Multimédia nappali gyakorlat

Javascipt és jQuery gyakorlás

A tananyaghoz készült videók az alábbi linkeken érhetőek el: https://www.youtube.com/watch?v=EYxN9NISH3A,

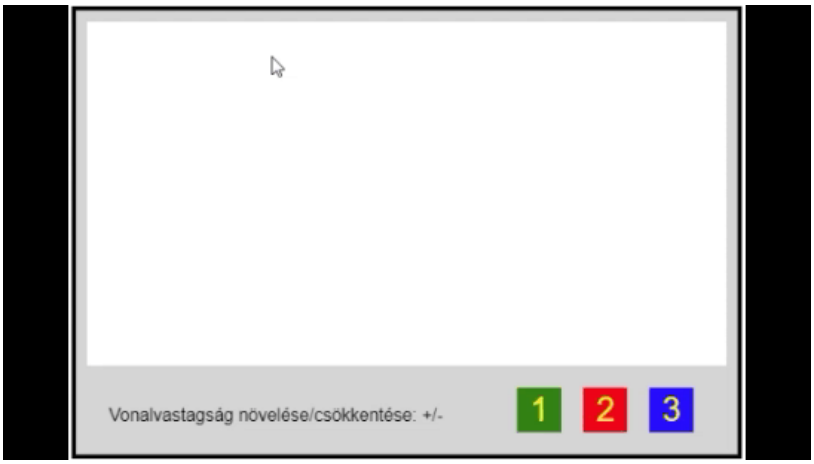
https://www.youtube.com/watch?v=xSbMlGf57fE

Utoljára módosítva
2020. április 13.

Kulcsszavak
Canvas, Javascript, jQuery, getElementById, getContext, addEventListener, getBoundingClientRect, moveTo, lineTo, beginPath, stroke, fillRect, fillStyle, lineWidth, font, fillText, clientX/Y, key, addClass, strokeStyle, appendTo, css, animate, this, each, removeClass, attr, on

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

<CANVAS> használatával készítsük el az alábbi egyszerű rajzolóprogramot!

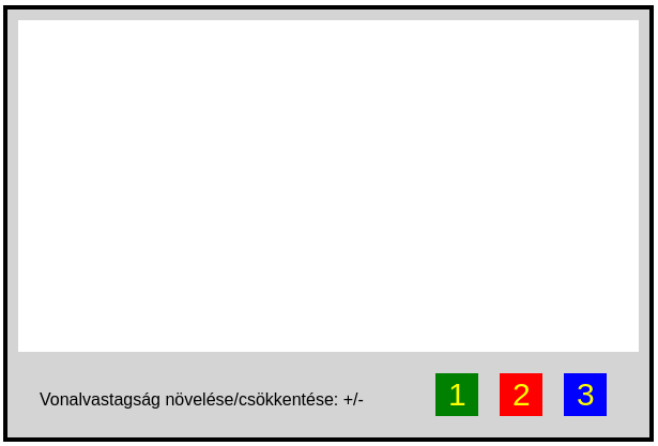


Melléklet a feladathoz: kiindulasi_alap.zip
Megoldás: megoldas.zip

Megoldás menete

A kiindulasi_alap.zip csak a feladat megoldásához szükséges kiindulási HTML állományt tartalmazza. Indítsuk el a PhpStorm-ot vagy Webstorm-ot és hozzunk létre egy új projektet. Ide másoljuk be a letöltött és kicsomagolt fájlt. Nyissuk meg a `canvas_paint.html`-t.

Tekintsük át, hogy mi az, ami már rendelkezésre áll és nem szükséges megírni. A kiindulási weboldal megnyitásakor az alábbiakat láthatjuk:



A rajzolóterület 400x600-as méretben definiált, továbbá adottak annak stílusformázási beállításai. Első lépésben jelöljük ki a `<canvas>` elemet annak érdekében, hogy tudjuk arra a későbbiekben rajzolni. Ezt a már korábban sokszor látott módon tehetjük meg:

```
let c = document.getElementById("ex");
let ctx = c.getContext("2d");
```

Maga a rajterületet alkotó elemek kirajzolása kitöltőszínnel és/vagy körvonallal az `init` függvényben került megvalósításra. Értelemszerűen, ahhoz, hogy mindez megjelenjen, szükséges a függvény meghívása is. Az itt definiált elemek a vizuális élményt hivatottak megalapozni a feladathoz.

```
init();

function init() {
  // rajzolomezo kirajzolasa kitoltoszinnel es korvonallal
  ctx.beginPath();
  ctx.fillStyle = "white";
  ctx.lineWidth = 4;
  ctx.fillRect(10, 10, 580, 310);
  ctx.stroke();

  // A valasztogombok kirajzolasa
  ctx.beginPath();
  ctx.fillStyle = "green";
  ctx.fillRect(400, 340, 40, 40);
  ctx.beginPath();
  ctx.fillStyle = 'red';
  ctx.fillRect(460, 340, 40, 40);
  ctx.beginPath();
  ctx.fillStyle = 'blue';
  ctx.fillRect(520, 340, 40, 40);

  // A valasztogombok magyarazo szovegenek kirajzolasa
  ctx.font = '16px Arial';
  ctx.fillStyle = "black";
  ctx.fillText('Vonalvastagsag novelese/csokkentese: +/-', 30, 370);

  // A valasztogombok feliratainak kirajzolasa
  ctx.font = '30px Arial';
  ctx.fillStyle = "yellow";
  ctx.fillText('1', 412, 370);
  ctx.fillText('2', 472, 370);
  ctx.fillText('3', 532, 370);

  // default vonalvastagsag beallitasa
  ctx.lineWidth = 1;
}
```

Ahhoz, hogy a fehér területre ténylegesen rajzolhassunk, a `<canvas>`-re egy eseményfigyelőt kell meghívni. Azért ezen HTML elemre, mert nem szeretnénk, ha a weboldal bármely területén az egérmozgás hatására rajzolhatnánk. Valójában ez nem is lenne lehetséges, mivel a rajzterület nem is definiált az egész weboldalra. Tehát, egérmozgás hatására hívjuk meg a `draw` függvényt, amit mindjárt definiálunk is. Hogy hogyan néz ki egy eseményfigyelés, már a korábbiakban megismertük.

```
c.addEventListener('mousemove', draw);
```

Ugyan egérmozgás bekövetkezte esetén akarjuk annak mozgását nyomonkövetni és rajzolni közben, de nem folyamatosan, ezért létrehozunk egy változót, mellyel mindezt befolyásolni fogjuk.

```
let isDraw = false;
```

Ahhoz, hogy rajzolni tudjunk, meg kell határoznunk az egér pontos pozícióját, amikor a canvas területén van. Ezt, a már ismert módon fogjuk megtenni. Amennyiben az `isDraw` értéke `true`, úgy lekérjük a `<canvas>` pozícióját is a `getBoundingClientRect` függvényhívás segítségével, mert az így kapott `top` és `left` értékek számítanak az egér pontos helymeghatározásánál a HTML elemen belül. Az egér (x, y) pozíciójának megadása a `clientX` és `clientY` értékekkel történik, melyekből a `<canvas>` offsetjeit ki kell vonni. Arra is figyelni kell, hogy csak akkor rajzoljunk, ha a fehér területen vagyunk az egérrel, ezért ennek vizsgálata is a függvény törzsét képezi. A `lineTo` paraméterében át kell adni az egér pozícióját, így, ha meghívjuk a kirajzoláshoz szükséges `stroke` függvényt, akkor az adott képpontra egy megadott színnel ellátott pontot rajzolhatunk ki. Abban az esetben, ha nem a fehér területen vagyunk az egérral, állítsuk az `isDraw` értékét `false`-ra.

```
function draw(e) {
  if (isDraw) {
    let rect = c.getBoundingClientRect();
    let mouseX = e.clientX - rect.left;
    let mouseY = e.clientY - rect.top;

    if (mouseX >= 10 && mouseX <= 590 && mouseY < 320 && mouseY > 10) {
      ctx.lineTo(mouseX, mouseY);
      ctx.stroke();
    } else {
      isDraw = false;
    }
  }
}
```

Azért, hogy ne rajzoljunk folyamatosan, amint a fehér területre mozgunk az egérrel, ezért az `isDraw` értékét csak akkor változtatjuk meg `true`-ra, amikor lenyomjuk az egér bal egérgombot. Ehhez egy újabb `addEventListener`-t kapcsolunk a `<canvas>`-re. A figyelt esemény a `mousedown` lesz. Újra szükséges az egér aktuális pozíciójának elérése, hogy a `moveTo` függvény paraméterében átadhassuk a rajzolandó objektum kiindulási pontját. A cél az, hogy különböző alakzatokat is kirajzolhassunk, ezért meghívjuk a `beginPath` függvény is.

```
c.addEventListener('mousedown', function (e) {
  isDraw = true;
  let rect = c.getBoundingClientRect();
  let mouseX = e.clientX - rect.left;
  let mouseY = e.clientY - rect.top;

  ctx.moveTo(mouseX, mouseY);
  ctx.beginPath();
});
```

Addig fog tartani egy-egy rajzolás, amíg el nem engedjük a bal egérgombot, tehát szükséges ennek a vizsgálata. Erre is beállítunk egy eseményfigyelőt. Mivel ezután nem akarunk tovább rajzolni, így az `isDraw` értékét `false`-ra állítjuk át.

```
c.addEventListener('mouseup', function (e) {
  isDraw = false;
});
```

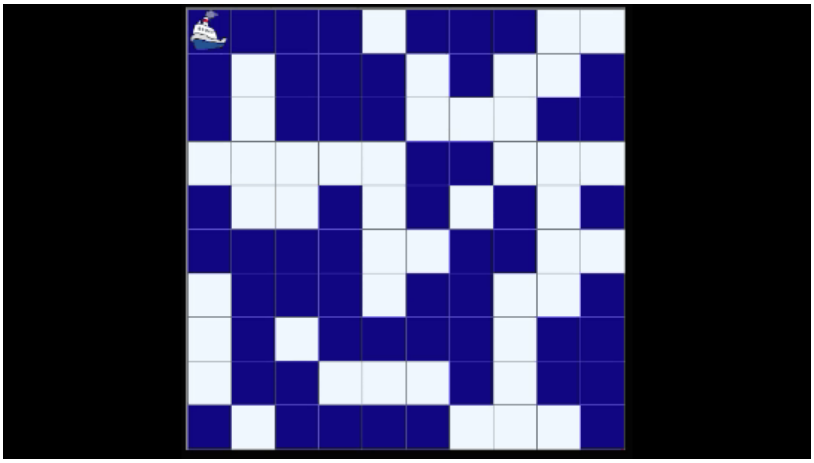
Azonfelül hogy már lehet rajzolni a fehér területre, a már előre megrajzolt elemeken szereplő számokhoz, valamint a `+, -` jelekhez is funkcionalitást kell rendelni. Ezek a rajtuk megadott billentyűk lenyomására történnek meg. Ebben az esetben az ablakra hívjuk meg az `addEventListener`-t. Definiálunk egy `changeColor` függvényt, melyben megvizsgáljuk a lenyomott billentyűt és beállítjuk a megfelelő színt, vonalvastagságot a rajzoláshoz.

```
window.addEventListener("keydown", changeColor);
```

Az esemény bekövetkeztekor szükséges a lenyomott billentyű vizsgálata. Erre már korábban számos példát láthattunk, így nem részletezzük.

```
function changeColor(e) {
  var key = e.key;
  if (key === '1') {
    ctx.strokeStyle = "green";
  } else if (key === '2') {
    ctx.strokeStyle = "red";
  } else if (key === '3') {
    ctx.strokeStyle = "blue";
  } else if (key === '+') {
    ctx.lineWidth += 1;
  } else if (key === '-') {
    ctx.lineWidth -= 1;
  }
}
```

Készítsük el az alábbi animációt jQuery használatával!



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 és HTML-t tartalmazza. Indítsuk el a PhpStorm-ot vagy Webstorm-ot és hozzunk létre egy új projektet. Ide másoljuk be a letöltött és kicsomagolt fájlokat. Először tekintsük meg a HTML fájlt. A feladatmegoldáshoz használni fogjuk a jQuery függvénykönyvtárat is, ezért szükséges hozzáadni a weboldalhoz. Egy külön Javascript fájlban találhatóak a már előre megírt függvények, valamint egy külső állományban a stílusformázási beállítások.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Icebreaker</title>
    <script src="../../../jquery-3.4.1.min.js"></script>
    <script src="script_full.js"></script>
    <link rel="stylesheet" href="style.css"/>
</head>
<body>
</body>
</html>
```

A HTML forrást megtekintve egy fekete hátterű weboldalt láthatunk.

```
body {
    margin: 0;
    padding: 0;
    background: black;
}
```

A játéktér 500x500-as méretű és középre igazított. Ehhez a `top` és a `left` értékét 50% állítjuk, így a játéktér bal felső sarka pont középen lenne, de még ebből levonjuk a szélesség és magasság felét, így a középpont már az oldal közepén lesz. 2 pixel széles szürke szegélyt is beállítunk.

```
#gamearea {
    position: absolute;
    top: 50%;
    left: 50%;
    margin-left: -250px;
    margin-top: -250px;
    width: 500px;
    height: 500px;
    border: solid gray 2px;
}
```

A pálya négyzetekből épül fel ls kétféle class-t vehetnek fel. Az egyik a `water`, mely esetében beállításra kerül a szín, elhelyezkedés, szegély. A `z-index`-szel azt állítjuk be, hogy egymáson lévő elemek közül melyik hol legyen. Minél nagyobb ez a szám, annál előrébb van az elem. Csak akkor működik az elemek megfelelő megjelenítése, ha az elem pozícionált. `Absolute` pozíció esetén kikerül a megjelenés folyamából, megadhatunk neki különféle elhelyezkedési értékeket, méretet.

```
.water {
    background: navy;
    position: absolute;
    z-index: 0;
    border: solid #666 1px;
}
```

Az `ice` class-t viselő elemek esetében csak a szín kerül beállításra, illetőleg majd látható lesz, hogy valójában felüldefiniálásra, mégpedig a `water` class-ban megadott érték tekintetében.

```
.ice {
    background: aliceblue;
}
```

A játékterületen egy hajó fog billentyűlenyomásra mozogni és ezt folyamatosan látni szeretnénk, ezért a `z-index` értékét 1-re állítjuk.

```
#ship {
    z-index: 1;
    position: absolute;
}
```

Tekintsük át a már előre definiált, deklarált változókat. A tábla mérete dinamikusan meghatározható, de minden esetben megegyezik a szélessége és magassága. A méretét az `N` változóval adhatjuk meg, tehát NxN-es táblákat hozhatunk létre. A `blockSize` változóban tároljuk az elem méretét. A hajó pozíciója a (0, 0)-ból fog kiindulni, és a mindenkori aktuális pozícióját a `shipPos` változóban érjük el. A játéktér tárolására a `gameArea`, míg a hajó tárolására a `ship` változókat definiáljuk. A hajót a kurzormozgató billentyűk segítségével fogjuk a táblán mozgatni, ezért ezeket azonosítóját egy-egy változóban szintén letároljuk.

```
let N = 10;
let blockSize = 500 / N;
let shipPos = {x: 0, y: 0};
let gameArea;
let ship;

//Nyilbillentyuk
let KEYLEFT = 'ArrowLeft';
let KEYUP = 'ArrowUp';
let KEYRIGHT = 'ArrowRight';
let KEYDOWN = 'ArrowDown';
```

Ahogyan fentebb említetésre került, a játéktér "parkettákból" épül fel. Ezek kirajzolására a `randomizeIce` függvény szolgál. Alapvetően minden négyzet felveszi a `water` classt és 50% eséllyel az `ice` classba is tartozik. For ciklussal végigmegyünk az NxN-es táblán, létrehozunk egy új `<div>`-et az egyes elemekhez és hozzáadjuk a `water` osztályt. A `Math` objectbeli `random` függvény segítségével generálunk egy véletlen számot [0, 1] intervallumban és amennyiben az 0.5 feletti értéket ad, akkor hozzáadjuk az aktuális elemhez az `ice` class-t is. Beállítjuk a CSS tulajdonságokat, minden egyes mező elhelyezkedése abszolút a játéktérhez képest és `blockSize` méretű lesz. Fontos, hogy megfelelően pozícionáljuk is a táblán a létrehozott HTML elemet. Végül, de nem elhanyagolható lépésként hozzáfűzzük a játéktérhez.

```
function randomizeIce() {
  for (let i = 0; i < N; i++) {
    for (let j = 0; j < N; j++) {
      let block = $('<div></div>');
      block.addClass('water');

      if (Math.random() > 0.5) {
        block.addClass('ice');
      }

      block.css({
        width: blockSize,
        height: blockSize,
        top: i * blockSize,
        left: j * blockSize
      });

      block.appendTo(gameArea);
    }
  }
}
```

A hajó létrehozására és kirajzolására létrehozzunk az `addShip` függvényt. A korábban deklarált `ship` változónak átadunk egy HTML image objektumot, beállítjuk a méretét, mely megfelel a mező méretének és hozzáfűzzük a játéktérhez.

```
function addShip() {
  ship = $('<img src="icebreaker.png" id="ship" />');
  ship.css({
    width: blockSize,
    height: blockSize,
  });

  ship.appendTo(gameArea);
}
```

A nyílbillentyűk segítségével mozog a hajó a 4 irányba. Ehhez létrehozunk egy eseménykezelőt, mely a billentyű lenyomásakor zajlik le. Az input paraméter egy esemény lesz. Lekérjük az aktuálisan lenyomott billentyű nevét és amennyiben az megegyezik valamelyik kurzormozgató billentyűével, akkor növeljük/csökkentjük a `shipPos` értékét. Miután beállítottuk a hajó új koordinátáját szükséges ellenőrizni, hogy az még a tartományon belülre mutat-e. Ha nem, akkor felveszi az előző pozícióját, nincs mozgás, míg ellenkező esetben animációval az új pozícióra kerül.

```
function moveShip(e) {
  let key = e.key;

  switch (key) {
    case KEYDOWN:
      shipPos.y++;
      break;
    case KEYUP:
      shipPos.y--;
      break;
    case KEYRIGHT:
      shipPos.x++;
      break;
    case KEYLEFT:
      shipPos.x--;
      break;
}

  // Tartomanyok ellenorzese
  if (shipPos.x < 0) {
    shipPos.x = 0;
  } else if (shipPos.x > N - 1) {
    shipPos.x = N - 1;
  } else if (shipPos.y < 0) {
    shipPos.y = 0;
  } else if (shipPos.y > N - 1) {
    shipPos.y = N - 1;
  } else {
```

```
    animateShip();
  }
}
```

A `moveShip` függvényben láthattuk, hogy a hajó tényleges mozgásáért a `animateShip` a felelős. Alapvetően az `animate` függvényt hívjuk meg, hogy animálja a hajót és módosítsa a pozícióját 1 ms alatt. A callback függvényben (az animáció végén hívandó) megvizsgáljuk, hogy a hajó `ice` class-al ellátott mezőre lép, akkor az osztály el kell távolítani.

```
function animateShip() {
  ship.animate({
    top: shipPos.y * blockSize,
    left: shipPos.x * blockSize
  }, 1, function () {
    // alternativ valtozat:
    // $('.ice').each(function(){
    gameArea.find('.ice').each(function () {
      if ($(this).css('top') === ship.css('top') && $(this).css('left') === ship.css('left')) {
        $(this).removeClass('ice');
      }
    });
  });
}
```

Amint az oldal betöltődött, akkor a `gameArea`-t elhelyezzük a `body`-ba az `appendTo` segítségével, majd kirajzoljuk a hajót, a mezőket és definiáljuk az ablakra az eseménykezelőt a billentyűlenyomás figyelésére.

```
$(function () {
  gameArea = $('<div></div>');
  gameArea.appendTo('body');
  gameArea.attr('id', 'gamearea');

  addShip();
  randomizeIce();
  $(window).on('keydown', moveShip);
});
```