Multimédia nappali gyakorlat

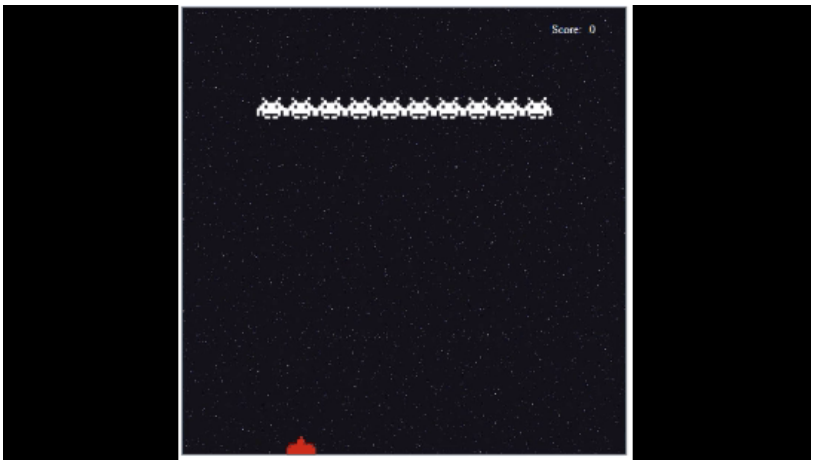
Ütközésdetektálás

A tananyaghoz készült videó az alábbi linken érhető el: https://www.youtube.com/...

Utoljára módosítva
2020. április 13.

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

Egészítsük ki a korábbi feladatot ütközésdetektálással és pontszámítással! (jQuery példa)



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 fájlt. Nyissuk meg a `space_invaders_jquery.html`-t.

A korábbi forráskódhoz képest a html állományban nem történt számottevő változás. A feladat részeként az eltalált ellenség számát a játéktéren meg kell jeleníteni a jobb felső sarokban. Ehhez egy új stílusformázási beállítást helyezünk el a `<style>` tag-ben. Beállítjuk a szöveg színét, illetve pozícióját is.

```
#score_tab {
  color: mintcream;
  position: relative;
  left: 500px;
  top: 20px;
}
```

A továbbiakban vizsgáljuk meg a `script.js` tartalmát és a feladatmegoldáshoz szükséges kiegészítéseket. Az aktuális pontszám kijelöléséhez hozzunk létre egy változót, melynek értéke kezdetben, természetesen, legyen 0.

```
let score = 0;
```

Ahhoz, hogy a játéktéren meg tudjuk jeleníteni az aktuális találatok számát, létre kell hozni számára egy új elemet. Ez az elem fogje felvenni a `score_tab` ID-t, így megkapva a korábba definiált formázási beállításokat. Nem csak a pontszámot jelenítjük meg, hanem előtte a `Score:` sztringet is. Maga a pontszám egy külön elemben jelenik majd meg mellette. Ehhez a `score` ID-t adjuk meg azért, hogy tudjunk majd erre a `<span>` elemre hivatkozni és módosítani a tartalmát, jelen esetben hozzáadni az aktuális pontszámot a `text` függvényhívás segítségével.

```
game_area.append('<div id="score_tab">Score:<span id="score" style="padding-left: 10px"></span></div>')
$('#score').text(score)
```

Annak érdekében, hogy változtathassuk a pontszámot, írjuk meg hozzá a szükséges függvényeket. Első körben azt fogjuk megtekinteni, hogy hogyan lehet lövést "létrehozni" egérkattintás és billentyűlenyomás segítségével egyaránt. A játéktérhez hozzáadunk egy egérkattintás esemény monitorozót és amennyiben ez bekövetkezik, meghívjuk az `init_shoot` függvényt, melyet a későbbiekben tárgyalunk.

```
$(game_area).on('click', init_shoot);
```

Az egérkattintás mellett a SPACE billentyű lenyomásával is szeretnénk az ellenségre tüzelni, ezért szükséges ennek a lenyomott billentyűnek a vizsgálata is a korábban már megírt `move_defender` függvényben.

```
if (pressed_key === 'ArrowRight') {
  ...
} else if (pressed_key === 'ArrowLeft') {
  ...
} else if (pressed_key === ' ') {
  init_shoot();
}
```

Miután beállítottunk különféle eseményfigyelést, tekintsük meg a lövésért felelős `init_shoot` függvényt. Azt szeretnénk, ha a lövedék úgy kerülne kilövésre, mintha a defender csövéből történt volna. Ezért a lövedék (x, y) pozíciójának a beállításával is foglalkozni kell. Ehhez lekérjük a defender aktuális x és y koordinátáját arra figyelve x irányban, hogy a középre pozícionálás miatt a `move_step` értékét is adjuk hozzá. Annak érdekében, hogy a lövés vizuálisan is megjelenjen, a `cartridge` változóban tároljuk el az ide vonatkozó képünket, tehát hozzunk létre egy `<img>` HTML elemet. Látni fogjuk majd, hogy valójában majd minden egyes lövésnél egy újabb elemet hozunk létre.

Ezt követően állítsuk be a `cartridge` (x, y) pozícióját és ügyeljünk arra, hogy a `position` formázási beállítást is megtegyük, hogy a lövedék megfelelően jelenjen meg a játéktéren. A lövéshez az `animate` függvényt fogjuk használni, mely a játéktér tetejéig írja le függőleges mozgását. Amint elérte a 0. koordinátapontot, akkor levesszük a játéktérről a `remove` függvény segítségével. Az ütközésdetektálás során a lövedékek és az ellenség koordinátáit fogjuk összehasonlítani, így szükséges lesz majd a nevezett elemek aktuális pozíciója is, ezért a lövedékhez egy `shoot` nevezetű osztálycímkét is hozzáadunk, majd az elemet a játéktérhez fűzzük.

```
function init_shoot() {
  let def_pos_x = defender.position().left + move_step;
  let def_pos_y = defender.position().top;
  let cartridge = $('<img src="bomb.jpg">');

  cartridge.css({
    top: def_pos_y,
    left: def_pos_x,
    position: 'relative'
  });

  cartridge.animate({
    top: 0
  }, 1000, function () {
    cartridge.remove()
  });

  cartridge.addClass('shoot');
  game_area.append(cartridge);
}
```

Miután tudunk lövéseket előidézni, már csak az ütközések vizsgálata van hátra. Ehhez szükséges, hogy azt folyamatosan figyelemmel kísérjük és megvizsgáljuk azok lehetséges bekövetkeztét. Ehhez elhelyezünk egy másik időzítőt az ellenség mozgása mellett, ami a `check_collision` függvényt fogja meghívni 1 ms-onként, melyben maga az ütközést detektáljuk.

```
setInterval(check_collisoin, 1);
```

Az ütközésdetektáláshoz a lövedék és az ellenség távolságát fogjuk megvizsgálni, ezért létrehozunk egy függvényt, melynek bemenő paramétere két object. Az egyik a vizsgált lövedék és ellenség x és y koordinátáját tartalmazza. Maga a távolság egy egyszerű Euklideszi távolságérték lesz.

```
function distance(a, b) {
  let dx = a.x - b.x;
  let dy = a.y - b.y;
  return Math.sqrt(dx * dx + dy * dy)
}
```

Az ütközésdetektáláshoz először vegyük sorra az `each` függvény segítségével a lövedékeket. Az aktuális elemet tároljuk el az `act_shoot` változóban. Ahhoz, hogy az előbb látott `distance` függvénynek át tudjuk adni a lövedék pozícióját, ahhoz azt le kell kérni. Mivel a lövedék felfelé irányuló mozgást végez, ezért az y koordinátája az elem kezdő y pozíciója lesz, tehát az a pont, ahol éppen y irányú mozgása során tart a lövedék. Továbbá, lekérjük az x pozíciót is és ebben az esetben is hozzáadjuk a `move_step` értékét a középre pozícionális miatt.

Nem csak a lövedékeken kell végigmenni az összehasonlításhoz, hanem az ellenségen is. Hasonlóan tesszük meg, mint az előzőekben. Azért láttuk el ezeket az elemeket osztálycímkével, hogy könnyebben össze tudjuk gyűjteni, amikor szükséges, mint pl. ebben az esetben. Az ellenség esetében annak aktuális (x, y) pozíciója az elem közepén lesz. Eztkövetően egy feltételes szerkezetben megvizsgáljuk a lövedék és az ellenfél távolságát és amennyiben az egy adott értéknél kisebb, akkor az aktuálisan vizsgált ellenfelet eltávolítjuk, illetve a lövedéket is, valamit a `score` változó értékét növeljük és ezt frissítjük is a játéktéren.

```
function check_collisoin() {
  $('.shoot').each(function () {
    let act_shoot = $(this);
    let act_x = act_shoot.position().left + move_step;
    let act_y = act_shoot.position().top;

    $('.enemy').each(function () {
      let e_pos_x = $(this).position().left + move_step;
      let e_pos_y = $(this).position().top + parseInt(enemy.css('height')) / 2;

      if (distance({x: act_x, y: act_y}, {x: e_pos_x, y: e_pos_y}) <= parseInt($(this).css('height')) /
          2) {
        $(this).remove();
        act_shoot.remove();
        score += 1;
        $('#score').text(score);
      }
    });
  });
}
```

Egészítsük ki a korábbi feladatot ütközésdetektálással és a megtett körök számításával! (Canvas példa)

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

Megoldás menete

A kiindulasi_alap.zip csak a feladat megoldásához szükséges kiindulási HTML állományt és a szükséges képeket 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 `kukacos_vaz.html`-t. Ebben a feladatban is egy korábban már látott példát fogunk továbbfejleszteni.

Ebben a példában ki fogjuk jelezni a játéktéren, hogy hányszor haladt át a kukac felett a repülő, miközben bombákat dobált úgy, hogy nem találta el a kukacot, amit mi irányítunk mindeközben. tekintsük meg először, hogy `<canvas>` elem használata esetén miként tudunk szöveges tartalmat elhelyezni a játéktéren, azaz jelen esetben a teljes vásznon. Ehhez létrehozunk egy `plane_rounds` változót, melyet 0 értékkel definiálunk.

```
1  let plane_rounds = 0;
```

Ezt követően el is helyezzünk a vásznon. A `font` változó értékeként beállítjuk a betűstílust, méretet, típust, majd a `fillText` függvényhívás segítségével kiíratjuk a 'Rounds: ' szöveget és a `plane_rounds` aktuális értékét az (500, 30)-as koordinátaponttól kezdve.

```
1  ctx.font = 'bold 14px Arial';
2  ctx.fillText('Rounds: ' + plane_rounds, 500, 30);
```

Ahhoz, hogy kiírjuk, hogy hányszor haladt át a kukac felett a repülőgép, nyilvánvalóan a `drawPlane` függvényhez kell hozzányúlni, mivel itt valósítottuk meg a repülőgép mozgatását. Akkor tekinthetjük áthaladtnak a repülőt, amikor újra a 600-as x koordinátapontra kerül, vagyis ekkor kelle hozzáadni 1-et a `plane_rounds` változó értékéhez.

```
function drawPlane() {
  ...
  if (planeX < 0 - planeW / 4) {
    planeX = 600;
    plane_rounds += 1
  }
}
```

Ebben az esetben is két pont Euklideszi távolságát fogjuk kiszámítani az ütközésdetektáláshoz, de ebben az esetben a bombák és a kukac pozíciója lesz a két vizsgált objektum.

```
function distance(a, b) {
  let dx = a.x - b.x;
  let dy = a.y - b.y;
  return Math.sqrt(dx * dx + dy * dy)
}
```

Az ütközésdetektáláshoz végigmegyünk a bombák listáján és lekérjük az aktuális elem x és y koordinátáját. Külön változóban a a kukac pozíciója is tárolva van, így annak lekérésére nincs szükség. A `distance` függvénynek tehát átadjuk a bomba és a kukac pozícióját és ebben az esetben is, amennyiben a távolságuk kisebb, mint egy küszöbérték, akkor különböző utasításokat hajtunk végre. A jQuery példában azt láthattuk, hogy az elemek közepére vagy a tetejének közepére pozícionáltuk a vizsgált pontokat, itt ezt nem tettük meg.

Amennyiben a kukacot eltalálja egy bomba, akkor azt szeretnénk, ha az egész animáció megállna. Ehhez létrehozunk egy flag-et, aminek kezdetben az értéke `true`, viszont találat esetén `false`-ra állítjuk át.

```
let is_animate = true;
```

Ezen felül kicseréljük a kukac képét is és onnantól letiltjuk az egérmozgást is.

```
function check_collision() {
  for (b in bombList) {
    let act_x = bombList[b].x;
    let act_y = bombList[b].y;

    if (distance({x: act_x, y: act_y}, {x: wormX, y: wormY}) < planeH / 8) {
      is_animate = false;
      wormImg.src = 'worm2.png';
      c.removeEventListener('mousemove', wormMouseMove);
    }
  }
}
```

Fontos, hogy az ütközésdetektálást is folyamatosan hajtsuk végre, ezért ezt is az `animate` függvényben kell meghívni. Ahhoz, hogy a bomba és a kukac ütközésekor leálljon az animáció, az `is_animate` flag-et false állítottuk és ezért a `requestAnimationFrame` ütközés után már nem hívódik meg.

```
function animate() {
  if (is_animate) {
    requestAnimationFrame(animate);
  }
  ...
  check_collision();
}
```