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, illetve a billentyű lenyomásokat? 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**). Az események kezelésére megadott függvényeknek mindig van egy fontos paramétere, ami az esemény adatait tárolja. Innen lehet lekérdezni a lenyomott billentyűket, valamint az egér pozícióját is.
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.
A billentyű lenyomás hasonló, csak azt célszerű a teljes ablakra létrehozni, hogy biztosan működjön:

```
window.addEventListener("keydown", keyDownListener);
window.addEventListener("keyup", keyUpListener);
```

Azt, hogy melyik billentyűt nyomtuk le, annak kódjával tudjuk ellenőrizni. Például a betű billentyűk kódja a betű nagy verziójának ascii kódjával egyezik meg: ’a’: 65, ’b’: 66, stb. Példa gyanánt megtekinthető az alábbi program, amely egy egyszerűen megvalósított, wasd gombokkal történő mozgást demonstrál (nyilván ez nem a legjobb kivitelezés, de a billentyű kezelés szemléltetésére megfelelő):

```
<script>
	class PlayerType{
		constructor(posx, posy){
			this.posx=posx;
			this.posy=posy;
			this.size=20;
			this.color="#888888";
			this.speed=3;
			this.up=false;
			this.down=false;
			this.right=false;
			this.left=false;
		}
	}

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

```
}

var player = new PlayerType(200,200);

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

	if (player.up && player.right){
		player.posx+=player.speed*1.41;
		player.posy-=player.speed*1.41;
	}
	else if (player.up && player.left){
		player.posx-=player.speed*1.41;
		player.posy-=player.speed*1.41;
	}
	else if (player.down && player.right){
		player.posx+=player.speed*1.41;
		player.posy+=player.speed*1.41;
	}
	else if (player.down && player.left){
		player.posx-=player.speed*1.41;
		player.posy+=player.speed*1.41;
	}
	else if (player.up){
		player.posy-=player.speed*2.00;
	}
	else if (player.down){
		player.posy+=player.speed*2.00;
	}
	else if (player.left){
		player.posx-=player.speed*2.00;
	}
	else if (player.right){
		player.posx+=player.speed*2.00;
	}
}


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);

	updatePosition();

	ctx.beginPath();
	ctx.arc(player.posx, player.posy, player.size, 0, 2 * Math.PI);
	ctx.fillStyle = player.color;
	ctx.fill();
}

function keyDownListener(event){
	if (event.keyCode==65) // a
	{
		player.left=true;
		player.right=false;
```

```
            }
            else if (event.keyCode==68) // d
            {
                  player.right=true;
                  player.left=false;
            }
            else if (event.keyCode==87) // w
            {
                  player.up=true;
                  player.down=false;
            }
            else if (event.keyCode==83) // s
            {
                  player.down=true;
                  player.up=false;
            }
      }

      function keyUpListener(event){
            if (event.keyCode==65) // a
                  player.left=false;
            else if (event.keyCode==68) // d
                  player.right=false;
            else if (event.keyCode==87) // w
                  player.up=false;
            else if (event.keyCode==83) // s
                  player.down=false;
      }
</script>
<body>
      <canvas id="myCanvas" width="500" height="500" style="border:1px
solid #000000;">
      </canvas>
</body>
<script>
      var myVar = setInterval(refreshCanvas, 30);
      window.addEventListener("keydown", keyDownListener);
      window.addEventListener("keyup", keyUpListener);
</script>
```

A Canvas lehetőséget ad képek kirajzolására is. Ehhez létre kell hoznunk egy Image objektumot, és meg kell adni, hogy melyik fájl tartalmazza magát a képet, amit be szeretnénk tölteni. Ha meg kell várni a betöltés végét, akkor az onload eseményre reagálva tudunk a betöltés utáni műveletet definiálni. A képeknek lehetnek áttetsző részei is. Az alábbi példában egy „smiley.png” fájlból betöltünk egy képet, és a betöltés végén beállítjuk, hogy a program működése elindulhat. A kép ettől még nem jelenik meg sehol, ahhoz meg kell majd hívni a drawImage függvényt. Láthatjuk, hogy az onload megadása előbb van, mint a fájlnév megadása.

```
var started = false;
smileyTemplate = new Image();
smileyTemplate.onload = function(){
      started = true;
};
smileyTemplate.src = 'smiley.png';
```

Lentebb található a teljes kód, amely betölti a smiley-t, mindig kirajzol egy véletlenszerű méretűt, ami követi az egeret. Az egérrel kattintva ez fix helyet kap a pályán és egy új, véletlenszerű méretű smiley keletkezik, ami majd ezután követi az egeret.

```
<script>
	class Smiley {
		constructor(posx, posy,  size) {
			this.posx=posx;
			this.posy=posy;
			this.size=size;
		}
	}

	var placedSmileys = [];

	cursorSmiley = null;
	var started = false;

	smileyTemplate = new Image();
	smileyTemplate.onload = function(){
		cursorSmiley = new Smiley(0, 0,
20+Math.floor(Math.random()*30));
		started = true;
	};
	smileyTemplate.src = 'smiley.png';

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

		for (var index=0; index<placedSmileys.length; index++){
			var current = placedSmileys[index];
			ctx.drawImage(smileyTemplate, current.posx-
current.size/2, current.posy-current.size/2, current.size, current.size);
		}

		if (cursorSmiley !== null){
			ctx.drawImage(smileyTemplate, cursorSmiley.posx-
cursorSmiley.size/2, cursorSmiley.posy-cursorSmiley.size/2,
cursorSmiley.size, cursorSmiley.size);
		}
	}

	function clickEvent(event){
		if (!started) return;
		var c = document.getElementById("myCanvas");
		var cavasRect = c.getBoundingClientRect();
		var xPos = event.clientX - cavasRect.left;
		var yPos = event.clientY - cavasRect.top;
		placedSmileys.push(cursorSmiley);
		cursorSmiley = new Smiley(xPos, yPos,
20+Math.floor(Math.random()*30));
	}

	function mouseMoveHandler(event){
		if (!started) return;
		if (cursorSmiley !== null){
			var c = document.getElementById("myCanvas");
			var cavasRect = c.getBoundingClientRect();
```

```
                    var xPos = event.clientX - cavasRect.left;
                    var yPos = event.clientY - cavasRect.top;
                    cursorSmiley.posx = xPos;
                    cursorSmiley.posy = yPos;
                }
        }

</script>
<body>
        <canvas id="myCanvas" width="500" height="500" style="border:1px
solid #000000;">
        </canvas>
</body>
<script>
        var myVar = setInterval(refreshCanvas, 30);
        document.getElementById("myCanvas").addEventListener('click',
clickEvent, false);
        document.getElementById("myCanvas").addEventListener('mousemove',
mouseMoveHandler);
</script>
```

3 Aszteroida bányászat

A világűr ismeretlen részeinek felfedezése izgalmas tevékenység, azonban az expedíciók erőforrásokat igényelnek, az űrhajók viszont korlátozott kapacitással rendelkeznek. Így ahhoz, hogy újabb és újabb expedíciókat indíthassunk, szükségünk van az űrben létrehozott bázisokra, amelyek az erőforrásokat előállítják. Az erőforrások nyersanyagokat igényelnek, amire a legjobb forrást az aszteroidák adják. Jelen feladatban is az a célunk, hogy egy aszteroida mező közelében kiépítsünk egy bázist, ami az aszteroidákból nyersanyagokat képes kibányászni.
A programban aszteroidák mozogjanak a pályán (tipikusan bejönnek az egyik szélén, majd egyenesen mozogva eljutnak a másik széléhez és eltűnnek). Az egyszerűség kedvéért csak egyfajta nyersanyagot, vasat gyűjtünk. Kezdetben az aszteroidákra kattintva lehet belőlük vasat kinyerni, de később tudunk majd lerakni bányász állomásokat is, amik automatikusan működnek. A végső cél pedig a bázis felépítése.

4 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. Nem használható 3rd party könyvtár vagy technológia, amit külön telepíteni kellene.

Beküldhető feladatok:

1) Készítsétek el a leírt programot:
   a) A pályán rendszeresen legyen pár aszteroida, amelyek egyenes vonalban mozognak, véletlenszerű irányban és sebességgel!
   b) Az aszteroidáknak legyen egy vas kapacitása, ami be lehet belőlük termelni!
   c) Egérrel kattintva egy aszteroidára abból valamennyi vasat termeljünk be (például 1-et), és a készletünk látszódjon is a honlapon.
   d) Lehessen a pályára bányász állomásokat helyezni (az egérrel választott pozícióba), melyek valamennyi vasba kerülnek (például 50-be), és egy bizonyos hatósugárral rendelkeznek. Minden bányász állomás másodpercenként 1-szer termel valamennyi vasat (például 1-et) valamelyik, a hatósugarában lévő aszteroidából.
   e) A végső cél a bázis leépítése legyen, amely nagy mennyiségű vasba kerül (például 50000-be). Ha ez megtörténik, akkor sikerrel jártunk és a játéknak vége. Ezt nem kell a pályára tenni, elég csak egy gombbal megvenni.
   f) Lehessen néhány fejlesztést venni, szintén vasból. Az oldal jelenítse is meg, hogy mit tudnak ezek, és mennyibe kerülnek.
      i) Pár ötlet fejlesztésekre (ezek csak ötletek, lehet mást is kitalálni, és 3-4 darab elég): a kattintás több vasat termel be; a kattintásnak is legyen hatósugara, és több aszteroidát is elérhessen egyszerre; a bányász állomás több vasat termel másodpercenként; a bányász állomás nagyobb hatósugárral rendelkezik; a bányász állomás egyszerre több aszteroidából is tud termelni; a bányász állomás magához tud vonzani aszteroidát, ami így sosem megy ki a hatósugarából.
   g) A tesztelés könnyítésére legyen egy gomb, amit megnyomva a program teszt módba lép: minden költséget (bányák, fejlesztések, bázis) 1-re állít. (Így ezek funkcionalitása hatékonyabban tesztelhető).

1100 0000$|_2$ pont