JavaScript Canvas

1 Vizuális megjelenítés

Az erőgyűjtés harmadik fordulójában arról lesz szó, hogy miként lehet vizuális megjelenítéseket, grafikát helyezni el egy weboldalon.

2 Canvas kezelése

A feladat megoldásághoz szükség lesz grafikai megjelenítésre, amire jó opciót ad a HTML-ben található Canvas. Linkek:
Link
Link

```
<canvas id="myCanvas" width="300" height="300" style="border:1px solid
#d3d3d3;">
Your browser does not support the HTML5 canvas tag.</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.strokeStyle = "#0000FF";
ctx.moveTo(50,50);
ctx.lineTo(200,50);
ctx.lineTo(200,100);
ctx.lineTo(50,100);
ctx.lineTo(50,50);
ctx.stroke();
ctx.fillStyle = "#FF0000";
ctx.beginPath();
ctx.arc(95, 50, 40, 0, 2 * Math.PI);
ctx.fill();
</script>
```

Próbáld ki!

A Canvas használatához az szükséges, hogy az oldalon legyen egy ilyen típusú elem, amit JavaScript segítségével el tudunk érni, majd hozzá kell férni ezen az elemen belül a rajzolási környezethez (context). A rajzolást magát azt innentől vonalak és ívek rajzolásával lehet megoldani. Fontos, hogy a canvas számára a 0,0 pozíció a bal felső sarokban van, az x koordináta jobbra, az y pedig lefelé növekszik. Egyéb fontos információ, hogy a Canvas a szövegeket radiánban méri, ahol a 0 fok jobbra van, a 90 fok (radiánban PI/2) pedig lefelé: Canvas arc().
Az ismételt rajzolást úgy oldhatjuk meg, ha beállítunk egy ismétlődő időzítőt, ami bizonyos időközönként meghívja a rajzoló függvényt. Ilyen esetben célszerű a canvas tartalmát a rajzolás előtt törölni is (lásd a lentebbi példában). Amennyiben sok objektum kirajzolására van szükség, célszerű azokat osztályokba szervezve eltárolni. Ilyen módon minden kirajzolás előtt módosítható a pozíciójuk is. Ezen felül az eseményekre a hagyományos módon reagálhatunk, ezzel befolyásolva a megjelenítést.
Vegyük például a lenti kódot, amely megjelenít 4 mozgó, pattogó labdát. Minden labdát egy külön objektumban tárol, amelyben összegyűjti azok adatait. A kirajzolás egy időzítőhöz kötve

meghívódik újra és újra, minden alkalommal törli a vásznat, és kirajzolja a labdákat az új helyzetükben. A labdák helyzetének frissítése külön függvénybe szedve található. Itt a megfelelő képletek megoldják a megfelelő mozgást, és a visszapattanást a szélekről. Ebben sok a matek, de akit érdekel, belenézhet részletesen. Az oldalon ezen felül van két gomb, ami szemlélteti, hogy a kirajzolás mellett az egyéb események is működnek.

```
<script>
	class Ball {
		constructor(posx, posy, color, direction, speed) {
			this.posx=posx;
			this.posy=posy;
			this.color=color;
			this.direction=direction;
			this.speed=speed;
			this.size=20;
		}
	}

	function toRadian(angle){
		return angle*Math.PI/180;
	}

	var ballObjects = [new Ball(50,40,"#ffcccc",toRadian(30),3.5),
		new Ball(150,80,"#aa00cc",toRadian(83),4.3),
		new Ball(250,340,"#005577",toRadian(-130),1.9),
		new Ball(450,140,"#aaffbb",toRadian(12),3)];

	var speedmod=0;

	function speedup(){
		if (speedmod<6){
			for (var index=0; index<ballObjects.length; index++){
				ballObjects[index].speed+=0.3;
			}
			speedmod++;
		}
	}

	function slowdown(){
		if (speedmod>-6){
			for (var index=0; index<ballObjects.length; index++){
				ballObjects[index].speed-=0.3;
			}
			speedmod--;
		}
	}

	function moveBall(ballObj, canvasWidth, canvasHeight){
		// Ball boundaries
		var xMinimum=ballObj.size;
		var xMaximum=canvasWidth-ballObj.size;
		var yMinimum=ballObj.size;
		var yMaximum=canvasHeight-ballObj.size;
		// Move the ball
		var xChange=ballObj.speed*Math.cos(ballObj.direction);
		var yChange=ballObj.speed*Math.sin(ballObj.direction);
		ballObj.posx+=xChange;
		ballObj.posy+=yChange;
		// Bounce back from the sides
		if (ballObj.posx<xMinimum)
```

```
            {
                  ballObj.posx=xMinimum+(xMinimum-ballObj.posx);
                  if (ballObj.direction>0) ballObj.direction=Math.PI-
ballObj.direction;
                  else ballObj.direction=-Math.PI-ballObj.direction;
            }
            else if (ballObj.posx>xMaximum)
            {
                  ballObj.posx=xMaximum-(ballObj.posx-xMaximum);
                  if (ballObj.direction>0) ballObj.direction=Math.PI-
ballObj.direction;
                  else ballObj.direction=-Math.PI-ballObj.direction;
            }
            if (ballObj.posy<yMinimum)
            {
                  ballObj.posy=yMinimum+(yMinimum-ballObj.posy);
                  ballObj.direction=-ballObj.direction;
            }
            else if (ballObj.posy>yMaximum)
            {
                  ballObj.posy=yMaximum-(ballObj.posy-yMaximum);
                  ballObj.direction=-ballObj.direction;
            }
      }

      function updateObjects(){
            var c = document.getElementById("myCanvas");
            var ctx = c.getContext("2d");
            var canvasHeight=c.offsetHeight;
            var canvasWidth=c.offsetWidth;
            for (var index=0; index<ballObjects.length; index++){
                  moveBall(ballObjects[index], canvasWidth, canvasHeight);
            }
      }

      function refreshCanvas() {
            var c = document.getElementById("myCanvas");
            var ctx = c.getContext("2d");
            var canvasHeight=c.offsetHeight;
            var canvasWidth=c.offsetWidth;
            ctx.clearRect(0, 0, canvasHeight, canvasWidth);

            updateObjects();

            for (var index=0; index<ballObjects.length; index++){
                  ctx.beginPath();
                  ctx.arc(ballObjects[index].posx, ballObjects[index].posy,
ballObjects[index].size, 0, 2 * Math.PI);
                  ctx.fillStyle = ballObjects[index].color;
                  ctx.fill();
            }
      }
</script>
<body>
      <canvas id="myCanvas" width="500" height="500" style="border:1px
solid #000000;">
      </canvas>
      <button type="button" onclick="speedup()">Speed up!</button>
      <button type="button" onclick="slowdown()">Slow down!</button>
</body>
<script>
```

```
      var myVar = setInterval(refreshCanvas, 30);
</script>
```

Próbáld ki!

Még egy érdekes dolgot tárgyalhatunk ki a Canvas kapcsán + hogy kezeljük az egérkattintásokat? Erre a válasz egyszerű: kell hozzá rendelni egy eseménykezelő függvényt, ami figyel a kattintás eseményre:

```
document.getElementById("myCanvas").addEventListener('click', clickEvent,
false);
```

A fenti példa konkrétan a **click** eseményre figyel, de meg lehet különböztetni direkt az egérgomb lenyomását és felengedését (**mousedown** és **mouseup**), valamint az egérmozgatását is (**mousemove**).
A következő lépés megtudni, hogy hol volt az egér a kattintás pillanatában. Nos, az egér pozíciót globális pozícióként adja meg az esemény (hol van az ablakban). Ha azt akarjuk megtudni, hogy ez a vásznon milyen pozíciót jelent, akkor át kell számolnunk:

```
var c = document.getElementById("myCanvas");
var cavasRect = c.getBoundingClientRect();
var xPos = event.clientX - cavasRect.left;
var yPos = event.clientY - cavasRect.top;
```

Ha ez megvan, akkor már azt kezdünk a kattintás tényével, amit szeretnénk.

Próbáld ki!

3 Osonó hacker part 2: az információ gyűjtés

Az előző forduló során a hacker sikeresen bejutott a vállalat szervertermébe, és rácsatlakozott az egyik gépre. Most következik az adatszerzés. Valamennyi ideje volt felkészülni, így ismer néhány támpontot arra vonatkozóan, hogy milyen módon tudja az adatokat megtalálni, de nincs egyszerű dolga. Sajnos az idő szűkös, és így nincs lehetősége minden adatot lemásolni, szelektálni kell. Viszont a szervereken lévő adatok egy része hasztalan számára.
Mint kiderült a vállalat rendszergazdái elég kaotikusan végezték a dolgukat, ezért az adataik nincsenek megfelelően rendszerezve. A hackernek véletlenszerűen kell keresgélni, remélve, hogy minél több értelmes adatra talál. Összedobott egy kicsi segédprogramot, ami segít neki az adatok elemzésében. Idő híján nem sikerült tökéletesen, de segít megkülönböztetni a fontos adatot a hasztalantól.
A mostani forduló során egy minigame elkészítése a feladat, amely során a hacker bőrébe bújva kutathatunk a hasznos adatok után. A játék egyfajta gyorsasági + mintafelismerős keverék lesz. Az oldalon megjelennek majd valamilyen formában a véletlenszerűen megtalált adatok. Pontosabban azok kódolt fajtái, hiszen a hacker segédprogramja a talált adatokból kódokat gyárt, amikről egyszerűbb felismerni, hogy melyik hasznos. A játékos feladat az lesz, hogy a megjelenő értelmes adatokra időben rákattintson, mielőtt a segédprogram tovább halad rajtuk.

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, ennek megfelelően kell működniük.

Beküldhető feladatok:

1) Készítsétek el a fentebb leírt játékot a következő módon:
   a) Definiáljátok az adatok kinézetét. Fontosabb kérdések és szempontok:
      i) Hogy fog kinézni egy kódolt adat? Lehet pl. számsor, vagy egyszerűbb alakzatok
      ii) Mi lesz a szabály arra, hogy mi jelöl hasznos adatot? Szám/szöveg adatként lehet benne valami rész szöveget keresni. Kis ábrák esetén meg lehet adni, hogy melyik jó. A lényeg, hogy minden megjelenő adatról egyértelműen el lehessen dönteni szemmel is 1-2 másodperc alatt, hogy megfelelő-e.
      iii) Az oldalon mindenképpen legyen egy szabályismertető, ami leírja a megjelenő adatok ismertetését, és elmondja, hogy mi számít megfelelőnek.
      iv) A tényleges, szerveren lévő adat csak képzelt, és nem kell foglalkozni semmilyen tényleges kódolással. Elég a „kódolt” formákkal dolgozni, és elképzelni, hogy jönnek valahonnan.
   b) Az hacker programja által talált és kódolt adatok véletlenszerűen jelenjenek meg az oldalon, célszerűen Canvas használatával.
      i) Az adat pozíciója legyen véletlenszerű, de akár randomizálható egy forgatás is rá.
      ii) Véletlen időpontokban jelenjenek meg az alakzatok, és egy adott idő után tűnjenek el (ez is lehet véletlen).
      iii) Célszerű arra figyelni, hogy egymáshoz túl közel ne jelenjenek meg.
   c) A játékos tudjon rákattintani a megjelenő adatokra, ezzel összegyűjtve őket.
   d) A program számoljon pontszámot. Természetesen csak a hasznos adat ér pontot, a feleslegesen begyűjtött adatokért járjon pontlevonás.
   e) A játék egy adott ideig tartson, majd érjen véget és közölje az eredményt.
   f) Legyen a játékhoz 2-4 képesség, amit a játékos egyszer el tud használni. 1-2 ötlet, de nyugodtan lehet sajátot kitalálni (legyenek benne a leírásban): időlassítást, hogy lassabban tűnjenek el a kódok; hitstreak, ami plusz időt ad, amíg elég gyorsan hiba nélkül játszunk; valami, ami segít megkülönböztetni a jó és rossz kódokat.
   g) Legyenek a játékban nehézségi fokozatok, amik alapján változnak a paraméterek. Például más az időzítés, a különböző kódok száma, a képességek száma. Lehet hardcore mód is, ahol az első hibázás a játék végét jeleni. Stb…

1100 0000$|_2$ pont