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 játékossá 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."[1]
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 (kaszkádolás) 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 gyökér elemre definiálunk egy stílust, akkor az többnyire az oldal összes elemére érvényes (a tulajdonságok örökölhetőségétől függően)."[2]
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 játékteret 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 JavaScrpit 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()* metódussal létrehozzuk a tábla sorait, illetve az *insertCell()* metódussal 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-tol szamolja a tombelemeket, de a puzzle 1-tol!
    }
}
```

Próbáld ki!

4 Keverjük össze a cellákat!

A HMTL 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 egéreseményeket!

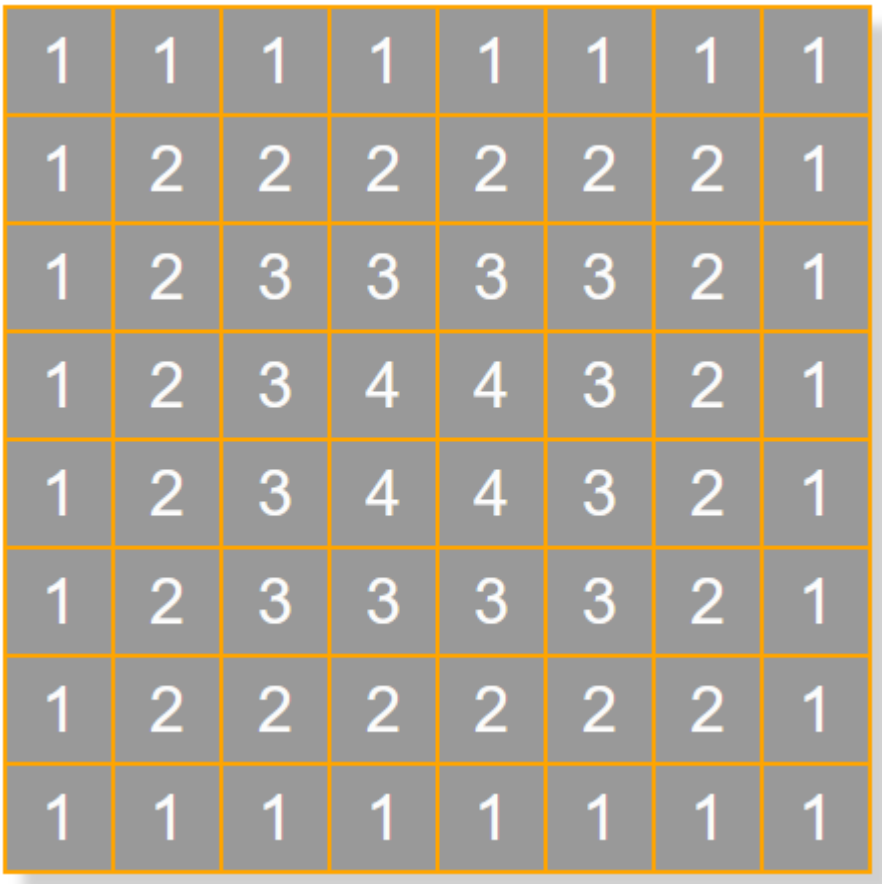
Elérkeztünk a játék lényegéhez, a mozaik léptetését kezelő metódushoz. Ha a játéktáblán olyan kockára kattintunk, amelyiknek üres cella az élszomszédja, akkor bizony helyet kell cserélniük egymással. Ehhez a felismeréshez az *if* utasításban kifejtett összetett logikai kifejezéssel jutunk el.

```
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.getEleme
ntById(obj.id).innerHTML;

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

Próbáld ki!

6 Reversi játék

Játék leírás: https://en.wikipedia.org/wiki/Reversi
A mostani feladat a Reversi játék megvalósítása. A játékot két játékos játssza egy négyzethálós játéktéren. A játékosok felváltva helyezik el korongjaikat, a még üres négyzetek valamelyikébe. Minden korong lehelyezése után az ellenfél közrefogott korongjai (vízszintesen, függőlegesen, átlósan) a saját színünkre fordulnak.
A játékot annyival módosítjuk azonban, hogy a végén a pontszámításban a mezők értéke eltérő lesz. Az a játékos nyer, akinek több pontja van a végén. A táblázat szélén lévő mezők 1-1 pontot érnek, befelé haladva pedig mindig eggyel többet. Az alábbi ábra mutatja a mezők értékét 8x8-as pálya esetén, de a pálya méret nem fix, más méretre is működnie kell.



7 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) Készítsétek el a fentebb leírt játékot a következő módon:
   a) Legyen egy négyzetrácsos pálya (egy táblázat), rajta az játék kezdő állapotával (középső 4 mezőn két-két korong, váltott helyzetben)!
   b) A táblázat méretét a játékos tudja a játék előtt beállítani (csak négyzetes lehet)!
   c) A korongokat jelölhetik mondjuk 'X' és 'O' karakterek, de a teljes pont elérése érdekében legyenek képekkel ábrázolva!
   d) Működjön maga a játék: korongok lerakása és átforgatás (vízszintes, függőleges, átlós)!
   e) A játék végén írja ki a program hogy ki nyert! A pontszámításban figyelembe kell venni, hogy a tábla közepén lévő korongok többet érnek, mint amik a szélén vannak.
   f) A játék mindig jelezze, hogy éppen melyik játékos következik!
   g) A játék során mindenki csak olyan helyre tehessen, ahol tud átforgatni! Ha egy játékos nem tud tenni, akkor kimarad.
   h) A játék vége feltétel legyen a hivatalos változatra! (Akkor van vége, ha már senki sem tud tenni, vagy betelt a tábla.)
   i) Adjatok hozzá egy órát, amely másodperc pontossággal kijelzi a játék megkezdése óta eltelt időt!
   j) Az átforgatás ne egyszerre történjen, hanem hullámszerűen! (A lerakott korongtól induljon, a messzebbiek később forduljanak át.) Amíg az animácó tart, addig ne lehessen korongot lerakni.

1100 0000$|_2$ pont

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