Eötvös Loránd Tudomány Egyetem
Informatika kar
Média és Oktatásinformatika Tanszék

Valós idejű ügyességi játék: Free-For-All 2D Aréna

Heizlerné Bakonyi Viktória
Mester Tanár

Sevcsik Marcell
Programtervező informatikus BS

Budapest, 2018

Tartalomjegyzék

1 Bevezetés

A számítógépes játékok „felfedezésekor” kerültem először kapcsolatba az informatikával és úgy gondolom, hogy az érdeklődésem e terület felé ennek köszönhető. A évek, és persze tanulmányaim során, rengetem ismeretet szereztem a számítástechnikáról, számtalan felhasználási területével ismerkedtem meg, a mindennapi életben körül vesznek – a számomra nélkülözhetetlen – informatikai eszközök és a szakmai gyakorlatom során sikerült betekintést kapnom a mélyebb programozás világába is.

Jelen téma választásomat azonban mégis az alapok, az első motivációk kiváltója a számítógépen játszható játékok adták. Évek óta a legsikeresebb játékok a valós idejű online játszható többjátékos élmények, ahol a résztvevők csapatban vetélkednek egymás ellen. A legújabb ilyen sikertörténet a *Battle Royale* játékmód, ahol közel 100 játékos küzd - mindenki-mindenki ellen - egy nagy játéktérben. Tudom, hogy egy ilyen játékmódot megvalósítani nem egyszerű feladat, de ami számomra rendkívül izgalmassá tesszi ezt a játékmódot, - informatikai szempontból – hogy vajon mi okozza azt a kellemetlen játékélményt az online játékosoknak, amit „akadásként” érzékelnek.

Akadás alatt nem arra gondolok, hogy a számítógépünk hardware-e nem elég erős, hogy folytonosan megjelenítse a grafikát, és nem is arra, hogy rossz/lassú az internet kapcsolatunk, ami miatt nem folytonos az információáramlás a kliens és szerver között. Hanem, hogy ha kliens oldalon minden ideális, akkor egy nagyobb, jól támogatott játék szerverei mégis miért nem képesek egy stabilkapcsolat megtartására. Ezen probléma a *Battle Royale* zsánernél könnyen megtapasztalható, mikor egy kisebb térben egyszerre - a tér méretéhez képest - nagyon sok játékos cselekszik vagy csupán tartózkodik egy időben, ekkor már-már garantált a játék „akadása”.

Célom egy olyan játék létrehozása volt, melynél ezzel a problémával szembesülhetek, megismerkedhetek. Egy olyan „lövöldözős” játékot terveztem, ahol a játékosok a billentyűzet segítségével mozognak, továbbá egérrel célozva lövedékeket lőnek egymásra. Akit eltalálnak az kiesik, aki pedig leadta a sikeres lövést az pontot szerez. A játékosoknak nem csak elmozdulással lenne módja a lövések elkerülésére, hanem tudnának az egységük köré pajzsot idézni, amikor nem lehet őket eltalálni. Vizuális megoldást tekintve: kétdimenziós, a játékos szemszögéből felülről nézet játéktér, amelynek aktuálisan azon kis része látható, ahol a játékos karaktere éppen elhelyezkedik.

Fontos, hogy a játék szórakoztató legyen bárki számára adhoc jelleggel. Ne legyen szükséges a játékkal órákat eltölteni, ahhoz, hogy sikereket lehessen elérni és a játékkal töltött idő kikapcsolódást jelentsen. Egy játék sikerét nagyrészben az elérhetősége határozza meg. Ezért választottam a platformfüggetlen web-es környezetet a kivitelezéshez. Ezen ötletek és tervek megvalósítása a ***Free-For-All 2D Aréna.***

2 Felhasználói dokumentáció

2.1 Feladat rövid ismertetése

A ***Free-For-All 2D Aréna*** egy valós idejű webes játék, ahol a fő cél a legtöbb pont összegyűjtésével a ranglétra tetejére jutni. A játékhoz a regisztráció opcionális, csupán néhány extra funkciót tesz elérhetővé.

A játékban résztvevők egy előre meghatározott kétdimenziós pályán mozognak, ahol egymásra adnak le lövéseket, melyek elől ki lehet térni vagy ki lehet védeni. Ha valakit eltalálnak akkor az a játékos ideiglenesen kiesik a játékból és a lövést leadó játékos pontot szerez. Kiesésnél az addig összeszedett pontok elvesznek, de a ranglétrán szerzett helyezés megmarad. A ranglétrán a három legtöbb pontot összegyűjtő játékos szerepel. A kiesett játékos azonnal visszaléphet a játékba.

2.2 Rendszerkövetelmények

A játék egy webszerveren érhető el, így csak egy böngészőre van szükség az oldal eléréséhez, a játék irányításához egér és billentyűzet szükséges. A böngészők többsége megfelelően megjeleníti az oldalt, de a ***Mozilla Firefox*** és ***Google Chrome*** böngészőket ajánlom, mivel ezeken volt részletesen tesztelve.

A webszerver helyi futtatásához ***Node.js***-re van csak szükség.

2.3 Telepítés

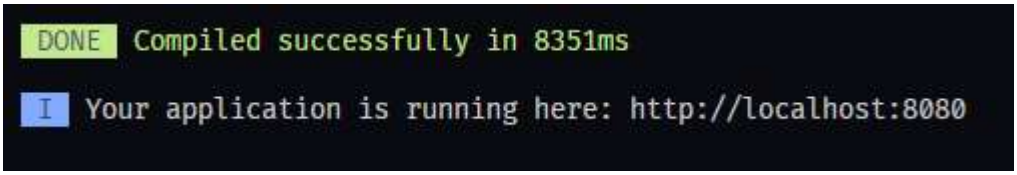
A webszerver lokális elindításához a következőket kell tenni:

1. Ha nincs még ***Node.js*** telepítve akkor a *https://nodejs.org/en/download/* oldalon a használt rendszernek megfelelő telepítőt kell letölteni és használni.
2. Nyitni két konzolablakot az alkalmazás gyökérkönyvtárában. A játékszervernek és a webszervernek is kell egy-egy.
3. Kiadni az **npm install** parancsot a függőségek letöltéséhez. (viszonylag hosszadalmas a lefutása).
4. Kiadni a **node gameServer/gameServer.js** parancsot a játékszerver elindításához. (l*ásd.: 2.1.ábra*)

```
$ node gameServer/gameServer.js
Server On
_
```

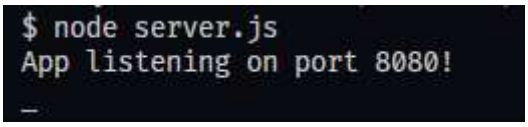
2.1. ábra.: Sikeres játékszerver indítás

5. A webszerver elindításának két módja van.
    a. Kiadni a **npm start** parancsot ezáltal elindítva a *http:localhost:8080*-as címen egy development szervert. (l*ásd.: 2.2.ábra*)



2 2. ábra.: Sikeres development szerver indítás

    b. Kiadni a **npm run build** parancsot a forráskód lefordításához, ekkor létrejön egy **/dist** mappa amit egy tetszőleges webszerverre lehet kitenni.
    Egy egyszerű HTTP szerver van az applikáció gyökérkönyvtárában, aminek indításának módja a konzolból a következő (l*ásd.: 2.3.ábra*):
        i. **node server.js**



***2.3**. ábra.: Sikeres HTTP szerver indítás*

Ha a játékszervert ki szeretnénk helyezni hálózatra akkor a következő kódrészleteket kell változtatni:

- ***./src/components/Game.vue::77.sor***: Itt a megfelelő játékszerver elérési útvonalát kell megadni, hogy elérje a kliens.

```
socket = io.connect("http://localhost:3000");
```

- ***./gameServer/gameServer.js::9.sor***: Itt a játékszerver által figyelt port-ot kell megadni.

```
const server = app.listen(3000);
```

2.4 Használat

Egy támogatott böngészőbe meg kell adni a webszerver elérési útvonalát, utána a weboldalon egér és billentyűzet használatával lehet navigálni.

2.4.1 Oldalak

A **Free-For-All 2D Aréna** egy olyan oldal, amelynek a tartalma dinamikusan változik. Az oldal tetején egy fekete fejléc, bal szélén logó látható, jobb szélén a regisztrációs opciók vannak elhelyezve, attól függően, hogy a felhasználó be van-e jelentkezve.

2.4. ábra.: Vendég felhasználó fejléc

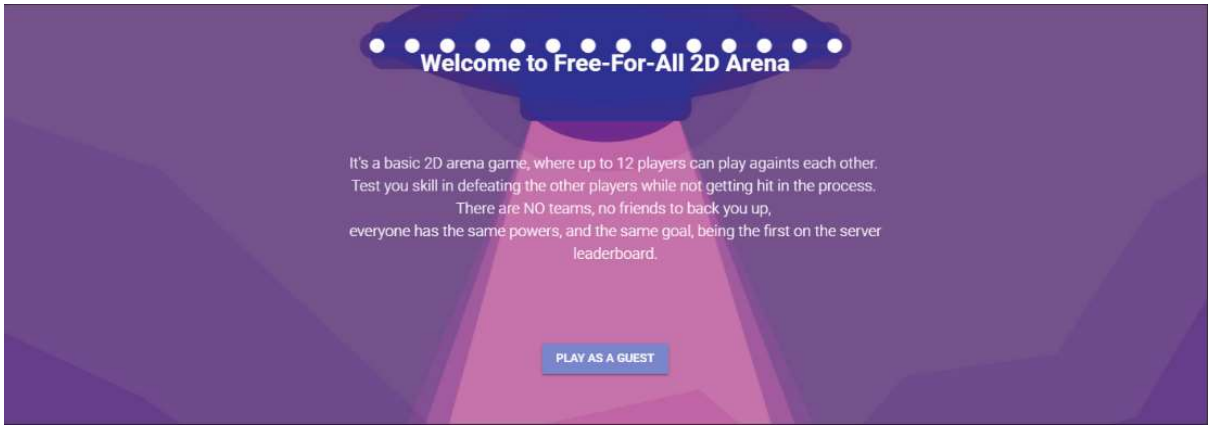
2.5. ábra.: Bejelentkezett felhasználó fejléc

2.4.1.1 Kezdőlap

A kezdőlapot látja meg a felhasználó először, amikor meglátogatja a weboldalt. Három ’kártya’ helyezkedik el egymás alatt különböző információkkal.

Üdvözlő kártya:

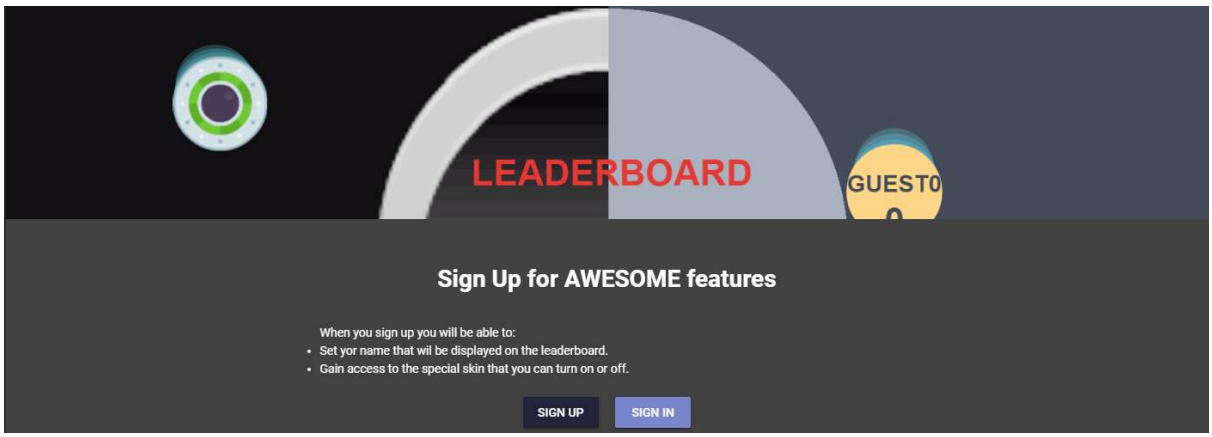
Röviden összefoglalja mi a weboldal célja, és ismerteti a játékot továbbá, ha a felhasználó nincs bejelentkezve akkor egy gomb (’***PLAY AS A GUEST***’) is látható, aminek segítségével beszállhat a játékba vendégként. (*lásd.: 2.6.ábra*)



2.6. ábra.: Üdvözlő kártya

Regisztrációs kártya:

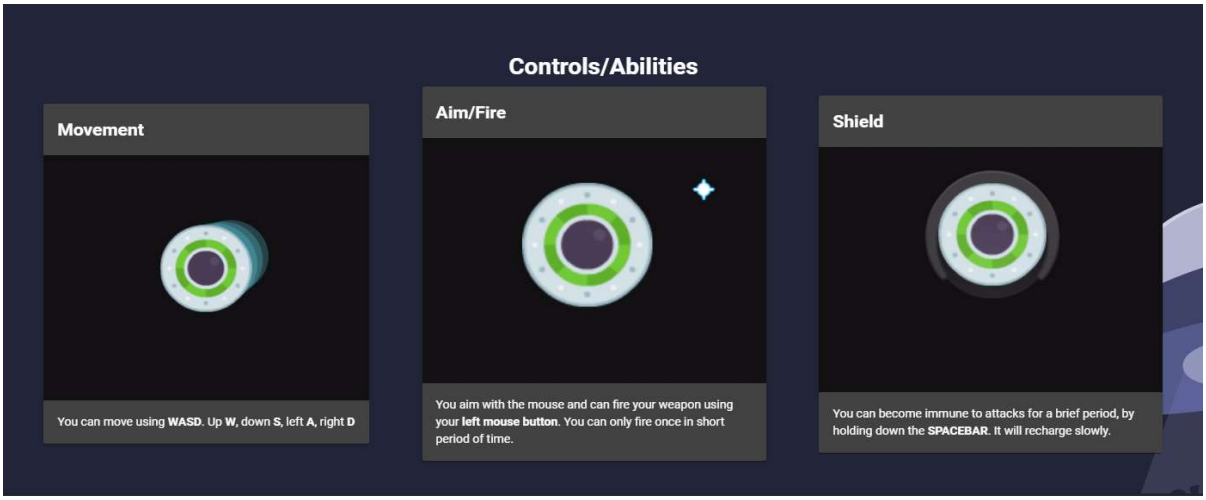
Csak akkor jelenik meg, ha a felhasználó nincs bejelentkezve. Ismerteti a regisztrációval járó szolgáltatásokat továbbá, gombokat is tartalmaz, amelyek a regisztrációs és bejelentkező oldalra irányítanak. (*lásd: 2.7. ábra*)



2.7. ábra.: Regisztrációs kártya

Tipp kártya:

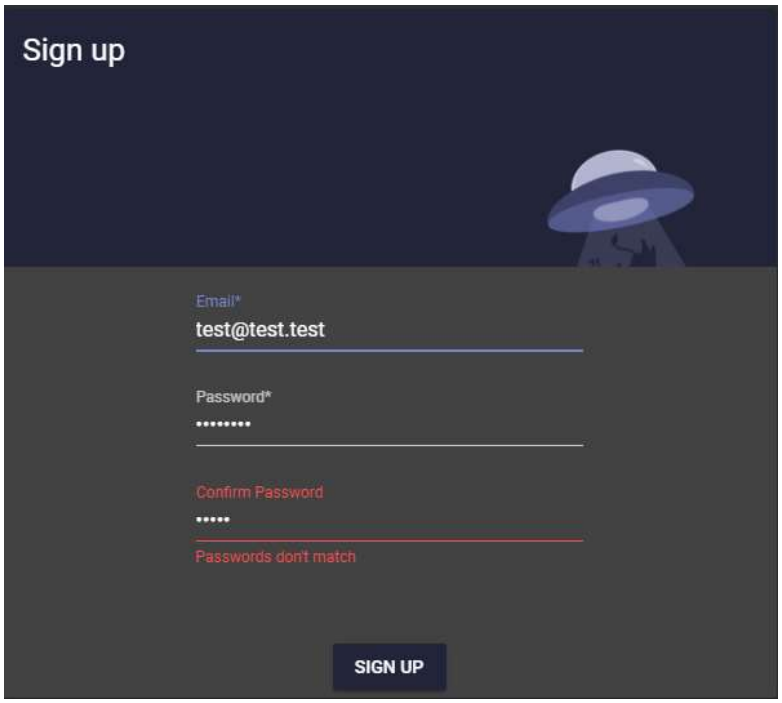
Bemutatja a játékos képességeit és azok használati módját. (*lásd.: 2.8. ábra*)



2. 8. ábra.: Tipp kártya

2.4.1.2 Regisztráció

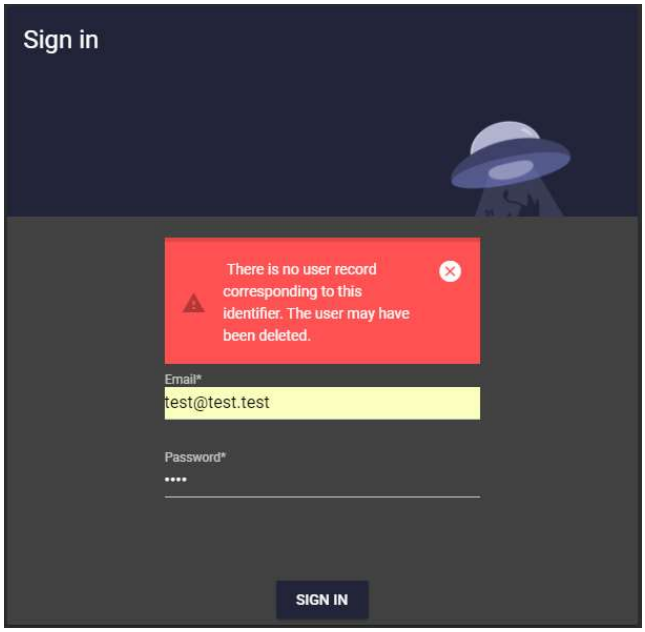
A lényegi funkció eléréséhez nem szükséges, de hasznos extra funkciókat tesz elérhetővé. A regisztrációhoz szükséges egy email cím és egy jelszó (amit meg kell ismételni). Az adatok megadása után a ***'SING UP'*** gombot megnyomva véglegesíteni kell. Hiba esetén (nem email címet adott meg; túl rövid a jelszó; etc.) a felület jelezni fogja a hiba eredetét szövegesen és vizuálisan is. Sikeres regisztráció után a bejelentkezés automatikusan megtörténik. (*lásd.:2.9*)



2.9. ábra.: Regisztráció

2.4.1.3 Bejelentkezés/Kijelentkezés

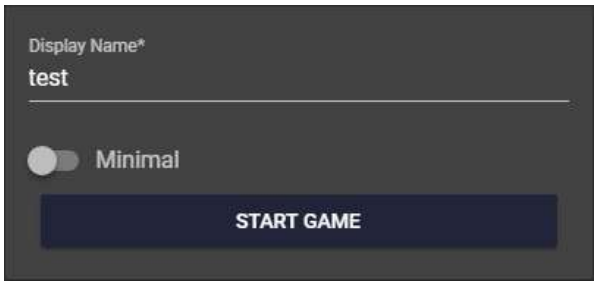
Már létező felhasználóval a ***'SIGN IN'*** menüpontba lehet bejelentkezni. Hibás adat megadás esetén jelzi a hibát. (*lásd.: 2.10.ábra*). Ha a felhasználó már be van jelentkezve akkor a ***'SIGN OUT'*** menüpont jelenik meg a ***'SIGN IN'*** helyett, amivel a felhasználó ki tud jelentkezni.



2.10. ábra.: Bejelentkezés

2.4.1.4 Home

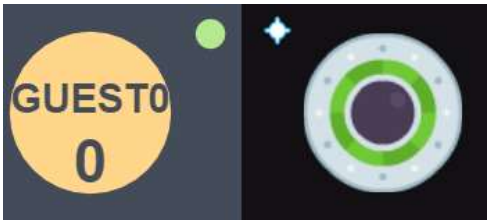
Bejelentkezés után ide kerül a felhasználó, ahol a ***'START GAME'***-re kattintva beléphet a játékba. Játékba szállás előtt meg kell adnia azt a nevet ('***Display Name***') amit látni fog a többi játékos, a felület generál egy nevet a felhasználó email címéből, de ezt tetszőlegesen lehet testre szabni a játék előtt. A felhasználó választhat a ***'Minimal'*** kapcsoló segítségével, hogy szeretné-e használni az regisztrációval járó játék textúrákat vagy sem. (*lásd.: 2.11.ábra*)



2.11. ábra.: Home

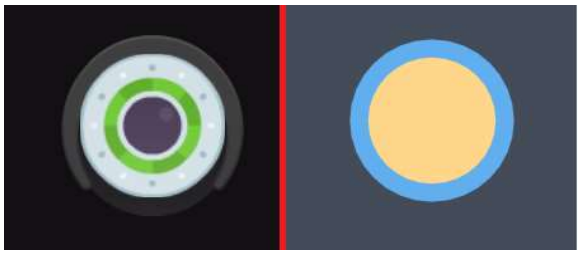
2.4.1.5 Játék

A játékos amikor belép a játékba a játéktér közepén található körnél (*lásd: 2.14.: ábra*) kezd és a billentyűzet **WASD** billentyűit használva tud mozogni. Minden játékos tud lövést leadni az egér gomb lenyomásával, a lövés irányát a kurzor aktuális pozíciója határozza meg, a lövések között egy kis időnek el kell telnie.



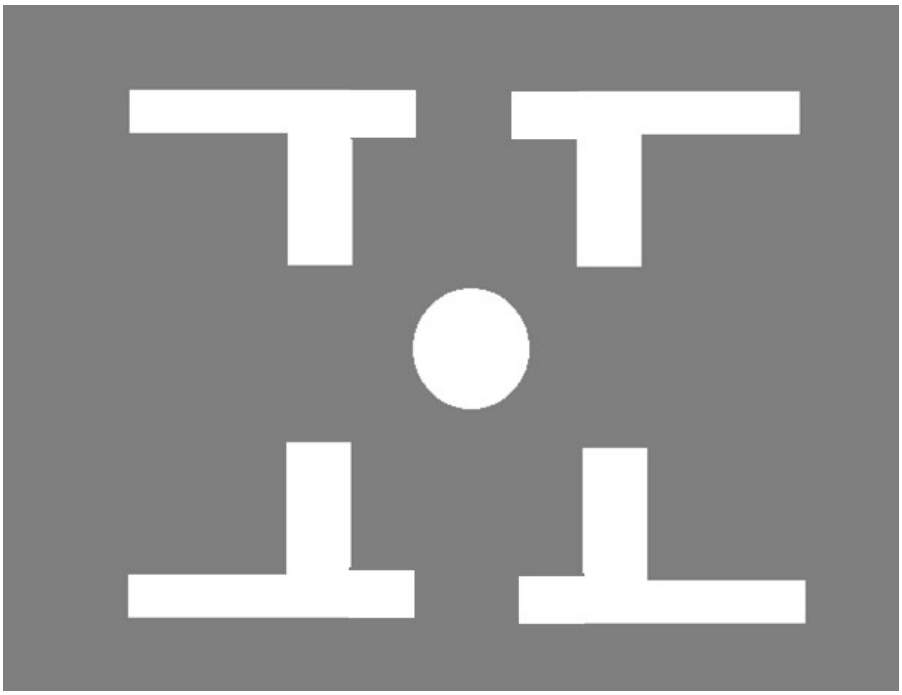
2.12. ábra.: Lövés

Továbbá a **SPACE** billentyű nyomva tartásával lehet rövid ideig immunissá válni a lövedékekkel szemben (*lásd.:2.13.ábra)*, használat után lassan újra töltődik miután ismét lehet használni. Ha a játékosok ütköznek akkor elpattannak egymástól viszont, ha egy mozdulatlan játékosnak mennek neki akkor csak a mozgó játékos pattan vissza.



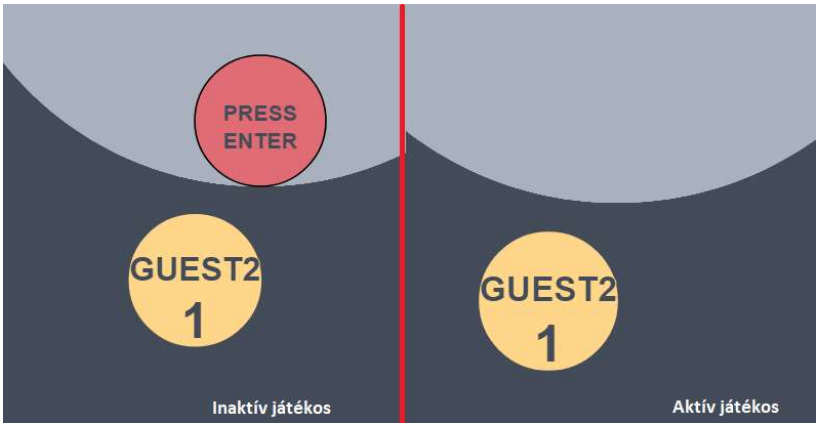
2.13. ábra.: Hárítás

A játéktér öt darab akadályt tartalmaz, ezen objektumokon a játékos és a játékos lövedéke nem tud átmenni, a pálya határai is hasonlóan működnek. A középen található kör tartalmazza a ranglétrát, ahol a három legtöbb pontot elérő játékos neve szerepel, nevük mellett meg a legtöbb pont, amit egy '*élet*' alatt elértek.



2.14. ábra.: Játéktér vázlat

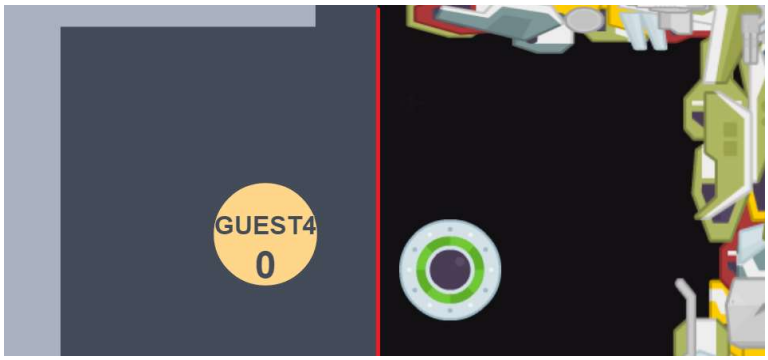
Pontot szerezni játékosok eltalálásával lehet, de ha eltalálnak egy játékost akkor annak az aktuális összeszedett pontjai elvesznek és visszakerül a játéktér közepére inaktív módban. Inaktív módban a játékos a középső objektum tizenkettő előre meghatározott pont egyikében vár arra, hogy visszalépjen a játékba, ezt az **ENTER** billentyű leütésével lehet megtenni. Mindeközben nem látható az ellenfelek számára, de az inaktív játékos lát minden aktív játékost, hogy biztonságosan vissza tudjon lépni a játékba. (*lásd.: 2.15.ábra*)



2.15. ábra.: Aktív vs Inaktív láthatóság

Egy játékos bármikor elhagyhatja a játékot, ekkor a ranglétrán elért helyezése megmarad azon a néven, amivel csatlakozott a játékba, ha változtat nevet és visszatér akkor a ranglétrán lévő régebbi még más néven levő eredménye NEM változik.

A játék, ha a felhasználó nincs bejelentkezve, ’*minimal*’ módban fog futni, ebben az esetben nincsenek textúrák, továbbá a játékos neve **Guest[X]** lesz. ’*Minimal*’ módban a játékos neve a saját avatárján szerepel, alatta pedig az aktuális pontja. Bejelentkezett felhasználó dönthet úgy, hogy ’*minimal*’ módban játsszon, de nem ez az alapértelmezett. (*lásd.: 2.16.ábra*)



2.16. ábra.: Minimal mód

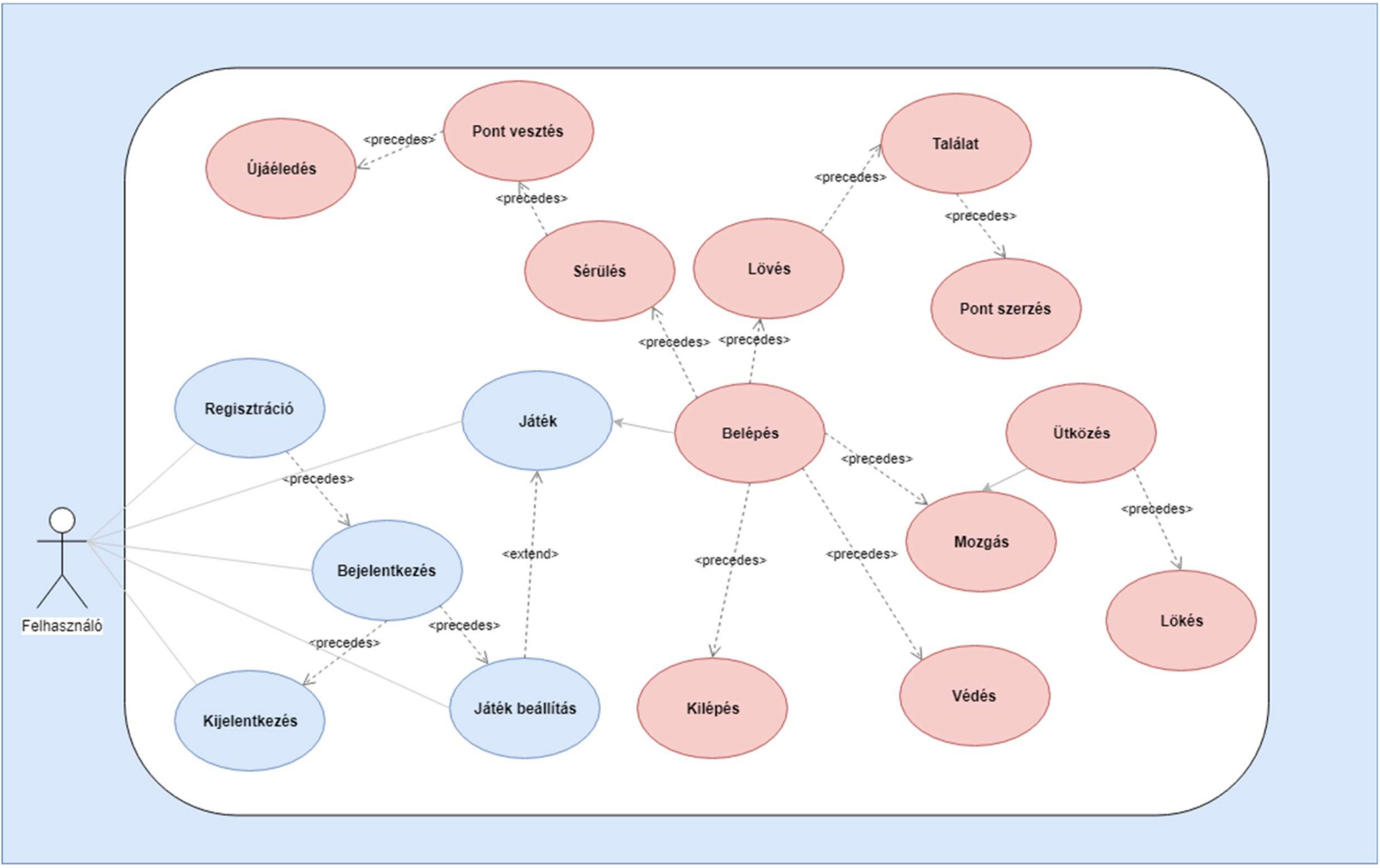
3 Fejlesztői dokumentáció

3.1 Megoldási terv

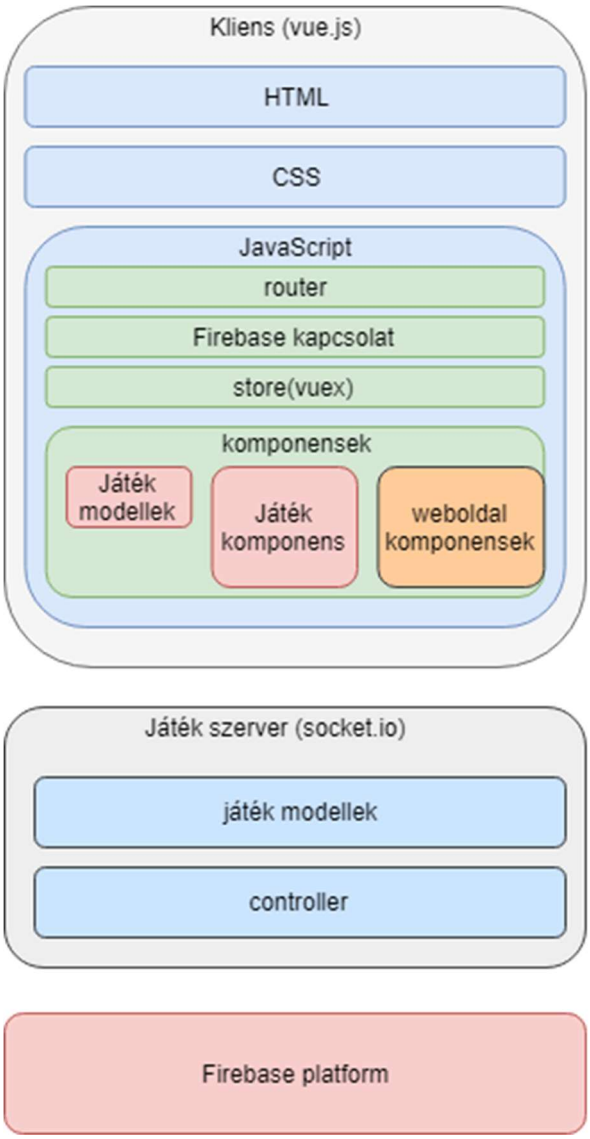
Mivel egy internetes, valós idejű, többjátékos webalkalmazásról van szó ezért két fő részre lehet bontani a problémát. Kell egy kliens oldali alkalmazás, ami maga a játék és egy webszerver ami a sok kliens közötti valós idejű kommunikációt oldja meg. Egy mellékes rész a játékhoz tartozó körítés, egy kezdő oldal, ahol a játék röviden el van magyarázva, valamiféle regisztrációs opciók többlet funkcióval, ami manapság minden webalkalmazásnál megszokott.

A webalkalmazás főbb funkcióit a 3.1.-es ***use case diagrammon*** lehet látni. Két felé lehet csoportosítani az alkalmazást a felhasználót figyelembe véve, maga a weboldal(*kék*) ami tartalmazza a regisztrációs funkciókat és a játékra(*piros*).

A teljes alkalmazás structúra a 3.2.-es ábrán látható.



3.1. ábra.: Use Case Diagram



3.2. ábra.: Alkalmazás struktúra

3.2 Használt technológiák

3.2.1 Szerver technológiák

A játékszerver **Node.js**[1]-en fut. A **Node.js** egy *open source*, *platform független*, skálázható, szerveroldali *JavaScript* futás idejű környezet. Lényegében lehetővé teszi, hogy szerver oldalon futtassunk *JavaScript*-et ezáltal dinamikussá téve a webapplikációnkat. Esemény vezérelt architektúrát alkalmaz aszinkron bemenet/kimenet kezeléssel, emiatt valós idejű webalkalmazások többségnél használva van.

Mivel egyszer több felhasználó közötti gyors, valós idejű, kommunikációra van szükség ezért az egyszerű *HTTP* alapú üzenet küldés túl lassú lenne, helyette a *WebSocket* technológiát érdemes használni, itt jön be a **Socket.io**[6].

A **Socket.io** egy *JavaScript* könyvtár kliens és szerver oldali (Node.js) alkönyvtárakkal, a *WebSocket* technológia egy tovább gondolása *(több funkcióval rendelkezik*). Hasonlóan a **Node.js**-hez esemény vezérelt és aszinkron műveletekre képes, rendkívül hasznos amikor egyszerre több felhasználó kéréseire kell reagálni és válaszolni.

A felhasználó hitelesítésére a **Firebase**[7] fejlesztői platformot használom, azon belül a **Firebase Auth** szolgáltatást. A **Firebase** egy *Google* által nyújtott szolgáltatás csomag mely segítségével gyorsan lehet jó minőségű webes és/vagy telefonos alkalmazásokat létrehozni. A **Firebase Auth** többek között lehetővé teszi, hogy csak kliens oldalon kelljen foglalkozni a felhasználó hitelesítésével.

Továbbá egy egyszerű **Express**[10] webszervert is tartalmaz a forráskód, aminek egyetlen célja a lefordított webalkalmzás hosztolása. **Express** egy Node.js webalkalmazás keretrendeszer, szerverek és API-k létrehozásához.

3.2.2 Kliens technológiák

Minden weboldal építő elemei a **HTML**, ami leírja milyen tartalom szerepel az oldalon, a **CSS** ami meghatározza a tartalom formázását, és a **JavaScript**[8] mely lehetővé teszi a weboldal dinamikusságát.

A felhasználói felület **Vue.js**[3] keretrendszeren készült, mely lehetővé teszi, hogy a weboldalt komponensekből építsük föl melyek tartalmazzák a hozzájuk szükséges HTML, CSS, és JavaScript elemeket, így egy adott komponens mindenhol egységesen fog működni és megjelenni.

Vuex[4] egy kiegészítő könyvtár, melynek segítségével az állapot kezelést oldjuk meg. A **vuex**-ben a weboldal állapotát egy vagy több kijelölt objektumban tároljuk és minden oldal ezen objektumokhoz fordul, ha valamit szeretne az állapottal tenni legyen az egy lekérés vagy egy módosítás.

A **CSS** egyszerű és konzisztens használatához a **Vuetify**[5]-t használjuk. A **Vuetify** egy **Vue.js**-hez készített *UI toolkit* ami a Google által kitalált **Material Design** specifikációt követi, fő célja a reszponzív megjelenés és konzisztens stílus biztosítása.

Vue.js saját fájl kiterjesztést is használ a megszokott **HTML**, **CSS**, és **JavaScript** fájlok mellett, mivel saját szintakszis és felépítés(*SFC*[1]) is társul hozzá. Emiatt hogyha azt szeretnénk, hogy a böngésző megtudja jeleníteni le kell fordítani egyszerű **HTML**, **CSS**, és **JavaScript**-re.

A 'fordítást' a **Webpack**[8] oldja meg, ami egy statikus modul csomagoló mely a függőségekből statikus eszközöket hoz létre. Rendkívül moduláris felépítésű, ezért a **vue-loader** hozzáadásával képes Vue.js fájlokkal is dolgozni. További fontosabb használt kiegészítő a **Babel**-hez szükséges **babel-loader.**

[1] Single File Components: Minden komponens egy fájlból áll ami minden szempontból definiálja, szóval tartalmaz CSS-t, HTML-t, és JavaScript-et is.

Mivel a webtechnológiák rohamosan fejlődnek, főleg a **JavaScript,** ezért a böngészők nem mindig tudnak lépést tartani, emiatt kompatibilitási hibák keletkeznek. Ennek megoldására jött létre a **Babel** mely a friss **JavaScript** szabványban (manapság **ES6**) íródott **JavaScript**-et lefordítja funkcióvesztés nélkül olyan régebbi szabványra melyet képesek a régebbi böngészők is értelmezni.

A játék a **HTML** *canvas* objektumára van kirajzolva, a *canvas* használatának megkönnyebbítésére a **p5.js**[2] JavaScript könyvtárat használjuk. Ahhoz, hogy a játék tudjon kétirányba kommunikálni a szerverrel szükséges a **Socket.io** kliens oldali könyvtára is.

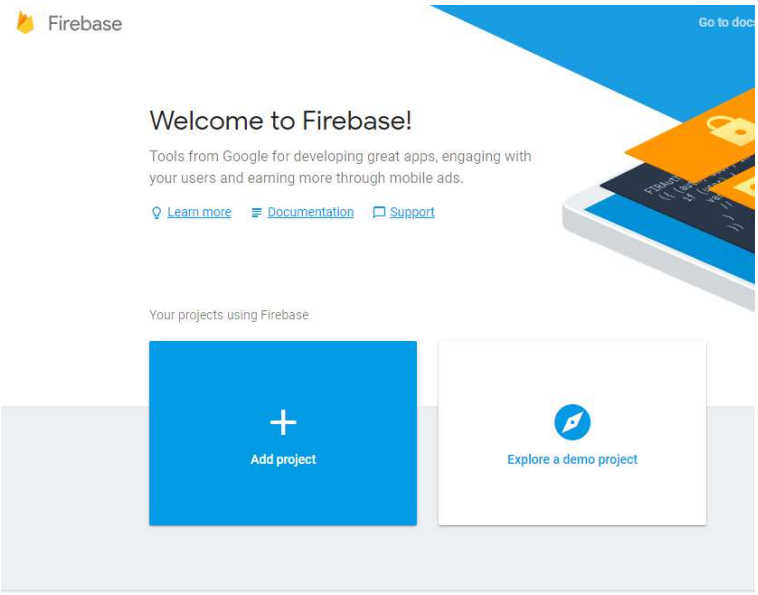
3.2.3 Egyéb technológiák

Egy nagyobb project fejlesztéséhez már-már elengedhetetlen valamilyen verziókövető rendszer használata. A verziókövetés gondoskodik arról, hogy minden változtatás el legyen tárolva és vissza lehessen nézni/állítani bármikor, ezáltal nyugodtan lehet kísérletezni fejlesztés közben. Az egész project elejétől a végéig használja a **Git** verziókövető rendszert.

3.3 Server oldali implementáció

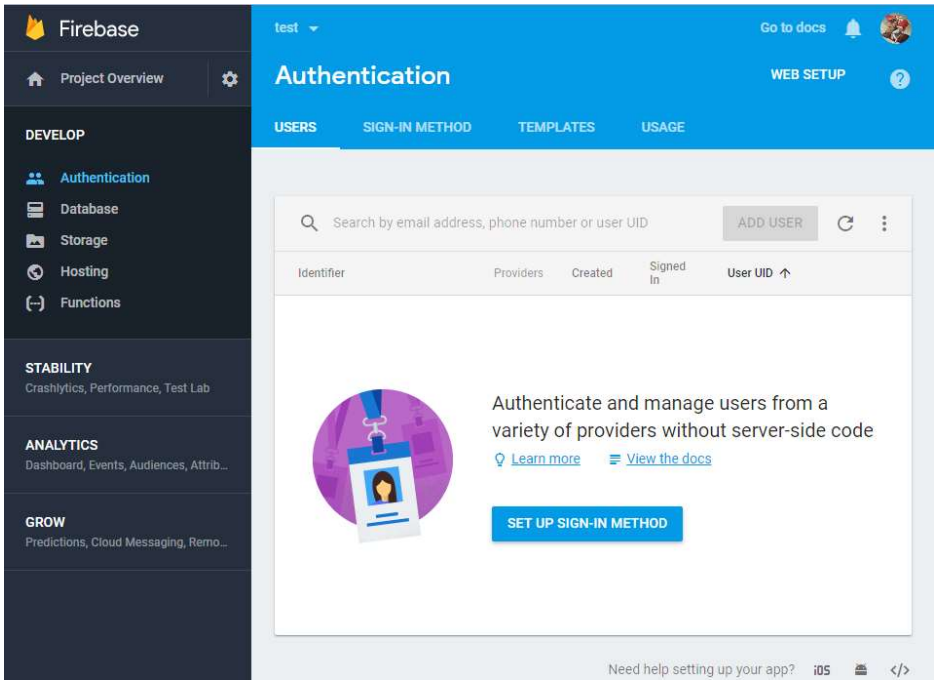
3.3.1 Autentikáció (*Firebase*)

A felhasználók hitelesítéséhez a *Google* által szolgáltatott **Firebase** platformot használjuk. Egy *Google* fiókkal kell rendelkezni, hogy hozzáférhessünk, kisebb alkalmazások esetén ingyenes persze ilyenkor vannak megkötések (limitált felhasználó létszám; stb.), de bármikor lehet a korlátozásokat csökkenteni és tovább bővíteni szolgáltatásokkal, fizetés fejében. Ha *Google* fiókkal rendelkezünk és ellátogatunk a *https://console.firebase.google.com* oldalra létrehozhatunk egy új projektet. (*lásd: 3.3.ábra*)

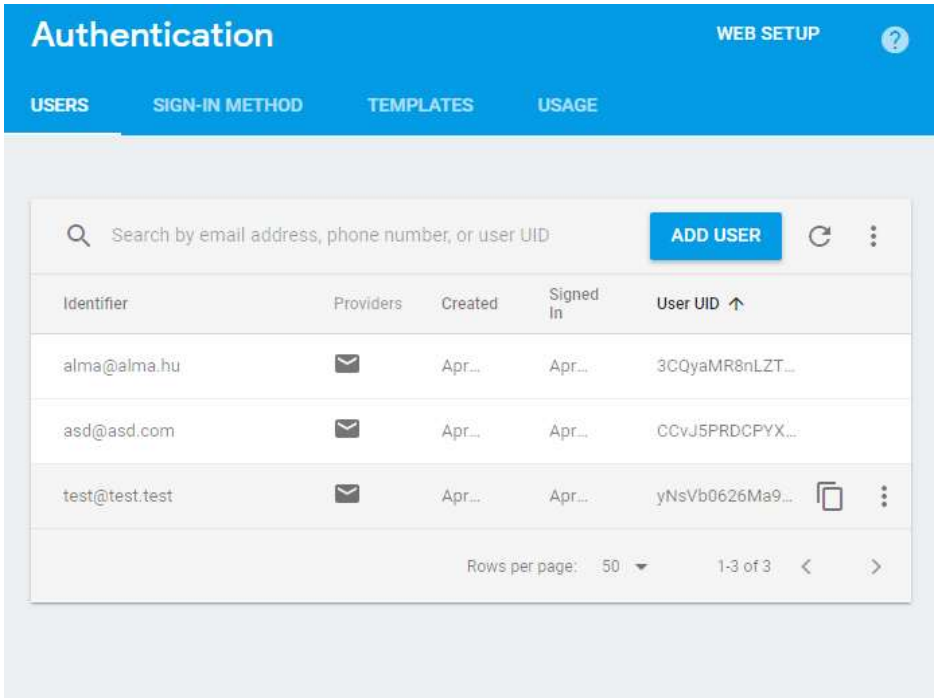


3.3. ábra.: Add project

Utána nevet és régiót adva a projectnek egy pici várakozás után elkészül az üres project. A project **Firebase console** *DEVELOP* menüpont alatt az *Authentication* almenüpontra kattintva lehet inicializálni a regisztrációs opciókat. (*lásd:3.4.ábra*) Inicializáció után itt lehet megtekinteni a regisztrált felhasználókat. (*lásd:3.5.ábra*)

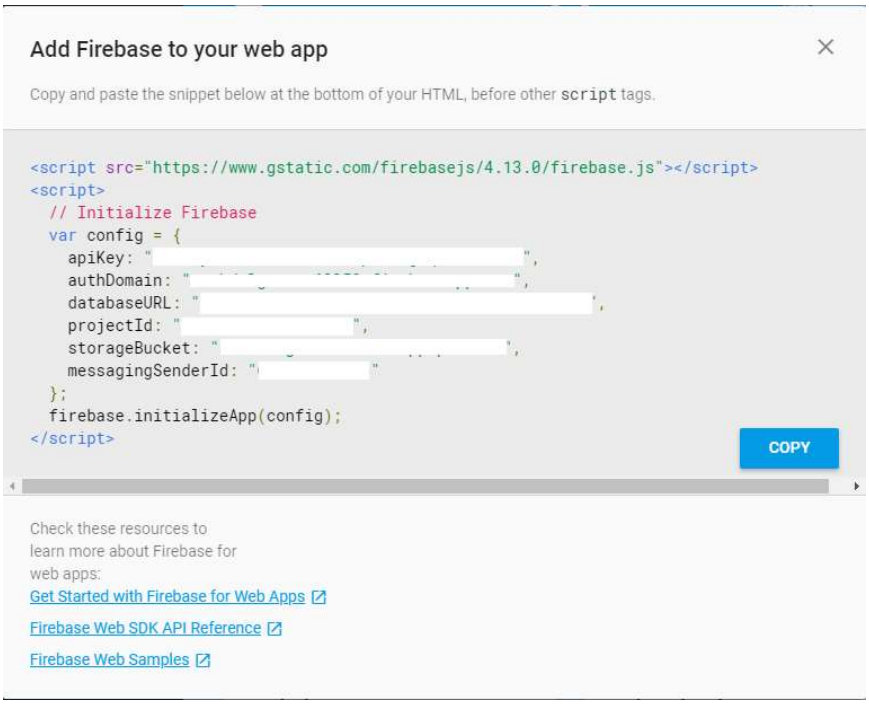


3.4. ábra.: Authentication



3.5. ábra.: Users

Ha elszeretnénk érni a klienssel akkor a *Project Overview* oldalon látható *Add Firebase to your web app* gombra kattintva megjelenik a szükséges *api kulcs* és más fontos információ, ami kell, hogy csatlakozzunk a szolgáltatáshoz. (*lásd:3.6.ábra*)



3.6. ábra.: Api kulcs

Használt metódusok

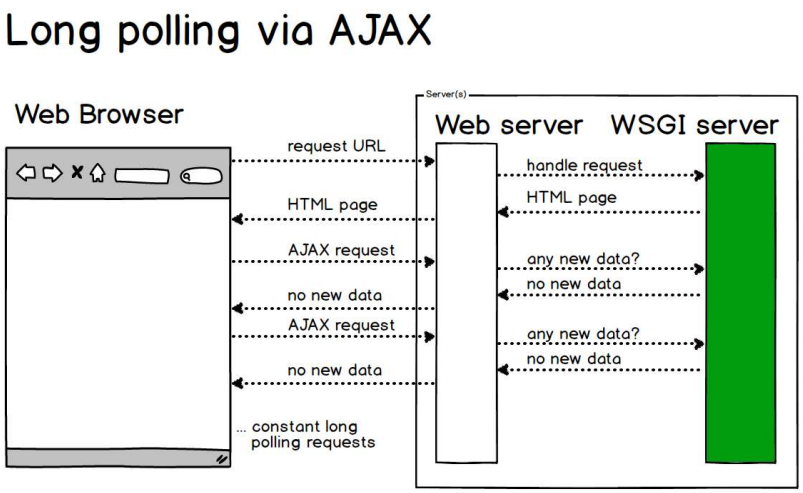
- ***firebase.auth().createUserWithEmailAndPassword(email, password)***:
  - Létrehoz egy új felhasználót az email és jelszó alapján. Sikeres létrehozáskor a felhasználó be lesz jelentkezve. Ellenőrzi, hogy volt-e már használva az email, és a jelszó erősségét is ellenőrzi.
- ***firebase.auth().signInWithEmailAndPassword(email, password)***:
  - Bejelentkezik a megadott email és jelszó párossal. Hibát dob, ha helytelen a jelszó vagy az email cím.
- ***firebase.auth().signOut()***:
  - Kijelentkezteti a felhasználót.
- ***firebase.auth().onAuthStateChanged()***:
  - Az aktuálisan bejelentkezett felhasználót a böngésző *IndexedDB*-jében tárolja el a *Firebase*. *IndexedDB* egy *NoSQL* tárolóhely a böngészőben, amiben majdnem bármit el lehet tárolni. Ha az ott lévő felhasználót használni szeretnénk akkor egy megfigyelőt (*obeserver-t*) állítunk rá, ami figyeli bejelentkezési állapotot és ha talál egy felhasználót akkor azt visszaadja nekünk, hogy dolgozni tudjunk vele.

3.3.2 Játékszerver

3.3.2.1 Miért WebSocket ?

A HTTP protokoll egy egyirányú protokoll, ahol a kliens szeretne valamit (általában egy weboldal tartalmát) aminek érdekében egy vagy több port-on felépíti a kapcsolatot a vágyott tartalmat szolgáló szerverrel, aki ezekután elküldi a kért tartalmat és bontja a kapcsolatot. Egyszerű blogok és hírportálok böngészéséhez teljesen megfelelő, de a böngészők és webes technológiák rohamos fejlődése miatt olyan alkalmazás típusok kezdtek megjelenni a weben, amik sokáig elérhetetlenek voltak, mint például játékok, chat szolgáltatások, dokumentum szerkesztés stb.

Ezen alkalmazásoknak már folytonos kommunikációra volt szüksége a szerver és kliens között, amit egy bizonyos szintig meg lehet oldani *polling* segítségével, ami annyit takar, hogy a kliens bizonyos időközönként megkérdezte a szervert, hogy *'történt-e valami'*, ekkor ugyanúgy felépítve, majd bontva a HTTP kapcsolatot.

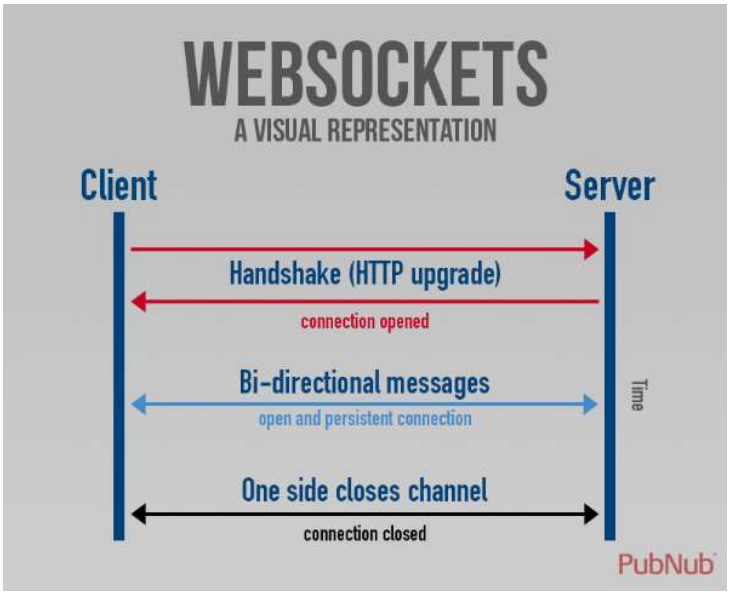


3.7. ábra.: polling[2]

Érezhető, hogy ez nem valami hatékony. Ha feltesszük, hogy van egy körökre bontott játék 2 játékos között, ahol a játékosnak van 2 perce, hogy lépjen és ha valaki hamarabb befejezi a körét akkor egyből a másik játékos jön. Ebben az esetben, ha kliens *'csak'* másodpercenként kérdezi, hogy léphet-e, akkor megeshet az az eset is, hogy 119 alkalommal kérdezte meg a szervert, de csak a 120. alkalomra kapta meg a várt választ a szervertől. Itt volt 119 teljesen felesleges üzenetváltás, ami csak terhelte a hálózatot és a szervert, és itt még nem is beszélünk igazi valós idejűségről.

Szerencsére 2011-ben szabványosítva lett a *WebSocket* protokoll, ami egy kétirányú, hosszú élettartamú kapcsolatra képes protokoll. Itt már lehetséges az, hogy a szerver szóljon a kliensnek, lehetséges egy folytonos kétirányú adatfolyam létesítése. Továbbá a WebSocket fejléce is sokkal kisebb, ami miatt jóval gyorsabb és kevésbé terheli meg a hálózatot.

[2] Forrás: https://www.fullstackpython.com/websockets.html



3.8. ábra.: WebSocket[3]

3.3.2.2 Játék állapot

Socket.io használva hozzunk létre egy *WebSocket* szervert, ami a játék állapotát tárolja és sugározza a játékosoknak, továbbá a játékosok által küldött információ alapján frissíti a játék állapotát. A **játék állapota (gameState)**, amit minden kliensoldalon történt változás után kisugározunk minden játékosnak. Ez egy lista melynek három eleme van melyek maguk is listák:

- ***játékosok listája (players)***
- ***lövések listája (shots)***
- ***ranglétra(leaderboard)***

További fontos játék állapotok:

- ***availableSpawn***: A játéktér közepén elhelyezkedő körre tizenkettő, egymástól és a kör középpontjától azonos távolságra lévő pontokat definiálunk. (kliens oldalon van ezen pontok pontos pozíciója) Minden játékoshoz tartozik egy ilyen pont, itt fog mindig létrejönni a karakterük vagy itt fognak inaktívan várni. Számon kell tartani, hogy melyik játékoshoz melyik *'spawn pont*' tartozik, hogy ne tudjanak egymáson létrejönni. Erre van a ***availableSpawn*** lista mely tartalmazza a szabad *'spawn pontok*'.
- ***shotPlayers***: egy lista mely tartalmazza azon játékosokat, akik inaktívak, ezen játékosok nincsenek benne a küldendő játékosok listájában, hogy a többi játékos ne lássa őket. Csak a szerver tartja számon jelenlétüket.
- ***guestCounter***: a vendégjátékosoknak dinamikusan kell nevet adni, egy számláló segítségével tesszük ezt.

[3] Forrás: https://www.pubnub.com/websockets/

3.3.2.3 Játékos modellje (szerver szerinti)

- ***id***: Minden *socket*-nek létrejöttekor kap egy egyedi azonosítót mely segítségével küldeni lehet neki parancsokat. Ezt az azonosítót használjuk a játékos azonosítására is.
- ***user***: A játékos egyedi felhasználó neve, amit regisztrációkor használt, a mi esetünkben ez egy email cím.
- ***name***: A játékos választott neve, aminek nem feltétlenül kell egyedinek lennie. Az a karaktersor amit megjelenítünk a ranglétrán.
- ***score***: A játékos aktuális pontszáma, ami nullázódik, ha eltalálják.
- ***bestScore***: A játékos kapcsolódása óta összegyűjtött legtöbb pontszáma.
- ***spawn***: A fentebb említett '*spawn pont*' sorszáma, ami a játékoshoz tartozik.
- ***positionX***: A játékos pozíciója az **X** tengely mentén.
- ***positionY***: A játékos pozíciója az **Y** tengely mentén.
- ***velocityX***: A játékos sebesség vektor **X** koordinátája.
- ***velocityY***: A játékos sebesség vektor **Y** koordinátája.
- ***shield***: Jelzi, hogy a játékos védekezik-e.

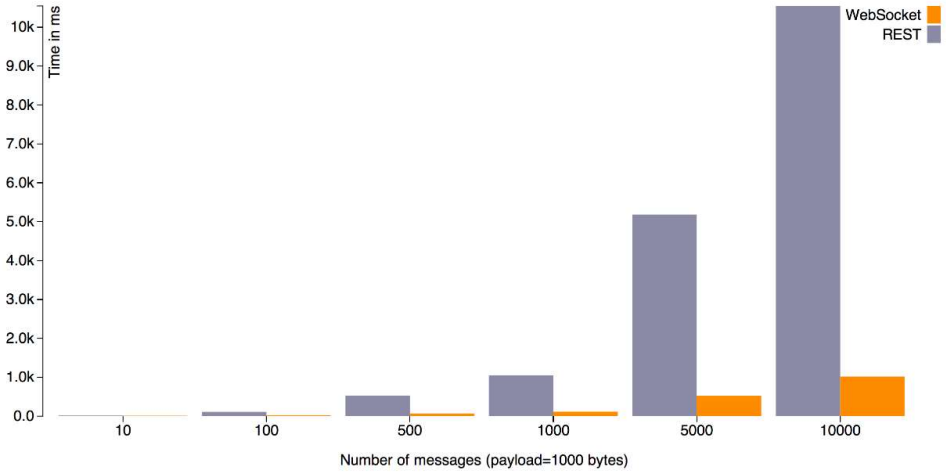
3.3.2.4 Lövedék modellje (szerver szerinti)

- ***user***: A lövedéket kilövő játékos felhasználó neve, a mi esetünkben ez egy email cím.
- ***id***: A lövedék azonosítója mely játékosokként egyedi.
- ***positionX***: A lövedék pozíciója az **X** tengely mentén.
- ***positionY***: A lövedék pozíciója az **Y** tengely mentén.
- ***velocityX***: A lövedék sebesség vektor **X** koordinátája.
- ***velocityY***: A lövedék sebesség vektor **Y** koordinátája.
- ***ttl***: Egy lövedék csak egy megadott ideig lehet játékban, utána megsemmisül. Ennek követésére használt számérték.

3.3.2.5 Események

A HTTP protokollal ellentétben a *WebSocket-nél* az üzenet küldés kétirányú és full-duplex melynek következtében egyszerre mindkét irányba haladhat az információ. Egyetlen TCP kapcsolatot használ egy klienssel való kommunikációhoz, aminél csak egyszer hajtja végre a *HTTP kézfogást* és csak akkor zárja le, ha a kliens lekapcsolódott, míg a HTTP minden kérésre nyit egy új TCP kapcsolatot, ahol lejátszódik a *HTTP kézfogás*. Sebesség összehasonlítás a *3.9. ábrán* látható.

A *WebSocket***,** és emiatt a **Socket.io** is, esemény vezérelt, ami annyit tesz, hogy eseményekre reagálnak és eseményeket keltenek a kommunikációban résztvevők. Minden eseménynek van egy tetszőleges neve. Amikor valaki küld egy eseményt annak általában van valami tartalma/rakománya, amit a címzett feldolgoz miután fogadta az eseményt. Kétfelé lehet csoportosítani az eseményeket a mi esetünkben, a szerver által figyelt események és a kliens által figyelt események. A kommunikáció menete a *3.10.ábrán* látható.



3.9. ábra.: Üzenetek száma és sebessége REST vs WebSocket[4]

Szerver által figyelt események:

- ***connection***: Amikor a kliens csatlakozik a szerverhez akkor jön létre ez az esemény.
  - Beépített *event,* a *socket* adatait kapja rakományként.
  - Ezen eseményen belül van definiálva, hogy az adott *socket*-en mely eseményekre, hogyan reagáljon a szerver. Továbbá ellenőrzi, hogy van-e még hely a játékban, ha nincs küld egy *goHome()* eseményt a kliensnek.

- ***userCheck***: Kliens azonnal kapcsolódás után küldi.
  - Rakományként egy *felhasználó nevet (**user**)* vár.
  - Ellenőrzi, hogy a rakományként megkapott felhasználó bent van-e már a játékban, ezzel kiszűrve azt a lehetőséget, hogy egy regisztrált felhasználó többször is belépjen. Ha nem kap semmit, mint rakomány akkor úgy gondolja, hogy vendég szeretne csatlakozni és generál egy nevet a *guestCounter* segítségével. Ha minden rendben van, vagyis új a játékos vagy vendég, ekkor sorsol neki egy kezdőpozíció sorszámot a *availableSpawn listából* és küld egy *spawn()* eseményt a kliensnek mely tartalmazza a *kliens azonosítóját*, *pozíció sorszámát* és egy mezőt ami vendégek esetén használt nevet tartalmazza.

- ***start***: Kliens sikeres kezdés után küldi.
  - Rakományként várja a
    - *kliens azonosítót (**id**)*
    - *felhasználó nevet (**user**)*
    - *játékos nevet (**name**)*
    - *játékos kezdőpozíciót (**spawn**)*
    - *játékos aktuális X és Y pozícióját (**positionX, positionY**)*.
  - Ellenőrzi, hogy kapott-e adatot, utána példányosít egy *Player*-t amit elment a *players* listába, és kiveszi az elérhető *'spawn pontok'* közül a már nem elérhetőt.

[4] Forrás: http://blog.arungupta.me/rest-vs-websocket-comparison-benchmarks/

- ***update***: Kliens helyi változás után küldi.
    - Rakományként várja a
        - *játékos aktuális X és Y pozícióját (**positionX, positionY**)*
        - *játékos aktuális sebességének X és Y érékét (**velocityX, velocityY**)*
        - *játékos aktuális pajzsának állapotát(**shield**)*
        - *a játékos aktuális pontját (**score**)*
        - *egy a játékos lövedékeit tartalmazó listát(**shots**),* mely elemei(***shot***) a következő adattagokat tartalmazzák*:*
            - *Lövedék azonosítója* (***shot.id**)*
            - *Lövedék X és Y koordinátája* (***shot.positionX, shot.positionY***)
            - *Lövedék élettartalma* (***shot.ttl***)
            - *Lövedék sebességének X és Y értéke* (***shot.velocityX, shot.velocityY***)

    - Ellenőrzi, hogy kapott-e adatot, utána megkeresi a küldő játékost a *players* listában, ha megtalálja akkor frissíti a küldött adatok alapján. Frissíti a ranglétrát (leaderboard) ha szükséges. Ha jött hozzátartozó lövedék adat akkor egyenként megkeresi a lövedékek között és az új adatok alapján frissíti. Ha egy lövedék *ttl* adattagja kisebb, mint nulla akkor frissíti a játék állapotot és kiküldi egy *heartbeat* eseménnyel mielőtt kivenné a saját *shots* listájából ezzel szólva a kliensnek, hogy az adott lövedéket el kell távolítani. Az adatok frissítése után minden *socket*-re küld egy *heartbeat* eseményt.

- ***enemyHit***: Kliens sikeres találatkor küldi.
    - Rakományként várja a *lelőtt játékoshoz tartozó azonosítót*(***targetId***)
    - Ellenőrzi, hogy kapott-e adatot, utána megkeresi a lelőtt játékost a *players* listában és ha megtalálja akkor figyelmezteti, hogy eltalálták és átteszi a *shotPlayers* listába ezáltal inaktívvá téve.

- ***respawn***: Kliens küldi, ha *'újjáéledt*' az inaktív játékos.
    - Rakományként nem vár semmit.
    - Megkeresi a *socket*-hez tartozó inaktív játékost a shotPlayers listában, ha megtalálja akkor átteszi a *players* listába ezáltal aktívvá téve az üzenő játékost.

- ***disconnect***: Kliens lecsatlakozásakor jön létre.
    - Beépített *event*, nincs rakománya.
    - Kiveszi az aktív vagy az inaktív játékosok közül a lecsatlakozó játékost, ha létezik, továbbá a lövedékeit is eltávolítja. Szól a klienseknek, hogy egy játékos elment.

Kliens által figyelt események:

- ***heartbeat***: Szerver küldi adatok frissítése után.
    - Rakománya a *játék állapota* mely három listát tartalmaz
        - *players:* Minden eleme egy *Player* objektum (a szerver modell alapján).
        - *shots:* Minden eleme egy *Shot* objektum (a szerver modell alapján).
        - *leaderboard:* egy lista, ami *user* és *score* adattaggal rendelkező elemeket tartalmaz.
    - A *players* és *shots* listán végig iterálunk, a *leaderboard*-ot meg felülírjuk.
      A *players lista* a fogadó játékost is tartalmazza, szóval amikor végig iterálunk a küldött játékosokon(*players*) figyelni kell rá, továbbá azt is figyelni kell, hogy új játékos-e az iteráció alatt éppen vizsgált játékos, ha igen akkor létre kell hozni egy új ellenfél játékost, ha meg a vizsgált játékos már létező, akkor annak összes adattagját frissíteni kell az új adatokkal.

      Lövedékeknél hasonló a helyzet, a fogadó játékos lövedékei is benne vannak a küldött lövedékek listájában(*shots*) amikre iteráció közben figyelni. Továbbá a lövedékekhez tartozó játékosokat is meg kell találni, azután eldönteni, hogy új a lövedék vagy sem. Új lövedék esetében a lövedék tulajdonosának a listájába kell belefűzni, már létező lövedéknél meg felülírjük az összes lehetséges adattagját a küldött adatokkal.

- ***spawn***: Szerver küldi, ha átment a felhasználó a **userCheck**-en
    - Rakománya a
        - ’*spawn pont*’ sorszáma (***spawnPoint***)
        - *socket* azonosítója (***id***)
        - vendégek esetében a vendég neve (***forGuests***)
    - A küldött és kliens oldali információk alapján inicializálja a játékost a kliens és küld egy *start()* eseményt a szervernek, ami után belép a játékba.

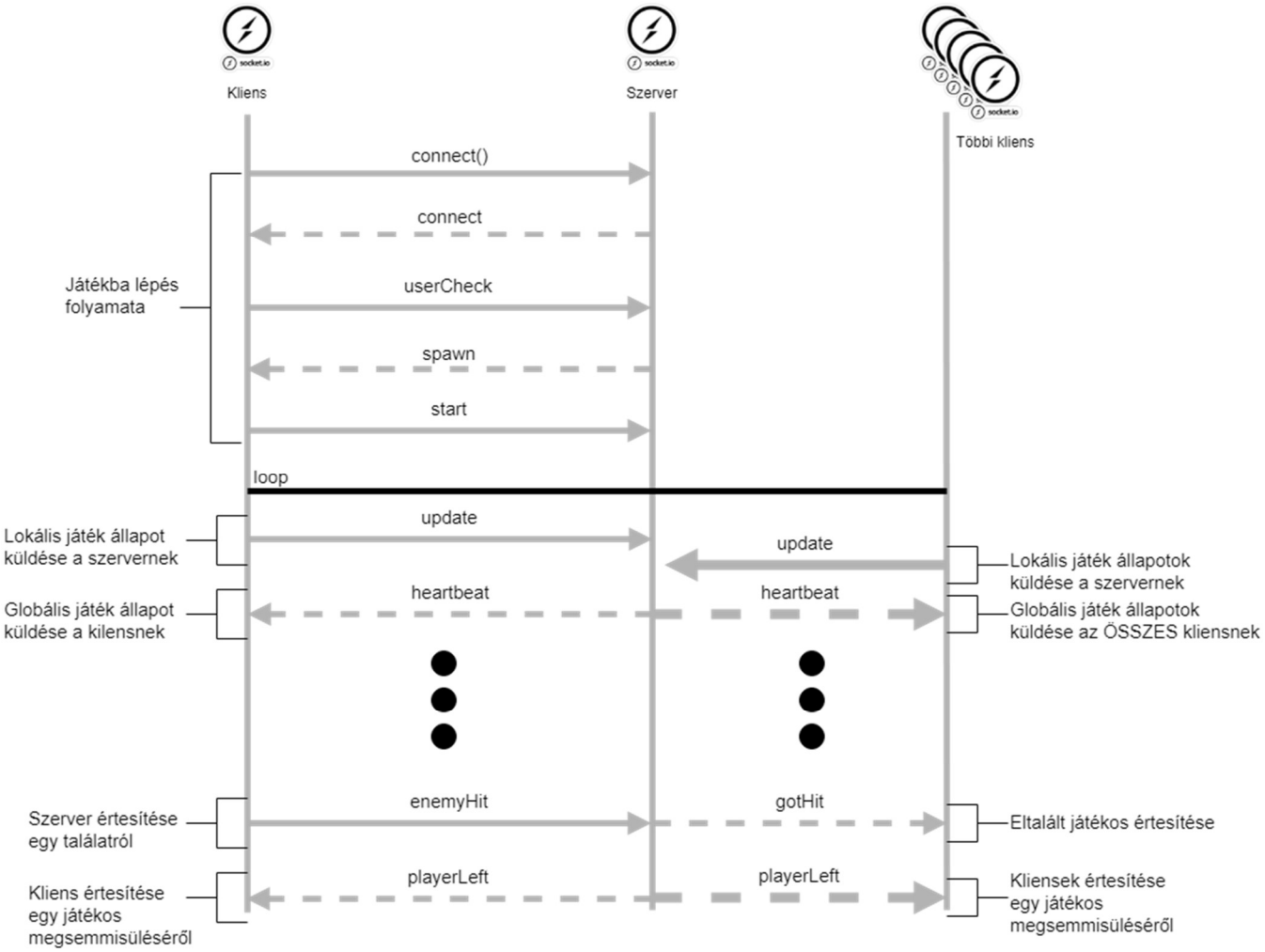
- ***goHome***: Szerver küldi, ha valami felhasználó által okozott hibát észlelt.
    - Rakománya egy *hibaüzenetet* tartalmaz.
    - Kliens visszalép a *Home* oldalra vagy vendég esetén a *Landing* oldalra, beállítja a hibát, amit megjelenít a kliens.

- ***playerLeft***: Szerver küldi, ha egy játékost ki kell törölni a helyi játék állapotból, mivel inaktív lett vagy elhagyta a játékot.
    - Rakománya az *eltávolítandó játékoshoz tartozó socket azonosító (**playerId**).*
    - Az azonosító alapján a kliens kitörli a helyi játék állapotból az érintett játékost.

- ***gotHit***: Szerver küldi, ha egy másik játékos eltalálta a címzett játékost
  - Rakomány a *lövő azonosítója* (***shooterId***).
  - A kliens inaktívvá teszi a játékost, ekkor bemozgatja a kezdőpozícióba és az aktuális pontját nullára állítja.

- ***connect_error***: Akkor keletkezik, ha nem lehet elérni a szervert.
  - Beépített *event*, rakománya a *hiba*(***error***).
  - Kliens visszalép a *Home* oldalra vagy vendég esetén a *Landing* oldalra, beállítja a hibát, amit megjelenít a kliens.

- ***event_error***: Akkor keletkezik, ha hiányos adatot küld a kliens a szervernek.
  - Rakománya a hibaüzenet(***error***).
  - Kliens visszalép a *Home* oldalra vagy vendég esetén a *Landing* oldalra, beállítja a hibát, amit megjelenít a kliens.



3.10. *ábra.: Kommunikáció menete*

3.3.3 Host szerver

Egyetlen célja a kliens alkalmazás elérhetővé tétele a weben. Mivel a kliens oldja meg az útvonalak kezelését ezért viszonylag egyszerű feladata van. Express keretrendszert használ, és egy egyszerű *middleware*-t, ami a felhasználó által lekért összes útvonalat megváltoztatja az */index.html* oldalra. Erre azért van szükség mivel a kliens oldali útvonal kezelő *SPA*[5]-ra van tervezve, ami nem tudja kezelni, ha egy másik oldalt kérünk le, (pl.: */bármi*) mivel csak egy oldal létezik és a többi tartalmat/'oldalt' csak be injektáljuk az oldalra.

3.4 Kliens oldali implementáció

A kliens oldalon egy **Vue.js** alapú Single Page Application(*SPA*) található. **Vue.js** egy *JavaScript* keretrendszer mely komponensekből állítja össze a weboldalt. Minden komponensnek van saját állapota és tud kommunikálni a többi komponenssel. Annak érdekében, hogy ne legyen a komponensek között kavarodás az egész weboldalra vonatkozó állapotok és azok változtatását megoldó metódusokat egy külön objektum végzi, ezt a **Vuex**-nek a Store objektuma teszi lehetővé. A háttérben a **Vue.js** egy *Virtuális Dokumentum Objektum Modellt* készít, mely lehetővé teszi, hogy egy oldal akkor tölt csak újra, ha arra valóban szükség van.

Annak eldöntéséről, hogy melyik lap töltődjön be, a **Vue Router** kiegészítő gondoskodik. A *HTML5 History API* segítségével a weboldalon való navigálás során új lapletöltés nélkül változtatja az aktuális oldal címét a böngésző címsorrában, biztosítja a keresőbarát URL-eket és a böngészési előzmények megfelelő kezelését a böngészőben.

A játék **p5.js**-t használva van megírva és bele van helyezve egy *vue* komponensbe, de csak minimális *vue* szintakszist használ. A **p5.js** egy gyors és egyszerű lehetőséget ad a *canvas* (és sok más) HTML objektum használatára.

3.4.1 Komponensek

Store

Nem kifejezetten egy komponens, de logikailag ide tartozik mivel mindegyik komponens globális állapotot változtató és globális állapottól függő részét ő kezeli. Benne tároljuk a globális állapotot, ő kezeli a *Firebase* autentikációt, a játék indítást és a hibakövetést. Rendelkezik metódusokkal melyek az állapotot változtatják, ezeket használva a többi komponens tud egymással kommunikálni. Rendelkezik lekérő metódusokkal melyek segítségével a többi komponens tudja olvasni a globális állapotot.

[5] Single Page Application: Egy oldalú alkalmazás, ahol nem több oldal van hanem minden tartalom egy oldalra van dinamikusan be injektálva.

App

Az alap komponens, amely az egész alkalmazás vázát adja és tartalmazza a többi alkomponenst. Része a fejléc és a töltő dialógus, így ezek konzisztensek az összes oldalon.

SingIn:

Kezeli és megjeleníti a bejelentkezési űrlapot. Email cím és jelszó megadási mezői vannak és egy bejelentkezés gomb. A *Store*-t kéri meg hogy a megadott felhasználóval jelentkezzen be és változtassa a globális állapot *user* mezőjét. Ha a *Store* hibát észlel, akkor a globális állapot *error* mezőjébe írja a hibát, amit a bejelentkező felület megjelenít. Drótváza látható a *3.11. ábrán.*

3.11. ábra.: SignIn drótváz

SingUp:

Kezeli és megjeleníti a regisztrációs űrlapot. Email cím, jelszó és jelszó megerősítő mezői vannak és egy regisztráció gomb. A *Store*-t kéri meg hogy a megadott adatokból hozzon létre egy új felhasználót és jelentkezzen be, szóval állítsa be a globális állapot *user* mezőjét. Ha a *Store* hibát észlel akkor a globális állapot *error* mezőjébe írja a hibát, amit a regisztrációs felület megjelenít. Drótváza látható a *3.12. ábrán.*

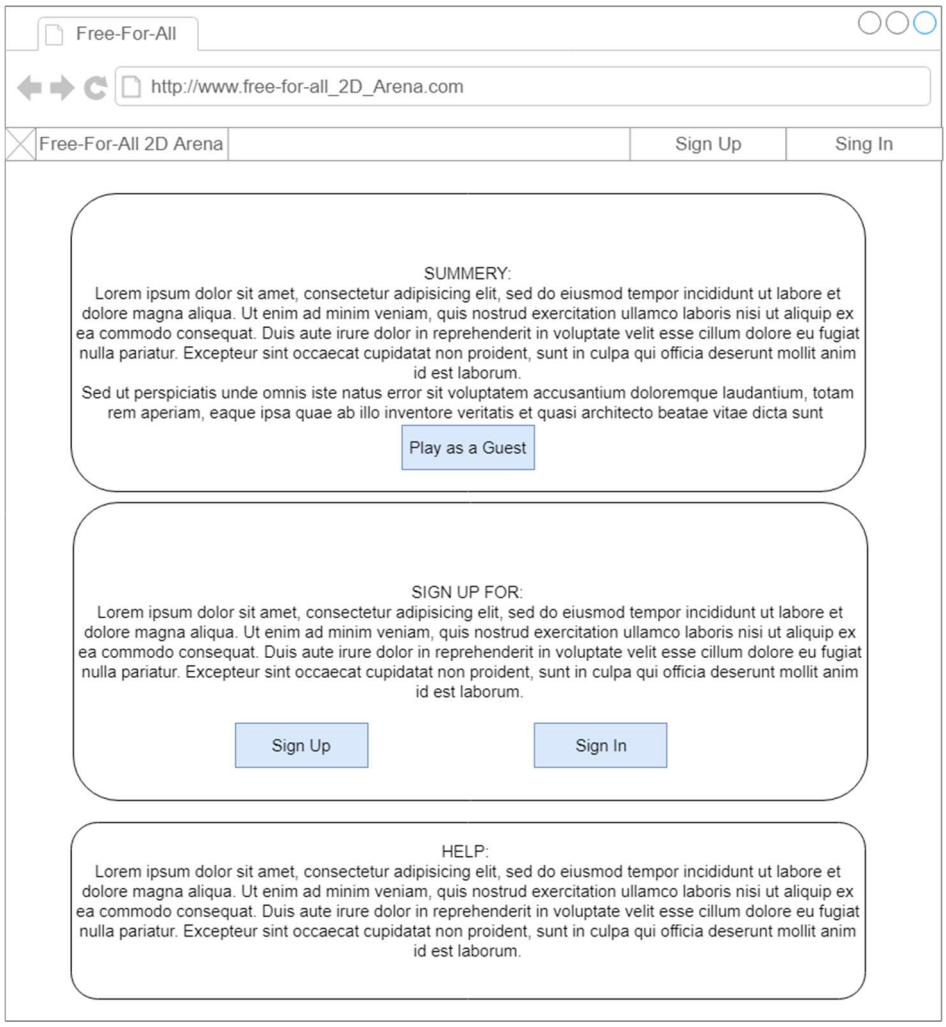
3.12. ábra.: SingUp drótváz

Landing:

A kezdőoldalt kezeli és jeleníti meg, amit egy látogató először lát. Három kártya objektum látható rajta.

- Összefoglaló kártya, ami bemutatja a weboldalt, és lehetőséget ad a játékba lépéshez. A *Play as a Guest* gomb átirányítja a játék oldalra a felhasználót, ez a gomb csak akkor jelenik meg ha a globális állapot *user* mezője *null* értéket vesz fel.
- Regisztrációra buzdító kártya mely ismerteti a regisztráció előnyeit. A *Sign In* gomb átirányít a bejelentkezés oldalra míg a *Sign Up* átirányít a regisztrációs oldalra.
- *Help* kártya mely ismerteti a játék irányításához használt gombokat.

Drótváza látható a *3.13. ábrán.*



3.13. ábra.: Landing drótváz

Home:

A bejelentkezett felhasználó Home oldalát kezeli és jeleníti meg. Tartalmaz egy beviteli mezőt melyben meglehet adni a játékban használt nevet(*displayName*), egy opció mezőt melyben el lehet dönteni, hogy a megjelenése a játéknak milyen legyen(*minimal*), és egy *Start Game* gombot.

A két mező tartalma *bind*-olva van a lokális állapot két változójához (*displayName*, *minimal*) melyek automatikusan kitöltődnek az oldal inicializálásakor a globális állapot alapján. Ha a mezők értéke változik akkor változik a változók értéke is (de csak a lokális változók). A *Start Game* gombot megnyomva szól a *Store*-nak, hogy állítsa a globális állapot *displayName* mezőjét a lokális *displayName* változó értékére, és a *minimal* mezőjét a lokális *minimal* változó értékére, utána meg irányítsa át a felhasználót a *Game* oldalra. Ha valami hiba történt akkor megjeleníti a globális állapot *error* mezője alapján.

Drótváza látható a *3.14. ábrán.*

3.14. ábra.: Home drótváz

Game:

A játék oldalt kezeli és jeleníti meg. A játékhoz tartozó grafikus asset-ek elérési útvonalát lokális állapot változókban tárolja melyek fordításkor a *Webpack* kicseréli a fordítás utáni végleges elérési útvonalra. Visszairányítja a *Home* oldalra és beállítja a globális állapot *error* mezőjét a *Store*-on keresztül, ha a következő esetek valamelyike előfordul:

- Ha a felhasználó be van jelentkezve, de nincsen megadva a *displayName.*
- Ha a szerver tele van.
- Ha a szerver nem elérhető.
- Ha a bejelentkezett felhasználó már jelen van a játékban.

A játékszerverhez való csatlakozáskor a *socket* objektumot eltárolja a globális állapot *socket* változójába is annak céljából, hogy ha a felhasználó hirtelen elhagyja a játékot a *socket* kapcsolat biztosan lezáródjon, mivel, ha nem záródik be a kapcsolat rendesen akkor a játékos beragad a játékba miközben nincs is benne ezáltal vissza sem tud lépni.

A játék pontos működéséről későbbiekben fogok beszélni.

NotFound:

A 404-es hibaoldal, ide irányítja a *router* a felhasználót, ha semelyik aloldalra sem illeszkedik a böngészőben leadott útvonal kérése.

3.5 Játék implementáció

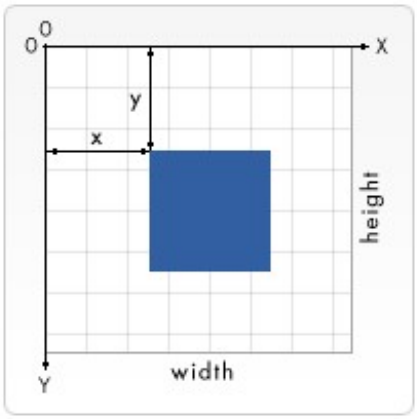
A játékot a **p5.js** nevezetű JavaScript könyvtár segítségével valósítjuk meg. A számunkra fontos részei a grafikában használt adat struktúrák, továbbá azok transzformációi(*Vector*) és a *canvas*-t manipuláló metódusai.

Három fő fázisa van a rajzolásnak a **p5.js**-ben:

1. ***preload***: Még az oldal betöltése előtt egyszer lefutó rész. Itt töltjük be a szükséges textúrákat, és itt hozzuk létre a *socket* kapcsolatot a szerverrel.
2. ***setup***: A rajzolást megelőző inicializációs lépés, ami csak egyszer fut le. Itt hozzuk létre a c*anvas*-t.
3. ***draw***: A rajzolási ciklus magját tartalmazza. Folytonosan hajtja végre a tartalmát, amikor elér a végére kezdi is elölről.

3.5.1 HTML Canvas

A canvas egy HTML elem mely a HTML 5 óta létezik és weboldalon való grafika megjelenítésére van kitalálva. *Raszter grafikán* alapul, így pixelekkel dolgozik és nem vektorokkal, *immediate mode*-ban rajzolja ki a képet, aminek hatására, ha a modell, ami alapján a rajz készült megváltozik, akkor az egész képet újra kell rajzolni, ezért van szükségünk egy ciklusra mely folyamatosan újra és újra rajzolja a képet. A koordináta rendszer, amit használ eltér a matematikában megszokottól, az *origó* (*az x = 0, y = 0 pont*) a bal felső sarokban van, az *Y* tengely értékei meg lefelé haladva növekednek, felfelé haladva csökkenek, az X tengely értékei viszont a szokott módon jobbra növekednek és balra csökkenek. (*lásd.: 3.15.ábra)*



3.15. ábra.: Canvas koordináta rendszer[6]

[6] Forrás: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Drawing_shapes

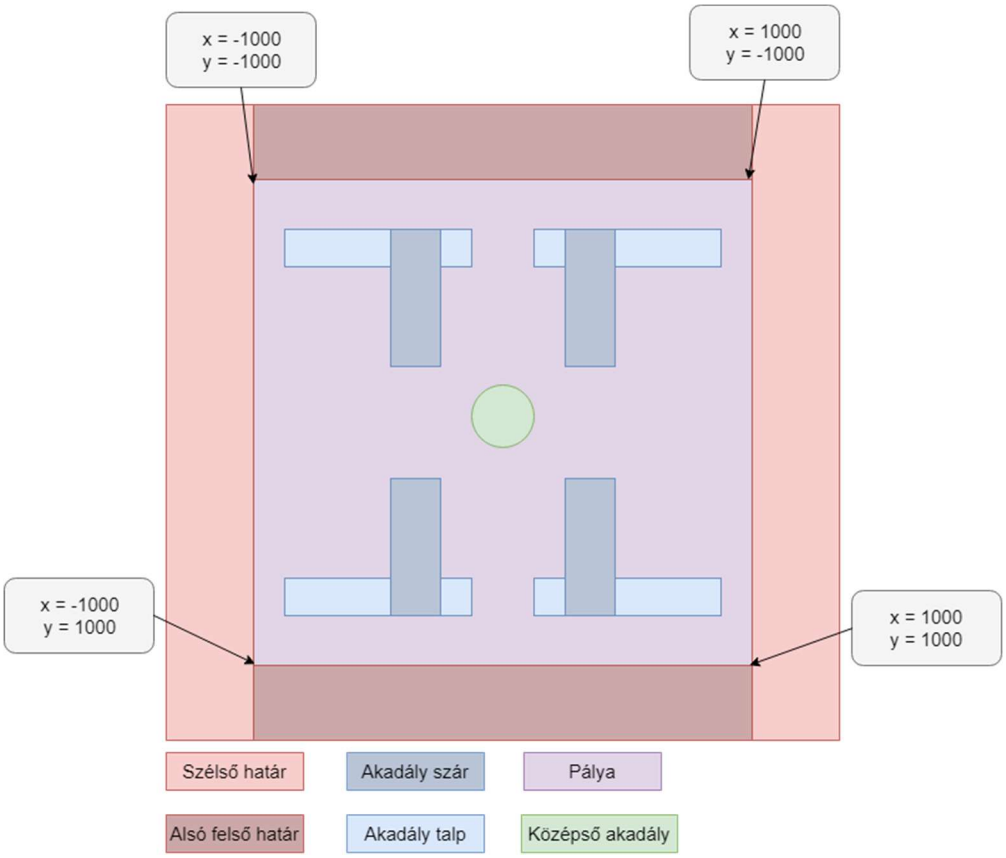
3.5.2 Játéktér felépítése

A játéktérnek két fontos mérete van, maga a bejárható pálya mérete és a játékos látó terének mérete. A *pálya* egy 2000x2000 képpontból álló négyzet míg a játékos látó tere nagyából ennek a negyede (ablak méretenként lehet egy kicsi eltérés). Persze ez egy mai átlagos kijelzőn elég kicsinek számít és majdnem beleférne az egész pálya egy böngésző ablakba, de transzformációk segítségével, rajzoláskor kilehet nagyítani és méretre lehet állítani, hogy csak a pálya egy része látszódjon.

A játéktér elemeinek megrajzolásának a következő a sorrendje:

0. *canvas* létrehozása és háttérszín megadása.
1. A háttérkép kirajzolása.
2. A falak és határok megrajzolása.
3. A játékosok és lövedékek megrajzolása.

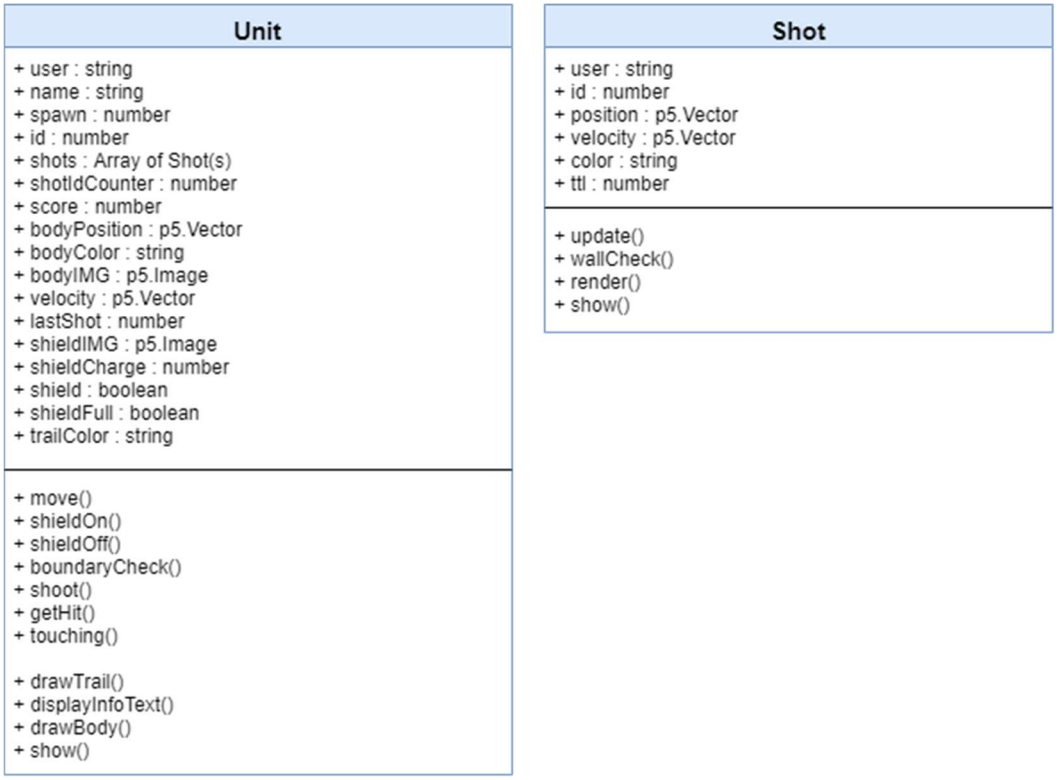
A falak és a határok téglalapok melyeket leírja a négysarkuk X és Y pozíciója a pályán. Egy akadály két falból áll ezáltal egy *T*-t vagy fordított *T*-t formálva, céljuk a játékosok mozgásának korlátozása továbbá fedezékként is funkcionál. A középső elem is egy akadály, de sajátos elbánást kap mivel egy kör. A *T* szárát adó fal hosszabb, hogy egymásba folyjon a két fal ezzel elérve, hogy vizuálisan is szebb, továbbá nem lehet így a két fal közé szorulni. A pálya négy szélén található egy-egy határ, hogy jelezzük a játékosoknak, hogy arra nem tudnak tovább menni. A határok, túl nyúlnak a pályán és a két szélső (bal és jobb) határ hosszabb, ezzel kitöltve a sarkokat. (*lásd.: 3.16. ábra*)



3.16. ábra.: Játéktér felépítése

3.5.3 Osztályok

A játékban dinamikusan megjelenő és dinamikusan változó elemeket osztályokként modelleztem. Egy játékost a *Unit* osztály ír le míg egy lövedéket a *Shot* osztály.



3.17. ábra.: Osztályok

3.5.3.1 Játékos(Unit)

Adattagok:

- ***user***: a játékost irányító felhasználó regisztrációnál megadott neve, ami a mi esetünkben egy email cím. Vendég játékosoknál a szerver adja a nevet(*GuestX*).
- ***name***: a felhasználó által választott név, ami felkerül a ranglétrára. Vendég játékosoknál a szerver által adott *GuestX* név lesz beállítva.
- ***spawn***: egy egész szám 0 – 11 között mely meghatározza a kezdési helyét. Minden játékban lévő játékosnak egyedi ez a szám.
- ***id***: a *socket* kapcsolat azonosítója.
- ***shots***: a játékos összes aktív lövedéke, mindegyik elem egy *Shot* objektum.
- ***shotIdCounter***: a játékos lövedék számlálója, célja az újonnan leadott lövedékeknek egyedi azonosító generálása.
- ***score***: a játékos aktuális pontja, amit a többi játékos legyőzésével lehet szerezni.

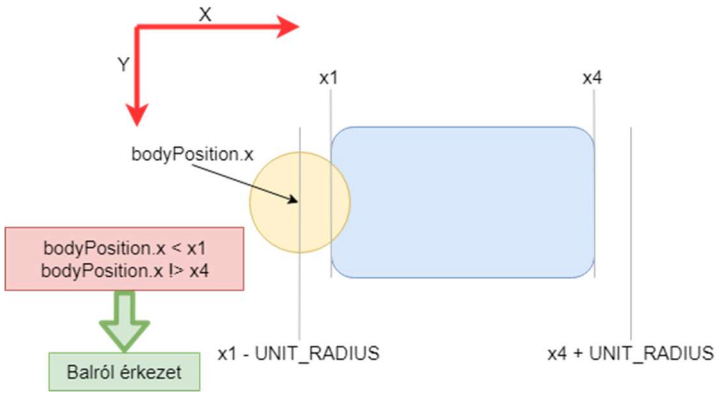
- ***bodyPosition***: a játékos aktuális pozícióját ábrázoló p5.Vector[7].
- ***bodyColor***: a karakter színét reprezentáló karaktersor.
- ***bodyIMG***: a játékoshoz tartozó textúra.
- ***velocity***: a játékos aktuális sebességét ábrázoló p5.Vector[7].
- ***lastShot***: egy szám mely meghatározza mennyi időnek kell még eltelnie, hogy a játékos tudjon lőni.
- ***shieldIMG***: a játékos pajzsát ábrázoló textúra.
- ***shieldCharge***: a játékos pajzsának töltöttségét jelző szám. Amíg van töltése addig tud védekezni a játékos. Használatkor folytonosan veszíti a töltést, de amikor nem használja akkor meg vissza töltődik, gyorsabban fogy, mint ahogy töltődik.
- ***shield***: jelzi (i*gaz vagy hamis*), hogy a játékos védekezik-e vagy sem, ha igen akkor nem lehet lelőni.
- ***shieldFull***: jelzi, hogy fel lett-e teljesen töltve a pajzs a legutóbbi használata óta, mivel csak akkor lehet újra használni a pajzsot, ha teljesen feltöltődött.
- ***trailColor***: a játékos 'hajtóművének' színét reprezentáló karaktersor.

Metódusok:

- ***move()***: figyeli mely mozgáshoz használt billentyű van lenyomva. A mozgás irányának megfelelően hozzáad/kivon egy előre definiált értéket a sebesség (*velocity*) vektor X vagy Y koordinátáihoz/ból. Utána megszorozza az új sebesség vektort a súrlódással ezzel periodikusan lassítva a mozgást melynek következtében, ha nem megyünk egyik irányba sem a játékos lassan megáll. Továbbá itt nézzük meg, hogy le van-e nyomva a védekezés gomb, ami alapján meghívja a védekezéssel foglalkozó metódusokat (*shieldOn, shieldOff*).
- ***shieldOn()***: a játékos *shield* adattagját igazra állítja és a *shieldCharge* adattag értékét csökkenti egy előre definiált értékkel.
- ***shieldOff()***: a játékos *shield* és *shieldFull* adattagját hamisra állítja és a *shieldCharge* adattag értékét növeli egy előre definiált értékkel, ha nincs tele. Ha tele van, akkor a *shieldFull* adattagját igazra állítja.
- ***boundaryCheck()***: ellenőrzi, hogy a játékos nem megy-e bele valamilyen akadályba vagy határba. Három fázisa van:
    - ***Határok ellenőrzése***: megnézzük, hogy az *X* és/vagy *Y* pozíciója meghaladja-e az 1000-t, vagy kisebb mint -1000, ha ez megtörténik akkor a túlmenő koordináta értékét egyenlővé tesszük azon határ értékével amin túl haladt.

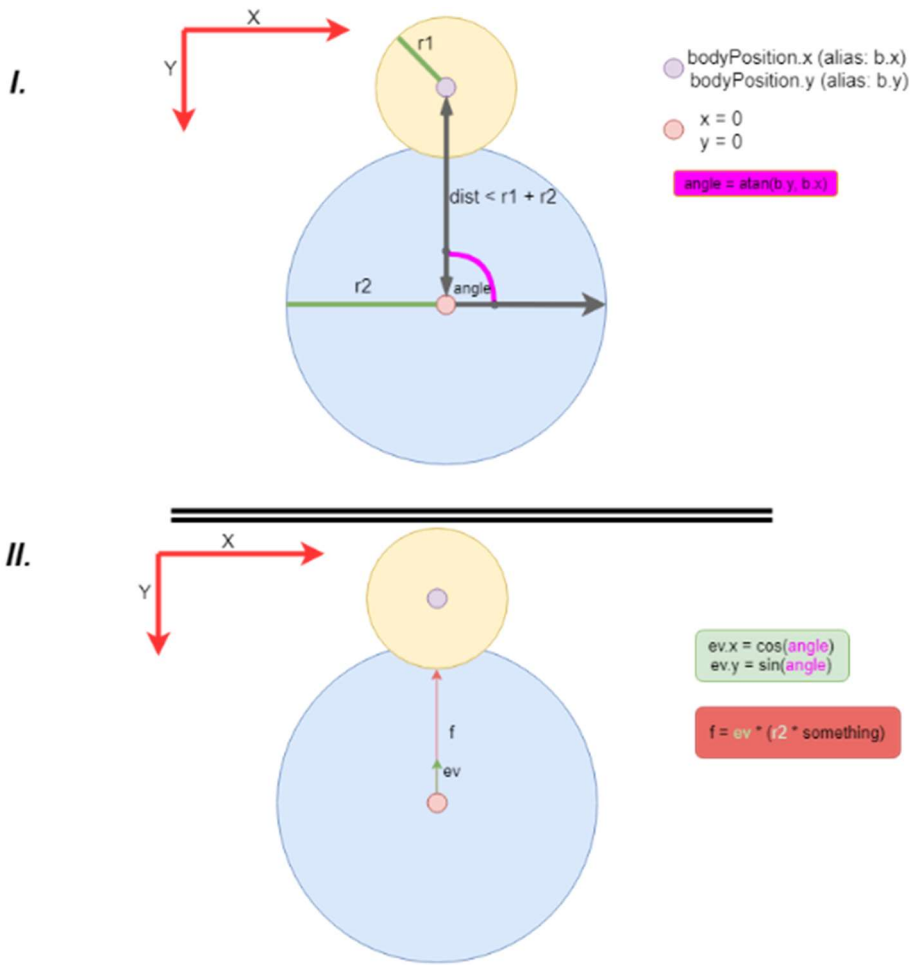
[7] A ***p5.js*** egy vektor implementációja mely beépített metódusai lehetővé teszi transzformációk használatát és két vektorral való számítások végzését.

- ***Akadályok ellenőrzése***: itt már trükközni kell, mivel azt egyszerű megmondani, hogy bene van-e az akadályban egy pont, de játékos nem egy pontból áll, hanem van egy sugara (*UNIT_RADIUS*) is amit figyelembe kell venni. Erre a megoldás, hogy azt figyeljük, hogy az akadály a sugárral meghosszabbított részét érinti-e, ekkor egyszerűen megtudjuk mondani, hogy már hozzáért továbbá, hogy melyik irányból. Erre látható egy példa a *3.18. ábrán*, ahol csak az X koordinátát figyeljük, hogy jobban átlátható legyen, de az Y koordinátánál nem különbözik a logika. Ha ilyen bekövetkezik akkor a megfelelő irányba vissza toljuk úgy, hogy lineárisan interpolálunk az akadály sugárral meghosszabbított határáig így adva egy ruganyos effektet.



3.18. ábra.: Akadály észlelés

- ***Középső elem ellenőrzése***: A középső elem alakjából kifolyólag kell egy külön ellenőrzés. Itt először megnézzük érintkeznek-e, ez akkor van, ha a középső elem sugarának és a játékos sugarának az összege kisebb vagy egyenlő a két elem középpontja közötti távolsággal. Ha ez igaz akkor *arcus-tangens* segítségével megkapjuk a játékos és középpont közötti vektor irányának a szögét, amelyből egy egységvektort képzünk, amit felszorzunk a középső elem sugarát megközelítő értékkel (*lásd.: 3.19.ábra*), hogy megkapjuk az érintkezés pontot és efelé lineárisan interpoláljuk a játékos pozícióját.



3.19. ábra.: Középső elem észlelése

- ***shoot()***: ha a játékos lőhet, létrehoz egy új *Shot* objektumot amit beletesz a *shots* listába és beállítja a *lastShot* adattagot ami szabályozza a lövések közötti idő különbséget.
- ***getHit()***: egy *Shot* objektumot kapva mint paraméter, ellenőrzi, hogy a játékos és a lövedék ütközött-e. Igaz vagy hamissal tér vissza.
- ***touching()***: egy *Unit* objektumot kapva mint paraméter, ellenőrzi, hogy a két játékos ütközött-e. Igaz vagy hamissal tér vissza.
- ***drawTrail()***: a játékos után megjelenő *'sugár'* megjelenítését oldja meg. Ezt úgy oldja meg, hogy a sebesség vektor szerinti előző öt pozícióra rajzol egy-egy, egyre kisebb kört.
- ***displayInfoText()***: kiírja a játékosra a nevét és pontszámát.
- ***drawBody***(): egy igaz-hamis értéket kapva paraméterül, meghívja a *drawTrail()* metódust, megrajzolja a játékost, meghívja a *displayInfoText()* metódust. Igaz paraméternél aktív játékost rajzol, hamis esetben inaktív játékost.
- ***show():*** egy igaz-hamis értéket kapva paraméterül, meghívja a *drawBody()* átadva a paraméterét. Továbbá csökkenti a *lastShot* adattagot.

3.5.3.2 Lövedék(Shot)

Adattagok:

- ***user***: a lövedék tulajdonosának a felhasználó neve.
- ***id***: a lövedék azonosítója.
- ***position***: a lövedék pozícióját reprezentáló p5.Vector objektum.
- ***velocity***: a lövedék sebességét reprezentáló p5.Vector objektum.
- ***color***: a lövedék színe.
- ***ttl***: a lövedék élettartalmát reprezentáló szám.

Metódusok:

- ***update()***: hozzáadja a lövedék aktuális pozíciójához a sebességet.
- ***wallcheck()***: ellenőrzi, hogy belement-e a lövedék egy akadályba, ha igen akkor a *ttl* adattagot 0-ra állítja.
- ***render()***: kirajzolja a lövedéket.
- ***show()***: meghívja a *wallCheck(), update(), render()* metódusokat ilyen sorrendben, és csökkenti a *ttl* adattagot.

3.5.3.3 Paraméterek

Egy játékban nagyon sok kis konstans érték van, amik a játék menetét és a játékteret befolyásolják, mint például az akadályok pozíciói, az érték, ami befolyásolja milyen gyakran lehet lőni stb. A *params.js* fájl tartalmazza ezen értékeket melyek mind csupa nagybetűs konstansok, funkciójukat egyértelműen leíró nevekkel.

Minden konstans nevében az első, illetve második szónak segítségével vannak csoportosítva.

- ***UNIT_***: Azon konstansok melyek a játékosok karakterére és képeségeire vonatkoznak.
  - ***UNIT_SHIELD_***: Azon konstansok melyek a játékos pajzs/védekezés képeségére vonatkoznak.
- ***SHOT_***: Azon konstansok melyek a lövedékre vonatkoznak.
- ***CANVAS_***: Azon konstansok melyek a játéktérre vonatkoznak.
  - ***CANVAS_OBSTACLES_***: Azon konstansok melyek a játéktérbeli akadályokra vonatkoznak.

3.5.4 Game loop

Itt a fő játék ciklust fogom leírni, ami egybe esik a rajzolási ciklussal. Fontos tudnivaló, hogy itt a szervertől érkező kommunikációt nem fogom részletezni mivel arról már korábban beszéltem, továbbá itt a fókuszban az áll, hogy a kliens oldalon mivel foglalkozik a játék. Továbbá minden rajzolási ciklus elején vannak transzformációk amivel később foglalkozunk.

1. A játékos modelljének frissítése az input szerint (meghívja a játékos *move()* metódusát). Ha történt változás akkor küldi a szervernek.
2. Végig megyünk az összes ellenségen és
    a) Ha a játékos aktív:
        i. Ellenőrizzük a játékos összes lövedékét, hogy eltalálta-e az ellenfelet egy lövedékkel, ami még él (szóval *ttl* > 0) úgy, hogy az ellenfélen nem volt pajzs. Ha igen akkor szól a szervernek, hogy valakit eltalált. Ha eltalálta, de volt rajta pajzs akkor a lövedék *ttl* adattagját 1-re állítja. Egyébként meg megy tovább.
        ii. Ellenőrizzük, hogy az ellenfél hozzáért-e a játékoshoz, ha igen akkor a sebességének inverzévének a többszörösével visszapattan a játékos.
    b) Mindig:
        i. Végig megyünk az ellenség összes lövedékén és ellenőrizzük, hogy élnek-e még (*ttl* > 0), ha igen akkor megjelenítjük a lövedékeket, ha nem akkor kivesszük a listából, ezáltal megsemmisítve.
        ii. Megjelenítjük az ellenséget.
3. Végig megyünk a játékos összes lövedékén és ellenőrizzük, hogy élnek-e még (*ttl* > 0), ha igen akkor megjelenítjük a lövedékeket, ha nem akkor kivesszük a listából ezáltal megsemmisítve.
4. Megjelenítjük a játékost.

Összegezve, a kliens csak a saját játékosával és lövedékeivel történő eseményekkel foglalkozik, az ellenfeleket és lövedékeiket csak megjeleníti és vagy megsemmisíti, nem figyeli, hogy két ellenfél ütközik vagy hogy az egyik lelőtte a másikat, ezek majd a szerver által küldött információból kiderülnek.

3.5.5 Transzformációk

Ahhoz, hogy játékos a *canvas* közepén helyezkedjen el és a háttér mozogjon alatta, kellenek transzformációk, mi a *p5.js* által implementáltakat használjuk. Ha végrehajtunk egy transzformációt vagy valami grafikai beállítást változatunk *p5*-ben akkor a változtatást követő összes rajzra vonatkozni fog.

Ez rendkívül limitáló, de a *p5* ad lehetőséget ezen beállítások tárolására és visszaállítására a *push()* és *pop()* metódusokkal. A *push()*-al eltároljuk a beállításainkat egy verembe és újakat állíthatunk be utána, amikor visszaakarjuk állítani a beállításainkat akkor a *pop()*-al kiszedjük a verem tartalmát és felülírjuk az aktuális beállításokat az eltároltakkal. Fontos, hogy egy *push()*-t MINDIG követnie kell egy *pop()*-nak. Mi minden elkülönülő elem rajzolásnál használunk *push()* és *pop()* függvényeket.

Általunk használt transzformációk:

- ***translate(x, y):*** A *translate* transzformáció eltolja az objektumokat a megadott értékekkel. Többszöri használat esetén a transzformáció hatásai összeadódnak.
- ***scale(s):*** Megnöveli vagy csökkenti az objektumok méretét a megadott érték szerint. Többszöri használat esetén a transzformáció hatásai összeszorzódnak.

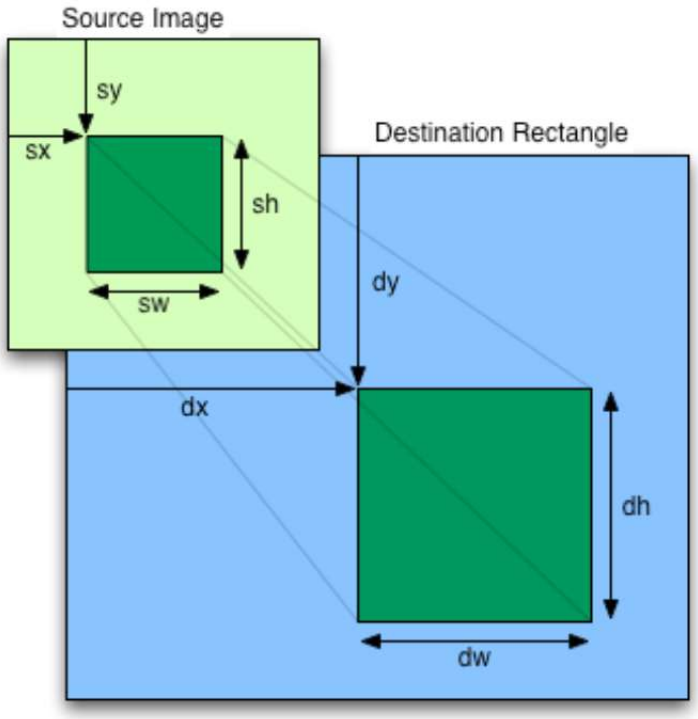
A rajzolási ciklus elején lévő három transzformáció felelős azért, hogy a rész elején említett effekt létrejöjjön.

1. ***translate(width/2, height/2)***: eltoljuk az egész *canvas* tartalmát, úgy hogy az *origó* legyen a *canvas* közepén.
2. ***scale(width * height / (width + height) / 450)***: A *canvas* teljes tartalmát növeljük/csökkentjük az ablak méretétől függően.
3. ***translate(-bodyPosition.x, -bodyPosition.y)***: Ismét eltoljuk az egész *canvas* tartalmát, hogy már a játékos legyen a *canvas* közepén.

3.5.6 Textúrák

A *p5.js*-ben implementált *p5.Image*-et használjuk a kép tárolására, egy kép eltárolásához a *loadImage()* függvényt kell használni, aminek a kép elérési útvonalát kell megadni mint paraméter. Az *image()* függvénnyel jelenítjük meg a képet aminek a kép X és Y koordinátáin kívül szélességet és magasságot is meg lehet adni, de az X és Y koordináta amit megadunk az az a pont ahol a kép jobb felső sarka fog elhelyezkedni. (*lásd.: 3.20. ábra*)

Emiatt amikor a körünkre szeretnénk egy körképet tenni úgy, hogy tudjuk a kör közép pontját és sugarát, akkor a mi képünk X koordinátája(*dx*) egyenlő a kör középpontjának x koordinátája mínusz a sugár, hasonlóan az Y koordináta(*dy*). A szélesség(*dw*) és a magasság(*dh*) pedig a sugár kétszerese.



3.20. ábra.: Kép koordináták[8]

A textúrákat nem én készítettem, hanem egy ingyenes kollekciót használtam, ami a https://kenney.nl/assets/space-shooter-redux oldalon érhető el.

[8] Forrás: https://p5js.org/reference/#/p5/image

3.6 Tesztelés

A program részletes tesztelése rendkívül fontos része a fejlesztésnek. A játék szerver vizsgálását *fehér doboz teszteléssel* oldottam meg, amiben segítgetett a *socket.io tester*[9] alkalmazás, amivel fel lehet jelentkezni eseményekre és küldeni lehet eseményeket JSON tartalommal. A frontend és játék tesztelését *tesztelési napló* vezetésével oldottam meg. A napló elkészítéséhez használt eszközök: *Chrome Developer Tools*[10], *Vue.js devtools*[11], *Firebase Console*[12]

3.6.1 Tesztelési napló

Esetek	Reakció
Regisztrációs űrlap	
Regisztrációs űrlap helyes kitöltése és elküldése.	Új felhasználó létrejön, *Firebase console-ban* is látható, bejelentkezett és átirányítódott a *Home* oldalra. Bejelentkezett felhasználó látható a böngésző *IndexedDB-jében*, továbbá a *Store user* adattagjába is bekerült.
Regisztrációs űrlap üresen hagyása és elküldése.	A felület *form*-ja jelezte, hogy üres az email mező és nem küldte el a szervernek a kérést.
Regisztrációs űrlap kitöltése nem helyes email formátummal és elküldése.	A felület *form*-ja jelezte, hogy az email helytelen formátumú és nem küldte el a szervernek a kérést.
Regisztrációs űrlap kitöltése különböző jelszóval és elküldése.	A felület *form*-ja jelzezte, hogy a két jelszó nem egyezik meg és nem küldte el a szervernek a kérést.
Regisztrációs űrlap kitöltése nem elég erős jelszóval (kevesebb mint 6 karakter).	A felület elküldte a *Firebase*-nek ami visszaküldött egy 400-as hibát, amit a felület megjelenített. A *Store error* adattagjába bekerült a hiba üzenet.

[9] Socket.io tester alkalmazás: http://appsaloon.github.io/socket.io-tester/
[10] A Chrome böngészőben az F12 megnyomásval lehet felhozni.
[11] Vue.js devtools böngésző kiegészítő: https://bit.ly/2IyfBBB
[12] A Firebase webes felülete: https://console.firebase.google.com

Bejelentkezés űrlap	
Bejelentkezés űrlap helyes kitöltése és elküldése.	A felület elküldte a Firebase-nek ami visszaküldött egy 200-as üzenetet. Átirányítódott a *Home* oldalra. A felhasználó látható a böngésző *IndexedDB-jében*, továbbá a *Store user* adattagjába is bekerült.
Bejelentkezés űrlap üresen hagyása és elküldése.	A felület *form*-ja jelezte, hogy üres az email mező és nem küldte el a szervernek a kérést.
Bejelentkezés űrlap kitöltése nem helyes email formátummal és elküldése.	A felület *form*-ja jelezte, hogy üres az email mező és nem küldte el a szervernek a kérést.
Bejelentkezés űrlap kitöltése nem helyes email címmel vagy jelszóval és elküldése.	A felület elküldte a *Firebase*-nek ami visszaküldött egy 400-as hibát, amit a felület megjelenített. A *Store error* adattagjába bekerült a hiba üzenet.
Home oldal /Játékba szállás	
Home űrlap helyes kitöltése, minimal opció hamis. Játék indítása.	A megadott név a *Store displayName* adattagjába került, a *Store minimal* adattagja hamis lett. Átirányítódott a Game oldalra, létre jött a *socket* kapcsolat és a *Store socket* adattagjába eltárolódott a *socket* objektum. Látható lett a játék textúrákkal és irányítani is lehetett.
Home űrlap helyes kitöltése, minimal opció igaz. Játék indítása.	A megadott név a *Store displayName* adattagjába került, a *Store minimal* adattagja hamis lett. Átirányítódott a Game oldalra, létre jött a *socket* kapcsolat és a *Store socket* adattagjába eltárolódott a *socket* objektum. Látható lett a játék textúrák nélkül és irányítani is lehetett.

Home űrlap üresen hagyása, minimal opció igaz vagy hamis. Játék indítása.	A felület jelezte, hogy ki kell tölteni a *Display Name* mezőt és nem lépett be a játékba.
Bejelentkezett felhasználó játék közben oldalt frissít.	A *Store* adattagjai alapértelmezett értékeket vettek fel. Megszűnt a *socket* kapcsolat a *Store* user és displayName adattagjai visszaállítódtak játék előtti állapotba. Megszűnt a karaktere a játékban. Kiolvasta az IndexedDB-ből a felhasználót és a *Store user* adattagjába rakta ezzel automatikusan bejelentkezett. Visszakerült a Home oldalra a felhasználó.
Vendég felhasználó játék közben oldalt frissít.	A *Store* adattagjai alapértelmezett értékeket vettek fel. Megszűnt a *socket* kapcsolat a *Store* user és displayName adattagjai visszaállítódtak játék előtti állapotba. Megszűnt a karaktere a játékban. Újra létrejött a *socket* kapcsolat, bekerült a *Store socket* adattagjába a *socket* objektum. A *Store user* és *name* adattagjaiba bekerült a játékszerver által generált név. Visszakerült a játékba a vendég, mint egy új játékos.
Felhasználó játékközben oldalt vált	Megszűnt a *socket* kapcsolat, a *Store* user és displayName adattagjai visszaállítódtak játék előtti állapotba. Megszűnt a karaktere a játékban. Betöltődött az oldal.
Már játékban lévő bejelentkezett felhasználó egy másik ablakból ismét játékba szált.	A Store minimal és displayName adattagjai a megadott értékeket vette fel. Létrejött a socket kapcsolat és a *Store socket* adattagjába eltárolódott a *socket* objektum. A játékszerver észlelte a hibát. A Store error adattagja be lett állítva a hiba. Át lett irányítva a Home oldalra. A felület megjelenítette a hibát.

Kezdőlap	
Vendég felhasználó a ’Play as a Guest’ gombra kattint.	Létrejött a *socket* kapcsolat. Bekerült a *Store socket* adattagjába a *socket* objektum. A *Store user* és *name* adattagjaiba bekerült a játékszerver által generált név. Bejutott a játékba.
Vendég felhasználó a ’Sign In’ gombra kattint.	Átirányítódott a bejelentkezés oldalra.
Vendég felhasználó a ’Sign Up’ gombra kattint.	Átirányítódott a regisztrációs oldalra.
Router	
Létező útvonal megadása a böngészőben. Útvonalhoz nem kell autentikált felhasználó.	A *router* átirányította a megfelelő oldalra.
Nem létező útvonal megadása a böngészőben.	A router átirányította a ’Page not found’ hibaüzenetet tartalmazó *NotFound* oldalra.
Létező útvonal megadása a böngészőben vendég felhasználóval. Útvonalhoz kell autentikált felhasználó.	A *router* átirányította a bejelentkezés oldalra.
Vendég felhasználó a böngészőbe /game oldalra konkrétan rákeres.	Létrejött a *socket* kapcsolat. Bekerült a *Store socket* adattagjába a *socket* objektum. A *Store user* és *name* adattagjaiba bekerült a játékszerver által generált név. Bejutott a játékba.
Bejelentkezett felhasználó a böngészőbe /game oldalra konkrétan rákeres.	A router átirányította a Home oldalra. Beállította a *Store error* adattagját. A felület megjelenítette a hiba üzenetet.
Játék	
Játékba lépett a felhasználó	Betöltődtek a képek, létrejött a *socket* kapcsolat. Elkezdődött a *kommunkáció* a *socket*-en.
Aktív játékos WASD gombokat váltakozva nyomja.	A játékba a karaktere mozgott az adott irányba. A mozgást a több játékos is látta.
Aktív játékos megnyomja az ENTER billentyűt.	Nem történt semmi.

Aktív játékos egyik egérgombot megnyomja.	A játékban keletkezett egy lövedék a játékos pozícióján és elindult a kurzor irányába. Egy kis idő után a lövedék megsemmisült. A többi játékos is látta a lövedéket és a többi játékosnál is megsemmisült a lövedék.
Aktív játékos nyomva tartja a SPACE billentyűt.	Kirajzolódott a pajzs a játékos körül, egy rövid idő után eltűnt, aztán egy hosszabb idő után megint megjelent. A többi játékos is látta a pajzsot.
Inaktív játékos megnyomja a SPACE vagy WASD billentyűt, vagy valamelyik egérgombot.	Nem történt semmi.
Inaktív játékos megnyomja az ENTER billentyűt.	Visszalépett a játékos a játékba. A többi játékosnál ismét látható volt. Ha sok böngésző ablak van megnyitva akkor előfordulhat, hogy valamelyik megakadályozza az ENTER esemény észlelését és ilyenkor nem történik semmi.
Lövedék akadályt ér.	A lövedék megsemmisült amint elérte az akadályt. A többi játékosnál is megsemmisült.
Lövedék pajzsot ér.	A lövedék megsemmisült amint elérte a pajzsot és a játékost épen hagyta. A többi játékosnál is megsemmisült a lövedék.
Lövedék játékost ér.	A játékos inaktív állapotba került a pálya közepén. A lelőtt játékos megszűnt a többi játékos játékában. A lövést leadó játékos pontszáma nőt.
Játékos játékossal ütközik.	A játékosok a sebességük szerint ellenkező irányba pattantak. Ha egy akadály közelében történik akkor megeshet, hogy az egyik játékos beleesik az akadályba, de ki tud jönni.
Játékos akadállyal, középső elemmel vagy határral ütközik.	Nem tudott belemenni az objektumokba, mindig enyhén visszapattant, kivéve a határnál ott csak nem tudott tovább menni.

3.6.2 Fehér doboz tesztelés

<table>
<tr><th rowspan="2">Esetek</th><th>Küldött esemény</th><th>Válasz</th></tr>
<tr><th colspan="2">Elvárt viselkedés</th></tr>
<tr><td rowspan="2">Bejelentkezett játékos ellenőrzése</td><td>Esemény: userCheck
Rakomány: test@test.com</td><td>Esemény: spawn
Rakomány: {
id: ’…’
forGuest: ’test@test.test’
spawnPoint: 1
}</td></tr>
<tr><td colspan="2">A szerver visszaküld egy spawn eseményt, ami tartalmaz egy objektumot, ami tartalmaz, egy id, forGuest, és spawnPoint adattagot. Ebből a forGuest-et nem vesszük figyelembe.</td></tr>
<tr><td rowspan="2">Vendégjátékos ellenőrzése</td><td>Esemény: userCheck
Rakomány: null</td><td>Esemény: spawn
Rakomány: {
id: ’…’
forGuest: GuestX’
spawnPoint: 9
}</td></tr>
<tr><td colspan="2">A szerver visszaküld egy spawn eseményt, ami tartalmaz egy objektumot, ami tartalmaz, egy id, forGuest, és spawnPoint adattagot. A forGuest-t tartalmazza a vendég nevét.</td></tr>
<tr><td rowspan="2">Bejelentkezett játékos ellenőrzése, a játékos már egyszer a játékban van.</td><td>Esemény: userCheck
Rakomány: test@test.com</td><td>Esemény: goHome
Rakomány: ’Already in game’</td></tr>
<tr><td colspan="2">A szerver jelzi a hibát egy goHome eseménnyel, mely tartalmazza a hiba üzenetet.</td></tr>
<tr><td rowspan="2">Játékba lépés, hiányos adattal.</td><td>Esemény: start
Rakomány: null</td><td>Esemény: event_error
Rakomány: Error:
start event =>
Invalid payload:
undefined
From socket
_M9OkT8c3vGIiO0jAAAA"</td></tr>
<tr><td colspan="2">A szerver észleli az adatok hiányát és jelzi az event_error segítségével, hogy valami baj történt a kliens oldalon játék kezdés előtt.</td></tr>
</table>

<table>
<tr><td rowspan="2">Játékba lépés, helyes adattal.</td><td>Esemény: start
Rakomány: {
user: "test@test@test",
id: "RjWfG0p0R_OnfyfvAAAA",
name: "test",
spawn: "9",
positionX: 0,
positionY: 0 }</td><td>létrejött szerveroldalon a játékos</td></tr>
<tr><td colspan="2">A szerver a helyes adatokból létrehoz egy új játékost és eltárolja.</td></tr>
<tr><td rowspan="2">Játék állapot frissítése, üres adattal.</td><td>Esemény: update
Rakomány: null</td><td>Esemény: event_error
Rakomány: Error:
update event =>
Invalid payload:
undefined
From socket
_M9OkT8c3vGIiO0jAAAA</td></tr>
<tr><td colspan="2">A szerver észleli az adatok hiányát és jelzi az event_error segítségével, hogy valami baj történt a kliens oldalon állapot frissítés közben.</td></tr>
<tr><td rowspan="2">Játék állapot frissítése, helyes adattal.</td><td>Esemény: update
Rakomány: {
positionX: 1,
positionY: 1,
velocityX: 1,
velocityY: 1,
shield: false,
score: 1
shots: [] }</td><td>Esemény: heartbeat
Rakomány: a játék aktuális állapota</td></tr>
<tr><td colspan="2">A szerver az adattokkal frissíti a játék állapotot és kiküldi az összes játékosnak.</td></tr>
<tr><td rowspan="2">Játék állapot frissítése, helyes adattal nem aktív játékos álltal.</td><td>Esemény: update
Rakomány: { … }</td><td>nem történik semmi</td></tr>
<tr><td colspan="2">Ha egy nem aktív játékos frissíteni próbálja a globális játék állapotot akkor nem történik semmi. Erre azért van szükség mert, ha egy játékos inaktív lesz akkor lehet, hogy miközben inaktív lett küldött egy frissítést emiatt nem dobhatunk hibát.</td></tr>
</table>

<table>
<tr><td rowspan="2">Játékos lelövése</td><td>Esemény: enemyHit
Rakomány: { targetId: … }</td><td>1.Esemény: gotHit
Rakomány: { shooterId: … }
2. Esemény: playerLeft
Rakomány: { playerId: … }</td></tr>
<tr><td colspan="2">Egy játékos találatakor szól a kliens a szervernek, ami szól az eltalált játékosnak, hogy eltalálták, utána pedig szól az összes játékosnak, hogy egy játékost le kell venni a pályáról.</td></tr>
<tr><td rowspan="2">Játékos lelövése, játékos megadása nélkül</td><td>Esemény: enemyHit
Rakomány: null</td><td>Esemény: event_error
Rakomány: Error:
enemyHit event =>
Invalid payload:
undefined
From socket:
FGYFsDO3voewnX5LAAAA</td></tr>
<tr><td colspan="2">A szerver észleli az adatok hiányát és jelzi az event_error segítségével, hogy valami baj történt a kliens oldalon a játékos lelövése közben.</td></tr>
<tr><td rowspan="2">Inaktív játékos aktiválása</td><td>Esemény: respawn
Rakomány: nincs</td><td>játékos aktiválódott</td></tr>
<tr><td colspan="2">A szerver aktiválja az inaktív játékost</td></tr>
<tr><td rowspan="2">Aktív játékos aktiválása</td><td>Esemény: respawn
Rakomány: nincs</td><td>Esemény: event_error
Rakomány: Error:
respawn event =>
No inactive player corresponding to socket:
FGYFsDO3voewnX5LAAAA</td></tr>
<tr><td colspan="2">A szerver észleli, hogy nincs a socket kapcsolathoz rendelt inaktív játékos és jelzi az event_error segítségével, hogy valami baj történt a kliens oldalon a játékos aktiválása közben.</td></tr>
</table>

4 Összefoglalás

Az elkészült játékot ismerősökkel és családtagokkal kipróbálva örömmel tapasztaltam, hogy egyszerű irányíthatósága miatt korosztálytól majdnem, hogy függetlenül élvezték a résztvevők a közös játékot. A játékba gyorsan lehet ki-be lépni, probléma nélkül. A weboldal felülete a modern és egyszerűen kezelhető.

Tehát a célomat úgy érzem elértem, miszerint egy szórakoztató, elkötelezettségmentes játékot sikerült létrehoznom. Továbbá tapasztalatot szereztem a játékszerver létrehozásánál a bevezetésben ismertetett problémával („akadás”). Már értem, hogy milyen nagy feladat a játék akadásmentes közvetítése még egy egyszerű játéknál is. Sok időt töltöttem a játék készítése során az optimalizálással. Nehéz megállapítani, hogy mely adatokat fontos helyileg kezelni és melyeket globálisan. Mindent nem lehet a szerveren kezelni, mivel anélkül is könnyen túl lehet terhelni. A kliens sem foglalkozhat a játékban történő összes eseménnyel, mivel itt is fennáll a túlterhelés veszélye továbbá lehetséges a kliensek összeakadása is. Meg kell találni az egyensúlyt a szerver és a kliens terhelése között, amint tapasztaltam ez a feladat komplikált és néha elrettentő, de én szívesen foglalkoznék vele a jövőben is, amikor már több tapasztalatot szerzetem a témában.

4.1 Tovább fejlesztési lehetőségek

Tesztelés során feljött jó pár ötlet, amivel ki lehetne egészíteni a játékot. Ezek közül felsorolok most párat, amelyek leginkább megragadták a fantáziámat:

- Többféle pálya: kisebb, nagyobb, más akadályokkal stb.
- Többféle hajó a játékosoknak: Játék kezdése előtt ki lehetne választani milyen kinézetű hajóban szeretnénk játszani.
    - Továbbá, ha a különböző hajók különböző tulajdonságokkal rendelkeznének
        - Gyorsabb, lassabb, kevésbé sérülékeny, stb.
- A pályán elszórt fegyverek vagy használati tárgyak, amit a játékosok felvehetnének és egy rövid ideig valamilyen szempontból erőssebek lennének.
    - Gyorsabb lövések, szuper pajzs, stb.
- A webfelületen egy a játékos ’*karrierje*’ során játszott játékok és a bennük elért eredmények alapján statisztikák megjelenítése.
- Több, szebb animációt tenni a játékba.
- Hang effekteket tenni a játékba.
- Érintő kijelző támogatása.

5 Irodalomjegyzék

[1] Node.js
URL: https://nodejs.org/docs/latest-v7.x/api/
(elérve: 2018.05.08)

[2] p5.js
URL: https://p5js.org/reference/
(elérve: 2018.05.08)

[3] Vue.js
URL: https://vuejs.org/
(elérve: 2018.05.08)

[4] Vuex
URL: https://vuex.vuejs.org/en/state.html
(elérve: 2018.05.08)

[5] Vuetifyjs
URL: https://vuetifyjs.com/en
(elérve: 2018.05.08)

[6] Socket.io
URL: https://socket.io/
(elérve: 2018.05.08)

[7] Firebase
URL: https://firebase.google.com/
(elérve: 2018.05.08)

[8] Webpack
URL: https://webpack.js.org/
(elérve: 2018.05.08)

[9] JavaScript referencia
URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
(elérve: 2018.05.08)

[10] Express
URL: https://expressjs.com/
(elérve: 2018.05.08)

[11] Weboldalon használt képek:

URL:

- https://dastrontm.deviantart.com/art/Simple-House-Vector-With-UFO-628298860)
- https://wallup.net/minimalistic-ufo-takes-cow/

(elérve: 2018.05.08)