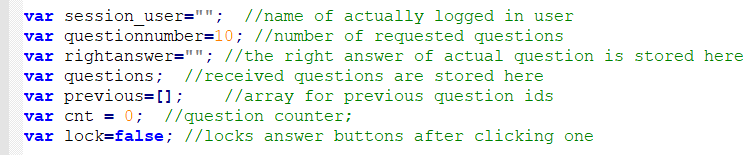
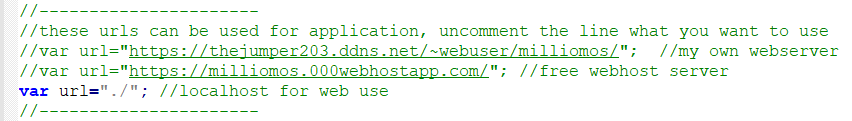
JavaScript

A játék JavaScript részénél arra törekedtünk, hogy minél jobban levegyük a terhet a backend-ről és inkább a frontend-en fussanak az algoritmusok. Ennek érdekében a játék alatt az adatbázisból csak egyszer kérünk ki adatot és azt a frontend-en tároljuk.



Webszerverek elérhetősége:



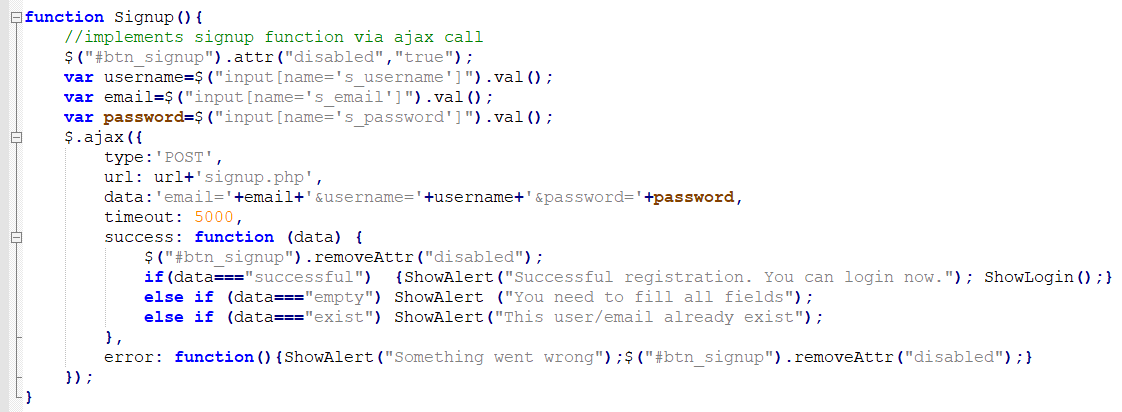
Kommentek törlésével könnyen állítható, hogy melyik webszerverre szeretnénk csatlakozni. Illetve localhost indítás is rendelkezésünkre áll.

Kommunikáció a backend-del

A dinamikus kommunikálás megvalósítását AJAX végezte. JavaScriptről php backend-re tudunk adatot küldeni, ami pedig az adatbázissal való kommunikációt valósítja meg. Ennek segítségével nem kell új html lapot küldeni, így nincs frissítés sem.

Regisztráció

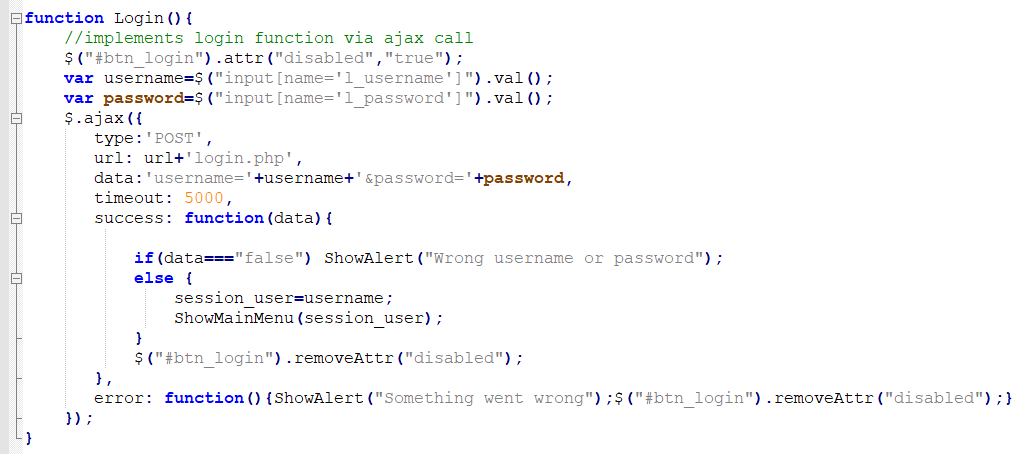
Regisztráció során csak pár új adatot kell az adatbázisba küldeni.



Változók értékeit adjuk át az ajax-nak. Típusa: POST. URL a célt jelöli, ez jelen esetben a signup.php fájl lesz. Az adatokat stringként, összefűzve küldjük el. A függvény futása végén alert-tel jelezzük a küldés sikerességét, sikertelenségét.

Bejelentkezés

A bejelentkezés hasonló a regisztrációhoz az ajax szempontjából. Itt is adatot küldünk a backend részére. Típusa tehát POST. AZ URL a login.php fájl. Az adatokat ugyan úgy stringként összefűzve küldjük el.



GetScores(), GetQuestions() függvények

Ezekkel a függvényekkel kérjük ki az adatbázisból a pontokat, kérdéseket. Ugyan úgy ajax-szal működik a kérés, majd az adatot változókban tároljuk mint az előbbiekben. Továbbá készült még ezekhez függvény ami a változókból a html elemekre tölti fel ezeket az adatokat.

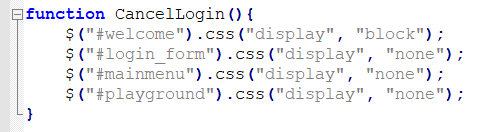
Offline mód

Egy helyi JSON fájlban is tároljuk a kérdések táblát. Offline módban nem az adatbázishoz intézünk kérést hanem erre a fájlra hivatkozunk és innen olvassuk ki a kéréseket.

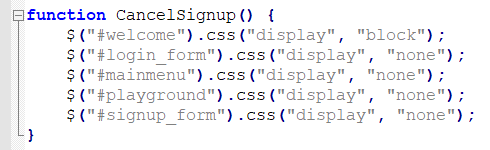
UI

A játék élvezhetősége érdekében egy html oldal fut végig és ezen történnek a változások a különböző függvények segítségével. Néhány példa:

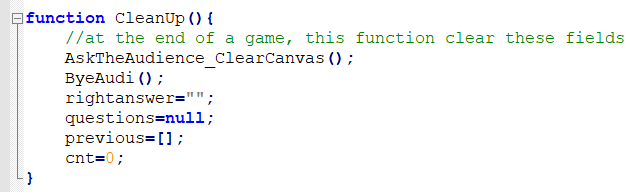
CancelLogin() metódus eltűnteti a a login formot és a kezdőképernyő állapotát hozza vissza.



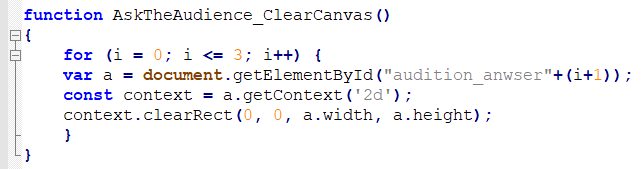
CancelSignup() metódus az előbb említett módszerrel a regisztrációs formot tűnteti el.



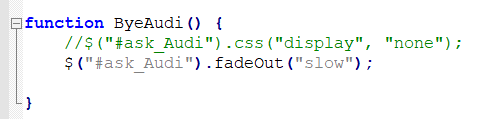
CleanUp() metódus a játék végén hívódik. Ennek az a feladata, hogy visszaállítson különböző kezdőértékeket, pl.: a jó válasz tárolását, a kérdéseket tároló tömböt, előző kérdés ID-ját tároló változót, kérdések számlálóját.



Ezen kívül itt hívódik meg az AskTheAudience\_ClearCanvas() nevű metódus, ami a közönség segítség kezdőértékeit állítja vissza.

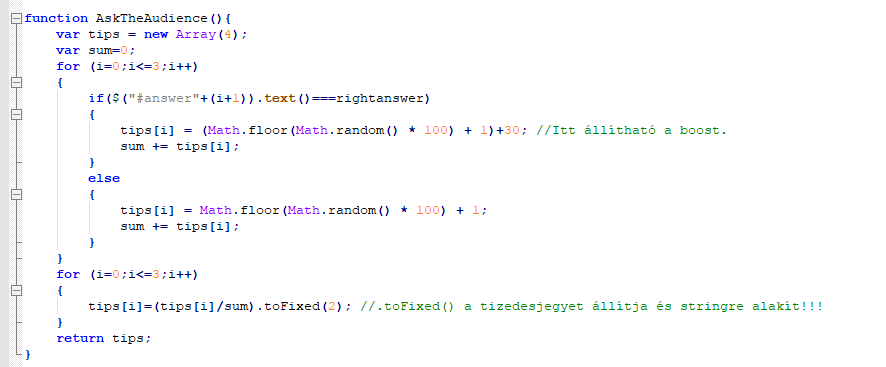


ByAudi() metódus eltűnteti a közönség segítség megjelenítését.

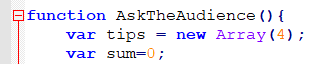
Segítségek

Közönség szavazás:

A cél az volt, hogy négy darab véletlenszerű törtet generáljunk, amiknek az összege egy egész számot alkot, hogy a közönség szavazását tudjuk szimulálni.



Ehhez létrehoztunk egy 4 elemű tömböt amiben a szavazat számokat különbözőképpen tároljuk. Kellett még egy a számok összegét tároló változó is.



Egy cikluson belül hozzuk létre a random számokat, de előtte minden lépésnél meg kell vizsgálnunk, hogy jó vagy rossz válasznak. Ehhez egy elágazásban kell összehasonlítanunk a gombunk által tárolt szöveget és a *rightanswer* globális változóban tárolt helyes választ. A html gomb nevére kell hivatkoznunk (amik *#answer1*, *#answer2*, *#answer3*, *#answer4* névre hallgatnak a html-ben), illetve annak .text() metódusára, úgy, hogy hozzá fűzzük az *i* változót is. Erre egyrészt azért van szükség, hogy a pontos nevét kapjuk meg a html gombnak másrészt pedig, hogy végig tudjunk lépdelni a többi gombon az *i* változtatásával. Tehát az elágazáson belül összehasonlítjuk a két stringet.

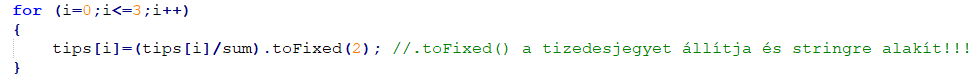


Ha egyezik, akkor a jó válasz gombjánál jár a ciklus. Tehát létrehozhatunk egy random számot a Math.floor(Math.random())-mal 1 és 100 között és hozzáadhatunk egy bizonyos értéket amivel növelhetjük a közönség „tudását”.



Itt +30-cal növeltük a random számot, ez 30% növelésnek fog megfelelni. Ezután a *sum* változóhoz adjuk ezt a számot, hogy a ciklus végén meglegyen a számok összege.

Az else ágon csak simán random számot generálunk, nem adunk hozzá semmit, hiszen a rossz válaszról van szó. Itt is hozzáadjuk a számot a *sum*-hoz. Ezen a szinten már fel van töltve a tömbünk egészekkel. Törtekké kell alakítanunk amiket ha összeadunk egyet kapunk. Egy újabb ciklus segítségével végig megyünk a tömbön és leosztjuk a számokat a *sum* változóval.

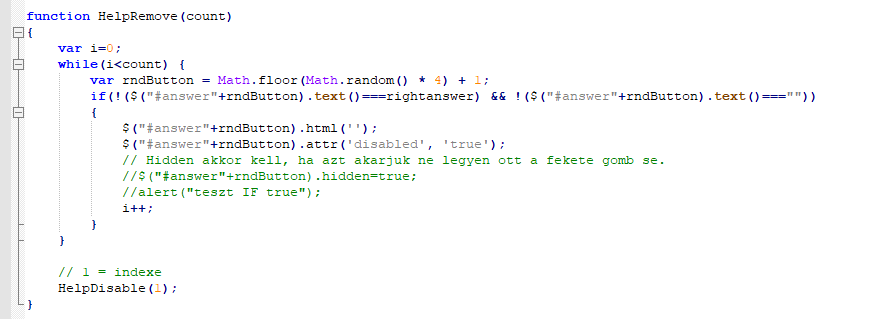


Hogy 2 tizedesig írjuk ki a számokat .toFixed(2) metódusát használtunk. Itt figyelni kell rá, hogy stringre változtatja a számokat.

A függvény végén egy return-nel visszaadjuk a *tips* tömbünket.

Felezés:

A felezés funkció két rossz választ vesz el véletlenszerűen.



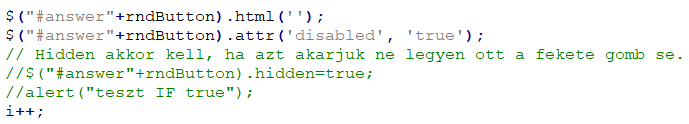
Először felveszünk egy *i=0* változót a ciklus lépéseinek követése végett. Majd egy *count* (alap értéke 2) nevű bemeneti paraméterrel ellátott while ciklusban generálunk 1 és 4 között véletlenszerűen egy számot (*rndButton* változóba), ami a gomboknak a számát fogja megadni.



Ezt a számot hozzá kell fűzni az *#answer* stringhez és így megkapjuk az egyik html válasz gombnak a nevét.



Ezt egy elágazáson belül tesszük meg amiben .text() metódusával az adott gomb text-jét tudjuk kikérni és ezt összehasonlítani a *rightanswer* globális változónkkal, így megvizsgáltuk, hogy az gombunk jó vagy rossz választ ad. Ha rossz választ akkor inaktívvá tesszük a gombot és növeljük az *i* változónkat.



Jó válasz esetén tovább megy a ciklus mindaddig amíg az *i* értéke el nem éri a *count* értékét azaz összesen 2 darab random rossz választ fog elvenni a függvényünk.

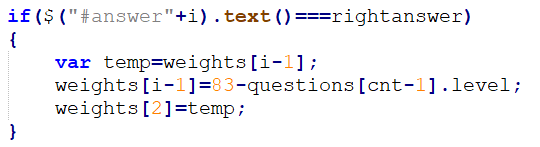
A függvény végén a HelpDisable() függvény hívását illesztettük be, ami inaktívvá teszi ezt a segítséget.

Einstein:

Einstein segítség hasonló a közönség szavazáshoz. A különbség az, hogy ez csak egy tippet ad, de azt nagyobb valószínűséggel. Minél előrébb járunk a játékban annál jobban romlik a pontossága.



Súlyozott random elven működik a függvény. A súlyozás értékeit a *weights* tömb tárolja. A következő for ciklus egy elágazás segítségével megkeresi, hogy melyik gombon lesz a jó válaszunk. Ezt az eddig megszokott módon .text() metódussal, összehasonlítja a html gomb stringjét és a *rightasnwer* változónkat. Ha egyezik akkor a tömbön, a gombnak megfelelő pozícióra írja át a 83-mat csökkentve a játékban elért jelenlegi szinttel (amit a *cnt* változó tárol). Ezzel gyakorlatilag a jó válasz gombjának a pozíciójára helyeztük a legnagyobb súlyozást.



Ezután felveszünk egy *sumweight* nevű változót, 0 kezdőértékkel. Ez fogja tárolni a súlyozások összegét. Egy for ciklussal végig megyünk a tömbön és az értékeket hozzá adjuk a változónkhoz. A következő lépésben kreálunk egy random számot 1 és a súlyozás értéke között. Ezt az *rnd* változó tárolja. Majd egy újabb for ciklussal végig lépdelünk a *weights* tömbön és minden ciklusban egy elágazással megnézzük, hogy az  *rnd* változónk kisebb-e a *weights* értékével. Ha igen visszatérünk egy *i+1* értékkel, ha pedig nem, kivonjuk az *rnd* változóból a tömb jelenlegi értékét.

