
# L2: Kliensoldali JavaScript

## Kliensoldali JavaScript

- Mielőtt nekifognánk a feladatok megoldásának, olvassuk el a minőségi követelményeket tartalmazó megjegyzéseket (`ubb-web-javascript-konvenciok.pdf`).
- Ettől a laborfeladattól kezdődően **elfogadási feltétel**, hogy az *ESLint* statikus kódellenőrző eszköz ne találjon hibákat a JavaScript kódunkban:
  - A repónk gyökerében található kellene legyen egy `package.json` állomány (ha nem látjuk, hívjunk `git pull`-t), amely körbeírja a szükséges eszközök telepítését. Telepítsük az `npm install` paranccsal ugyanazon mappából. Ez jelenleg csak egyszer szükséges.
  - Bizonyosodjunk meg, hogy a következő állományok egyeznek a példarepó gyökerében megadott állományokkal: `.eslintrc.yml`, `.prettierrc.yml`
  - Telepítsük a hivatalos *ESLint* Visual Studio Code **extension**t. Így konzolról is futtathatjuk az eszközt (`npx eslint .` a gyökérből), illetve a VSCode is jelez hibákkal.
  - Állítsuk be a Visual Studio Code-unkat, hogy a *Prettier*t használja JavaScript (`js` kiterjesztésű) állományok formázására.
- Az alábbi feladatokhoz használjunk kliensoldali HTML-t, CSS-t és főként **JavaScript**et.
- **FIGYELEM:** Minden kliensoldalra célzott állományt (pl. `js`-t, amit böngészőben fogunk futtatni) helyezzünk a repónk `public` nevű alfolderébe, másképp az `eslint` nem tekinti kliensoldalinak, s hibákat fog jelezni.

---

### 2.1. Feladat: *Aritmetika kérdés-válasz párosítás*

- Készítsünk egy űrlapot (`form`) az alábbi adatok bekérésére alkalmas elemekkel:
  - név
  - egy-egy checkbox pipa minden alapműveletnek (összeadás/kivonás/szorzás/osztás)
  - hány kérdés lesz generálva (ez egy szám 5 és 10 között)

- Amikor a formot kitöltöttük, egy gombbal indíthassuk el a játékot. Ettől a ponttól a form elemeit ne lehessen módosítani többet. Ha a gombot újra megnyomjuk, a játék induljon újra, s a formban újra tudjuk módosítani az értékeket.
- Generáljunk megadott számú konkrét feladatot, amely random módon tartalmazza a kiválasztott műveleteket. Az operandusok legyenek `0` és `100` közötti értékek.
  - pl. ha az összeadás és szorzás van kiválasztva, lehetnek ilyen generált feladatok: `3+21`, `5*10`, stb.
- Jelenítsük meg egy baloldali oszlopban a generált műveleteket úgy, hogy mindegyik egy színes téglalapban jelenjen meg, egy jobboldali oszlopban pedig az eredmények jelenjenek meg, véletlenszerű sorrendben, szintén színes téglalapokban. Lehet `canvas`-t vagy más generált elemeket használni.
- A felhasználó feladata a műveletek párosítása a megfelelő eredményekkel: előbb a baloldali oszlopból választ egy téglalapot, rákattintva (*váljon is láthatóvá, hogy kiválasztottuk*, azaz keretezzük be egy másik színnel), majd a jobboldali oszlopból válasszunk egy eredményt (ekkor a baloldali kiválasztott téglalap köré rajzolt keret tűnjön el, és a két téglalapot kösse össze egy színes vonal).
- Ha végeztünk minden párosítással, váljon láthatóvá, hogy melyik párosítások helyesek illetve melyek helytelenek.

---

## 2.2. Feladat: *Tic-tac-toe*

- Írjunk tic-tac-toe (avagy X-O) játékot, amit egy `n`x`n`-es táblán lehet játszani, és `m` egy sorba/oszlopba/átlóra letett szomszédos karakter jelenti a nyerést. Az `n`-et és `m`-et lehessen megadni egy legördülő listából (`select`). Alapértelmezetten `n = m = 3`.
- Két játékos játszik egymás ellen, és mindig legyen jól láthatóan feltüntetve, hogy éppen melyik a soros.
- A tábla megjelenítése történhet bármilyen módon (pl. képekkel egy táblázatban, `canvas`on, generált elemekkel, stb.). Ellenőrizzük, és írjuk ki, ha nyert valaki, vagy írjuk ki a végén, hogy döntetlen, amennyiben egyik játékos sem nyert.
- A játékot egy gomb megnyomásával lehessen újrakezdeni.

---

## 2.3. Feladat: *Hol a kép?*

Írjunk egy játékot, amelyben pontosan azonosítani kell egy rövid ideig megjelenő információ pozícióját. A játék a következő szabályok alapján működjön:

- Egy `canvas`on véletlenszerű pozíciókon (ld. `Math.random`) rövid ideig egy kép jelenik meg.
- A felhasználónak a megjelent *(és a rövid idő után eltűnt)* kép helyére kell kattintania.
- Minél közelebb eltalálja a felhasználó a kép pozícióját, annál több pontot kap. Minden klikk után jelenjen *rövid ideig* a távolság a kép közepe és a klikk között, valamint az emiatt nyert összeg.
- A játékos aktuális pontszáma mindig legyen látható a `canvas` jobb felső sarkában.
- A játéknak legyen több szintje, attók függően, hogy mennyi ideig jelenik meg a kép (pl. `0.5s` könnyű, `0.25s` közepes, stb.).
- A játéknak egy megadott számú forduló után van vége, ekkor az összpontszám és a „Játék vége” üzenet íródjon ki az oldalra.

## 2.4. Feladat: *Távirányítós kocsi*

- Egy `canvas` elemre rajzolt „pálya” közepén jelenítsünk meg egy (alaphelyzetben balról-jobbra irányba néző) **autót** (lehet más kép is; nem kötelező, hogy autó legyen).
- Készítsünk egy `form`ot az alábbi választási lehetőségekkel:
  - idő milliszekundumban, amely beállítja, hogy mennyi időnként lépjen az autó egyet (ld. `setTimeout`, `clearTimeout`)
  - radio gombok, melyből a parancsot választhatjuk ki (előre/hátra mozdul, jobbra/balra fordul)
  - egy szám típusú bemeneti mező, melyben a parancs paraméterét adjuk meg:
    * „előre” vagy „hátra” esetén a lépésszámot adjuk meg
    * fordulás esetén pedig az elfordulás szögét adhatjuk meg valamilyen formában
- Az autó mozgása történjen automatikusan ezek alapján.
- Ha az autó kifut a pályáról, jelenjen meg annak ellentétes oldalán. Egy gombra kattintva legyen visszaállítható az alaphelyzet.

## 2.5. Feladat: *Hatból négyet játék*

A játék kezdetén jelenjenek meg a játékos adatai az oldal egy jól látható helyén. Szabályok:

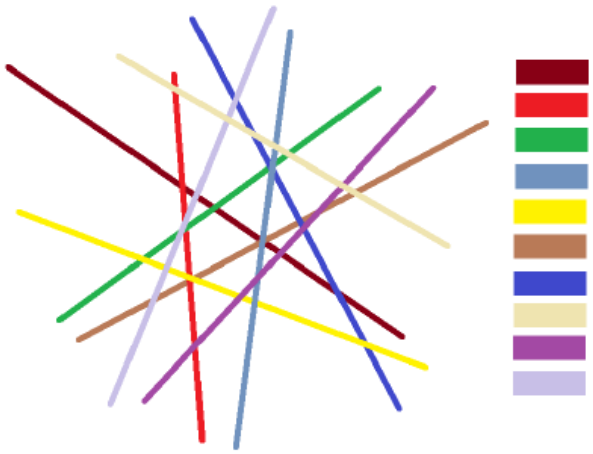
- A gép kiválaszt (véletlenszerűen generál) 6 megadott színből 4-et (egyik szín sem ismétlődhet), ezek sorrendjét is meghatározva. Ezt kell a játékos kitalálja.
- A játékos tippel (pl. négy legördülő listából vagy radio gombokkal választja ki a színeket, de tetszés szerint másképp is megoldható), majd a gép kiértékeli a tippet:
  - megmondja, hogy hány **pontatlan találat** (a szín talál, de a pozíciója nem) van
  - megmondja, hogy hány **pontos találat** (a szín és a pozíció is talál) van.
- A játékban hagyományosan fekete négyzet jelöli a pontatlan válaszokat, illetve fehér négyzet jelöli a pontos találatokat. Mi is hasonlóan oldjuk meg.
- A négyzeteket egy, az oldalon előre elhelyezett `canvas` elemen jelenítsük meg.
- A játék végén megjelnik az eredmény: kitalálta-e a játékos, s ha igen, akkor hány lépésből. A játékos életkorától függően más-más szöveget írjunk ki.

Lényeges, hogy a gép válaszából csak a helyes vagy helytelen találatok **száma** derül ki, az, hogy konkrétan melyik színre vonatkozik, az nem. Ha a játékos még nem találta ki a színeket a helyes sorrendben, akkor ismét tippelhet. A tippek és válaszok egymás alatt jelenjenek meg. Max. 8-at tippelhet a játékos, ha nem sikerül kitalálnia a 4 színt a helyes sorrendben, akkor veszített, különben nyer.

*Tipp:* véletlenszám generálása: `Math.random` segítségével.

## 2.6. Feladat: *Szövegszerkesztő*

- Adott két szövegdoboz (`textarea`) – az egyikbe szóközzel elválasztott szavakat írunk, ezek lesznek a figyelt szavak. A másikba szöveget írunk. A második szövegdoboz alatt megjelenik a beírt szöveg megfelelőképpen kiszínezve, a gépeléssel egyidőben, a következő szabály szerint:
  - Ha a második szövegdobozba írt szövegben valamely szó (akár részszóként is) szerepel az első szövegdobozban, akkor *pirossal* jelenítjük meg, a többi karakter színe *fekete*.



1. ábra. Pálcikák

  - Ha pl. az első szövegdobozban az **aaa** és **aab** szavak szerepelnek, akkor az **abaa** szó nincs kiszínezve, míg az az **aaab** teljesen piros.

- Egy checkbox pipával lehessen beállítani, hogy a rendszer *case-sensitive*–e – ha nincs kipipálva, a kis- és nagybetűk nem számítanak, de ha ki van, akkor pontos egyezés szükséges

---

## 2.7. Feladat: *Kő, papír, olló játék*

- Készítsünk egy űrlapot (`form`), amely bekéri a felhasználó becenevét, s hogy mennyi pénze van. Ha megkezdődött a játék, a játékos beceneve és pénze legyen feltüntetve jól látható módon.
- A játékos három megfelelő kép *(kő, papír vagy olló)* közül választva adja meg a saját tippjét, a gép pedig generálja a sajátját. Miután valamelyik képre kattintottunk, csak a kiválasztott kép maradjon látható, illetve a gép tippje is váljon láthatóvá. A nyertes fizet *x* összeget *(a gép kezdeti pénzösszegét generáljuk véletlenszerűen)*. Az alábbi szabály adja meg, hogy ki a nyertes:
  - Ha a két fél tippje megegyezik, nem nyert senki
  - A kő elcsorbítja az ollót *(tehát a kő nyer)*
  - Az olló elvágja a papírt *(az olló a nyertes)*
  - A papír becsomagolja a követ *(a papír nyer)*
- Tüntessük fel, hogy ki nyert, aktualizáljuk a játékos (és a gép) pénzösszegét, majd egy gombra kattintva folytathatjuk a játékot egy újabb fordulóval mindaddig, amíg óhajtjuk, illetve mindkét félnek van pénze.

---

## 2.8. Feladat: *Pálcikák*

- Rajzoljunk 10 db. színes pálcikát egy `canvas`ra (a szakaszok végpontjainak koordinátáit, illetve a tíz különböző színt megadhatjuk előre, a színek sorrendjét viszont generáljuk véletlenszerűen). A színkódoknak megfelelő téglalapokat helyezzünk el a kép jobb oldalán az ábrán szemléltetett módon.
- A feladat az, hogy a pálcikákat a megfelelő sorrendben *távolítsuk el*, a színkódokra kattintva: mindig a legfelső pálcikát kell elvenni. Ha a felhasználó a helyes színkódra kattintott, akkor a megfelelő pálcika tűnjön el a rajzról, különben a felhasználónak eggyel kevesebb „élete” marad (a kezdeti háromból, amit szintén jelenítsünk meg, tetszés szerint numerikus vagy grafikus formában). Egy „ismétlés” gombra kattintva ismétlődjön a játék.

- *Tipp:* véletlenszám generálása: `Math.random` segítségével.

---

### 2.9. Feladat: *Memóriajáték*

- Készítsünk egy memóriajátékot, amely a következő szabályok alapján működik:
  - Egy számszerű szöveges mezőbe beírhatunk egy 10-nél nagyobb páros számot – `n`.
  - A játék indításakor megjelenik `n` felfordított kártya. Ha egyre ráklikkelünk, megfordul s láthatjuk a képet az alján. A következő klikkünk ennek egyedi párjára kell essen – ha ez nem sikerül, mindkét kártyát visszafordítjuk. Ha megtaláljuk párját, felfordulva marad mindkettő, s a ciklus újrakezdődik.
  - A játék addig tart, míg minden párt sikeresen megtaláltunk.
- A rendszer már a játék elején generálja ki a képpárok helyzetét (ld. `Math.random`).
- Jelenjen meg az oldal alján egy gomb, amellyel akármikor újraindíthatjuk a játékot – ezzel újragenerálódik a kártyák helye is.
-

---

### 2.10. Feladat: *Itt a piros, hol a piros*

- Készítsünk egy űrlapot (`form`), amely bekéri a felhasználó becenevét, s hogy mennyi pénze van (egy pénznemet választhassunk ki egy legördülő listából). Ha megkezdődött a játék, a játékos beceneve és pénze legyen feltüntetve jól látható módon.
- Ha megkezdődött a játék, jelenjen meg a játékos neve az oldal egy jól látható helyén. A játék szabályai a következők:
  - A játékos 3 kép közül választva tippelhet arra, hogy „hol a piros” *(a képet, amire tippelt „fordítsuk fel”, hogy látható legyen az eredmény)*.
  - Ha eltalálta a helyes *(véletlenszerűen generált)* pozíciót, *(a pénznemtől függő)* `x` összeget nyer, különben kétszer annyit veszít.
  - A játék addig ismétlődik, míg a játékos meg nem unja *(„meguntam” gombra kattintva)* vagy a pénze a minimális alá csökken.
  - Mindig legyen kiírva, hogy a játékosnak mennyi pénze maradt, illetve a nyerés vagy vesztés függvényében más-más kép jelenjen meg.

*Tipp:* véletlenszám generálása: `Math.random` segítségével.

## Megjegyzések

- **A példaprogramokkal és a leadással kapcsolatos információk az `ubb-web-lab0-setup.pdf` állományban találhatóak**.
- A létrehozott `HTML` oldalak teljes mértékben helyesek kell legyenek úgy szintaktikailag, mint szemantikailag. Ennek ellenőrzése céljából alábbi három eszközt használjuk:
  - A szintaktikai ellenőrzésre használjuk a W3C `HTML` validátorát: `http://validator.w3.org`
- További kódolási konvenciókat találunk az `ubb-web-javascript-konvenciok.pdf` állományban.
- Próbáljunk meg az adott követelmények betartása mellett (esetleg annak ellenére) az oldal esztétikájára is figyelmet fordítani.
- A példaprogramok ezen témához tartozó része: `2-javascript/`