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**).
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>
```

3 Elektromos autók Középbitföldén

Középbitfölde népe számára most kezd mindennapossá válni az elektromos autók használata. Kezdenek ezek a járművek annyira elterjedni, hogy a Fő Bittanács eldöntötte, ideje kiépíteni a szükséges infrastruktúrát. Hogy elkerüljék a baleseteket és a túlzott javítási költségeket, az elektromos autók tulajdonosai kötelesek az akkumulátorokat rendszeresen cserélni. A használt akkumulátorokat ellenőrzik, és ha még biztonságosnak ítélik, újra forgalomba helyezik.
Az akkumulátor cserére és tesztelésre azonban speciális műhelyeket kell kiépíteni, és szeretnék, ha minél kevesebbre lenne szükség. Úgy kell ezeket elhelyezni, hogy a városoktól ne legyenek túl messze, hiszen a tulajdonosoknak el kell oda vezetni. A dolgot nehezíti, hogy az autók három fajta akkumulátorral rendelkeznek, és mindegyikre kell csere lehetőséget biztosítani.
A feladat egy puzzle játék megvalósítása, ahol a játékosnak a műhelyek helyeit kell meghatároznia úgy, hogy városok igényeit lefedje. A játékban lesznek városok, amiknek lesznek igényei, és a játékosnak a műhelyeket kell lehelyezni ezek közé, megfelelő helyekre.

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.

Beküldhető feladatok:

1) Készítsétek el a leírt programot:
   a) Jelenjenek meg a térképen városok, amelyeknek megvannak az igényei, vagyis, hogy a lakosok melyik fajta akkumulátort használják a három közül (ez legyen is látható).
   b) A játékos tudjon lehelyezni műhelyeket. Minden műhelyhez legyen adva, hogy milyen típus akkumulátort tud cserélni.
   c) A játékban legyen néhány kézzel elkészített feladvány
      i) Lehessen kiválasztani, hogy melyik feladványt indítsa el a játék
      ii) A feladványokban a városok legyenek fix helyen, mindegyik 1 akkumulátor típussal (bonyolultabb feladványban lehet olyan város is, ahova kettő típus is kell)
      iii) A feladvány mondja meg, hogy mennyi lehelyezhető műhelyünk van, és melyik milyen akkumulátort tud cserélni
   d) Legyen egy adott távolág, ami a kiszolgálási sugár. Egy műhely csak akkor tud egy várost ellátni akkumulátorokkal, ha ezen a távolságon belül van, és a megfelelő típusú akkumulátort tudja kezelni.
   e) A játékos feladata, hogy a rendelkezésre álló műhelyeket a térképre rakja úgy, hogy minden város akkumulátor igényeit ki tudják szolgálni.
      i) A lerakás közben a játék jelezze, hogy melyik városok vannak a határon belül.
      ii) A játék ellenőrizze a leadott megoldást, hogy tényleg minden város közelében van-e megfelelő műhely.
         (1) Ha valahol hiányt talál, jelezze
         (2) Ha minden jó, akkor gratuláljon a megoldáshoz

1100 0000$|_2$ pont