Multimédia nappali gyakorlat

jQuery animáció (időzítések), eseménykezelés

A tananyaghoz készült videó az alábbi linken érhető el: https://www.youtube.com/watch?v=sdgHH3AFkQA

Utoljára módosítva
2020. április 2.

Kulcsszavak
jQuery, append, parseInt, clone, on, css, setInterval, clearInterval, addClass, each, position, animate

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

Készítsük el az alábbi animációt!

Melléklet a feladathoz: kiindulasi_alap.zip
Megoldás: megoldas.zip

Megoldás menete

A kiindulasi_alap.zip csak a feladat megoldásához szükséges képeket és a jquery-3.4.1.min.js-t tartalmazza. Indítsuk el a PhpStorm-ot vagy Webstorm-ot és hozzunk létre egy új projektet. Ide másoljuk be a letöltött és kicsomagolt fájlokat. Ezt követően hozzunk létre egy HTML fájlt.

A feladatot jQuery használatával oldjuk meg, ezért szükségszerű a `jquery-3.4.1.min.js` hozzáadása a weboldalhoz. Adjunk hozzá egy üres JavaScript állományt is – ahova a megoldást fogjuk írni – és nevezzük el `script` néven.

Hozzunk létre a `<body>`-ban egy `<div>`-et, mely a játéktér lesz. Adjuk hozzá a gamearea ID-t. Ezzel fogunk majd hivatkozni a továbbiakban erre az elemre.

index.html:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Space invaders - jQuery</title>
  <script src="jquery-3.4.1.min.js"></script>
  <script src="script.js"></script>
</head>
<body>
  <div id="gamearea"></div>
</body>
</html>
```

Játéktérre vonatkozó beállítások

A játéktér létrehozását követően néhány stílusformázási beállítás szükséges. Ahogyan a `<canvas>` esetében is láthattuk, úgy az általunk definiált `<div>` esetében is megadjuk a játéktér méreteit. Ehhez a megoldás során index.html `<head>`-jében elhelyezzük a stílusformázási beállításokat.

A játéktér kialakításához néhány stílusformázási beállítás szükséges. Legyen a pályánk 600x600-as, melyhez melyhez a `width` és `height` tulajdonságoknak kell átadnunk a megfelelő értékeket, illetve rendelkezzen 2 pixel széles folytonos szürke szegéllyel is, mely könnyen megadható a `border` tulajdonság értékeinek megadásával. Ahhoz, hogy a megoldásban látható csillagos háttér legyen az animáció alatt, állítsuk be a `backgorund-image` értékét is, melyhez az `url` függvényben meg kell adni a kép relatív vagy abszolút elérési útvonalát.

```
width: 600px;
height: 600px;
border: 2px lightslategray solid;
background-image: url('bg.jpeg');
```

Szeretnénk, ha a játéktér a weboldal közepén helyezkedne el, ezért, ezért a margókat beállítjuk a következőképpen:

```
margin: 0 auto;
```

Mindemellett, annak érdekében, hogy megfelelően tudjuk majd pozícionálni a játéktéren elhelyezett objektumainkat, ezért a `position` értékét is beállítjuk.

```
position: relative;
```

Ez azt fogja jelenteni, hogy a HTML elem a normál helyzetéhez képest van elhelyezve. Később kitérünk erre, hogy ez pontosan mit jelent.

Tehát, a teljes beállítás a jatéktér esetében az alábbiképpen néz ki:

```
<style>
  #gamearea {
    width: 600px;
    height: 600px;
    border: 2px lightslategray solid;
    margin: 0 auto;
    position: relative;
    background-image: url('bg.jpeg');
  }
</style>
```

Tekintsük meg a további teendőket, mielőtt az animáció tényleges elkészítéséhez kezdenénk. A `script.js` állományban hozzunk létre egy változót a játéktérnek, illetve annak szélességének és magasságának tárolására.

```
// jatekterulet
let game_area;
// jatekterulet szelessege, hosszusaga
let ga_width, ga_height;
```

Az oldal betöltődését követően a `game_area` változónak értékül adjuk a kijelölt `gamearea` ID-vel ellátott HTML elemet és lekérjük a korábban létrehozott változóinkba a játéktér szélességét és magasságát. Ezeket az elemre jellemző értékeket a `css` függvényhívás segítségével érjük el. Az így kapott érték szöveges adattípusú, viszont ahhoz, hogy a későbbiekben fel tudjuk használni ezeket az értékeket, `int`-té kell alakítani. Erre szolgál a `parseInt` függvényhívás.

```
$(document).ready(function () {
  game_area = $('#gamearea');

  // a jatekter szelessegenek lekerdezese
  ga_width = parseInt(game_area.css('width'));
  // a jatekter magassaganak lekerdezese
  ga_height = parseInt(game_area.css('height'));
});
```

Az űrhajó mozgatása a kurzormozgató billentyűk segítségével

Miután a játékteret előkészítettük, áttérhetünk arra, hogy miként megjelenítsük az űrhajót is. Hozzunk létre változót az űrhajónak, az űrhajó szélességének, magasságának és az elmozdulásának megadására, mely majd a mozgatáskor kap fontos szerepet. Alapvetően azt szeretnénk, hogy változtatható legyen az elhelyezett ellenség száma, tehát dinamikusan számítsuk ki azoknak, illetve magának az űrhajónak is a szélességét, hogy egy síkba eshessen az ellenséggel. Ehhez létrehozzuk az alábbi változókat a megadott értékekkel:

```
let start_ey = 100;
// a megjelenitendo ellenseg veg y koordinataja
let end_ey = 500;
// megjelenitendo ellenseg szama
let enemy_num = 10;
// az ellenseg szelessege
let offset_x = (end_ey - start_ey) / enemy_num;
```

Ami számunkra jelenleg fontos lesz, az az `offset_x` értéke, mely aszerint fog alakulni, hogy mennyi ellenséget szeretnénk megjeleníteni és a pálya mekkora intervallumában. Ezek alapján tehát a változók deklarálása, definiálása:

```
// urhajo
let defender;
// az urhajo szelessege, magassaga
let def_width = offset_x, def_height;
// az urhajo elmozdulasanak merteke
let move_step = def_width / 2;
```

A játéktér css tulajdonságainak megadásánál már szó volt arról, hogy a `position` tulajdonság beállítása fontos abból a szempontból, hogy megfelelően tudjuk elhelyezni a további elemeinket az animációhoz. Állítsuk be az űrhajó `position` értékét `absolute`-ra, mely azt fogja eredményezni, hogy az elem az első nem statikus szülőelemhez viszonyítva kerül majd elhelyezésre. Ez esetben az szülőelem a `game_area` lesz.

```
#defender {
  position: absolute;
}
```

Ezt követően írjuk meg a függvényt az űrhajó inicializálására. Ehhez először adjuk meg a képet, melyet megjeleníteni szeretnénk, miután betöltődött az oldal.

```
defender = $('<img src="def.png" id="defender">');
```

Hozzunk létre egy `init_defender` függvényt, melyben először állítsuk be az űrhajó szélességét, melyet korábban definiáltunk már. A magasság beállítása nem szükséges, hiszen méretarányosan fog skálázódni a szélességhez képest, viszont az űrhajó pozíciójának beállításához szükségünk van rá, ezért a szélesség beállítását követően kérjük le a `css` függvényhívás segítségével a skálázódott magasságot a `def_height` változóba. Az űrhajó pozícionálásához a `top` tulajdonság értékét állítsuk be a játéktér és az űrhajó magasságának különbségére. Ezáltal az űrhajó a a számunkra megfelelő y koordinátába kerül.

```
function init_defender() {
  // az urhajo szelessegenek beallitasa
  defender.css({
    width: def_width,
  });

  // az urhajo magassaga
  def_height = parseInt(defender.css('height'));

  // az urhajo y poziciojanak beallitasa
  defender.css({
    top: ga_height - def_height,
  });
}
```

Az `on` függvényhívás segítségével nem csak arra van lehetőségünk, hogy azt monitorozzuk, hogy egy adott billentyűt lenyomtunk-e, hanem számos más esemény bekövetkeztét is megvizsgálhatjuk. Ilyen az is, hogy a elérhető-e már az adott tartalom, jelen esetben a megjeleníteni kívánt kép. Az `init_defender` meghívását ehhez az eseményhez fogjuk kötni, mivel a függvényben olyan értékeket kívánunk beállítani, melyek csak abban az esetben lesznek elérhetőek, ha már a kép is.

```
1  defender.on('load', function () {
2    init_defender();
3  });
```

Következő lépésben az űrhajó mozgatását tekintjük át, először kurzormozgató billentyűk segítségével. Ehhez létrehozunk egy `move_defender` függvényt, ami bemenő paraméterben egy bekövetkezett eseményt kap, jelen esetben egy billentyű lenyomásáho tartozó információkat. Annak vizsgálata, hogy milyen billentyű került lenyomásra hasonlóan fog történni, mint a `<canvas>` esetében. Ahhoz, hogy utasítást tudjunk rendelni egy-egy lenyomott gombhoz, el kell kérni, hogy melyik gombról van szó.

```
1  let pressed_key = ev.key;
```

A `key` a lenyomott billentyű azonosítója szövegesen. Miután megvan a lenyomott billentyű, akkor meg kell vizsgálni, hogy azok közül történt-e a lenyomás, melyek fontosak számunkra. Amennyiben igen, úgy szeretnénk arrébb léptetni az űrhajót balra vagy jobbra, attól függően, hogy melyik nyilat nyomtuk le. Mivel szeretnénk, ha csak a pályán belül tudna mozogni az űrhajó, ezért meg kell azt is vizsgál, hogy egyáltalán még továbbléphet-e az űrhajó az adott billentyű lenyomására. Az elmozdulás mértékét a `move_step` változónak már korábban átadtuk. Figyelni kell arra is, hogy nem biztos, hogy pont egy fél űrhajónyi hely van még a széleknélm, ezért ezekben az esetekben még léphetünk tovább, de csak a végpontokig. Tehát, az alábbiképpen néz ki a függvény:

```
1  function move_defender(ev) {
2    // lenyomott billentyu
3    let pressed_key = ev.key;
4    // ha a lenyomott billentyu a jobbra nyil
5
6    if (pressed_key === 'ArrowRight') {
7      // annak vizsgalata, hogy a jatekteren belul vagyunk-e meg
8      if (parseInt(defender.css('left')) + def_width < ga_width) {
9        defender.animate({
10         left: '+=' + move_step
11       }, 1)
12     } else {
13       defender.animate({
14         left: ga_width - def_width
15       }, 1)
16     }
17   // ha a lenyomott billentyu a balra nyil
18   } else if (pressed_key === 'ArrowLeft') {
19     // annak vizsgalata, hogy a jatekteren belul vagyunk-e meg
20     if (parseInt(defender.css('left')) - move_step > 0) {
21       defender.animate({
22         left: '-=' + move_step
23       }, 1)
24     } else {
25       defender.animate({
26         left: 0
27       }, 1)
28     }
29   }
30 }
```

Ahhoz, hogy monitorozni tudjuk az esemény bekövetkeztét, egy eseménykezelőt rendelünk az ablakhoz.

```
1  $(window).on('keydown', move_defender);
```

Az űrhajó mozgatása egér segítségével

Miután a kurzusmozgató nyilakkal tudjuk mozgatni az űrhajót, nézzük meg, hogy miként tehetjük meg ugyanezt az egérrel. Létrehozunk egy `mousemove_defender` neveztű függvényt, melynek egy bekövetkezett

A `<canvas>` példák esetében már láttuk, hogy az egér pozíciójának meghatározásához fontos volt azt is figyelembe venni, hogy pontosan hol helyezkedett el a HTML elem. Jelen esetben is fontos, hogy az általunk korábban a weboldalhoz hozzáadott `game_area` mekkora offset-tel rendelkezik.

```
let div_pos = game_area.offset();
```

Az egér aktuális x pozícióját a tartalomhoz képest a `clientX` értékeként kapjuk meg. Ebből le kell vonjuk a bal oldali offset-et és az űrhajó elmozdulását. Ebben az esetben is fontos, hogy megvizsgáljuk, hogy még a játéktéren belül vagyunk-e, ha elmozdulunk az űrhajóval jobbra vagy balra az egér pozíciójába x irányban.

```
function mousemove_defender(e) {
  let div_pos = game_area.offset();
  let mouse_pos_x = Math.ceil(e.clientX - div_pos.left - move_step);

  if (mouse_pos_x > 0 && mouse_pos_x < ga_width - def_width) {
    defender.css({
      left: mouse_pos_x,
    });
  }
}
```

Nyilvánvalóan az esemény bekövetkeztét ebben az esetben is figyelni kell, viszont most csak a `game_area` elemen fogjuk az egérmozgást kezelni és meghívni a `mousemove_defender` függvényt.

```
game_area.on('mousemove', mousemove_defender);
```

Az ellenség létrehozása, mozgatása a játéktéren

Az űrhajó megjelenítése és mozgatása mellett változtatható mennyiségű ellenséget is szeretnénk megjeleníteni, melyek adott időközönként közelítenek az űrhajó felé, de ha elérnek egy adott koordinátát, akkor utána már nem mozognak. Ehhez létrehozunk egy változót az ellenséget tartalmazó kép számára és egy tömböt, amibe az ellenségeket fogjuk tárolni különböző paraméterekkel.

```
// ellenseg
let enemy;
// megjelenitendo ellenseg szama
let enemy_num = 10;
// tomb az ellensegnek
let enemy_array = [];
```

Az ellenség `<img>` tagjének megadása.

```
enemy = $('<img src="enemy.png">');
```

Az űrhajónál már láthattuk, hogy akkor hívtuk meg a kirajzoló függvényt, amikor már elérhető volt. Ebben az esetben is így járunk el. Az ellenségre vonatkozó inicializáló függvény a `init_enemy` nevet viseli.

```
enemy.on('load', function () {
  init_enemy();
});
```

Korábban definiáltuk az ellenség számát és létrehoztunk egy tömböt is az ellenség tárolására. Töltsük fel a megadott számú ellenséggel, melyek y koordinátája megegyezik, az x pozíciójuk pedig legyen egymás mellett. Ezt úgy érhetjük el, hogy a kiindulási x koordinátához hozzáadjuk az ellenség szélességének és az aktuális ellenség számának szorzatát. Ahhoz, hogy több ugyanolyan képet is ki tudjunk rajzolni, szükséges több `<img>` tag-ben definiált elem. Ezt elérhetjük úgyis, hogy a korábban eltárolt képből másolatot készítünk a `clone` függvény segítségével. Itt most ezt is egy tulajdonságként fogjuk számon tartani a pozíciókkal együtt az egyes tömbbeli elemre.

```
function init_enemy() {
  // az osszes ellenseg poziciojat es a megjelenitendo kepet egy eltaroljuk az erre a celra korabban
       letrehozott tombben
  for (let i = 0; i < enemy_num; i++) {
    // az x koordinatat novelni kell minden ellenseg eseteben, hogy egymas mellett helyezkedjenek el
    enemy_array.push({
      x_pos: start_ex + i * offset_x,
      y_pos: start_ey,
      imgObj: enemy.clone()
    })
  }
  // az ellenseg tomb elemeinek kirajzolasa
  draw_enemy();
}
```

Látható, hogy a tömb feltöltése után meghívásra kerül a `draw_enemy` függvény. Az elemek megfelelő pozíción valü megjelenítése ebben a függvényben definiált. Ehhez a meglévő `enemy_array`-en végigmegyünk és az aktuális elem (x, y) pozícióját, valamint a megjelenítendő képet lekérjük. Ezt a képet kell hozzáadni a játéktérhez azért, hogy ténylegesen megjelenjen. Fontos, hogy beállítsuk a kép korábban definiált szélességét is, hogy minden elem megfelelően látszódjon. Ezenfelül hozzáadunk még egy `enemy` nevezetű class-t is az aktuális HTML elemhez, mely az ellenség mozgatásánál kap szerepet.

```
function draw_enemy() {
  // vegigmegyunk az ellenseg tomb minden elemen
  for (let e in enemy_array) {
    // vesszuk az aktualis elemet
    let act_enemy = enemy_array[e];
    // lekerjuk a megjelenitendo kepet
    let act_img = act_enemy.imgObj;
    // hozzaadjuk az ellenseg kepet a jatekterulethez
    game_area.append(act_img);
    // beallitjuk az aktualis ellenseg (x, y) koordinatajat es a kep szelesseget
    act_img.css({
      left: act_enemy.x_pos,
      top: act_enemy.y_pos,
      width: offset_x
    });
  // hozzaadjuk az enemy class-t
  act_img.addClass('enemy');
  }
}
```

A feladat része, hogy az ellenség folyamatosan közelítsen az űrhajó felé. Az időzítéshez ebben a megoldásban a `setInterval` függvényt használjuk, mely esetében meg kell mondani, hogy melyik függvényt akarjuk az általunk megadott időközönként (miliszekundumban kifejezve) meghívni, hogy újra és újra hajtódjon végre. Az időzítést majd szeretnénk felfüggeszteni, ha elérjük a 300-as y koordinátát az ellenség mozgása közben, ezért létrehozunk egy `mov_int` változót az időzítésnek.

```
mov_int = setInterval(moving_enemy, 1000);
```

A `moving_enemy` függvényben definiáljuk az ellenség mozgását. Ezt fogjuk 1000 miliszekundumonként meghívni. Azt szeretnénk, ha ezekben az időpillanatokban az összes ellenség 20 pixellel közelítene az űrhajóhoz, ezért begyűjtjük az `enemy` class-al ellátott HTML elemeket és végigiterálunk rajtuk. Egy HTML objektum pozíciója nem csak a `css` függvényhívással érhető el, hanem a `position` is visszaadja az elem elhelyezkedését. Ebben az esetben ezt fogjuk használni, mégpedig a `top` értékét. Ha elérjük a 300-as y koordinátát, akkor nem szeretnénk tovább lépkedni az ellenséggel, ezért a `clearInterval` függvényhívás paraméterében megadott `mov_int` időzítést tudjuk törölni.

```
function moving_enemy() {
  // az osszes ellenseg poziciojat megvaltoztatjuk egyenkent
  $('.enemy').each(function () {
    // ha az aktualis pozicio 300px alatt van, akkor noveljuk az ellenseg y koordinatajat 20px-el
    if ($(this).position().top < 300) {
      $(this).css({
        top: '+=20'
      })
    } else {
      // egyebkent toroljuk az idozitest
      clearInterval(mov_int)
```

```
    }
  });
}
```