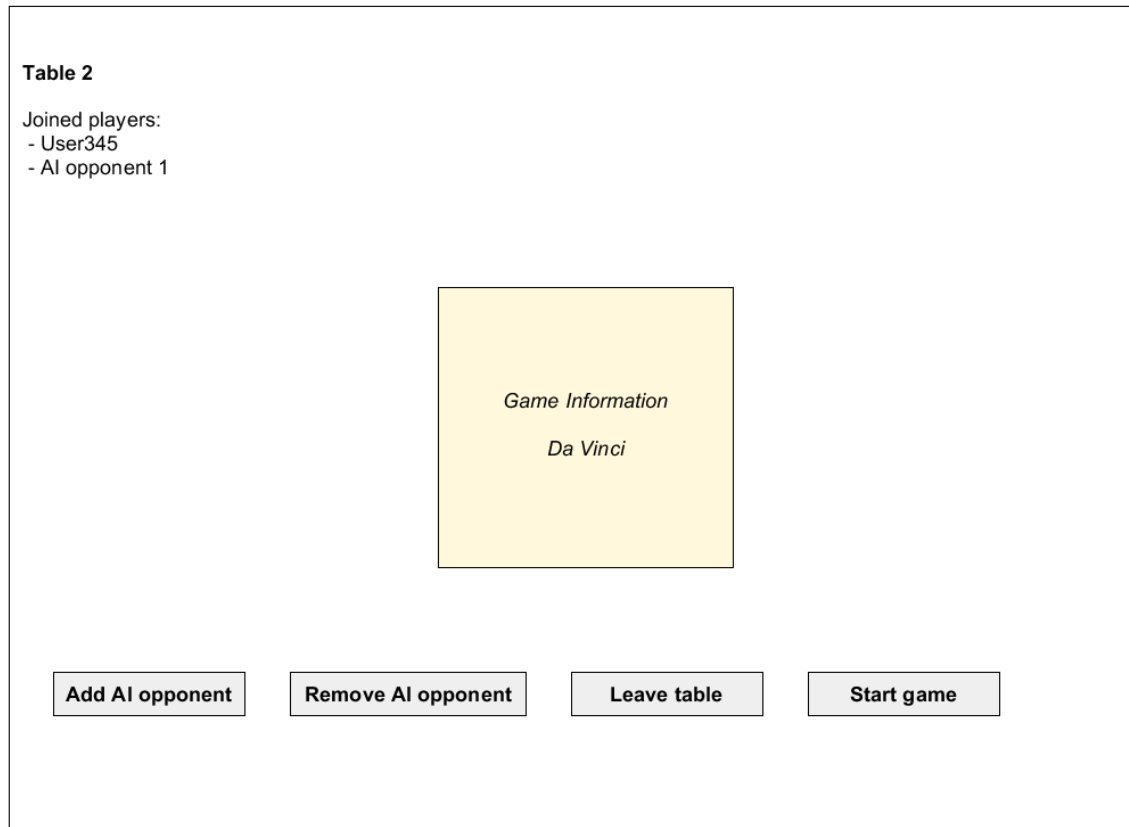
Ezen a nézeten jelenítenénk meg az elérhető asztalokat, sorolnánk fel az azokhoz csatlakozott játékosokat, és adnánk lehetőséget, hogy a jelenlegi felhasználó új asztalt hozhasson létre.

Hello User345!		Logout						
Open tables:								
<table border="1"><tr><td>Table 1</td></tr><tr><td>Game name: Da Vinci</td></tr><tr><td>Joined players: - User749 - User221</td></tr><tr><td><input type="button" value="Join table"/></td></tr></table>	Table 1	Game name: Da Vinci	Joined players: - User749 - User221	<input type="button" value="Join table"/>	<table border="1"><tr><td>New Table</td></tr><tr><td><input type="button" value="Create new table"/></td></tr></table>		New Table	<input type="button" value="Create new table"/>
Table 1								
Game name: Da Vinci								
Joined players: - User749 - User221								
<input type="button" value="Join table"/>								
New Table								
<input type="button" value="Create new table"/>								

3.15. ábra. Asztal választó nézet

Választott asztal nézet

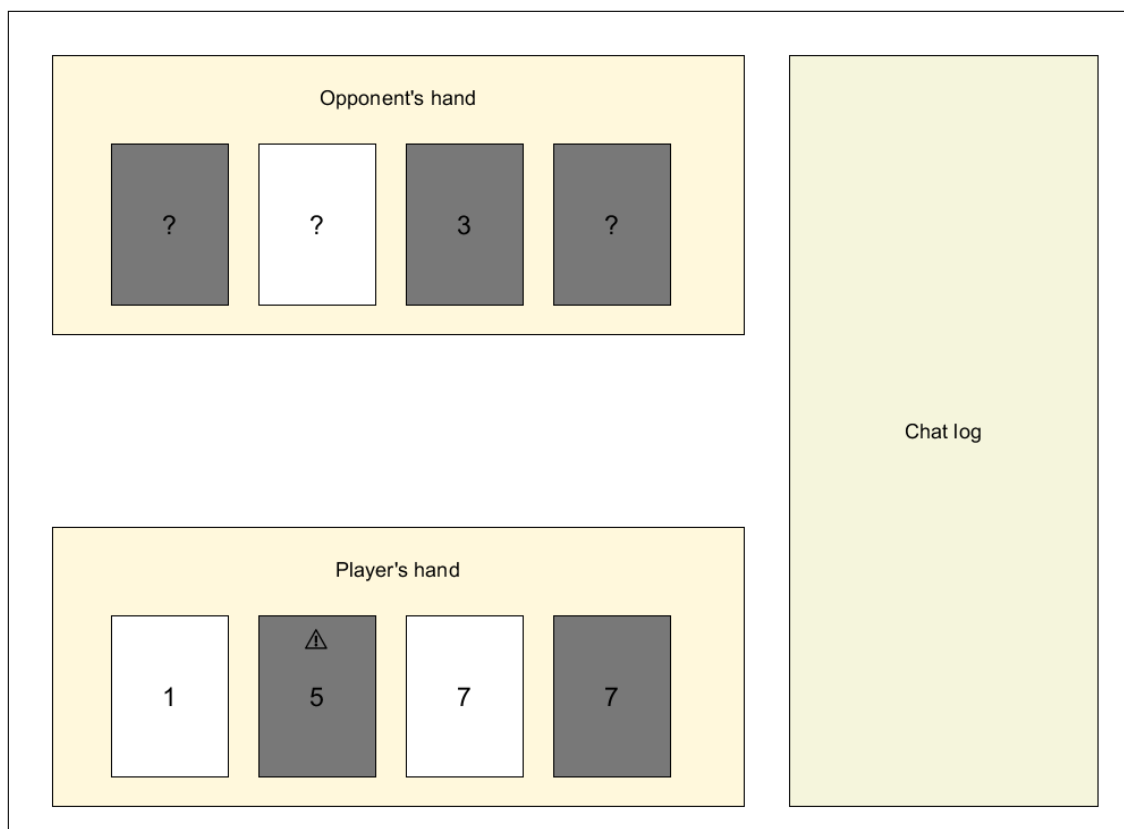
Ezt a nézetet látnánk akkor, ha csatlakozunk egy asztalhoz és várunk arra, hogy az elinduljon. E tudjuk indítani a játékot és tudunk gépi ellenfeleket hozzáadni vagy elvenni az asztaltól.



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

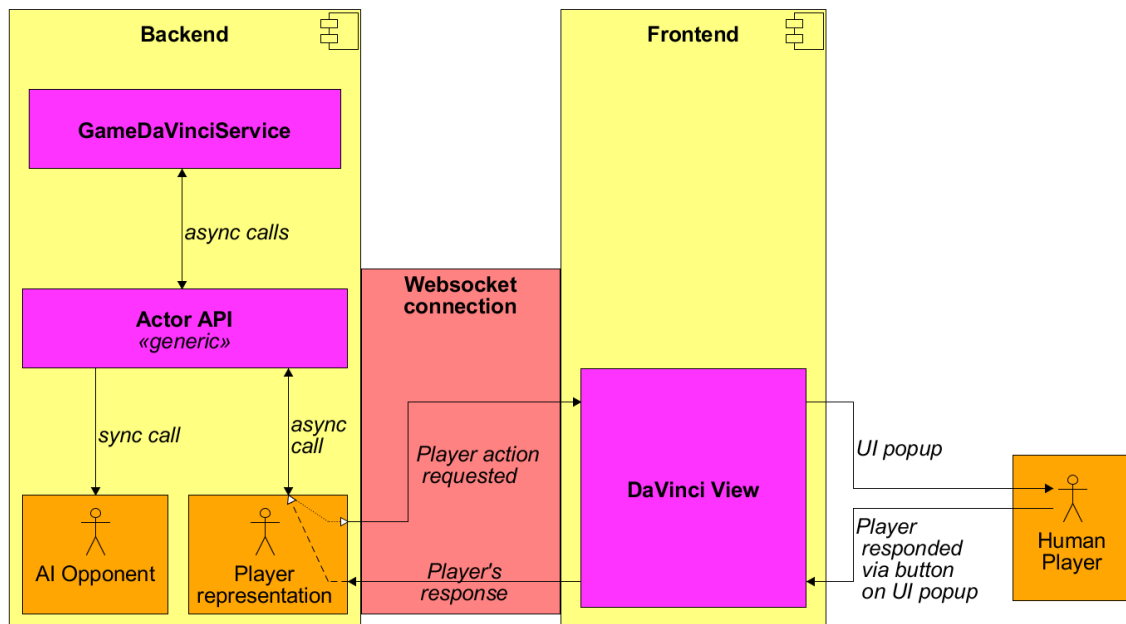
3.2.4. Hálózati kapcsolatok terve

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 elvágvá 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.

Topik neve	Küldő fél	Fogadó fél	Küldött adat	Megjegyzés
table-list	szerver	kliens	Aktuális asztalok listája	Ha a szerveren változik a futtatott asztalok állapot, a kliensek e módon tudnak róla értesülni.
server-start	szerver	kliens	n/a	Esetleges szerver újra indulás esetén tud a kliens reagálni egy oldal újratöltéssel.
user-connected	kliens	szerver	n/a	Esetleges kliens újra csatlakozás esetén így jelzi a kliens a szerver felé, hogy már képes folytatni a játékot.
game-init	szerver	kliens	SetupInfo	A küldött adat tartalmazza a játékosok neveit és a játék beállításait.
public-hand-update	szerver	kliens	PublicHand	Ellenfelek kódsorai, amely elemek publikusak azok értékkel is el vannak látva.
private-hand-update	szerver	kliens	GamePiece lista	Saját kódsorunk legfrissebb állapota.
message-info	szerver	kliens	Rendszer üzenet	Szerver által küldött üzenet a játék állapotával kapcsolatban.
guess	szerver	kliens	n/a	A szerver kéri a klienst a jelzett akció végrehajtására.
pick-color	szerver	kliens	Választható színek listája	A szerver kéri a klienst a jelzett akció végrehajtására.
take-extra-action	szerver	kliens	Választható extra akciók listája	A szerver kéri a klienst a jelzett akció végrehajtására.
game-over	szerver	kliens	Nyertes játékos neve	A szerver kéri a klienst a jelzett akció végrehajtására.
pick-color	kliens	szerver	Választott szín	A kliens válaszol a szerver által kért akcióval.
guess	kliens	szerver	Játékos által tett tipp	A kliens válaszol a szerver által kért akcióval.
take-extra-action	kliens	szerver	Választott extra akció	A kliens válaszol a szerver által kért akcióval.
game-over	kliens	szerver	n/a	A kliens válaszol a szerver által kért akcióval (A felhasználó reagált a játék vége feliratra.)

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 így módon vannak kommunikálva a megfelelő kliens felé és ugyanezen topikok használatával is válaszol ezen kérésekre a kliens.



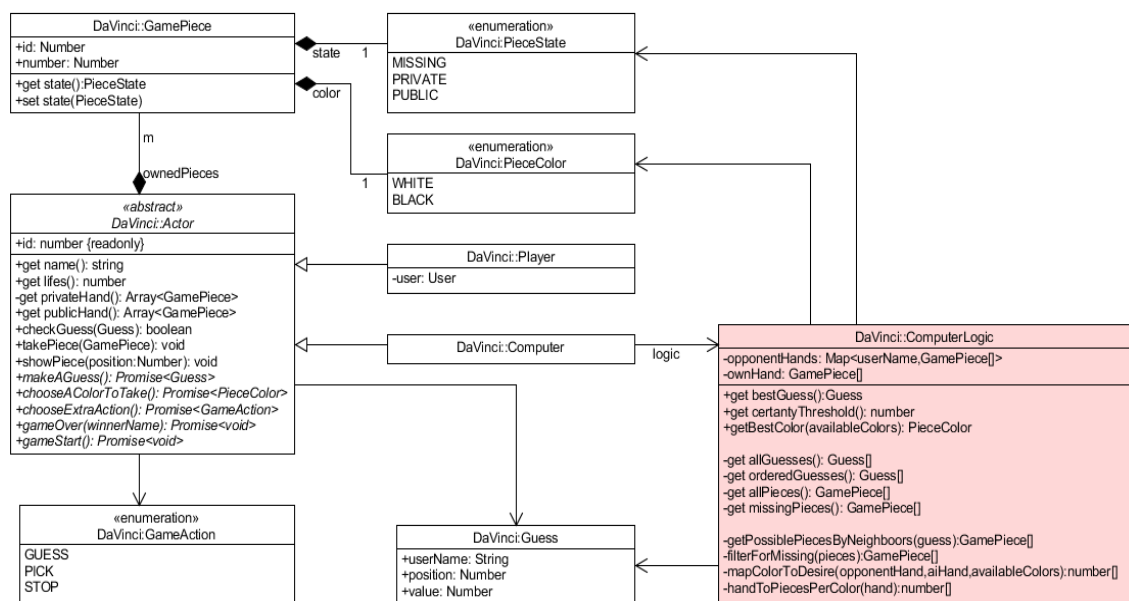
3.19. ábra. Felhasználót igénylő lépések lebonyolítása

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 használunk 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

Függvénynév	Argumentumok	Visszatérési típus	Megjegyzés
get bestGuess	n/a	Guess	Megadja a logika által számolt legmagasabb helyességi eséllyel rendelkező tippet.
get certaintyThreshold	n/a	number	Maximálisan vállalható kockázatot adja meg. Ha választhat extra akciót a gépi ellenfél, ez a szám adja meg, hogy fogja-e vállalni a következő tippel járó kockázatot, azaz van-e egy olyan tippje melynek helyességre való esélye magasabb mint a maximálisan vállalható kockázat. Kiszámolásához felhasználja fel nem húzott elemek számát és egy heurisztikát mely megállapítja mennyi az esélye annak, hogy a következő körben kiessen.
getBestColor	availableColors	PieceColor	Megadja a logika által számolt legkívánatosabb színt. Ellenfelként kiszámolja a mapColorToDesire függvény segítségével, hogy melyik színt szeretnénk, majd az értékeket a különböző játékosoktól színenként átlagolja, normalizálja, létrehoz egy valószínűségi változót és annak segítségével eldönti a választott színt.
get allGuesses	n/a	Guess[]	Megadja az összes olyan elemet melyet tehetünk tippet.
get orderedGuesses	n/a	Guess[]	Kiszámolja az összes tippelhető elemre, hogy hány darab értéket vehetnek fel a jelenlegi információink szerint és a lehetséges értékek darabszáma szerint növekvő sorrendbe rendezzük a vissza adott listát.
get allPieces	n/a	GamePiece[]	Elő állítja a játékban előforduló összes elemet és vissza adja azokat egy listában.
get missingPieces	n/a	GamePiece[]	Leszűri az allPieces által generált listát azon elemekre melyek nem a mi birtokunkban van és nem valamelyik ellenfelünknek van már felfedett állapotban.
getPossiblePiecesBy Neighbors	targetPiece	GamePiece[]	Megadja egy ellenfél privát kód elemére a jelenlegi információinkból számolható lehetséges értékek halmazát.
filterForMissing	GamePiece[]	GamePiece[]	Kiszűri azon elemeket a listából melyek nincsenek benne a missingPieces listában, azaz azon elemek listáját szeretnénk megkapni mely elemeket semelyik játékosnál sem látunk.
mapColorToDesire	opponentHand, ownHand, availableColors	number[]	Heurisztika segítségével kiszámolja, hogy az ellenfél által használt színek és a saját kódunk színei alapján melyik színt mennyire szeretnénk felhúzni. Minél több elemet ismerünk egy színből, annál jobb tippeket tudunk adni, de ha majd minden vagy minden elem egy adott színből játékban van, akkor könnyedén kitalálhatja bárki bármelyik elem értékét. Több játékosnál ez a visszahatás kevésbé érvényes.
handToPiecesPerColor	GamePiece[]	number[]	Megadja egy rendezett tuple-ben hogy a kapott kódelemek listájában, melyik színből hány darab van.

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 játékosról 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.

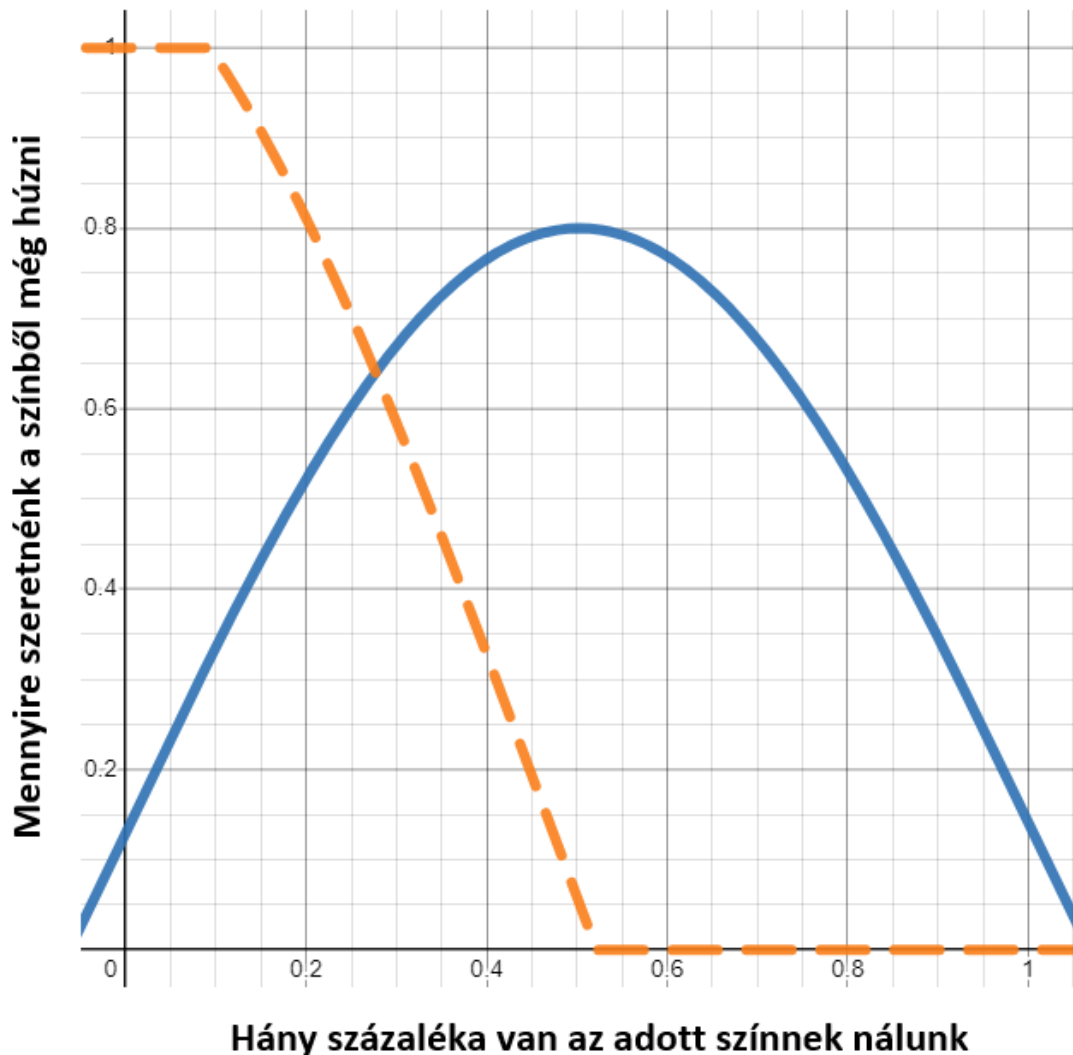
E megoldás hiányosságai:

- 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 elemeinek 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 tartjuk 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 megfejtethjük 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, ellenfelünknel 2, a dobozban még 5.
- Fekete színből nálunk 3 darab van, ellenfelünknel 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:

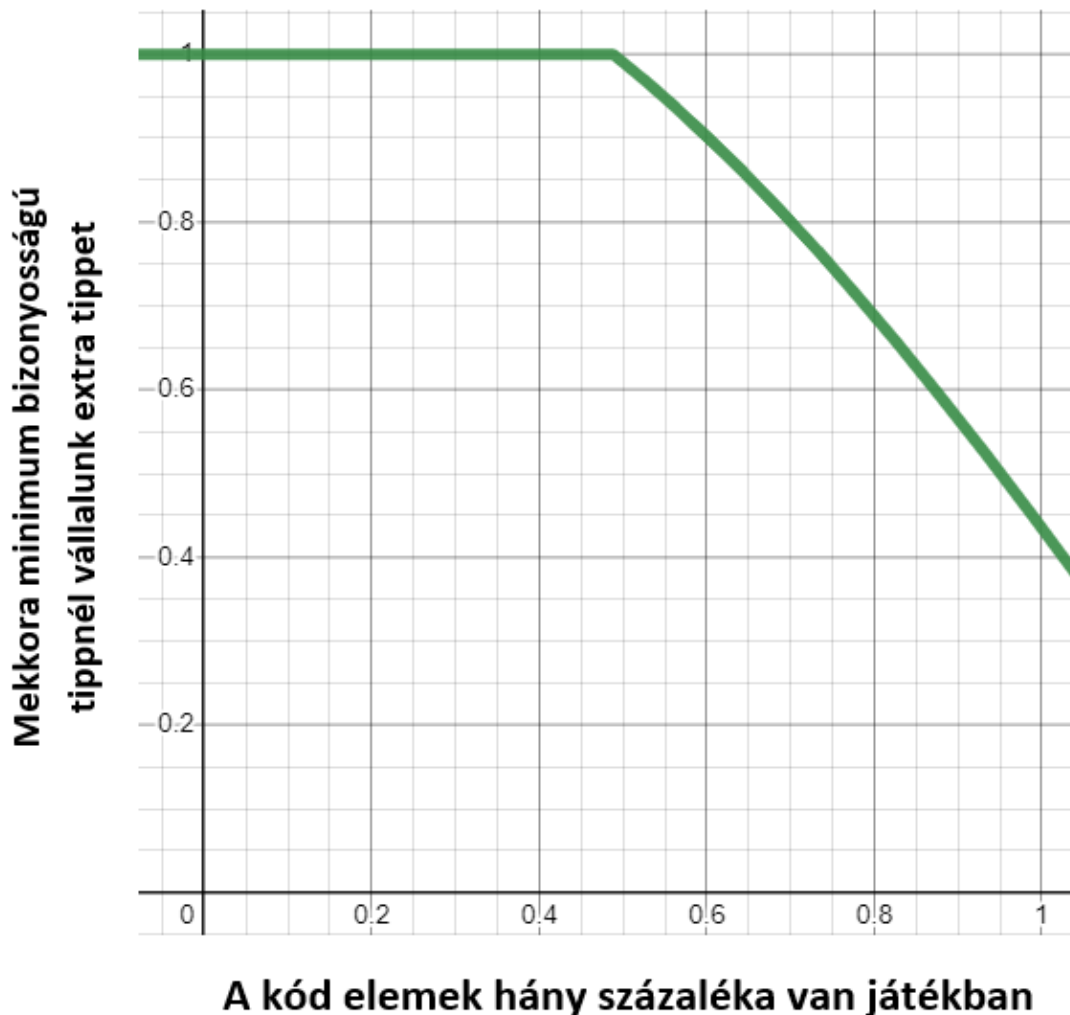
- Nem veszi figyelembe, hogy a húzott kódelem privát vagy publikus lesz-e.
- Talán pontosabb képet kaphatnánk, ha minden húzható elemre kiszámolnánk, hogy tippelésnél melyik húzott kártya birtokában könnyebbülne meg legjobban a dolgunk, ezt színekre összegeznénk és ennek fényében döntenénk a színről.

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 késég 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 találhatók.

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%-a 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* szerveret használtunk. Ezen kívül érdemes még megemlíteni az *RxJS* könyvtárat mely az adatok "push" alapú elven való folyását tette lehetővé.

```
const app = express();
app.use(bodyParser.json());
app.use(cookieParser());
const port = 3000;

app.route('/api/user/login').get(getUser);
app.route('/api/user/login').post(loginUser);
app.route('/api/user/pair-socket').post(checkCsrf, authenticate, pairSocket);
app.route('/api/user/logout').get(logoutUser);
app.route('/api/table/list').get(listTables);
app.route('/api/table/create').post(checkCsrf, authenticate, createTable);
app.route('/api/table/join').post(checkCsrf, authenticate, joinTable);
app.route('/api/table/leave').post(checkCsrf, authenticate, leaveTable);
app.route('/api/table/start').post(checkCsrf, authenticate, startTable);
app.route('/api/table/add-ai').post(checkCsrf, authenticate, addAi);
app.route('/api/table/remove-ai').post(checkCsrf, authenticate, removeAi);
```

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 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 szálát.

```
startTable(user: User, gameTitle: string) {  
  const tableToStart = this.findTableByUser(user);  
  tableToStart  
    .start(gameTitle)  
    .catch(error => console.log(error))  
    .finally(() => {  
      console.log('table-game ended');  
      this.deleteTable(tableToStart.id);  
    });  
  this.tablesEmitter.next(this.tables);  
}
```

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.

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 hajtottott volna végre, miközben a háttérben kommunikáltunk a klienssel, a felhasználó reagált a kérésre, az visszaérkezett a websocket kapcsolaton, hogy majd feldolgozásra kerüljön sor.

```
this.infoEmitter.next(`${currentActor.name} is guessing.`);  
const guess = await currentActor.makeAGuess();  
lastGuessResultIsCorrect = this.checkAndHandleGuess(guess);
```

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 megvalósítva, 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 mozgására 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

A megvalósított program alapvető logikájának helyes működését egység tesztekkel vizsgáltam, a felhasználói felületet pedig végfelhasználói tesztesetekkel ellenőriztem.

3.4.1. Egység tesztek

Game::TableGameInfo osztály egység tesztjei

Tesztelt komponens	Teszt célja	Teszt módszere
canGameStart függvény	Ellenőrizze, hogy játékosok megfelelő számosságánál IGAZ értéket kapjunk.	Teszteljük a függvényt minimum, maximum, majd az értékhatarok közti értékekkel.
canGameStart függvény	Ellenőrizze, hogy játékosok nem megfelelő számosságánál HAMIS értéket kapjunk.	Teszteljük a függvényt minimumnál kisebb és maximumnál nagyobb értékekkel.

3.28. ábra. Game::TableGameInfo osztály egység tesztjei

Array::shuffle függvény egység tesztjei

A program megvalósításakor implementációra került egy segédfüggvény mely egy tömb elemeit véletlenszerű sorrendbe keveri. Ez a játékosok sorrendjénél és a játék elemek húzási sorrendjének előállításakor volt hasznos.

Tesztelt komponens	Teszt célja	Teszt módszere
Tömb megkeverése	Ellenőrizze, hogy egy tömb megkeverése után csak és kizárólag az eredeti elemek szerepelnek a kapott tömbben.	Vegyünk egy tetszőleges tömböt, keverjük meg azt, ezután ellenőrizzük hogy a tömb mérete nem változott és, hogy minden elem megtalálható a meg kevert tömbben az eredeti tömbből.

3.29. ábra. Array::shuffle függvény egység tesztjei

DaVinci::ComputerOpponentLogic osztály egység tesztjei

Részben teszteltük a gépi ellenfél logikáját megvalósító osztályt is. Mivel a pontos működéshez erős heurisztika van használva, amely könnyedén változtatható, ezért a tesztek írásakor a nagyon általános, szélsőséges helyzetekben előforduló egyértelmű állapotok tesztelésén volt inkább a hangsúly. Csak fekete dobozos tesztek készültek.

Tesztelt komponens	Teszt célja	Teszt módszere
Legjobb tipp módszere	Ha csak egy elem nem ismert és minden elem játékban van, akkor tegyünk helyes tippet	Vegyünk egy két elemmel játszott játékot, ha az egyik elemet ismerjük, ki kell tudnunk találni a másikat is.
Legjobb tipp módszere	Ha egy színből minden elem játékban van de még van köztük PRIVÁT az ellenfelünknel, akkor találnunk kell egy olyan tippet, melyben 100%-ig biztosak vagyunk.	Vegyük a játék egy olyan állapotát amikor egy színből minden elem valamelyik játékosnál van és számoljuk ki a legjobb tippet, majd ellenőrizzük annak bizonyossági faktorát, hogy 100%-e.
Legjobb tipp módszere	A legjobb tipp ne legyen 100%-os bizonyosságú, ha túl sok az ismeretlen.	Vegyünk egy olyan állapotot, ahol a legjobb tippünkre több érték opció is van. Ekkor ellenőrizzük, hogy a bizonyossági érték alacsonyabb legyen mint 100%.
Kívánt szín választás módszere	A választott szín legyen olyan amelyet választhatunk is.	Vegyünk egy olyan játékot ahol egy színből tudjon csak a gép választani, ellenőrizzük, hogy ilyenkor a választott szín a választható színek közül van.
Extra lépés választásának módszere	Ha a legjobb tippünk 100%-os bizonyosságú, akkor mindenképpen tippeljünk	Vegyünk egy tetszőleges állapotot ahol a legjobb tippünk 100%-os bizonyosságú, ezután ellenőrizzük, hogy a választott extra akció ebben a pillanatban a tippelés lenne e.

3.30. ábra. DaVinci::ComputerOpponentLogic osztály egység tesztjei

DaVinci::ComputerOpponent osztály egység tesztjei

Miként a *ComputerOpponent* osztály az *Actor* absztrakt osztály-t implementálja, a lenti függvények tesztelése lefedi az *Player* osztály azonos függvényeinek azonos implementációit is.

Tesztelt komponens	Teszt célja	Teszt módszere
Élet számolása	Ellenőrizze, hogy a számolt életek értéke megegyezik-e az elvártal.	Vegyünk egy kódsort melynek ismerjük a szabályok szerinti életeinek számát, majd hasonlítsuk össze a számolt értékkel.
Élet számolása	Ellenőrizze, hogy a számolt életek értéke megváltozik-e ha a kódsort manipuláljuk.	Vegyünk egy kódsort kérdezzük le az életek számát, majd manipuláljuk pozitívan és negatívan is a kódsort és lépésenként vizsgáljuk az életek számát.
checkGuess függvény	Ellenőrizze, hogy a tipp kiértékelése helyesen történik-e.	Vegyünk egy kódsort és adjunk rá egy helyes és egy helytelen tippet. Azt várjuk, hogy a helyes esetben IGAZ, míg másik esetben HAMIS értéket kapunk.
Publikus kódsor olvasása	Ellenőrizze, hogy a publikus kódsor megfelelő módon tartalmazza-e a kódelemeket.	Vegyünk egy kódsort, vegyünk annak publikusan látható változatát, majd hasonlítsuk össze. Ahol privát elem volt, ott az értéknek rejtve kell lennie.
Publikus kódsor olvasása	Ellenőrizze, hogy ha kódsor tartalma változik, akkor a publikus kódsor tartalma is változik.	Vegyünk egy kódsort, manipuláljuk annak tartalmát, és vizsgáljuk lépésenként a publikus kódsort.

3.31. ábra DaVinci::ComputerOpponent osztály egység tesztjei

DaVinci::GamePiece osztály egység tesztjei

A *GamePiece* osztályra részletes egység teszteket írtunk, mivel a játék főbb logikáit ő kezeli. Például a kódelemek állapotok közti állapot átmeneteket teszteltük részletesen, a kód elemek szabály szerinti kód sorba rendezését és a játékhoz szükséges *érték elrejtése* funkciót is.

Tesztelt komponens	Teszt célja	Teszt módszere
Eltárolt állapot	Ellenőrizze, hogy a helyes állapotban vagyunk e az objektum létrehozása után.	Létrehozunk objektumokat több állapottal is és ellenőrizzük az elmentett állapotukat.
Állapot átmenet	Ellenőrizze, hogy PRIVÁT állapotba csak MISSING állapotból tudunk átmenni.	Megpróbálkozunk MISSING és PUBLIC állapotokat PRIVÁT-ra állítani, előbbi esetén sikeres értékadást várunk, míg utóbbi esetén hibát.
Állapot átmenet	Ellenőrizze, hogy PUBLIC állapotba BÁRMELY állapotból át állhatunk.	Megpróbálkozunk MISSING és PRIVATE állapotokat PUBLIC-ra állítani, mindkét esetben sikeres értékadást várunk.
Állapot átmenet	Ellenőrizze, hogy MISSING állapotba SEMMILYEN állapotból sem állhatunk át.	Megpróbálkozunk PUBLIC és PRIVATE állapotokat MISSING-re állítani, mindkét esetben hibát várunk.
Állapot átmenet	Ellenőrizze, hogy az aktuális állapottal megegyező állapotba az "átmenet" nem megengedett.	Megpróbálkozunk minden elemet a saját aktuális állapotára állítani még egyszer. Minden esetben hibát várunk.
Rendezés	Ellenőrizze, hogy szám értékek alapján megfelelően hasonlítunk össze két elemet.	Vegyünk egy két színből és két értékből álló négy elemű, értékpárok halmazát. Ezután hasonlítsuk össze a halmaz nem azonos értékkel bíró elemeit egymással szám értékeik alapján és ellenőrizzük a kapott eredményeket.
Rendezés	Ellenőrizze, hogy szín értékek alapján megfelelően hasonlítunk össze két elemet.	Vegyünk egy két színből és két értékből álló négy elemű, érték párok halmazát. Ezután hasonlítsuk össze a halmaz nem azonos színnel bíró elemeit egymással szín értékeik alapján és ellenőrizzük a kapott eredményeket.
Rendezés	Ellenőrizze, hogy egy példa listája a kódelemeknek az elvárt sorrendbe lenne rendezve.	Vegyünk egy példa kódlistát melynek tudjuk a megfelelő rendezését. Rendezzük az algoritmusunk segítségével majd hasonlítsuk össze a kapottat a várt eredménnyel.
Érték elrejtése	Ellenőrizze, hogy az elem értéke el lett rejtve.	Vegyünk egy tetszőleges elemet és ellenőrizzük, hogy az elrejtésből kapott element olvasható e a kód elem szám értéke.
Érték elrejtése	Ellenőrizze, hogy az elrejtés nem változtatja meg az elem további adatait, mint például színét.	Vegyünk tetszőleges elemeket különböző színekkel, elrejtjük őket és ellenőrizzük, hogy minden adatuk a szám értéken kívül továbbra is megfelelő.

3.32. ábra. DaVinci::GamePiece osztály egység tesztjei

3.4.2. Végfelhasználói tesztesetek

	Esemény	Várt eredmény
1	Weboldal első elérése	Betölt a weboldal, látjuk a bejelentkező felületet ahol megadhatjuk a felhasználó nevünket, és a felület alatt a jelenleg futó asztalok listáját látjuk, vagy a "There are no open tables at the moment" feliratot.
2	Bejelentkezés	Bejelentkezéskor jobb felső sarokban egy zöld notifikáció jelenik meg jelezve a sikeres bejelentkezést. Ezután a megjelenő felületen bal felső sarokban a fejlécen köszöntő üzenetet találunk a nevünkkel, jobb felső sarokban a kijelentkezés gombot. A fejléc alatt jelennek meg az éppen futó táblák sorban, amennyiben ilyen nincs akkor a "There are no open tables at the moment." feliratot látjuk, ezen kívül még az új asztal létrehozásához látunk egy gombot.
3	Kijelentkezés	Kijelentkezéskor egy piros notifikáció jelenik meg jelezve a sikeres kijelentkezést. Ezután a weboldal első elérésekor részletezett felületet kell látnunk.
4	Asztalhoz csatlakozás	A fejlécen továbbra is látjuk az üdvözlő üzenetet a nevünkkel és a kijelentkezés gombot. A felületen ezen kívül látnunk kell az aktuálisan csatlakozott játékosokat az asztalnál és a hozzá adott gépi ellenfeleket. A felületen továbbá látnunk kell az elérhető játékokat, azok nevét, és a játékhoz szükséges információkat (játékos létszám például). A játékos listában a nevünk mellett találjuk meg az asztal elhagyása gombot. Ezen kívül az asztalhoz csatlakozáskor minden más bejelentkezett játékosnak is a megfelelő felületeken látniuk kell az új állapotot.
5	Új asztal létrehozása	Új asztal létrehozásánál ugyanazt a felületet kell látnunk mint amit csatlakozáskor látunk azon különbségekkel, hogy képesek vagyunk megfelelő létszámnál elindítani a játékot és képesek vagyunk gépi ellenfelet hozzáadni az asztalhoz. Ezen kívül új asztal létrehozásánál minden más bejelentkezett felhasználónak is a megfelelő felületeken látniuk kell az új asztalt.
6	Gépi ellenfél hozzáadása	Gépi ellenfél hozzáadásakor minden az asztalhoz csatlakozott felhasználónál megjelenik a gépi ellenfél a csatlakozott játékosok listájában.
7	A játék elkezdése	Átváltunk a játék nézetre, betöltődik a játék. A fejlécben marad az üdvözlő felirat és kilépés gomb. A játék megjeleníti a játékosok és saját kezünket és elkezd a játékot annak kezdő lépéseivel. Jobb oldalt a játék üzeneteiben kell tudnunk követni a végre hajtott lépéseket.
8	Da Vinci játék kezdése	Játékos számtól függően a játéknak fel kell ajánlania 3 vagy 4 kezdő elemhez a szín választást.
9	Lépés a játékban	Az ellenfelek lapjaira rá hover-elve meg kell jelennie a gomboknak a számok felett és alatt amivel tippet lehet beállítani és véglegesíteni. Helyes tipp esetén extra akciót kell választani, helytelen esetén szint kell választani (de csak akkor ha van még több színből kód elem az osztónál).
10	Extra akció	Extra akció csak akkor jár ha tettünk egy sikeres tippet. Extra akció esetén a felületen egy felugró ablakban kell tudnunk gombok segítségével választani az extra akciók közül. ("Continue guessing" vagy "Take a piece"). Választás után a választásnak megfelelő akciónak kell következnie.
11	Helytelen tipp	Megjelenik egy felugró ablak a "Please pick a color for a PUBLIC piece." felirattal és szint kell választanunk egy publikus elemmel. (de csak akkor ha van még több színből kód elem az osztónál). Ezután a húzott elem bekerül a megfelelő helyére a kódsorunkba és publikus jelzéssel van el látva. (háromszögben felkiáltó jel)
12	Játék vége	Automatikusan megjelenik egy felugró ablak a "Congratulations! You won!" vagy {Nyertes játékos neve} won the game. Thanks for playing! Good luck next time!" felirattal és egy "Okay" gombal. A gombra kattintva automatikusan az asztalok listázása nézetre ugunk vissza.

3.34. ábra. Végfelhasználói tesztesetek

Fekete doboz teszteseteket a felhasználói történetek (3.2.-es 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/>