Multimédia nappali gyakorlat

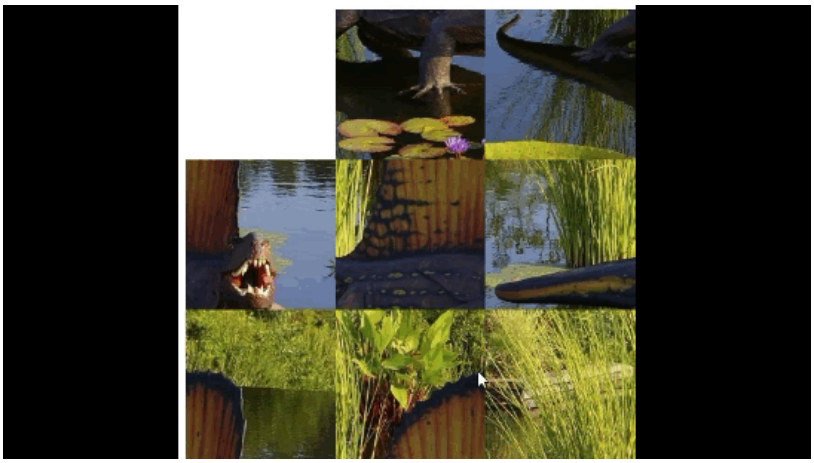
Puzzle és almaszedő játék

A tananyaghoz készült videó az alábbi linken érhető el: https://www.youtube.com/...

Utoljára módosítva
2020. április 29.

Észrevételek, javaslatok
✉ mkatona@inf.u-szeged.hu

Puzzle játék (<canvas>)



Megoldás: megoldas.zip

Megoldás menete

Indítsuk el a PhpStorm-ot vagy Webstorm-ot és hozzunk létre egy új projektet, amiben legyen egy `index.html` is. A játék elkészítéséhez szükséges képet megtaláljátok a megoldás csomagjában. Tekintsük meg lépésről lépésre, hogy miként lehet egy egyszerű puzzle játékkot elkészíteni. Először is hozzunk létre egy `<canvas>` HTML elemet a `<body>`-ban. Jelen példában egy `main` ID-vel és class-al ellátott DIV-be ágyaztuk. A mérete legyen 480x480 pixel. Ez megegyezik a mellékelt kép méretével, melyet szét fogunk darabolni.

```
<div id="main" class="main">
  <canvas id="puzzle" width="480px" height="480px"></canvas>
</div>
```

A megoldás első része nem tartalmaz újdonságot, mert már sokszor láttuk, hogy ahhoz, hogy a vászonra tudjuk elhelyezni bármit, először azt el kell érni. A `boardSize` változó a játéktér szélességét (és magasságát is, mivel NxN-es) tárolja. A `tileCount` segítségével állíthatjuk be, hogy hány egyenlő részre kívánjuk szeletelni a képet. Ebből egyértelműen következik, hogy a `tileSize`, azaz egy-egy puzzle mérete a játéktér méretének és a csempék számának hányadosából adódik. Az aktuális egérkattintás pozícióját a táblán a `clickLoc` változóban tároljuk le, míg az üres négyzet pozícióját az `emptyLoc`-ban. Annak a vizsgálatára, hogy megfelelő sorrendbe raktuk-e a kép darabjait, a `solved` flaget használjuk. A `boardParts` változót a táblát alkotó daraboknak deklaráljuk.

```
var context = document.getElementById('puzzle').getContext('2d');

var boardSize = document.getElementById('puzzle').width;
var tileCount = 3;
var tileSize = boardSize / tileCount;

var clickLoc = {x:0, y:0};
var emptyLoc = {x:0, y:0};
var solved = false;

var boardParts;
```

Adjuk hozzá a vászonhoz azt a képet, melyet darabokra fogunk osztani. Eddig `onload()` függvényhívás segítségével vizsgáltuk meg, hogy a kép elérhető-e már, de ezt úgy is megtehetjük, hogy arra eseményfigyelőt helyezünk el és a figyelt esemény a `load` lesz. Amint betöltődött a kép, meghívjuk a kirajzolásért felelős függvényt.

```
var img = new Image();
img.src = 'dimetrodon.jpg';
img.addEventListener('load', drawTiles, false)
```

Először letöröljük a vásznat. Mivel ezt mindig meg fogjuk hívni, amikor megváltoztatjuk a csempék pozícióját, így nem lesz látható az "elmosódás" effektusa. 2D-s tömbként fogjuk reprezentálni a táblát, melyet fel is töltünk. Jelen példában egy 3x3-as táblánk van. Ez így képzelhető el:

(0, 0)	(0, 1)	(0, 2)
(1, 0)	(1, 1)	(1, 2)
(2, 0)	(2, 1)	(2, 2)

Tehát, sorra vesszük az összes pozíciót és megvizsgáljuk, hogy az aktuális pozíció megegyezik-e az üres pozícióval vagy megoldottuk-e már a feladványt. Már korábban számos példát láthattuk a `drawImage` meghívására, ám azokat a bemenő paramétereit nem használtuk, melyekkel egy adott képet részekként kirajzolhatunk.

```
function drawTiles() {
  context.clearRect ( 0 , 0 , boardSize , boardSize );

  for (var i = 0; i < tileCount; ++i) {
    for (var j = 0; j < tileCount; ++j) {
      var x = boardParts[i][j].x;
      var y = boardParts[i][j].y;

      if(i != emptyLoc.x || j != emptyLoc.y || solved == true) {
        context.drawImage(img, x * tileSize, y * tileSize, tileSize, tileSize, i * tileSize, j *
            tileSize, tileSize, tileSize);
      }
    }
  }
}
```

Inicializáljuk a táblánkat.

```
setBoard();
```

Ehhez végimegyünk a 2D-s pályán és megadott pozíciókat írunk le. Ennek köszönhetően nem lesznek egymés után az aktuális darabok. Ezt csak 1x fogjuk meghívni. Beállítjuk az üres pozíció helyét is, illetve `false`-ra inicializáljuk a megoldás vizsgálatához használt változót.

```
function setBoard() {
  boardParts = new Array(tileCount);
  for (var i = 0; i < tileCount; ++i) {
    boardParts[i] = new Array(tileCount);
    for (var j = 0; j < tileCount; ++j) {
      boardParts[i][j] = {
        x: (tileCount - 1) - i,
        y: (tileCount - 1) - j
      }
    }
  }
  emptyLoc.x = boardParts[tileCount - 1][tileCount - 1].x;
  emptyLoc.y = boardParts[tileCount - 1][tileCount - 1].y;
  solved = false;
}
```

A táblára beállítunk egy kattintást figyelő eseményt. Amennyiben ez bekövetkezik, lekérjük a kattintott pozíciót, valamint a vászon elhelyezkedésére vonatkozó offset-et, melyet kivonunk belőle. Annak érdekében, hogy abban a koordinátarendszerben kapjuk meg a kattintási pozíciót, amit korőábban bemutattam, elosztjuk mindezt egy darab méretével. Ezt követően szomszédságot vizsgálunk a kattintott és az üres pozícióra nézve.

Amennyiben 4 szomszédja az üresnek a kattintott, úgy elmozgatjuk azt az elemet. Ezt először csak a tömbben végezzük el, majd vizuálisan is megjelenítjük. Annak a vizsgálatát is elvégezzük, hogy sikerült-e minden elemet a megfelelő helyre rakni az elmozgatást követően. Amennyiben igen, akkor felugrik egy ablak a "You solved it!" felirattal 500 ms-t követően.

```
document.getElementById('puzzle').onclick = function(e) {

  clickLoc.x = Math.floor((e.pageX - this.offsetLeft) / tileSize);
  clickLoc.y = Math.floor((e.pageY - this.offsetTop) / tileSize);

  if (distance(clickLoc.x, clickLoc.y, emptyLoc.x, emptyLoc.y) == 1) {
    slideTile(emptyLoc, clickLoc);
    drawTiles();
  }
  if (solved) {
    setTimeout(function() {alert("You solved it!");}, 500);
  }
};
```

Kiszámítjuk az elemek távolságát a már ismert módon.

```
function distance(x1, y1, x2, y2) {
  return Math.abs(x1 - x2) + Math.abs(y1 - y2);
}
```

A csempék "elmozgatása" során megvizsgáljuk, hogy mely pozícióból hova szeretnénk léptetni. Amennyiben nem megoldott még a kirakó, úgy beállítjuk az új pozícióját a kattintott elemnek és az üres elemet is az új pozícióra helyezzük el. Ezt követően ellenőrizzük, hogy már minden elem jó helyen van-e.

```
function slideTile(toLoc, fromLoc) {
  if (!solved) {
    boardParts[toLoc.x][toLoc.y].x = boardParts[fromLoc.x][fromLoc.y].x;
    boardParts[toLoc.x][toLoc.y].y = boardParts[fromLoc.x][fromLoc.y].y;
    boardParts[fromLoc.x][fromLoc.y].x = tileCount - 1;
    boardParts[fromLoc.x][fromLoc.y].y = tileCount - 1;
    toLoc.x = fromLoc.x;
    toLoc.y = fromLoc.y;
    checkSolved();
  }
}
```

Egyszerűen végigmegyünk a tömbön és megnézzük, hogy sorrendben vannak-e az elemek egymás után. Alapvetően azt feltételezzük, hogy igen, erre utal a `true` értéket kapott `flag` változó, de amennyiben nem egymást követőek az elemek, ezt átállítjuk, majd végül átadjuk a `solved` értékének az így kapottat.

```
function checkSolved() {
  var flag = true;
  for (var i = 0; i < tileCount; ++i) {
    for (var j = 0; j < tileCount; ++j) {

      if (boardParts[i][j].x != i || boardParts[i][j].y != j) {
        flag = false;
      }
    }
  }
  solved = flag;
}
```

Almaszedő játék (jQuery)

Megoldás: megoldas.zip

Megoldás menete

Indítsuk el a PhpStorm-ot vagy Webstorm-ot és hozzunk létre egy új projektet, amiben legyen egy `index.html` is. A megoldás során minden képet fel fogunk használni, amit a megoldás csomagjában van. jQuery segítségével fogjuk megoldani a feladatot. A szkriptet egy külső állományban helyezzük el, melyet a weboldalhoz hozzá kell adni. Egy külön stílusfájl is adott ebben az esetben, melynek tartalmát most nem fogjuk részletesebben megvizsgálni, csak utalások lesznek.

Létrehozunk egy DIV-et a játéktérnek és azon belül elhelyezünk két feliratot és mellettük egy üres elemet, melybe az aktuális életet, valamint pontszámot fogjuk majd beszúrni.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Almaszedo</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.0/jquery.min.js"></script>
<script src="almaszedo.js"></script>
<link rel="stylesheet" href="almaszedo.css">
</head>
<body>
  <div id="gamearea">
  <p id="lives">Lives: <span></span></p>
  <p id="score">Score: <span></span></p>
  </div>
</body>
</html>
```

Az alábbiakban tekintsünk meg néhány definiált változót. A `NUM_COLS` azt fogja megadni, hogy hány oszlopban eshetnek le különböző gyümölcsök. Az itt megadott érték befolyásolni fogja az elemek méretét is és a kosár elmozdulását. A `lives` az életek, a `score` az elkapott gyümölcsök számát kapja majd értékül. Annak a befolyásolására, hogy milyen sűrűn jelenjenek meg új gyümölcsök, a `ticks` változót fogjuk használni. A későbbiekben majd az is kiderül, hogy hogyan.

A megoldás csomagban megtalálhatóak gyn.png (n=1..4) fájlok, melyek elérési útvonalát a `fruits` tömbben letároljuk azért, hogy majd random generálhassunk egy számot és az adott tömbindexen lévő gyümölcs képét jelenítsük meg, amikor egy új példányt helyezünk el a játéktérre.

```
var NUM_COLS = 5;
var lives = 10;

var ticks = 0;
var score = 0;

var fruits = [];
for (i=0; i<4; i++){
  fruits[i] = new Image();
  fruits[i].src = "gy"+(i+1)+".png";
}
```

A `gameArea` változónak átadjuk a játéktért reprezentáló DIV-et, melynek beállítjuk az ablak teljes széltét és hosszát méreteknek. Felhasználva akorábban megadott értéket, hogy hány oszlopban szeretnénk, ha megjelennének a gyümölcsök, kiszámítjuk a lépésközt is. Ennyivel fogjuk a kosarat mozgás közben arrébb léptetni.

```
var gameArea = $('#gamearea');
gameArea.width($(window).width());
gameArea.height($(window).height());
var stepWidth = gameArea.width() / NUM_COLS;
```

Ezt követően adjuk hozzá a kosarat a játéktérhez. Ahogyan már említettem korábban, a kosár szélessége az elmozdulás mértéke lesz, a magassága méretarányosan ehhez alakul. Beállítjuk az y pozícióját is.

```
var bowl = $('<img src="bowl.png">');
bowl.appendTo(gameArea).css({
  position: 'absolute',
  bottom: '0px',
  width: stepWidth
});
```

Az egér x irányú mozgását fogja lekövetni a kosár mozgása. Ehhez most nem a játéktér elemére definiáltuk az eseményfigyelőt, de ezt is megtehettük volna, hanem a `body`-ra, mégpedig azért, mert a kettő mérete ebben az esetben megegyezik. Emlékezzünk vissza, hogy az ablak méretét adtuk át a játéktér méretének. Ennek ellenére az offset-et szükséges lekérni és a pozícióba kalkulálni. Arra is figyelni kell, hogy ne hagyjuk el a kosárral a játékteret. Ezután már csak be kell állítani a pozícióját. Először kiszámítjuk, hogy hanyadik oszlopban lehet a kosár (`Math.round((x - stepWidth / 2) / stepWidth)`), a közepéhez viszonyítunk a kosárnak és megszorozzuk az elmozdulás mértékével.

```
$('body').on('mousemove', function(e){
  var p = gameArea.offset();
  var x = e.pageX - p.left;

  if (x<stepWidth/2 || x>gameArea.width()-stepWidth/2) {
    return;
  }
  bowl.css({
    left: Math.round((x - stepWidth / 2) / stepWidth)  * stepWidth
  })
});
```

A kiindulási életet és ponszámot szintén hozzáadjuk a játéktérhez.

```
$('#lives span').text(lives);
$('#score span').text(score);
```

Beállítunk egy időzítést a gyümölcsök születésére és magára a játékra. 50 ms-onként meghívodik a `timer` függvény, ami az egész játékmechanizmust magában foglalja.

```
setInterval(timer, 50);
```

Először tekintsük meg ezt a függvényt egészében, majd bontsuk részekre annak megértéséhez. Le fogjuk kezelni az életek alakulását, a gyümölcsök elkapását, a megjelenésük gyakoriságát, ...

```
function timer(){
  if (lives == 0){
    gameArea.find('.fruit').remove();
    return;
  }
  if (ticks % Math.max(3, 25-Math.floor(score/3)) == 0){
    makeFruit();
  }
  ticks++;

  gameArea.find('.fruit').each(function(){
    var tPos = $(this).position();
    var bPos = bowl.position();
    var dx = (tPos.left + $(this).width()/2) - (bPos.left + bowl.width()/2);
    var dy = (tPos.top + $(this).height()/2) - (bPos.top + bowl.height()/2);

    if (dy>0) {
      $(this).addClass('missed');
    }
    if (dy < 0 && dy > -stepWidth/2 && dx < stepWidth / 2 && dx > -stepWidth/2) {
      score++;
      $(this).stop().remove();
      $('#score span').text(score);
    }
  });
}
```

A `timer` függvény minden egyes meghívásakor növelni fogjuk a `ticks` értékét. Amennyiben ez egy másik számmal vett modulo értéke 0, akkor fogunk új gyümölcsöt létrehozni. Az, hogy ez miként definiáljuk az teljesen ránk van bízva. Jelen példában az elért pontszám is beleszámít a kalkulációba. Minél több pontot értünk el, annál nagyobb lesz a játéktéren lévő gyümölcsök számossága. Egyszerre egy fog megjelenni, de nagyobb gyakorisággal.

```
if (ticks % Math.max(3, 25-Math.floor(score/3)) == 0){
  makeFruit();
}
ticks++;
```

Tekintsük meg, hogy miként néz ki a gyümölcsök megjelenése, majd szedjük szét részeire.

```
function makeFruit(){
var pos = Math.floor(Math.random()*NUM_COLS);

var wrapperDiv = $('<div>');
wrapperDiv.addClass('fruit');
wrapperDiv.css({
  width: stepWidth,
  height: stepWidth,
  left: stepWidth * pos + "px"
});
var divImage = $('<div>');
divImage.addClass('imageholder');

makeRandomFruitBg(divImage);

divImage.appendTo(wrapperDiv);
wrapperDiv.appendTo(gameArea);
var animTime = 5000 - score*100;
if (animTime < 1000) {
  animTime = 1000;
}

wrapperDiv.animate({
  top: gameArea.height() - wrapperDiv.height()/2
  }, animTime, function(){
    $(this).remove();
    if (lives>0){
      lives--;
      $('#lives span').text(lives);
    }
  });
}
```

Először generálunk egy random számot arra, hogy hanyadik oszlopban jelenjen meg a gyümölcs. Létrehozunk egy külső DIV-et a gyümölcsnek, melynek beállítjuk az aktuális pozíciót is, valamint a méreteit. Hozzáadjuk a `fruit` osztályt is a DIV-hez.

```
var pos = Math.floor(Math.random()*NUM_COLS);
var wrapperDiv = $('<div>');
wrapperDiv.addClass('fruit');
wrapperDiv.css({
  width: stepWidth,
  height: stepWidth,
  left: stepWidth * pos + "px"
});
```

Az előző lépésben generált DIV-hez fogjuk hozzáadni azt a másik DIV-et, ami ténylegesen tartalmazza a gyümölcs képét, melynek lesz egy `imageholder` nevű osztálycímkéje. A gyümölcsök megjelenítéséhez generálunk egyet a listából. Majd a képet tartalmazó DIV-et hozzáadjuk a `wrapperDiv`-hez, azt pedig a játéktérhez.

```
var divImage = $('<div>');
divImage.addClass('imageholder');

makeRandomFruitBg(divImage);

divImage.appendTo(wrapperDiv);
wrapperDiv.appendTo(gameArea);
```

Korábban összegyűjtöttük egy tömbbe a lehetséges gyümölcsök útvonalát, képét. Generálunk egy random számot 0 és 4 között, majd ezt a képet vesszük ki a tömbből és jelenítjük meg. Természetesen, a megfelelő méreteket is beállítjuk.

```
function makeRandomFruitBg(divImage){
  var i = Math.floor(Math.random()*fruits.length);
  divImage.css({
    width: stepWidth,
    height: stepWidth,
    'background-size': stepWidth,
    'background-image': "url("+fruits[i].src+")"
  });
}
```

A gyümölcsök esését a már korábban megszokott módon fogjuk elvégezni az `animate()` függvény segítségével. A gyümölcsök pörgése egy CSS utasít eredménye, viszont a leesés sebessége attól fog függeni, hogy mennyi a pontszámunk. Hasonlóan, ahogyan az előbb már láthattuk. A leggyorsabb leesés 1000 ms alatt fog bekövetkezni, ami kezdetben 5000 ms lesz, míg nem szerzunk pontot. Ezek után már csak be kell állítani a gyümölcsök pozícióját, valamint a mozgatását. Amikor a gyümölcs fele elérte a maximális y pozíciót, akkor az eltűnik, valamint életet veszítünk, amennyiben még van. Az életek módosított számát is frissítjük a játéktéren.

```
var animTime = 5000 - score*100;
if (animTime < 1000) {
  animTime = 1000;
}
wrapperDiv.animate({
  top: gameArea.height() - wrapperDiv.height()/2
  }, animTime, function(){
    $(this).remove();
    if (lives>0){
    lives--;
    $('#lives span').text(lives);
  }
});
```

Amennyiben így az életeink száma 0-ra csökken, akkor az összes gyümölcsöt levesszük a játéktérről.

```
if (lives == 0){
  gameArea.find('.fruit').remove();
  return;
}
```

Azt is szükséges megvizsgálni a pontszerzés érdekében, hogy sikerül-e elkapni a kosárral a gyümölcsöt. Ehhez összegyűjtük az összes gyümölcsöt és lekérjük azok aktuális pozícióját, valamint a kosár koordinátáit is. Kiszámítjuk a két pont távolságát és amennyiben ez egy adott küszöbértéken belül van, növeljük a pontok számát, leállítjuk az animációt és levesszük a játéktérről. A pontszámot is frissítjük. Ha a gyümölcsöt nem sikerül elkapni, mert az y koordináták távolsága > 0, akkor hozzáadjuk az elemhez a `missed` class-t és ezért az elszürkül, mert ez a beállítás tartozik ehhez az osztályhoz a CSS fájlban.

```
gameArea.find('.fruit').each(function(){
  var tPos = $(this).position();
  var bPos = bowl.position();
  var dx = (tPos.left + $(this).width()/2) - (bPos.left + bowl.width()/2);
  var dy = (tPos.top + $(this).height()/2) - (bPos.top + bowl.height()/2);
  if (dy>0) {
    $(this).addClass('missed');
  }
  if (dy < 0 && dy > -stepWidth/2 && dx < stepWidth / 2 && dx > -stepWidth/2) {
    score++;
    $(this).stop().remove();
    $('#score span').text(score);
    }
  });
}
```