

# JavaScript folytatás

## 2. Samuel Loyd feladvány

Bizonyára sokan ismeritek [Samuel Loyd](#) 1878-ban kitalált Boss-Puzzle játékát. A korábban "[15-ös játék](#)" néven ismertté vált feladványban a 4x4 férőhelyet tartalmazó sík keret 1-től 15-ig számozott négyzetlapocskái a 16. "üres hely" vándoroltatásával rendezhetők sorba. Az évek múlásával a lapocskák számozását mozaikra szétvágott képek váltották fel, mintegy játékosává téve a eredeti matematikai probléma megoldását. Igen, ez a Tili-Toli néven elhíresült tologatós játék, amelyet most a virtuális síkban fogunk kifaragni.

**Bemenet:** A böngésző munkaterületén kialakított játéktér.

**Kimenet:** A puzzle elemei, valamint a számított eredmények.

**Változók:** Játéktér sor és oszlop elemei, illetve a HTML dokumentum azonosított elemei.

**Algoritmus:** A HTML kód "puzzle" azonosítójú táblázatát feltöltjük a row vektorba ágyazott cell tömb formázott elemeivel. A tömbelemek az egéresemények bekövetkezésekor meghívják a mozgatási műveletek eljárását, ami a kijelölt dokumentumelemeket felcseréli. A sikeres megoldás a [DOM modell](#) metódusaira épül.

**Eseménykezelés:** Az onclick [esemény](#) bekövetkezésekor végrehajtandó utasításokat a mystep() függvényben rögzítjük.

Az alábbiakban rendszerezzük a megoldáshoz szükséges ismereteket. A platformfüggetlen megoldás elkészítéséhez gyakorlatilag egy böngésző programra, és esetleg egy egyszerűbb editorra van szükség.

## 2 Hogyan épül fel a DOM modell?

"A Document Object Model egy platform- és nyelvfüggetlen interfész, amely hozzáférést biztosít a programok és szkriptek számára a dokumentumtartalmakhoz. Modellezi a dokumentum szerkezetét és lehetővé teszi annak tartalmi és vizuális változtatását. Lényegében összeköttetést képez a weblapok és a script- vagy programozási nyelvek között.

Minden tulajdonságot, metódust és eseményt, ami a webfejlesztő számára rendelkezésre áll a weboldalak szerkesztése és változtatása során, objektumokba rendszerez. (pl. a *document* objektum jelöli a dokumentumot, a *table* objektum jelöli a HTML táblázatokat, stb.) Ezek az objektumok hozzáférhetőek a script-nyelvek számára az újabb böngészőkben.

A DOM-ot leggyakrabban [JavaScript-tel együtt](#) használják. Azaz a kód JavaScript-ben van írva, de a DOM-ot használja a weboldalhoz és elemeihez való hozzáférés során.

A DOM-ot azonban úgy tervezték hogy független legyen minden programozási nyelvtől, ezért a dokumentum szerkezeti modellje egyetlen, önálló és konzisztens API-ból érhető el. Bár a továbbiakban a JavaScriptre fogunk összpontosítani, a DOM-ot tulajdonképpen bármilyen nyelvből elérhetjük.

A [World Wide Web Consortium \(W3C\)](#) meghatározta a [standard DOM](#)-ot, amit W3C DOM-nak neveznek. Ma már a legfontosabb böngészők ezt támogatják, ezzel lehetővé teszik browserfüggetlen alkalmazások létrehozását."<sup>[1]</sup>

Kezdetben arra ügyeljünk, hogy minden tartalmi elem kapjon azonosítót az id=" " attribútumban.

## 2

## Miért használjunk CSS-t a formázáshoz?

A CSS szabvány leírása 1996. december 17-n látott napvilágot a W3C honlapján. "A CSS jelentése Cascading Style Sheets, azaz egymásba ágyazott stíluslapok. A HTML oldalaink megjelenését befolyásoló egyszerű szabványról van szó, mely segítségével meghatározhatjuk, hogy hogyan (és hol) jelenjenek meg az egyes HTML elemek (paragrafusok, címsorok, stb.), többek között befolyásolhatjuk a színüket, méretüket, elhelyezkedésüket, margóikat, stb. Az egymásba ágyazhatóság (kaszkádozás) arra utal, hogy több stíluslapot, meghatározást is megadhatunk egyszerre, illetve egy stílus lehet több elemre is érvényes, amit egy másik stílussal felüldefiniálhatunk. A stílusok öröklődnek az oldal hierarchiája szerint, ha például a gyöker elemre definiálunk egy stílust, akkor az többnyire az oldal összes elemére érvényes (a tulajdonságok öröklhetőségétől függően)."<sup>[2]</sup>

Stíluslapot négyféleképpen használhatunk a dokumentumban:

1. Beágyazott stíluslapként a HTML oldal fejlécében:

```
<head>
<style>
h1{
  color:#1f2839;
  font-size:2.5em;
  text-shadow: 6px 6px 2px #d0d0d0;
}
</style>
</head>
```
2. Külső stíluslapra hivatkozással:

```
<head>
<link rel="stylesheet" href="kulso.css" type="text/css">
</head>
```
3. Importálással:

```
<style>
@import url(http://www.mypage.hu/style/other.css);
</style>
```
4. Elemhez rendelt stílusként a style="" attribútumban leírva.

```
<h1 style="color:#1f2839;font-size:2.5em;text-shadow: 6px 6px 2px
#d0d0d0;">Címsor</h1>
```

A CSS formázások a [boxmodellre](#) épülnek, a részletes leírások és [példák](#) a [w3schools.com](#) oldalon elérhetőek. További magyar nyelvű források: [A dobozmodell](#); [CSS alapok](#); [CSS kijelölők](#); [HTML5.0 + CSS3](#). A stíluslapok alkalmazásával az alapvető formázási beállítások módosítása a tartalmi elemek változtatása nélkül megoldható. Az alábbiakban nézzük meg a kirakójáték kódjában használt CSS beállításokat. A *body* elemnél beállítjuk a használt betűtípust és a középre igazítást. A *section* elemnél megadjuk a játékkeret is magába foglaló dokumentumrész jellemzőit. Meghatározzuk a *h1* és *p* elemek stílusát, majd definiáljuk a *puzzle* azonosítóhoz rendelt elem megjelenítési beállításait.

```

<style type="text/css" media="screen,projection,print">
  body{
    font-family: Arial, Helvetica, sans-serif;
    margin: auto;
  }
  section{
    min-width:400px;
    padding:20px;
    margin: 10px auto 10px auto;
  }
  h1,p{
    text-align:center;
  }
  h1{
    color:#1f2839;
    font-size:2.5em;
    text-shadow: 6px 6px 2px #d0d0d0;
  }
  #puzzle {
    background:orange;
    font-size:2em;text-align:center;
    margin: auto;
    border-color: orange;
    box-shadow: 10px 10px 5px #d0d0d0;
  }
</style>

```

### 3 Töltsük fel a játéktáblát új cellákkal!

A HTML dokumentumban helyet foglalunk *puzzle* azonosítóval a játéktáblának. A sorokat és a cellákat még nem definiáljuk, mert azokat a JavaScript kóddal fogjuk előállítani. Elhelyezünk egy gombot, amelynek lenyomása az onclick esemény révén indítja el a mozaik keverését.

```

<section>
  <h1>Tili-Toli puzzle</h1>

  <table id="puzzle" border="1"></table>

  <p><input type="button" value="Keverés" onclick="shuffle()"></p>
</section>

```

#### Próbáld ki!

A HTML kódban csak egy üres táblát definiáltunk *puzzle* azonosítóval. Ezért most az *insertRow()* módszerrel létrehozzuk a tábla sorait, illetve az *insertCell()* módszerrel a sorok celláit, majd beállítjuk az így implementált cellák tulajdonságait, és az eseménykezeléshez használni kívánt függvényhivatkozást is. Figyeljétek meg, hogy minden cella kapott egy azonosítót, amit a *tt[i][j].id=i\*ncell+j;* (vagyis sorszám\*cellák\_száma+cellaszám) kifejezéssel számoltunk ki. Ez az azonosító bármikor konvertálható sor és oszlop indexre.

```

var nrow = 5;
var ncell = 5;
var tt = new Array(nrow);
var newrow;

for(var i=0;i<nrow;i++){
    newrow=document.getElementById("puzzle").insertRow(i);
    tt[i] = new Array(ncell);
    for(var j=0;j<ncell;j++){
        tt[i][j]=newrow.insertCell(j);
        tt[i][j].id=i*ncell+j;
        tt[i][j].onclick=function(){mystep(this);};
        tt[i][j].style.width="50px";
        tt[i][j].style.height="50px";
        tt[i][j].style.color="#fcfcfc";
        tt[i][j].style.background="#ff3333";
        tt[i][j].innerHTML=parseInt(tt[i][j].id)+1;
        // A JS 0-tól számolja a tombelemeket, de a puzzle 1-től!
    }
}

```

[Próbáld ki!](#)

## 4 Keverjük össze a cellákat!

A HTML kód letöltésekor a játéktábla a kirakott állapotot mutatja, így meggyőződhetünk arról, hogy valamennyi szám bekerült a játéktérbe. A játék indításához azonban, össze kellene kevernünk a mozaikot. A *keverés* gombhoz rendelt metódus ezt a feladatot irányítja. Működése nagyon egyszerű, hiszen egy kocka négy irányba mozoghat, csak arra kell vigyáznunk, hogy a tábla sor vagy oszlopszámánál nagyobb értéket ne adjon a léptető metódusnak.

```

function shuffle(){
    var xwalker;
    var ywalker;
    for(var i=0;i<nrow*ncell*6;i++){
        rowwalker=Math.floor(empty/ncell);
        cellwalker=empty%ncell;

        switch(Math.floor(Math.random()*4)) {
            case 0: // right or left
                cellwalker+=(cellwalker<ncell-1)?1:-1;
                break;
            case 1: // left or right
                cellwalker+=(cellwalker>0)?-1:1;
                break;
            case 2: // down or up
                rowwalker+=(rowwalker<nrow-1)?1:-1;
                break;
            case 3: // up or down
                rowwalker+=(rowwalker>0)?-1:1;
                break;
        }
        mystep(document.getElementById((rowwalker*ncell+cellwalker).toString(
    )));
    }
}

```

[Próbáld ki!](#)

## 5 Kezeljük le az egéreseeményeket!

Elérkeztünk a játék lényegéhez, a mozaik léptetését kezelő metódushoz. Ha a játéktáblán olyan kockára kattintunk, amelyiknek üres cella az élszomszédja, akkor bizony helyet kell cserélniük egymással. Ehhez a felismeréshez az *if* utasításban kifejtett összetett logikai kifejezéssel jutunk el.

```
function mystep(obj) {
    var nid = parseInt(obj.id);
    var nempty = parseInt(empty);

    if (nid+ncell==nempty || nid-ncell==nempty ||
        nid+1==nempty && nempty%ncell!=0 ||
        nid-1==nempty && nid%ncell!=0) {

        document.getElementById(empty.toString()).innerHTML=document.getElementById(obj.id).innerHTML;

        document.getElementById(empty.toString()).style.background=document.getElementById(obj.id).style.background;
        document.getElementById(obj.id).style.background="#d0d0d0";
        document.getElementById(obj.id).innerHTML="";
        empty=obj.id;
    }
}
```

[Próbáld ki!](#)

## 6 Gyümölcs szedő játék

*Most hallottuk a hírekben, hogy holnapra hatalmas jégeső várható. A kertünk tele van érő gyümölcsökkel, és a jég mindet tönkre fogja tenni! Gyorsan, van még egy napunk, szedjük be, amit csak tudunk! Minél többet meg tudunk menteni, annál jobb...*

A második fordulóban egy gyümölcs szedő játék megvalósítása a cél. A játékban egy négyzetrácsos pályán (a kert) kell mozogni, és adott mennyiségű lépésben minél több gyümölcsöt összeszedni. Minden mezőben egy-egy szám lesz, ez az adott helyen beszédhető gyümölcsök mennyisége.

## 7 Egy kis ráhangolódás a feladatra

A leckében tárgyaltak összefoglalásaként nézzünk meg a feladat alapjait. Az alábbi kód kirajzol egy táblázatot, aminek cellái a játéktábla elemei. Az szeretnénk, hogy a tábla könnyen módosítható és kezelhető legyen, ezért JavaScript segítségével építjük fel. A script elején megadjuk a méretet. A táblázat minden mezőjébe egy generált számot is ír a script. Illetve az eseménykezelés kód átszínezi egy mezőt, amire rákattintunk. Figyeljük meg a CSS beállítások elhelyezését és tartalmát, a dokumentumelemek azonosítását, a komponensek futásidejű

létrehozásának módszerét és az eseménykezelést. Gondoljátok át, vajon miért került most a script a HTML kód végére?

```
<!DOCTYPE html>
<html lang="hu">
<head>
<title>Számok</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />

<style type="text/css" media="screen,projection,print">
body{
    font-family: Arial, Helvetica, sans-serif;
    margin: auto;
}
section{
    min-width:400px;
    padding:20px;
    margin: 10px auto 10px auto;
}
h1,p{
    text-align:center;
}
h1{
    color:#1f2839;
    font-size:2.5em;
    text-shadow: 6px 6px 2px #d0d0d0;
}
#palya {
    background:orange;
    font-size:2em;text-align:center;
    margin: auto;
    border-color: orange;
    box-shadow: 10px 10px 5px #d0d0d0;
}
</style>
</head>
<body>

<section>
    <h1>Játék</h1>

    <table id="palya" align="center">
    </table>
</section>

<script>

    var nrow = 8;
    var ncell = 8;
    var tt  = new Array(nrow);
    var newrow;

    function init()
    {
        for(var i=0;i<nrow;i++){
            newrow=document.getElementById("palya").insertRow(i);
            tt[i] = new Array(ncell);
            for(var j=0;j<ncell;j++){
                tt[i][j]=newrow.insertCell(j);
                tt[i][j].id=i*ncell+j;
            }
        }
    }
</script>
```

```

        tt[i][j].onclick=function(){cellClicked(this);};
        tt[i][j].style.width="50px";
        tt[i][j].style.height="50px";
        tt[i][j].style.color="#fcfcfc";
        tt[i][j].style.background="#999999";
        tt[i][j].innerHTML=(i+j)%5+1;
    }
}

function clearBackground()
{
    for(var i=0;i<nrow;i++){
        for(var j=0;j<ncell;j++){
            tt[i][j].style.background="#999999";
        }
    }
}

function cellClicked(obj)
{
    clearBackground();
    var row      = parseInt(obj.id/ncell);
    var column    = obj.id%ncell;
    tt[row][column].style.background="#000000";
}

</script>

<script> init(); </script>
</body>
</html>

```

## **8 Oldd meg a feladatokat!**

### **Kedves Bakonyi Bitfaragók!**

Eljött a feladat megoldásának ideje.

**Fontos: A feladatokat egyben kell beküldeni, minden fájlt egy tömörített állományba csomagolva. A fájlokat helyi gépről nyitjuk meg, nem szerverről, így kell működniük.**

## Beküldhető feladatok:

1) A feladatok, amik rátok várnak:

- a) Az oldal betöltésekor generálódjon egy pálya, minden mezőben egy véletlen generált számmal.
- b) Legyen egy megadott lépésszám, amit az oldal ki is jelez. Ennyi körünk van minél több gyümölcsöt összeszedni.
- c) Valahogy jelöljétek, hogy melyik mezőn áll éppen a játékos.
- d) A játék legelején választhassuk ki, hogy melyik mezőről szeretnénk indulni. Az ott található gyümölcsöket egyből leszüreteljük, de ez még nem számít lépésnek.
- e) Innentől kezdve minden lépéssel csak fel/le/jobbra/balra léphetünk egyet.
- f) Amikor egy mezőre odalépünk, az ott található gyümölcsöket leszüreteljük. Az összesített mennyiség mindig legyen látható.
- g) Ha a lépések elfogytak, akkor érjen véget a játék és jelenítse meg a végső pontszámot.
- h) Lehessen új próbálkozást kérni ugyanazon a pályán, az eredeti értékek visszaállításával. Ilyenkor az eddigi próbálkozások közül a legjobbat jelenítse meg az oldal emlékeztetőnek.
- i) Legyen 2-3 különleges képesség, melyek egyszer-egyszer aktiválhatóak a játék során. Lehet kreatívnak lenni a képességekkel, és azzal is lehet variálni, hogy melyiknek a használata vesz el lépést. Pár ötlet, de nyugodtan lehet mást is kitalálni: átlós mozgás, teleportálás másik mezőre, ugrás 2-3 mező távolságra, 1-1 gyümölcsnyi szüret a szomszédos mezőkről, helyben maradva szüretelés valamelyik szomszéd mezőről.
- j) Valósítsatok meg stílusos megjelenítést. Például lehetnek változatos típusú gyümölcsök, látszódnak a különbség a gyümölcsökkel teli fák és a már leszüretelték között.

**1100 0000**<sub>2</sub> pont

(A feladatok elkészítése során nem kell követni a megadott mintákat, azt teljes egészében saját elképzelések alapján újra írhatjátok, csak arra figyeljétek, hogy funkcionalitást ne veszítsen az alkalmazás.)