

LAB0 3: 00 JS

Oefening 1: Bouncing Ball

Intro

Maak een nieuw project aan met HTML & JavaScript file. Voorzie een Canvas tag en spreek deze aan in JavaScript. Maak een globale variabele aan waarin je de context aanspreekt.

Definieer in een klasse "Bol" in dezelfde script file. We zullen later nog zien hoe je je code kan opsplitsen in meerdere files, voorlopig werken we binnen 1 bestand:

```
class Bol {  
  constructor(x, y) {  
    this.x = x;  
    this.y = y;  
    this.velocityX = 1;  
    this.velocityY = 3.3;  
  }  
}
```

Definieer voor deze klasse een draw functie. Hierin teken je een cirkel op de globale context, op basis van z'n properties (this.x, this.y).

In de init functie van je javascript maak je 1 globale Bol aan en roep je een globale draw functie op (niet die van bol, maar een aparte functie). Deze globale draw functie koppel je aan requestAnimationFrame zodat deze uitgevoerd wordt aan 60fps. In deze functie roep je de draw functie van de bol op:

```
const draw = () => {  
  ctx.clearRect(0, 0, canvas.width, canvas.height);  
  bol.draw();  
  window.requestAnimationFrame(draw);  
};
```

Test de applicatie. Je zou een bol moeten zien staan op het scherm.

Bouncing Ball

We zullen de draw functie van de bol nu uitbreiden zodat deze over het scherm kaatst. Eerst en vooral verhoog je de x en y waarden met de velocity waarden:

```
this.x += this.velocityX;  
this.y += this.velocityY;
```

Voer daarna nog een check uit of de bol de randen van de canvas overschrijdt. Indien dit het geval is, dan pas je de velocity aan:

```
if(this.x > canvas.width) {  
    this.velocityX = -Math.abs(this.velocityX);  
}  
if(this.x < 0) {  
    this.velocityX = Math.abs(this.velocityX);  
}  
if(this.y > canvas.height) {  
    this.velocityY = -Math.abs(this.velocityY);  
}  
if(this.y < 0) {  
    this.velocityY = Math.abs(this.velocityY);  
}
```

Vectoren

Momenteel wordt de 2D informatie (positie & snelheid) nog opgeslaan in aparte properties. We zullen deze 2D informatie opslaan in aparte objecten (vectoren).

Definieer een nieuwe klasse Vector die een x en y positie bijhoudt:

```
class Vector {  
    constructor(x, y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Vervang in de bol klasse de .x en .y properties door een instantie van de vector klasse en de velocityX en velocityY properties door een tweede instantie.

```
constructor(x, y) {
  this.location = new Vector(x, y);
  this.velocity = new Vector(1, 3.3);
}
```

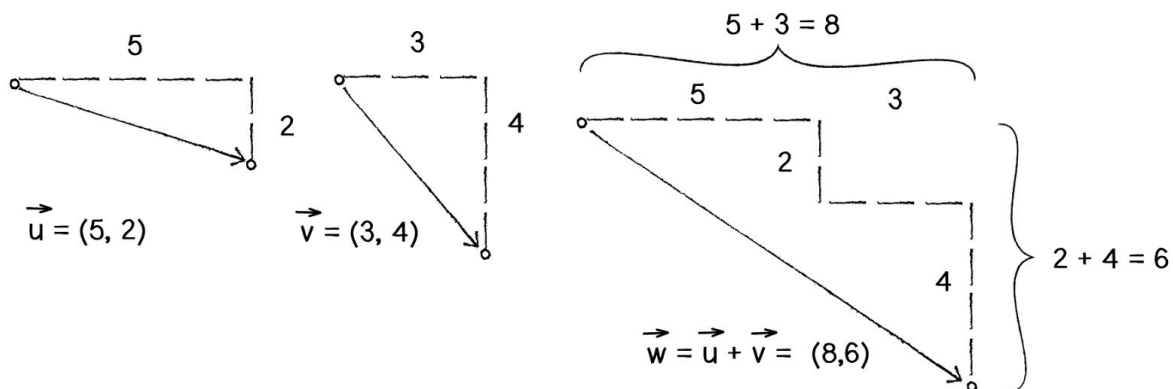
Pas de code verder aan, zodat de bouncing ball opnieuw werkt.

Vectoren optellen

Momenteel tellen we de x en y posities van de 2 vectoren apart op in onze bol klasse. We kunnen dit echter ook doen in de vector klasse.

Definieer een nieuwe functie add in de vector klasse, waarmee we een andere vector kunnen optellen:

```
add(otherVector) {
  this.x += otherVector.x;
  this.y += otherVector.y;
}
```



Maak vervolgens gebruik van deze add functie in de draw loop van de Bol.

Meerdere Ballen

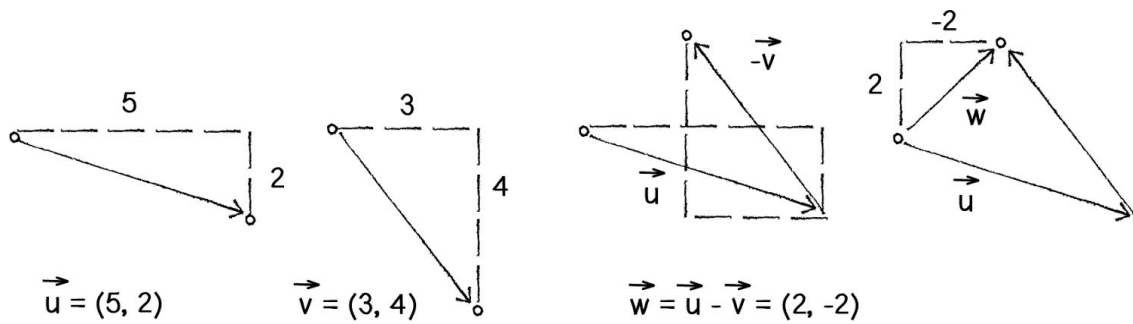
Breidt de code uit, zodat er meerdere bollen op het scherm bewegen.

Gebruik hiervoor een globale array waarin je de bollen opslaat. In de draw functie overloop je vervolgens de bollen en voer je hun draw functie uit.

Vectoren aftrekken

We zullen onze Vector klasse uitbreiden. In latere oefeningen zullen we vectoren moeten aftrekken, herschalen, de grootte berekenen of normaliseren.

Vectoren aftrekken van elkaar komt neer op de x en y componenten af te trekken.



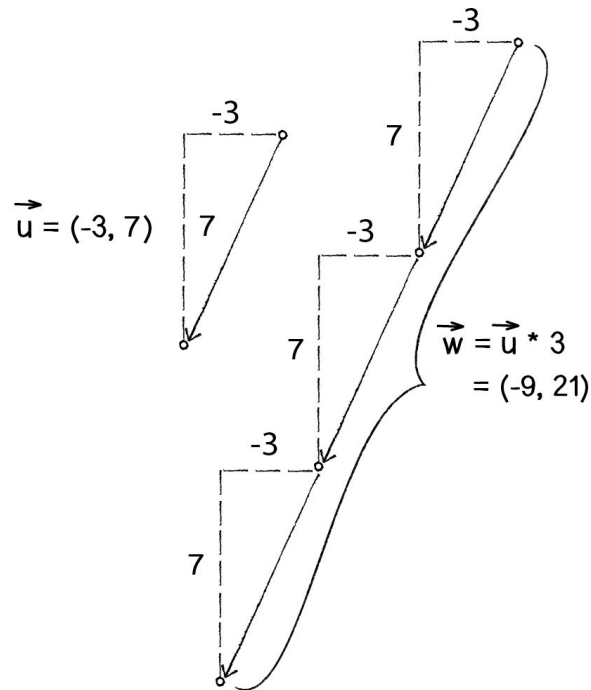
```

sub(v) {
  this.x -= v.x;
  this.y -= v.y;
}

```

Vectoren schalen

Een vector verscalen doe je door beide componenten met een bepaalde factor te vermenigvuldigen. Aangezien een deling hetzelfde is als een vermenigvuldiging, geldt hier dezelfde regel.

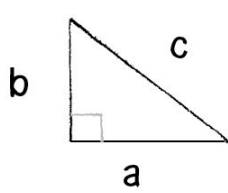


```
mult(n) {
  this.x *= n;
  this.y *= n;
}

div(n) {
  this.x /= n;
  this.y /= n;
}
```

Lengte van een vector

De lengte van een vector berekenen komt neer op het toepassen van de stelling van pythagoras:



$$a^2 + b^2 = c^2$$

or

$$c = \sqrt{a^2 + b^2}$$

```
mag() {
  return Math.sqrt(this.x*this.x + this.y*this.y);
}
```

Vectoren normaliseren

Nauw samenhangend met het berekenen van de lengte van een vector, is het normaliseren van een vector. Dit wil zeggen dat we een vector met een bepaalde lengte, zullen herleiden naar een vector met lengte 1.

Dit bekom je door de vector te delen door z'n lengte.

Genormaliseerde vectoren zullen we gebruiken bij het uitvoeren van krachten op een element.

```
normalize() {
  const m = this.mag();
  if(m !== 0) {
    this.div(m);
  }
}
```

Oefening 2: Versnelling

Dupliceer de vorige oefening.

We zullen deze uitbreiden zodat de ballen richting de muispositie bewegen.

Move through edges

We zullen de bol niet langer laten kaatsten tegen de randen, maar door de randen heen bewegen. Pas hiervoor de check aan in de draw functie van de Bol:

```
if(this.location.x > canvas.width) {  
    this.location.x = 0;  
} else if(this.location.x < 0) {  
    this.location.x = canvas.width;  
}  
if(this.location.y > canvas.height) {  
    this.location.y = 0;  
} else if(this.location.y < 0) {  
    this.location.y = canvas.height;  
}
```

Versnelling

We zullen onze velocity laten aanpassen door een versnellingsvector.

Maak hiervoor een nieuwe property aan in de Bol klasse:

```
this.acceleration = new Vector(-0.001, 0.01);
```

In de draw functie van de Bol, tel je deze acceleratie op bij de velocity:

```
this.velocity.add(this.acceleration);
```

Test de applicatie – de bollen beginnen aan een bepaalde snelheid en versnellen alsmaar verder.

Limiteren snelheid

De bollen blijven alsnog sneller bewegen. Voeg een `limit()` functie toe aan de vector klasse om de snelheid (magnitude) van een vector te beperken tot een bepaald maximum.

Pas je kennis van vectoren toe om deze limit functie te schrijven:

```
limit(max) {
  if (_____ > _____) {
    _____();
    _____(max);
  }
}
```

Roep deze functie op in de draw functie van de Bol, zodat de snelheid niet groter wordt dan 10:

```
this.velocity.limit(10);
```

Random beweging

Wanneer je de acceleration in de draw loop randomized, krijg je een leuk effect waarbij de bollen willekeurig over het scherm bewegen.

```
this.acceleration.x = 1 - Math.random() * 2;
this.acceleration.y = 1 - Math.random() * 2;
```

Beweging richting muis

Definieer een globale variabele voor de muispositie en initialiseer deze in de init functie:

```
mouse = new Vector(canvas.width / 2, canvas.height / 2);
```

Koppel vervolgens een mouse move event:

```
canvas.addEventListener(`mousemove`, event => mousemoveHandler(event));
```


In de mouse move handler doe je een update van de mouse vector:

```
const mousemoveHandler = event => {  
  mouse.x = event.clientX;  
  mouse.y = event.clientY;  
};
```

We zullen de acceleratie vector instellen zodat deze zich richt naar deze muis coördinaten.

Dit doe je door de positie van de muis af te trekken van de locatie van de bol:

```
const dir = new Vector(mouse.x, mouse.y);  
dir.sub(this.location);
```

Als we deze vector onmiddellijk zouden toepassen als acceleratie, dan zou deze te groot zijn. We zullen de vector dus normalizeren en vermenigvuldigen met een bepaalde maximum snelheid:

```
dir.normalize();  
dir.mult(0.5);
```

Stel deze dir vector vervolgens in als acceleratie vector:

```
this.acceleration = dir;
```

Test de applicatie. De bollen bewegen richting & rond de muispositie.

Vector Clones

Een nadeel van onze vector methods is dat deze altijd werken op een instantie van een vector.

Wanneer we 2 vectoren van elkaar willen aftrekken en het resultaat opslaan in een nieuwe vector, kunnen we niet zomaar gebruik maken van de sub functie, aangezien deze de properties van de vector zelf aanpast:

```
sub(v) {  
  this.x -= v.x;  
  this.y -= v.y;  
}
```

Je moet dus een kopie maken van de vector en deze kopie gebruiken.

Momenteel maken we op deze manier een kopie van de mouse vector:

```
let dir = new Vector(mouse.x, mouse.y);
```

We zullen een clone() methode toevoegen aan de vector klasse, om dit korter te kunnen schrijven:

```
clone() {  
    return new Vector(this.x, this.y);  
}
```

Maak nu gebruik van deze clone() methode om de muis vector te kopiëren:

```
let dir = mouse.clone();
```

Static methods

Momenteel gebeurt het berekenen van het verschil tussen 2 vectoren in 2 stappen: eerst maak je een kopie van de eerste vector, en vervolgens trek je de 2^{de} vector af van deze kopie:

```
let dir = mouse.clone();  
dir.sub(this.location);
```

We zullen een static method toevoegen aan de Vector klasse, zodat dit ook korter geschreven kan worden in onze code:

```
static sub(v1, v2) {  
    let copy = v1.clone();  
    copy.sub(v2);  
    return copy;  
}
```

In onze applicatie zelf, kunnen we opnieuw 2 lijnen code samenvoegen tot 1 instructie:

```
let dir = Vector.sub(mouse, this.location);
```

Return This

Met een klein handigheidje kunnen we de code in onze static sub functie nog compacter maken. De functie moet een instantie van onze nieuwe vector returnen.

Pas de “gewone” instantie functie aan, zodat deze “this” returned:

```
sub(v) {  
  this.x -= v.x;  
  this.y -= v.y;  
  return this;  
}
```

Aangezien de gewone sub functie nu een instantie returned, kan je je return statement vroeger plaatsen in de static sub functie:

```
static sub(v1, v2) {  
  let copy = v1.clone();  
  return copy.sub(v2);  
}
```

Het kan zelfs nog compacter nu: we hebben de copy variabele niet meer nodig, en kunnen de clone() en sub() functies chainen:

```
static sub(v1, v2) {  
  return v1.clone().sub(v2);  
}
```

Schrijf nu zelf ook static functies voor de add, mult en div functionaliteiten.

EXTRA: Platform Game

Je zal een canvas rendering engine schrijven voor een eenvoudige platform game. In de startbestanden vind je een ES5 versie van deze game, met een DOM rendering engine (rendering gebeurt als DOM elementen / tables – ter referentie: de tutorial die deze game opbouwt in ES5 kun je vinden op http://eloquentjavascript.net/15_game.html)

- Zet de ES5 code om naar ES6.
- Schrijf een CanvasDisplay klasse die de game rendert op een Canvas in plaats van de DOMDisplay klasse (die met DOM elementen werkt)