



---

# INFORMATICA

---

Basis games



DECEMBER 2024 V1

AUTEUR: ing. A. Haksteen

## Inhoudsopgave

Inleiding.....	2
Spel ontwerp.....	3
1. Spel zonder interactie (animatie) .....	5
1.1 Een raster zonder gebruik te maken van javascript .....	5
1.2 Layouts.....	8
1.2.1 Grid Layout .....	8
1.2.2 Flexbox Layout .....	9
1.3 Raster met opmaak .....	11
1.4 Raster met bewegend blokje inclusief snelheids-regeling.....	13
1.5 Blokje zoekt (zelf) een blauwblokje in een doolhof .....	15
1.6 De 3 (zelf) bewegende ballen .....	19
1.6.1 Beschrijving van de werking van het JavaScript.....	21
1.6.2 OOP in Javascript.....	21
1.6.3 Opbouw van een class .....	21
1.7 Drie tegen elkaar botsende ballen .....	25
2. Met interactie en gebruik makend van THREE.JS.....	30
2.1 Eenvoudig 3d- kanon die kan schieten en bewegen .....	30
2.1.1 Three.js gebruik.....	31
2.1.2 Wat is Three.js? .....	32
2.1.3 Three.js code .....	34
2.1.4 Uitleg Three.js gebruikte onderdelen .....	35
2.2 Schieten met een eenvoudig kanon op een vliegend object .....	37
2.3 spel uitbreiding .....	41
2.3.1 Elk spel heeft zijn begin en zijn einde.....	41
2.3.2 Het bijbehorend verhaaltje .....	41
2.3.3 Toevoegen van geluid .....	46
2.4 De tank .....	47
2.5 kruisraketten i.p.v. kogels .....	50
3. Quiz game.....	53
4.0 Het MVC-Architectuurpatroon.....	56
4.1 Wat is MVC?.....	56
4.2 De MVC-architectuur toegelicht.....	57
4.3 Waarom gebruiken ontwikkelaars MVC?.....	57
4.4 Hoe wordt MVC toegepast?.....	57
4.5 MVC voorbeeld: Spelletje hoger-lager.....	58

## Inleiding

Deze module is ontworpen om je te begeleiden bij het ontwikkelen van eenvoudige spelletjes en je vertrouwd te maken met de technieken die daarbij komen kijken. We beginnen met het opzetten van een basisstructuur, zoals een raster, die op verschillende manieren kan worden opgebouwd. Daarna wordt in het raster (de afbakening) een bewegend object toegevoegd, zoals een blokje, dat interactief over het scherm beweegt. Stap voor stap werken we naar een animatie waarin een rood blokje, met behulp van een algoritme, een blauw blokje in een doolhof moet vinden. De theoretische achtergronden en de daarbij horende opbouw worden hierbij toegelicht.

Na deze eerste stappen introduceert de module bewegende elementen, zoals drie zelf bewegende ballen die tegen “muren” en elkaar botsen. Hierbij wordt de wet van behoud van energie toegepast. Dit deel van de module maakt gebruik van Object-Oriented Programming (OOP) en class-diagrammen. Dankzij deze techniek kun je eenvoudig meerdere objecten met dezelfde eigenschappen op een eenvoudige wijze creëren en beheren.

De volgende fase introduceert het gebruik van **THREE.JS**, een krachtige bibliotheek voor 3D-programmering. Hiermee kun je een kanon maken dat heen en weer beweegt en waarmee je kunt schieten. Vervolgens wordt het spel uitgebreid met een ruimteschip dat je met het kanon kunt uitschakelen. Ook kun je geluidseffecten toevoegen om de spelervaring te verbeteren. Om de uitdaging te vergroten, kun je een extra ruimteschip en bommen introduceren die het kanon proberen uit te schakelen.

Voor een meer realistische ervaring kun je het eenvoudige kanon vervangen door een tank met wielen en een draaibare loop. Deze tank kan bewegen en schieten. Je kunt dit uitbreiden met geavanceerde wapens, zoals kruisraketten. Deze raketten kunnen na het afvuren versnellen, van richting veranderen en nauwkeurig hun doel treffen.

Tijdens de module leer je essentiële basisprincipes, zoals hoe OOP werkt en hoe je botsingen detecteert met behulp van wiskundige formules. Dit vormt een solide basis waarop je zelf verder kunt bouwen. Zelfonderzoek en experimenteren zijn hierbij onmisbaar. Je wordt aangemoedigd om aanvullende informatie te zoeken, bijvoorbeeld via Google of door tools zoals ChatGPT te gebruiken.

Daarnaast wordt in deze module het belang van een verhaallijn benadrukt. Een goed verhaal kan de aantrekkelijkheid van je spel aanzienlijk vergroten. Denk ook aan spelmechanieken zoals het bijhouden van scores, tijd en andere variabelen die het spel dynamischer maken.

Deze module biedt een uitgebreid stappenplan waarmee je zowel de technische als de creatieve aspecten van spelontwikkeling kunt verkennen. Ik wens je veel plezier met het bouwen van je eigen spellen en hoop dat je geniet van het proces! Op pagina 51 is nog een voorbeeld gegeven van een quiz-spel.

Daarna volgt nog een uitleg over MVC, wat buiten de lesstof valt, maar wel belangrijk is voor een applicatie ontwikkelaar. Er zijn twee spelletjes toegevoegd die volgens het MCV model zijn gemaakt. De code is zo opgebouwd dat je de code vrijwel direct kan kopiëren. Mooier zou het zijn dat de HTML-code, CSS en JS apart in de directory geplaatst zouden worden.

We beginnen met een beknopte versie van het spel ontwerp. Wat moeten we doen om een spelletje te kunnen creëren?

# Spel ontwerp

Het ontwerpen van een spel vereist een gestructureerde aanpak. Hieronder staan de belangrijke stappen en documentaties die je moet maken:

## 1. Idee en Conceptontwikkeling

### Wat te doen?

- Bepaal het doel en de kernmechanieken van het spel.
- Definieer het thema, genre, en de doelgroep.
- Maak een korte beschrijving van wat het spel uniek maakt.

### Documentatie:

- **Conceptdocument:** Beschrijf het doel, thema, doelgroep, en waarom het spel interessant is.
- **Elevator pitch:** Een korte samenvatting van je spel, zoals je het aan iemand in 1 minuut zou uitleggen.

## 2. Spelregels en Mechanica

### Wat te doen?

- Schrijf gedetailleerde regels op.
- Beschrijf hoe spelers het spel beginnen, voortgang maken en winnen of verliezen.
- Definieer interacties, zoals bewegingen, acties en beperkingen.

### Documentatie:

- **Spelregels-document:** Overzicht van alle regels en spelmechanismen.
- **Flowchart:** Een visuele weergave van hoe het spel zich ontwikkelt.

## 3. Ontwerp van Levels en Elementen

### Wat te doen?

- Maak schetsen of prototypes van levels, kaarten, of borden.
- Ontwerp personages, obstakels, wapens, of andere spelobjecten.
- Zorg dat moeilijkheidsgraden en voortgang gebalanceerd zijn.

### Documentatie:

- **Leveldesign-document:** Beschrijving van elk level, inclusief uitdagingen en doelen.
- **Objectbeschrijving:** Uitleg van wat elk object doet

## 4. User Interface (UI) en User Experience (UX)

### Wat te doen?

- Ontwerp hoe de speler informatie ziet, zoals scores, tijd, of gezondheidsbalken.
- Zorg dat de interface intuïtief en aantrekkelijk is.

### Documentatie:

- **UI/UX-design-document:** Schetsen of wireframes van schermen, menu's en HUD.

## 5. Technische Specificaties

### Wat te doen?

- Kies technologieën (engine, taal, frameworks) en platformen (PC, mobiel, console).
- Bepaal systeemvereisten en beperkingen.

### Documentatie:

- **Technisch ontwerp-document:** Beschrijft technologieën, algoritmes, en resources.

## 6. Prototyping en Testen

### Wat te doen?

- Maak een werkend prototype of papieren versie.
- Test de spelregels en mechanieken.
- Verzamel feedback en verbeter het spel.

### Documentatie:

- **Testrapport:** Informatie over bugs, feedback en aanpassingen.
- **Iteratieverslagen:** Aantekeningen van wat er na testen is aangepast.

## 7. Productie

### Wat te doen?

- Maak de definitieve versie van het spel.
- Voeg grafische elementen, geluidseffecten en animaties toe.

### Documentatie:

- **Projectplan:** Beschrijving van wie wat doet en binnen welke tijdlijn.
- **Assetlijst:** Overzicht van benodigde afbeeldingen, geluiden, scripts, etc.

## 8. Publicatie en Handleiding

### Wat te doen?

- Voorzie het spel van een handleiding voor de speler.
- Test of alles werkt op het gekozen platform.
- Publiceer het spel.

### Documentatie:

- **Handleiding:** Uitleg voor spelers over het doel, regels, en bediening.
- **Promotiemateriaal:** Beschrijving, screenshots, trailers, etc.

## 9. Onderhoud en Updates

### Wat te doen?

- Zorg voor een systeem om bugs op te lossen.
- Plan updates voor nieuwe functies of content.

### Documentatie:

**Changelog:** Overzicht van wijzigingen en verbeteringen.

**Feedbacklogboek:** Suggesties van spelers en prioriteiten voor toekomstige updates.

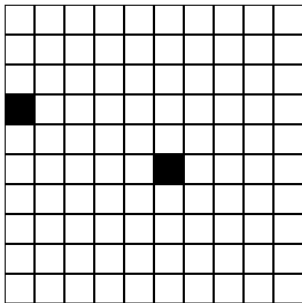
### Overzicht van Documentatie:

- Conceptdocument
- Spelregels-document
- Leveldesign-document
- UI/UX-design-document
- Technisch ontwerp-document
- Testrapport
- Handleiding
- Changelog

# 1. Spel zonder interactie (animatie)

## 1.1 Een raster zonder gebruik te maken van javascript

### Doolhof Raster



```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <title>Doolhof Raster</title>
  <style>
    /* Stijl voor het doolhof */
    .maze {
      display: flex;
      flex-wrap: wrap;
      width: 220px; /* Breedte voor een 10x10 raster */
    }

    .maze .cell {
      width: 20px;
      height: 20px;
      border: 1px solid #000;
      background-color: #fff;
    }

    /* Specifieke cellen zwart maken */
    .maze .cell:nth-child(31) { /* Cel 4,1 (31e cel in volgorde) */
      background-color: black;
    }

    .maze .cell:nth-child(56) { /* Cel 6,6 (56e cel in volgorde) */
      background-color: black;
    }
  </style>
</head>
<body>

  <h1>Doolhof Raster</h1>

  <!-- Container voor het doolhof -->
  <div class="maze">
    <!-- 100 cellen voor een 10x10 raster -->
    <!-- Kopieer de divs hieronder om een raster te maken -->
    <div class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div>
  </div>
</body>
```

```

<div class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div>
<div class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div>
<div class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div>
<div class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div>
<div class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div>
<div class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div>
<div class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div><div class="cell"></div><div class="cell"></div><div
class="cell"></div><div class="cell"></div>
</div>
</body>
</html>

```

Dit HTML-document maakt een visueel raster van 10x10 cellen met CSS-styling en HTML. Hier is een uitleg van de belangrijkste onderdelen:

- HTML-structuur

`<!DOCTYPE html>`

Geeft aan dat het document HTML5 gebruikt. Dit is standaard bij moderne webpagina's.

`<html lang="nl">`

Start het HTML-document en specificeert dat de inhoud in het Nederlands is (belangrijk voor zoekmachines en schermlezers).

`<head>`

Bevat meta-informatie over het document, zoals de tekenset en de titel.

`<body>`

De inhoud van de webpagina. Hier staat de titel van het raster (`<h1>`) en het raster zelf (`<div class="maze">`).

`<div class="maze">`

De container voor het raster. Binnen deze container staan 100 `<div>`-elementen, elk met de klasse `cell`. Samen vormen deze de individuele cellen van het raster.

- CSS-styling

De styling in <style> maakt het raster visueel aantrekkelijk:

Container (het raster zelf):

```
.maze {
  display: flex;      /* Cellen worden naast elkaar geplaatst totdat breedte is gevuld */
  flex-wrap: wrap; /* Laat de cellen naar de volgende rij springen als de breedte is gevuld */
  width: 220px;      /* Breedte gebaseerd op 10 cellen van elk 20px */
}
```

width: 220px;

Zorgt voor een vaste breedte van 10 cellen (20px breedte + 1px rand per kant = 22px per cel × 10 cellen = 220px).

Individuele cellen:

```
.maze .cell {
  width: 20px;
  height: 20px;          /* Elke cel is 20x20 pixels */
  border: 1px solid #000; /* Geeft elke cel een zwarte rand */
  background-color: #fff; /* Achtergrondkleur is wit */
}
```

Specifieke cellen zwart maken:

```
.maze .cell:nth-child(31) {
  background-color: black; /* Verander achtergrondkleur naar zwart */
}
```

```
.maze .cell:nth-child(56) {
  background-color: black; /* Verander achtergrondkleur naar zwart */
}
```

## Werking in detail

Het raster heeft een breedte van 220px, wat precies past bij 10 cellen van 22px breed (20px plus randen).

De flexbox (display: flex) zorgt ervoor dat de cellen automatisch op de volgende rij worden geplaatst als de breedte vol is.

### Wat zou je hiermee kunnen doen?

- Uitbreiding naar een groter raster:  
Verander de width van .maze en voeg meer cellen toe.

Interactiviteit toevoegen:

- Gebruik JavaScript om interactie met de cellen mogelijk te maken, zoals klikken om een cel zwart te maken.
- Automatisch cellen genereren met JavaScript:  
Vervang de handmatig gekopieerde <div>-tags door een script dat 100 cellen genereert.
- Dit HTML-programma genereert een raster met javascript, maar zonder opmaak en is **niet zichtbaar**.

```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
```



```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Raster</title>

<style>
</style>

</head>
<body>
  <div id="maze"></div>

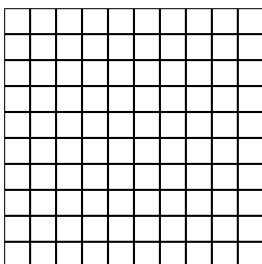
  <script>
    const maze = document.getElementById("maze");

    // Raster genereren
    for (let i = 0; i < 100; i++) {
      const cell = document.createElement("div");
      maze.appendChild(cell);
    }
  </script>
</body>
</html>

```

In de code wordt een raster gegenereerd door div-elementen toe te voegen aan het element met het ID "maze". Echter, het raster wordt niet zichtbaar omdat de div-elementen geen stijlen hebben toegewezen zoals breedte, hoogte of een achtergrondkleur. Hierdoor blijven ze standaard onzichtbaar of alleen als een lege ruimte zichtbaar.

Wanneer we tussen de `<style>` `</style>` CSS-code toevoegen zal het element #maze een raster met 10 bij 10 cellen creëren. Elke cel heeft een vaste breedte en hoogte van 20 pixels en een zwarte rand van 1 pixel rondom. Het zal er zo uit komen te zien.



## 1.2 Layouts

Om het raster zichtbaar te kunnen maken kunnen we blijkbaar **kiezen tussen 2 typen layouts**:

### 1.2.1 Grid Layout

De grid-opmaak maakt gebruik van rijen en kolommen om cellen te organiseren. De grootte van elke rij/kolom is gebaseerd op het herhalen van een bepaald aantal eenheden (in dit geval 10 rijen/kolommen van 20 pixels breed).

#### Voordelen:

- Gemakkelijk om specifieke cellen te positioneren en te stylen.
- Betere controle over hoe cellen worden gerangschikt (bijvoorbeeld overlappende cellen vermijden).
- Ideaal voor ingewikkelde, meer geavanceerde lay-outs met meerdere niveaus.

```

<style>
  #maze {
    display: grid;
    grid-template-columns: repeat(10, 20px); /* 10 kolommen van 20 pixels breed */
    grid-template-rows: repeat(10, 20px); /* 10 rijen van 20 pixels hoog */
  }

```

```
#maze div {
  border: 1px solid black; /* Maak een zichtbaar raster */
  width: 20px; /* Dit is overbodig en heeft totaal geen invloed */
  height: 20px; /* Dit is overbodig en heeft totaal geen invloed */
}
</style>
```

### 1. #maze: Stijl voor het hoofdcontainer-element

Het eerste blok CSS is gericht op het styling van het element met het ID maze. Dit is de container die het raster bevat.

- #maze geeft aan dat de stijl alleen geldt voor het element met het ID maze.
- display: grid; verandert het element in een **grid container**. Dit betekent dat er een rasterstructuur wordt toegepast.
- grid-template-columns: repeat(10, 20px); definieert hoe breed de kolommen in het raster moeten zijn. In dit geval zijn er 10 kolommen van elk 20 pixels breed.
- grid-template-rows: repeat(10, 20px); definieert hoe hoog de rijen in het raster moeten zijn. Er zijn 10 rijen van elk 20 pixels hoog.

### 2. #maze div: Stijl voor de afzonderlijke cellen

Vervolgens wordt de stijl toegepast op de individuele cellen die in het raster verschijnen:

- #maze div richt zich op elk afzonderlijk div-element binnen het element maze.
- border: 1px solid black; creëert een zichtbare rand rond elke cel.
- width: 20px; en height: 20px; zorgen ervoor dat elke cel een vaste breedte en hoogte heeft van 20 pixels.

#### Stap voor stap werking:

1. **Container veranderen naar een grid:** #maze maakt van het container-element een grid. Dit betekent dat het element is opgedeeld in kolommen en rijen die elk een vaste grootte hebben.
2. **Gridkolommen en -rijen:** grid-template-columns en grid-template-rows definiëren de verdeling van de grid-cellen in termen van breedte en hoogte.
3. **Celdetails:** #maze div zorgt ervoor dat elke individuele cel dezelfde visuele styling heeft: een zwarte rand en een vaste breedte en hoogte van 20 pixels.

## 1.2.2 Flexbox Layout

De flexbox maakt gebruik van een lineaire opmaak waarbij items naast elkaar of onder elkaar worden geplaatst. Het raster wordt door de breedte van de container bepaald.

#### Voordelen:

- Simpel op te zetten voor kleinere tabellen.
- Flexibeler voor eenvoudige opmaak zonder geavanceerde cel-positionering nodig te hebben.
- Ideaal voor eenvoudige lay-outs zoals bijvoorbeeld een lijst met items.

```
<style>
  /* Stijl voor het raster */
  #maze {
    display: flex;
    flex-wrap: wrap;
    width: 200px; /* Breedte voor een 10x10 raster */
  }
  #maze div {
    width: 20px;
    height: 20px;
    border: 1px solid #000; /* Zwarte randen voor de cellen */
    box-sizing: border-box; /* Zorgt ervoor dat randen + padding geen invloed hebben op de cel grootte */
  }
</style>
```

## Overzicht display items:

Waarde	Omschrijving	Gebruiksscenario	Voorbeeld
block	Blokniveau-element	Hele blokken of secties die de volledige breedte innemen	display: block;
inline	Inline-element	Elementen die horizontaal naast andere elementen worden geplaatst	display: inline;
inline-block	Inline met blokniveau stijlen	Elementen met flexibele breedte en hoogte, naast andere inline-elementen	display: inline-block;
flex	Flexbox container	Dynamische lay-out van items met rijen en kolommen	display: flex;
inline-flex	Inline-flexbox container	Inline-flex, met rijen en kolommen zoals flexbox	display: inline-flex;
grid	Grid container	Complexe lay-outs met gerichte kolommen en rijen	display: grid;
inline-grid	Inline-grid container	Grid in inline context	display: inline-grid;
table	Tabular lay-out, maakt van element een tabel	Gebruik voor tabulaire gegevensstructuren	display: table;
inline-table	Inline-tabellen	Tabellen met inline inhoud	display: inline-table;
table-row	Enkelvoudige rij in een tabel	Gebruik voor individuele rijen in tabellen	display: table-row;
table-cell	Cell binnen een tabel	Gebruik voor tabellen met afzonderlijke cellen	display: table-cell;
none	Element volledig verborgen	Verwijdert het element van de DOM	display: none;

Wanneer we de CSS-body toevoegen, dan komt het raster midden op de pagina te staan.

```
<style>
  body {
    font-family: Arial, sans-serif; /* Het lettertype dat wordt gebruikt voor tekst op de webpagina */
    display: flex;                  /* Het body-element wordt omgezet in een flex-container */
    flex-direction: column; /* Flex-items (kinderen van body) worden gerangschikt in een verticale rij (kolom) */
    align-items: center;          /* Flex-items worden horizontaal gecentreerd binnen de container */
    margin: 0;                    /* De standaard marges van het body-element worden verwijderd */
    padding: 20px;                /* Ruimte van 20 pixels rondom de inhoud van het body-element */
  }

  #maze {
    display: grid;
    grid-template-columns: repeat(10, 20px);
    grid-template-rows: repeat(10, 20px);
  }

  #maze div {
    border: 1px solid black;
    box-sizing: border-box;
  }
</style>
```

- De **HTML-body** bevat de inhoud van je pagina, en de **CSS-body** bepaalt hoe die inhoud eruitziet en zich gedraagt.

### font-family: Arial, sans-serif;

- **font-family: Arial, sans-serif;** Dit bepaalt het lettertype dat wordt gebruikt voor de tekst op de webpagina.
  - **Arial** is een sans-serif lettertype, wat betekent dat het geen 'stijlen' aan de letters toevoegt, zoals lijntjes aan de uiteinden van karakters.
  - **sans-serif** valt terug naar het standaard systeemlettertype als Arial niet beschikbaar is.

### display: flex;

- **display: flex;** Hiermee wordt het body-element veranderd in een flexbox-container. Een flexbox maakt het eenvoudiger om inhoud (zoals tekst, afbeeldingen of andere elementen) in een flexibele layout te organiseren.

```
flex-direction: column;
```

- **flex-direction: column;** Binnen de flexbox worden de kinderen (elementen in het body-element) gerangschikt in een verticale rij, van boven naar beneden.

## align-items: center;

- **align-items: center;** Binnen de flexbox worden de items gecentreerd op de verticale as, wat betekent dat ze in het midden van de flexbox worden geplaatst (horizontaal en verticaal gecentreerd).

**margin: 0;**

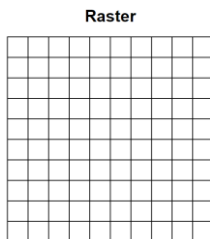
- **margin: 0;:** Dit verwijdert alle standaardmarges die de browser automatisch toevoegt aan het lichaam van de pagina. Hierdoor wordt de volledige inhoud strak tegen de randen van de browser geplaatst.

**padding: 20px;**

- **padding: 20px;** Dit voegt een interne ruimte toe rondom de inhoud van het body-element. De tekst en andere inhoud worden 20 pixels ingesloten van de randen van de browser.

### 1.3 Raster met opmaak

**Het raster staat nu midden op de pagina en is ook groter**



```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Raster</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      display: flex;
      flex-direction: column;
      align-items: center;
      margin: 0;
      padding: 20px;
    }
    #maze {
      display: grid;
      grid-template-columns: repeat(10, 40px); /* Raster: 10 kolommen */
      grid-template-rows: repeat(10, 40px); /* Raster: 10 rijen */
      gap: 0;
      border: 1px solid black; /* Uniforme rand rond het raster */
      box-sizing: border-box; /* Inclusief borders bij de grootte */
    }
    .cell {
      background-color: white; /* Witte achtergrond voor cellen */
      border: 1px solid black; /* Duidelijke randen voor elke cel */
      width: 40px;
      height: 40px;
      box-sizing: border-box; /* Inclusief borders bij de grootte */
    }
  </style>
</html>
```

```

</style>
</head>
<body>
  <h1>Raster</h1>
  <div id="maze">
    <!-- Dynamisch gegenereerde cellen -->
  </div>

  <script>
    const maze = document.getElementById("maze");

    // Raster genereren
    for (let i = 0; i < 100; i++) {
      const cell = document.createElement("div");
      cell.classList.add("cell");
      maze.appendChild(cell);
    }
  </script>
</body>
</html>

```

### Uitleg:

#### **gap: 0;:**

- Dit verwijdert de ruimte (afstand) tussen de rijen en kolommen in het grid. Normaal gesproken voegt gap een standaardruimte toe tussen de grid-items, maar hier wordt dat uitgesloten door gap: 0 toe te passen.

#### **border: 1px solid black;:**

- Dit zorgt ervoor dat een zichtbare rand van 1 pixel dik wordt toegepast rondom elke cel, zowel horizontaal als verticaal. Dit maakt de randen duidelijk zichtbaar en zorgt voor een duidelijke scheiding tussen de cellen.

#### **box-sizing: border-box;:**

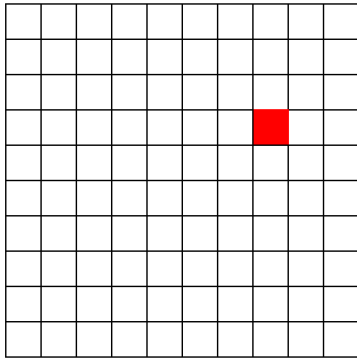
- Dit zorgt ervoor dat de breedte en hoogte van de cellen inclusief hun randen worden gemeten. Dit betekent dat de randen (1px) en inhoud van de cellen deel uitmaken van de totale breedte en hoogte. Dit voorkomt dat de randen en interne marges buiten het beoogde formaat van de cel vallen.

#### **.cell { ... }:**

- Elke individuele cel binnen het grid krijgt dezelfde styling. Dit houdt in dat elke cel een witte achtergrond heeft (background-color: white), een zwarte rand (border: 1px solid black), een vaste breedte en hoogte van 40px (width: 40px en height: 40px), en de genoemde box-sizing.

## 1.4 Raster met bewegend blokje inclusief snelheids-regeling

### Raster en Bewegend Blokje



 Snelheid: 300ms

```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Raster en Bewegend Blokje</title>
</head>
<body>
  <div id="maze-container">
    <div id="maze">
      <div class="cell">
        <div id="red-block">

```

```

    left: 0;
    transition: top 0.3s ease, left 0.3s ease; /* Vloeiende beweging */
  }

  #controls {
    margin-top: 20px;
  }
</style>
</head>
<body>
  <h1>Raster en Bewegend Blokje</h1>
  <div id="maze-container">
    <div id="maze">
      <!-- Dynamisch gegenereerde cellen -->
    </div>
    <div id="red-block"></div>
  </div>
  <div id="controls">
    <input type="range" id="speed" min="100" max="1000" step="100" value="300">
    <label for="speed">Snelheid: <span id="speed-value">300ms</span></label>
  </div>

  <script>
    const maze = document.getElementById("maze");
    const redBlock = document.getElementById("red-block");
    const speedSlider = document.getElementById("speed");
    const speedValue = document.getElementById("speed-value");

    // Raster genereren
    for (let i = 0; i < 100; i++) {
      const cell = document.createElement("div");
      cell.classList.add("cell");
      maze.appendChild(cell);
    }

    // Variabelen voor het rode blokje
    let currentRow = 0;
    let currentCol = 0;
    let direction = 1; // Richting: 1 voor rechts, -1 voor links
    let interval = null;

    // Functie om het blokje te verplaatsen
    const moveBlock = () => {
      redBlock.style.top = `${currentRow * 40}px`;
      redBlock.style.left = `${currentCol * 40}px`;

      // Richting aanpassen als de rand wordt bereikt
      if (direction === 1 && currentCol >= 9) {
        direction = -1; // Keer om naar links
        currentRow++; // Ga naar de volgende rij
      } else if (direction === -1 && currentCol <= 0) {
        direction = 1; // Keer om naar rechts
        currentRow++; // Ga naar de volgende rij
      } else {
        currentCol += direction; // Beweeg in de huidige richting
      }

      // Stoppen als we het raster verlaten
      if (currentRow >= 10) {
        currentRow = 0; // Herstart bovenaan
      }
    };
  </script>

```

```

    currentCol = 0;
    direction = 1; // Begin opnieuw naar rechts
  }
};

// Snelheid aanpassen
const updateSpeed = () => {
  clearInterval(interval); // Stop oude interval
  const speed = parseInt(speedSlider.value);
  speedValue.textContent = `${speed}ms`;
  interval = setInterval(moveBlock, speed); // Start nieuwe interval
};

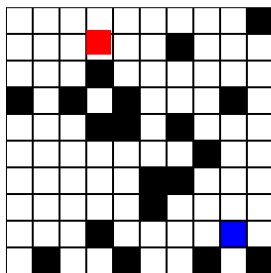
// Initialisatie
speedSlider.addEventListener("input", updateSpeed);
updateSpeed(); // Start animatie met standaard snelheid
</script>
</body>
</html>

```

## 1.5 Blokje zoekt (zelf) een blauwblokje in een doolhof

Hieronder is een voorbeeld gegeven van een doolhof waarin het rode blokje zelf opzoek gaat naar het blauwe blokje.

### Doolhof met Blokje



```

<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <title>Doolhof met Blokje</title>
  <style>
    /* Stijl voor het doolhof */
    .maze {
      display: flex;
      flex-wrap: wrap;
      width: 220px;
    }

    .maze .cell {
      width: 20px;
      height: 20px;
      border: 1px solid #000;
      background-color: #fff;
      position: relative;
    }

    /* Stijl voor het blokje */
    #block {
      width: 20px;

```



```

    height: 20px;
    background-color: red;
    position: absolute;
    transition: left 0.2s, top 0.2s;
  }
</style>
</head>
<body>

<h1>Doolhof met Blokje</h1>

<!-- Container voor het doolhof -->
<div id="mazeContainer"></div>

<script>
  // Functie om een doolhof te genereren
  function generateMaze(rows, columns) { /* Algoritme genereren doolhof */
    const mazeContainer = document.getElementById('mazeContainer');
    mazeContainer.innerHTML = ""; // Wis het vorige doolhof
    const maze = document.createElement('div');
    maze.classList.add('maze');
    maze.style.width = `${columns * 22}px`; // Voeg extra ruimte toe voor de randen

    for (let row = 0; row < rows; row++) {
      for (let col = 0; col < columns; col++) {
        const cell = document.createElement('div');
        cell.classList.add('cell');
        // Zorg ervoor dat het eerste hokje nooit zwart is
        if ((row === 0 && col === 0)) {
          cell.style.backgroundColor = 'fff'; // Startpositie blijft wit
        } else if ((row === rows - 1 && col === columns - 1) || Math.random() < 0.2) {
          cell.style.backgroundColor = '#000'; // Muur
        } else if (row === rows - 2 && col === columns - 2) {
          cell.style.backgroundColor = 'blue'; // Uitgang
        }
        maze.appendChild(cell);
      }
    }
    mazeContainer.appendChild(maze);
  }

  // Functie om het blokje te laten bewegen
  function moveBlock() { /* Algoritme zoeken blauwe blokje */
    const block = document.getElementById('block');
    const mazeCells = document.querySelectorAll('.cell');
    const exitCell = document.querySelector('.maze .cell:last-child'); // Laatste cel is de uitgang
    const blueCell = document.querySelector('.maze .cell[style="background-color: blue;"]'); // Blauwe cel

    let currentCellIndex = 0; // Startpositie van het blokje
    let currentCell = mazeCells[currentCellIndex];
    block.style.left = currentCell.offsetLeft + 'px';
    block.style.top = currentCell.offsetTop + 'px';

    const moveInterval = setInterval(() => {
      if (currentCell === exitCell) {
        clearInterval(moveInterval);
        alert('Gefeliciteerd! Het blokje heeft de uitgang gevonden.');
```

```

clearInterval(moveInterval);
alert('Gefeliciteerd! Het blokje heeft het blauwe hokje gevonden.');
```

```

generateMaze(10, 10); // Nieuw doolhof genereren
moveBlock(); // Start het spel opnieuw
} else {
  const possibleMoves = [
    mazeCells[currentCellIndex + 1], // Rechts
    mazeCells[currentCellIndex + 10], // Onder
    mazeCells[currentCellIndex - 1], // Links
    mazeCells[currentCellIndex - 10], // Boven
  ];
  const validMoves = possibleMoves.filter(move => move &&
!move.style.backgroundColor.includes('rgb(0, 0, 0)') && (move.offsetLeft === currentCell.offsetLeft ||
move.offsetTop === currentCell.offsetTop));
  const nextMove = validMoves[Math.floor(Math.random() * validMoves.length)];

  if (nextMove) {
    currentCell = nextMove;
    currentCellIndex = Array.from(mazeCells).indexOf(currentCell);
    block.style.left = currentCell.offsetLeft + 'px';
    block.style.top = currentCell.offsetTop + 'px';
  }
}
}, 500);
}
// Genereer het doolhof en laat het blokje bewegen bij het laden van de pagina
window.onload = function() {
  generateMaze(10, 10); // 10x10 doolhof
  moveBlock();
};
</script>

<!-- Het blokje -->
<div id="block"></div>
</body>
</html>

```

Laten we de HTML-code en de werking van het doolhof en het algoritme voor het rode blokje dat op zoek gaat naar het blauwe blokje uitleggen.

## 1. Opbouw van het Raster/Doolhof:

Het gebruikte algoritme om het doolhof te genereren, kan worden aangeduid als een random maze generation algorithm met probabilistische muurplaatsing. Dit algoritme genereert willekeurig muren door een bepaalde kans (zoals 20%) te gebruiken om zwarte cellen toe te voegen in plaats van vrije cellen.

- **Grid/Cellen:**
  - De grid van het doolhof bestaat uit cellen (div.cell), die een breedte en hoogte van 20 pixels hebben. Deze cellen worden weergegeven met een zwarte rand van 1 pixel en een achtergrondkleur, die kan variëren afhankelijk van de functie van de cel (start, muur, uitgang).
- **Zwarte muren:**
  - De zwarte cellen worden willekeurig gegenereerd met een kans van 20%, via de check `Math.random() < 0.2`. Dit zorgt voor het blokkeren van doorgang, en deze cellen krijgen een zwarte achtergrond (`cell.style.backgroundColor = '#000'`).

- **Startpositie en Uitgang:**

- De startpositie wordt altijd wit (`cell.style.backgroundColor = '#fff';`) en de uitgang wordt als blauw aangegeven (`cell.style.backgroundColor = 'blue';`).

## 2. Algoritme van het Rode Blokje:

Het gebruikte algoritme om het blauwe blokje te vinden, is een willekeurig pad zoekalgoritme gecombineerd met een depth-first search (DFS)-achtige benadering. Hierbij wordt willekeurig een van de aangrenzende cellen gekozen, mits deze niet geblokkeerd is (niet zwart).

- Het rode blokje (`#block`) probeert door het doolhof te bewegen door aangrenzende geldige cellen te volgen.
- **Bewegingen:**
  - De mogelijke bewegingen zijn naar rechts, beneden, links of boven, afhankelijk van welke cel niet geblokkeerd wordt door een zwarte muur (`!move.style.backgroundColor.includes('rgb(0, 0, 0)')`).
- **Validatie van Bewegingen:**
  - Het algoritme valideert de mogelijke nieuwe posities door te controleren of ze niet een zwarte muur bevatten (`!move.style.backgroundColor.includes('rgb(0, 0, 0)')`) en of ze valide grenzen volgen.
- **Interval:**
  - De bewegingen worden geregeld door een interval van 500 milliseconden, waarbij elke stap een nieuwe positie wordt gekozen uit de geldige aangrenzende cellen.

## 3. Werking van het Algoritme:

- **Startpositie:**
  - Het blokje begint op de cel `mazeCells[0]`, wat correspondeert met de bovenste linkerbovenhoek van het doolhof.
- **Mogelijkheden:**
  - Het algoritme genereert een lijst van mogelijke volgende cellen om te bewegen (`possibleMoves`), controleert of ze geldig zijn, en kiest een willekeurige volgende stap (`nextMove`).
- **Einddoelen:**
  - Het blokje stopt en genereert een nieuw doolhof wanneer het de uitgang of het blauwe doelwit bereikt. Deze doelen stoppen de interval met `clearInterval(moveInterval)` en geven een alert om de overwinning aan te geven.

### Hoe weet het rode blokje dat hij het blauwe blokje heeft gevonden?

Het rode blokje weet dat het blauwe blokje is gevonden door middel van een eenvoudige controle of het zich bevindt op dezelfde cel als de blauwe cel. Dit wordt bereikt met de volgende code:

```
const blueCell = document.querySelector('.maze .cell[style="background-color: blue;"]'); // Blauwe cel
```

Hiermee wordt de cel met de blauwe achtergrond geïdentificeerd. Vervolgens vergelijkt het algoritme de huidige positie van het rode blokje met de positie van de blauwe cel:

```
if (currentCell === blueCell) {
  clearInterval(moveInterval);
}
```

```

alert('Gefeliciteerd! Het blokje heeft het blauwe hokje gevonden.');
```

```

generateMaze(10, 10); // Nieuw doolhof genereren
moveBlock(); // Start het spel opnieuw
}

```

- **Controle:**

- `currentCell === blueCell` controleert of de huidige cel waarin het rode blokje zich bevindt, dezelfde cel is als de blauwe cel. Als dit waar is, stopt het algoritme en wordt er een melding weergegeven.

Dit zorgt ervoor dat zodra het rode blokje zich in de cel bevindt met blauwe achtergrond, het algoritme stopt en een overwinning wordt gemeld.

#### En verder....

- **Grid opbouw:** Er worden cellen gegenereerd, met een variatie aan achtergrondkleuren (wit voor start, blauw voor eind, zwart voor muren).
- **Rode Blokje:** Beweegt door het doolhof met een algoritme dat aangrenzende cellen valideert, en beweegt willekeurig naar een geldige positie, stoppen wanneer de blauwe cel of uitgang wordt bereikt.

Tijdens het laden van de pagina, genereert de `generateMaze(10, 10)` een 10x10 doolhof en begint de functie `moveBlock()` om het blokje te bewegen.

## 1.6 De 3 (zelf) bewegende ballen

Drie ballen worden gecreëerd die continu bewegen, waarbij hun positie in elk frame wordt bijgewerkt. Het canvas, de ruimte waarin de ballen zich bewegen, wordt bij elk nieuw frame schoongemaakt. In elk frame worden de ballen opnieuw getekend op hun bijgewerkte positie. De nieuwe positie van elke bal wordt berekend door de huidige positie (x, y) te veranderen met de snelheid (dx, dy) van die bal. Als een bal een rand van het canvas bereikt, een botsing, keert de bewegingsrichting om.



HTML:

```

<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Stuiterende Ballen</title>
  <style>
    canvas {
      display: block;
      margin: 0 auto;
      background-color: #f0f0f0;
    }
  </style>
</head>
<body>

```

```

<canvas id="myCanvas" width="800" height="600"></canvas>
<script src="script.js"></script>
</body>
</html>

```

### Java script script.js

```

// Klasse voor de Ballen
class Bal {
  constructor(x, y, dx, dy, radius, kleur) {
    this.x = x;
    this.y = y;
    this.dx = dx; // Snelheid in x-richting
    this.dy = dy; // Snelheid in y-richting
    this.radius = radius;
    this.kleur = kleur;
  }
  // Methode om de bal te tekenen op de huidige positie
  teken(context) {
    context.beginPath();
    context.arc(this.x, this.y, this.radius, 0, Math.PI * 2, false);
    context.fillStyle = this.kleur;
    context.fill();
    context.closePath();
  }
  // Methode om de bal te laten bewegen = volgende positie
  beweeg(canvas) {
    if (this.x + this.radius > canvas.width || this.x - this.radius < 0) {
      this.dx = -this.dx;
    }
    if (this.y + this.radius > canvas.height || this.y - this.radius < 0) {
      this.dy = -this.dy;
    }
    this.x += this.dx;
    this.y += this.dy;
  }
}

// Initialiseren van het canvas en context
const canvas = document.getElementById('myCanvas');
const context = canvas.getContext('2d');

// Ballen creëren
const ballen = [
  new Bal(100, 100, 2, 3, 30, 'red'),
  new Bal(200, 200, 3, 2, 30, 'green'),
  new Bal(300, 300, 4, 3, 30, 'black')
];

// Animatie functie
function animatie() {
  context.clearRect(0, 0, canvas.width, canvas.height); // Canvas schoonmaken

  ballen.forEach(bal => {
    bal.teken(context); // Bal tekenen, op nieuwe positie.
    bal.beweeg(canvas); // Bal bewegen, dx en dy worden met 1 verhoogd = volgende positie
  });

  requestAnimationFrame(animatie); // Volgende frame aanroepen
}

// Start de animatie
animatie();

```

### 1.6.1 Beschrijving van de werking van het JavaScript

Het script beschrijft de werking van een eenvoudig systeem waarin meerdere **ballen** (cirkelvormige objecten) op een canvas worden weergegeven en bewogen. Hier is een gedetailleerde beschrijving van de werking:

### 1.6.2 OOP in Javascript

De opkomst van objectgeoriënteerd programmeren (OOP) was een paradigmaverschuiving, dat wil zeggen dat een totaal andere stijl van programmeren is.

**Voordelen van het gebruik van een class in JavaScript:**

1. **Herbruikbaarheid:**
  - Met een class kun je eenvoudig herbruikbare code creëren. Je definieert eenmalig een sjabloon (template) en kunt dit herhaaldelijk gebruiken voor meerdere objecten met dezelfde structuur en gedrag.
2. **Code-organisatie:**
  - Door het gebruik van classes wordt je code beter georganiseerd en leesbaarder. Objecten met dezelfde structuur en methoden worden gecentraliseerd in een enkele class definitie.
3. **Encapsulatie:**
  - Klassen bieden een manier om data en methoden samen te bundelen, waardoor data beschermd is tegen directe toegang en manipulatie. Dit verhoogt de stabiliteit en integriteit van het object.
4. **Oplossen van complexe problemen:**
  - Klassen maken het mogelijk om complexere systemen te modelleren door objecten te creëren die specifieke gedrag patronen hebben, zoals interacties tussen meerdere objecten.
5. **Onderhoudbaarheid:**
  - Bij wijzigingen aan de structuur van een class, hoeft alleen de class zelf aangepast te worden. Alle objecten die van de class afstammen, profiteren automatisch van deze wijzigingen.

### 1.6.3 Opbouw van een class

```
// Klasse voor de Ballen
class Bal {
  /* Properties: De bal moet een positie, snelheid en een kleur hebben */
  constructor(x, y, dx, dy, radius, kleur) {
  }
  /* Methodes: De bal moet getekend worden en moet kunnen bewegen */
  teken(context) {
  }
  beweeg(canvas) {
  }
}
```

#### **Klasse**

Een **klasse** in JavaScript bestaat uit verschillende onderdelen, waaronder een naam, een constructor, properties, en methoden.

- **Naam**

De naam van de class is Bal.

- **Constructor:**

Doel van de constructor: De constructor initialiseert een bal met de opgegeven posities, snelheden, radius en kleur. Dit zijn de zogenaamde **properties**. Het zorgt ervoor dat een nieuw Bal-object gecreëerd kan worden met specifieke eigenschappen zoals positie, snelheid en visuele kenmerken.

```
constructor(x, y, dx, dy, radius, kleur) {  
    this.x = x;  
    this.y = y;  
    this.dx = dx; // Snelheid in x-richting  
    this.dy = dy; // Snelheid in y-richting  
    this.radius = radius;  
    this.kleur = kleur;  
}
```

- **Properties:**

Het doel van de properties in dit geval is om de essentiële eigenschappen van een bal vast te leggen, zoals zijn positie, snelheid, afmeting en kleur. Dit maakt het mogelijk om de staat van de bal te beheren en te manipuleren in relatie tot zijn beweging en visuele representatie binnen een applicatie.

1. **x**: De **x**-coördinaat van de positie van de bal.
2. **y**: De **y**-coördinaat van de positie van de bal.
3. **dx**: De snelheid in de **x**-richting (richting en snelheid).
4. **dy**: De snelheid in de **y**-richting (richting en snelheid).
5. **radius**: De straal van de bal.
6. **kleur**: De kleur van de bal.

- **Methodes:**

Doel van de methode: Visualisatie van de bal door hem te tekenen en te laten bewegen met zijn opgegeven eigenschappen.

**teken(context) = huidige positie:**

- Doel: De bal wordt op het canvas getekend.
- Beschrijving:
  - Begin een pad met context.beginPath().
  - Teken een cirkel met context.arc(this.x, this.y, this.radius, 0, Math.PI \* 2, false).
  - Vul de cirkel met de opgegeven kleur via context.fillStyle.
  - Vul het pad met context.fill().
  - Sluit het pad af met context.closePath().

```
// Methode om de bal te tekenen  
teken(context) {  
    context.beginPath();  
    context.arc(this.x, this.y, this.radius, 0, Math.PI * 2, false);  
    context.fillStyle = this.kleur;  
    context.fill();  
    context.closePath();  
}
```

**beweeg(canvas) wordt toekomstige positie**

De methode heeft als doel om de bal real-time te verplaatsen op basis van zijn snelheid (dx en dy), terwijl het ervoor zorgt dat de bal niet uit het canvas verdwijnt.

```
// Methode om de bal te laten bewegen  
beweeg(canvas) {  
    if (this.x + this.radius > canvas.width || this.x - this.radius < 0) {  
        this.dx = -this.dx;  
    }  
}
```

```

    if (this.y + this.radius > canvas.height || this.y - this.radius < 0) {
        this.dy = -this.dy;
    }
    this.x += this.dx;
    this.y += this.dy;
}

```

## Het initialiseren van het canvas en van de context

```

// Initialiseren van het canvas en context
const canvas = document.getElementById('myCanvas');
const context = canvas.getContext('2d');

```

### Doel van het initialiseren van het canvas en context:

- **Canvas Selectie:**
  - Het canvas-element wordt geselecteerd door zijn id te vinden met behulp van `document.getElementById('myCanvas')`. Dit geeft toegang tot het HTML <canvas>-element in het DOM (Document Object Model), waarin grafische inhoud kan worden getekend.
- **Context Opvragen:**
  - Met `canvas.getContext('2d')` wordt de 2D-context van het canvas verkregen. Deze context is nodig voor het uitvoeren van 2D-graphics en biedt methoden zoals `beginPath()`, `arc()`, `fill()` en andere om grafische elementen zoals lijnen, cirkels, vierkanten, en tekst te tekenen.
- **Waarom dit nodig is:**
  - Zonder het initialiseren van het canvas en het verkrijgen van de 2D-context, kan er geen grafische output plaatsvinden op het canvas. Dit stelt het script in staat om interactie te hebben met het canvas, objecten zoals balobjecten te tekenen en bewegingen te simuleren.
  - Dit proces is cruciaal voor het uitvoeren van animaties en andere grafische manipulaties op het web.

## Het creëren van de 3 ballen

Hier wordt de rode, groene en zwarte bal gecreëerd. De radius van elke bal is even groot. De ballen starten op verschillende posities en hebben verschillende snelheden.

```

// Ballen creëren
const ballen = [
    new Bal(100, 100, 2, 3, 30, 'red'),
    new Bal(200, 200, 3, 2, 30, 'green'),
    new Bal(300, 300, 4, 3, 30, 'black')
];

```

### • Animatie:

De animatie()-functie:

- Schoont het canvas door `clearRect()` te gebruiken om de voorgaande frame te verwijderen.
- Lijkt op elke iteratie elke bal te tekenen door de `teken()`-methode te gebruiken en vervolgens te verplaatsen met behulp van de `beweeg()`-methode.
- `requestAnimationFrame(animatie)` wordt gebruikt om de animatie aan te roepen en een soepel, voortdurend bewegende animatie mogelijk te maken.
- **Verloop**
  - De `teken()`-methode tekent elke bal op zijn nieuwe positie.
  - De `beweeg()`-methode controleert of de bal tegen de randen van het canvas botst. Als dit het geval is, keert de richting van de bal om (dx of dy wordt omgekeerd).
  - Vervolgens wordt op elke frame de animatie herhaald met behulp van `requestAnimationFrame()`.



Door deze opzet blijft het canvas voortdurend worden bijgewerkt en blijven de ballen bewegen binnen het canvas.

- **Botsing:**

De botsing tussen de wanden van het canvas en de bal wordt bepaald door het controleren of de positie van de bal (gegeven door zijn **x**- en **y**-coördinaten) overeenkomt met de randen van het canvas.

Laten we dit stap voor stap bekijken:

**Hoe de botsing wordt berekend:**

- **Berekeningen voor botsingen met randen:**

- **Horizontale botsing:**

- De bal botst tegen de linker- of rechterkant van het canvas wanneer zijn **x**-positie plus zijn **radius** groter is dan de breedte van het canvas (`canvas.width`) of kleiner dan 0.
    - Dit wordt gecontroleerd in de `beweeg(canvas)`-methode met de volgende if-voorwaarde:

```
if (this.x + this.radius > canvas.width || this.x - this.radius < 0) {  
    this.dx = -this.dx; // Snelheid omkeren in x-richting  
}
```

- **Verticale botsing:**

- De bal botst tegen de boven- of onderkant van het canvas wanneer zijn **y**-positie plus zijn **radius** groter is dan de hoogte van het canvas (`canvas.height`) of kleiner dan 0.
    - Dit wordt gecontroleerd in dezelfde `beweeg(canvas)`-methode met de volgende if-voorwaarde:

```
if (this.y + this.radius > canvas.height || this.y - this.radius < 0) {  
    this.dy = -this.dy; // Snelheid omkeren in y-richting  
}
```

- **Wat gebeurt er bij botsing?:**

- Wanneer een botsing plaatsvindt met een van de randen:

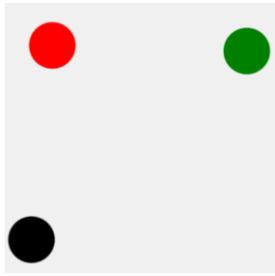
- **Horizontaal:** De snelheid in de **x**-richting (`dx`) wordt omgekeerd (`this.dx = -this.dx`).
    - **Verticaal:** De snelheid in de **y**-richting (`dy`) wordt omgekeerd (`this.dy = -this.dy`).

- **Waar in de software wordt dit berekend?**

- De berekeningen voor botsingen met de randen vinden plaats in de `beweeg(canvas)`-methode van de `Bal`-klasse. Dit gebeurt als eerste in de methode, voordat de bal wordt verplaatst. Hier wordt gecontroleerd of de bal de randen van het canvas bereikt en zo ja, worden de snelheidscomponenten omgekeerd om de botsing te simuleren.

## 1.7 Drie tegen elkaar botsende ballen

Deze code voegt botsingen tussen de ballen toe. Dit maakt het gedrag van de ballen complexer en realistischer.



**Code:**

```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Stuiterende Ballen met Botsing</title>
  <style>
    canvas {
      display: block;
      margin: 0 auto;
      background-color: #f0f0f0;
    }
  </style>
</head>
<body>
  <canvas id="myCanvas" width="800" height="600"></canvas>
  <script>
    class Bal {
      constructor(x, y, dx, dy, radius, kleur) {
        this.x = x;
        this.y = y;
        this.dx = dx;
        this.dy = dy;
        this.radius = radius;
        this.kleur = kleur;
      }

      teken(context) {
        context.beginPath();
        context.arc(this.x, this.y, this.radius, 0, Math.PI * 2, false);
        context.fillStyle = this.kleur;
        context.fill();
        context.closePath();
      }

      beweeg(canvas) {
        if (this.x + this.radius > canvas.width || this.x - this.radius < 0) {
          this.dx = -this.dx;
        }
        if (this.y + this.radius > canvas.height || this.y - this.radius < 0) {
          this.dy = -this.dy;
        }
        this.x += this.dx;
        this.y += this.dy;
      }
    }
  </script>
</body>
</html>
```

```

botsMetAndereBal(andereBal) { /* controleren of twee ballen met elkaar botsen */
  const dx = andereBal.x - this.x;
  const dy = andereBal.y - this.y;
  const afstand = Math.sqrt(dx * dx + dy * dy);
  return afstand < this.radius + andereBal.radius;
}

behandelBotsing(andereBal) { /* bewegingen veranderen bij botsing tussen 2 ballen */
  const dx = andereBal.x - this.x;
  const dy = andereBal.y - this.y;

  const afstand = Math.sqrt(dx * dx + dy * dy);
  const eenheidX = dx / afstand;
  const eenheidY = dy / afstand;

  const snelheidDit = this.dx * eenheidX + this.dy * eenheidY;
  const snelheidAnder = andereBal.dx * eenheidX + andereBal.dy * eenheidY;

  this.dx -= snelheidDit * eenheidX;
  this.dy -= snelheidDit * eenheidY;
  andereBal.dx -= snelheidAnder * eenheidX;
  andereBal.dy -= snelheidAnder * eenheidY;

  this.dx += snelheidAnder * eenheidX;
  this.dy += snelheidAnder * eenheidY;
  andereBal.dx += snelheidDit * eenheidX;
  andereBal.dy += snelheidDit * eenheidY;
}
}

const canvas = document.getElementById('myCanvas');
const context = canvas.getContext('2d');

const ballen = [
  new Bal(100, 100, 2, 3, 30, 'red'),
  new Bal(200, 200, 3, 2, 30, 'green'),
  new Bal(300, 300, 4, 3, 30, 'black')
];
function animatie() {
  context.clearRect(0, 0, canvas.width, canvas.height);

  for (let i = 0; i < ballen.length; i++) {
    for (let j = i + 1; j < ballen.length; j++) {
      if (ballen[i].botsMetAndereBal(ballen[j])) {
        ballen[i].behandelBotsing(ballen[j]);
      }
    }
  }

  ballen.forEach(bal => {
    bal.teken(context);
    bal.beweeg(canvas);
  });

  requestAnimationFrame(animatie);
}
animatie();
</script>
</body>
</html>

```

## Wat gebeurt er wanneer 2 ballen tegen elkaar aan botsen?

Wanneer twee ballen in dit programma tegen elkaar botsen, gebeurt het volgende:

- **De botsing wordt gedetecteerd**

De methode **botsMetAndereBal** wordt gebruikt om te controleren of de twee ballen elkaar raken. Dit gebeurt door de afstand tussen de centra van de twee ballen te berekenen. Als deze afstand kleiner is dan de som van hun stralen (radius), betekent dit dat ze botsen.

### Formule voor botsdetectie:

$$\text{Botsing} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} < (r_1 + r_2)$$

### Betekenis van de variabelen:

- $x_1, y_1$  : De x- en y-coördinaten van het centrum van de eerste bal.
- $x_2, y_2$  : De x- en y-coördinaten van het centrum van de tweede bal.
- $r_1$  : De straal (radius) van de eerste bal.
- $r_2$  : De straal (radius) van de tweede bal.
- $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ : De afstand tussen de twee ballen, berekend met de stelling van Pythagoras.
- $r_1 + r_2$ : De som van de stralen van de twee ballen.

### Logica:

Als de afstand tussen de twee ballen ( $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ ) kleiner is dan de som van hun stralen ( $r_1 + r_2$ ), dan raken de ballen elkaar en vindt er een botsing plaats.

In code wordt dit als volgt geïmplementeerd:

```
botsMetAndereBal(andereBal) {  
  const dx = andereBal.x - this.x;  
  const dy = andereBal.y - this.y;  
  const afstand = Math.sqrt(dx * dx + dy * dy);  
  return afstand < (this.radius + andereBal.radius);  
}
```

### Formules voor botsingsfysica:

1. **Behoud van impuls:** De impuls in de richting van de botsing wordt behouden. Dit betekent dat de totale impuls vóór de botsing gelijk blijft aan de totale impuls ná de botsing.

$$m_1 v_{1i} + m_2 v_{2i} = m_1 v_{1f} + m_2 v_{2f}$$

waarbij:

- $m_1$  en  $m_2$  de massa's van de twee ballen zijn.
- $v_{1i}$  en  $v_{2i}$  de snelheden vóór de botsing zijn.
- $v_{1f}$  en  $v_{2f}$  de snelheden ná de botsing zijn.

**Behoud van kinetische energie:** De totale kinetische energie vóór en ná de botsing blijft behouden onder ideale omstandigheden, tenzij er dissipatie (zoals wrijving) is.

$$\frac{1}{2} m_1 v_{1i}^2 + \frac{1}{2} m_2 v_{2i}^2 = \frac{1}{2} m_1 v_{1f}^2 + \frac{1}{2} m_2 v_{2f}^2$$

### Formule voor stuiterende snelheden:

Om de nieuwe snelheden van de ballen na de botsing te berekenen, kan de volgende afleiding worden gebruikt:

$$v_{1f} = \frac{(m_1 - m_2)v_{1i} + 2m_2v_{2i}}{m_1 + m_2}$$

$$v_{2f} = \frac{(m_2 - m_1)v_{2i} + 2m_1v_{1i}}{m_1 + m_2}$$

Waarbij:

- $v_{1f}$  en  $v_{2f}$  de snelheden na de botsing zijn.
- $v_{1i}$  en  $v_{2i}$  de snelheden vóór de botsing zijn.
- $m_1$  en  $m_2$  de massa's van respectievelijk de eerste en tweede bal zijn.

### In de praktijk:

In de code wordt gebruik gemaakt van een variant van bovenstaande concepten waarbij alleen de snelheden in de richting van de botsing worden aangepast, door eenheid vectoren te gebruiken:

$$v_{1f} = v_{1i} - (v_{1i} \cdot \text{eenheidX}) \times \text{eenheidX}$$

$$v_{2f} = v_{2i} - (v_{2i} \cdot \text{eenheidX}) \times \text{eenheidX}$$

### Code: Methode voor het behandelen van botsing

```
behandelBotsing(andereBal) {  
  const dx = andereBal.x - this.x;  
  const dy = andereBal.y - this.y;  
  const afstand = Math.sqrt(dx * dx + dy * dy);  
  const eenheidX = dx / afstand;  
  const eenheidY = dy / afstand;  
  
  const snelheidDit = this.dx * eenheidX + this.dy * eenheidY;  
  const snelheidAnder = andereBal.dx * eenheidX + andereBal.dy * eenheidY;  
  
  this.dx -= snelheidDit * eenheidX;  
  this.dy -= snelheidDit * eenheidY;  
  andereBal.dx -= snelheidAnder * eenheidX;  
  andereBal.dy -= snelheidAnder * eenheidY;  
  
  this.dx += snelheidAnder * eenheidX;  
  this.dy += snelheidAnder * eenheidY;  
  andereBal.dx += snelheidDit * eenheidX;  
  andereBal.dy += snelheidDit * eenheidY;  
}
```

### Uitleg:

#### Afstand en eenheidvector:

- De afstand tussen de centra van de ballen wordt berekend.
- De eenheidvector wordt berekend om de richting van de botsing te bepalen.

#### Snelheid aanpassen:

- De snelheden in de botsingsrichting worden afzonderlijk aangepast voor beide ballen, zodat ze elkaar "stuiteren".
- Wat er gebeurt wanneer alle ballen tegelijkertijd met elkaar botsen

Wanneer drie ballen tegelijk tegen elkaar botsen, worden de botsingen op een manier verwerkt die rekening houdt met de interactie tussen alle drie de ballen. Dit kan complexer zijn dan de botsingen van twee ballen, omdat er meerdere snelheidsaanpassingen nodig zijn om de juiste nieuwe snelheden te berekenen.

#### **Proces van botsing tussen drie ballen:**

##### **1. Botsingsdetectie:**

- Voor elk paar van de drie ballen wordt gecontroleerd of ze botsen door de afstand tussen hun centra te berekenen.
- Dit wordt gedaan door vergelijkbare logica als in de methode `botsMetAndereBal()`.

##### **2. Fysica van botsingen:**

- Voor elk paar dat botst, wordt de `behandelBotsing()`-methode gebruikt om de snelheden aan te passen zoals eerder beschreven.

##### **3. Interactie tussen meerdere ballen:**

- Als meerdere ballen tegelijkertijd botsen, wordt de snelheid van elke bal aangepast op basis van de interactie met de andere betrokken ballen.
- Dit betekent dat elk paar dat botst, afzonderlijk zijn snelheden aanpast, en vervolgens worden de aangepaste snelheden gebruikt om verdere interacties te verwerken.

#### **Implementatie in code:**

Bijvoorbeeld, in een animatiefunctie waar meerdere ballen bewegen, kan de botsingsdetectie plaatsvinden tussen elk paar ballen (combinaties van 3 ballen). Daarna worden de botsingsmethodes per paar toegepast.

#### **Bijvoorbeeld:**

```
function animatie() {
  context.clearRect(0, 0, canvas.width, canvas.height);

  for (let i = 0; i < ballen.length; i++) {
    for (let j = i + 1; j < ballen.length; j++) {
      if (ballen[i].botsMetAndereBal(ballen[j])) {
        ballen[i].behandelBotsing(ballen[j]);
      }
    }
  }

  ballen.forEach(bal => {
    bal.teken(context);
    bal.beweeg(canvas);
  });
  requestAnimationFrame(animatie);
}
```

Wanneer drie ballen botsen, wordt de methode `behandelBotsing()` voor elk paar uitgevoerd, wat resulteert in meerdere aanpassingen van de snelheden. De uiteindelijke beweging van elke bal zal rekening houden met de interactie met de andere twee ballen.

## 2. Met interactie en gebruik makend van THREE.JS

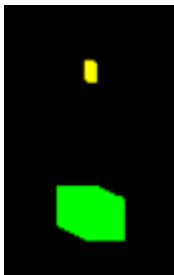
Hierbij een link naar 2 boeken (.pdf) waarin THREE.JS wordt uitgelegd met interessante voorbeelden :

[threejs tutorial.pdf](#)

[Learning Three.js\\_ The JavaScript 3D Library for WebGL - Second Edition \( PDFDrive \).pdf](#)

Ik heb getracht om de voorbeelden die ik gegeven heb zoveel mogelijk uit te leggen, maar uiteindelijk moet je het wel zelf doen. Advies is dan ook als je iets niet snapt of je wilt je er meer in verdiepen maak gebruik van GOOGLE of CHATGPT.

### 2.1 Eenvoudig 3d- kanon die kan schieten en bewegen



#### Code:

```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Kanon</title>
  <style>
    body { margin: 0; overflow: hidden; }
    canvas { display: block; }
  </style>
</head>
<body>
  <script src="https://cdn.jsdelivr.net/npm/three.js@r128/three.min.js"></script>
  <script>
    // Scene, Camera en Renderer instellen
    const scene = new THREE.Scene();
    const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
    const renderer = new THREE.WebGLRenderer();
    renderer.setSize(window.innerWidth, window.innerHeight);
    document.body.appendChild(renderer.domElement);

    // Lichten toevoegen
    const light = new THREE.DirectionalLight(0xffffff, 1);
    light.position.set(1, 1, 1).normalize();
    scene.add(light);

    // Kanon maken
    const cannonGeometry = new THREE.BoxGeometry(1, 1, 1);
    const cannonMaterial = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
    const cannon = new THREE.Mesh(cannonGeometry, cannonMaterial);
    cannon.position.set(0, -10, 0);
    scene.add(cannon);

    // Kogel maken
```

```
const bulletGeometry = new THREE.BoxGeometry(0.2, 0.5, 0.2);
const bulletMaterial = new THREE.MeshBasicMaterial({ color: 0xffff00 });
let bullets = []; // Array voor meerdere kogels
let bulletSpeed = 0.5; // Snelheid van de kogels

// Camera positie
camera.position.z = 30;

// Eventlisteners voor toetsenbordinput
document.addEventListener('keydown', function(event) {
  if (event.key === 'z' || event.key === 'Z') {
    cannon.position.x -= 0.5; // Beweeg kanon naar links
  }
  if (event.key === 'm' || event.key === 'M') {
    cannon.position.x += 0.5; // Beweeg kanon naar rechts
  }
  if (event.code === 'Space') {
    // Maak een nieuwe kogel en voeg toe aan het array
    const newBullet = new THREE.Mesh(bulletGeometry, bulletMaterial);
    newBullet.position.set(cannon.position.x, cannon.position.y + 1, 0);
    newBullet.visible = true;
    bullets.push(newBullet);
    scene.add(newBullet);
  }
});

// Functie om kogels te verwijderen als ze uit beeld zijn
function updateBullets() {
  bullets = bullets.filter(bullet => bullet.visible);
}

// Animatie loop
function animate() {
  requestAnimationFrame(animate);

  // Beweeg de kogels omhoog
  bullets.forEach(bullet => {
    bullet.position.y += bulletSpeed;
    if (bullet.position.y > 15) {
      bullet.visible = false; // Verberg de kogel als het buiten beeld is
    }
  });

  updateBullets(); // Verwijder onzichtbare kogels

  renderer.render(scene, camera);
}

animate();
</script>
</body>
</html>
```

### 2.1.1 Three.js gebruik

<https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js> is de URL van een gedeelde versie van de bibliotheek **Three.js**, die een JavaScript-API biedt voor 3D-graphics in een browseromgeving.



## 2.1.2 Wat is Three.js?

Three.js is een populaire bibliotheek waarmee ontwikkelaars interactieve 3D-graphics kunnen maken in webapplicaties. Het stelt ontwikkelaars in staat om eenvoudig 3D-modellen, scenes, lichten, en andere grafische elementen te creëren en te manipuleren met behulp van slechts HTML, CSS en JavaScript.

### Doel van Three.js:

- **Creëren van 3D-graphics:** Het biedt een eenvoudigere manier om 3D-modellen en -animaties in de browser weer te geven.
- **Cross-browser ondersteuning:** Three.js zorgt ervoor dat 3D-graphics werken in alle moderne webbrowsers.
- **Interactiviteit:** Het ondersteunt gebruikersinteracties zoals muis- en toetsenbordinput om 3D-scenes te manipuleren.
- **Efficiëntie:** Dankzij de optimalisaties en efficiënte rendering-engine is het mogelijk om complexe 3D-scenes te maken zonder dat het ten koste gaat van de prestaties.

### Waarvoor wordt het gebruikt?

- **3D-games:** Voor het ontwikkelen van 3D-spellen in de browser.
- **Visualisaties:** Voor interactieve 3D-visualisaties zoals architecturale visualisaties, productvisualisaties, en wetenschappelijke simulaties.
- **Simulaties en educatie:** Voor het maken van interactieve simulaties en leermiddelen in 3D.

Ik zou zeggen bekijk de site eens: [cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js](https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js)

```
!function(t,e){"object"==typeof exports&&"undefined"!=typeof mod
(this,(function(t){"use strict";const
e="128",n=100,i=300,r=301,s=302,a=303,o=304,l=306,c=307,h=1e3,u=
78,D=33779,I=35840,N=35841,B=35842,z=35843,F=37492,O=37496,H=230
rt{addEventListener(t,e){void 0===this._listeners&&(this._listen
0===this._listeners)return!1;const n=this._listeners;return void
t=n.indexOf(e);-1!==t&&n.splice(t,1)}}dispatchEvent(t){if(void 0
e=0,i=n.length;e<i;e++)n[e].call(this,t);t.target=null}}const s
t=4294967295*Math.random()|0,e=4294967295*Math.random()|0,n=4294
"+st[e>>16&15|64]+st[e>>24&255]+"-"+st[63&n|128]+st[n>>8&255]+"-
Math.max(e,Math.min(n,t))}function ut(t,e){return(t%e+e)%e}funct
Math.pow(2,Math.ceil(Math.log(t)/Math.LN2))}function ft(t){retur
```

**Note!** Het nadeel is dat je wel internet-verbinding moet hebben anders werkt het programma niet. Je kunt de gegevens ook kopiëren vanaf de site en in een tekstfile opslaan, bijvoorbeeld **Ajax.txt**. `<script src="Ajax.txt"></script>` i.p.v.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>
```

In de code wordt Three.js gebruikt om 3D-graphics te creëren voor een kanon en kogels.

### Kanon Creëren:

In de Three.js-code die je hebt gedeeld, wordt een eenvoudig kanon gecreëerd en bediend met toetsenbordinput om te bewegen en schieten. Laten we elk van deze onderdelen bespreken:

```
const cannonGeometry = new THREE.BoxGeometry(1, 1, 1);
const cannonMaterial = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
const cannon = new THREE.Mesh(cannonGeometry, cannonMaterial);
cannon.position.set(0, -10, 0);
scene.add(cannon);
```

- **BoxGeometry:** Creëert het basismodel van het kanon (een kubus).
- **MeshBasicMaterial:** Geeft het kanon een groene kleur.

- `position.set(0, -10, 0)`: Zet de beginpositie van het kanon.

#### Kanon Bewegen:

```
document.addEventListener('keydown', function(event) {
  if (event.key === 'z' || event.key === 'Z') {
    cannon.position.x -= 0.5; // Beweeg kanon naar links
  }
  if (event.key === 'm' || event.key === 'M') {
    cannon.position.x += 0.5; // Beweeg kanon naar rechts
  }
});
```

- Gebruik van `keydown`-eventlistener om beweging naar links of rechts te regelen.

#### Kanon Laten Schieten:

```
document.addEventListener('keydown', function(event) {
  if (event.code === 'Space' && !bulletFired) {
    bullet.position.set(cannon.position.x, cannon.position.y + 1, 0); // Kogel positie boven het kanon
    bullet.visible = true; // Maak de kogel zichtbaar
    bulletFired = true; // Markeer dat de kogel is afgevuurd
  }
});
```

- Als de spatiebalk wordt ingedrukt en er nog geen kogel is afgevuurd, wordt de kogel ingesteld boven het kanon en zichtbaar gemaakt.

#### Class-gebaseerde benadering (optioneel):

Hoewel in de code geen expliciete klasse wordt gebruikt voor het kanon, kan het kanon wel worden behandeld als een klasse:

```
class Cannon {
  constructor(scene) {
    this.geometry = new THREE.BoxGeometry(1, 1, 1);
    this.material = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
    this.mesh = new THREE.Mesh(this.geometry, this.material);
    this.mesh.position.set(0, -10, 0);
    scene.add(this.mesh);
    this.bulletGeometry = new THREE.BoxGeometry(0.2, 0.5, 0.2);
    this.bulletMaterial = new THREE.MeshBasicMaterial({ color: 0xffff00 });
    this.bullet = new THREE.Mesh(this.bulletGeometry, this.bulletMaterial);
    this.bullet.visible = false;
    scene.add(this.bullet);
    this.bulletFired = false;
    this.bulletSpeed = 0.5;
    document.addEventListener('keydown', (event) => this.handleInput(event));
  }
  handleInput(event) {
    if (event.key === 'z' || event.key === 'Z') {
      this.mesh.position.x -= 0.5;
    }
    if (event.key === 'm' || event.key === 'M') {
      this.mesh.position.x += 0.5;
    }
    if (event.code === 'Space' && !this.bulletFired) {
      this.fireBullet();
    }
  }
  fireBullet() {
    this.bullet.position.set(this.mesh.position.x, this.mesh.position.y + 1, 0);
    this.bullet.visible = true;
    this.bulletFired = true;
  }
}
```

```

update() {
  if (this.bulletFired) {
    this.bullet.position.y += this.bulletSpeed;
    if (this.bullet.position.y > 15) {
      this.bullet.visible = false;
      this.bulletFired = false;
    }
  }
}
}
}

```

Met een klasse wordt het makkelijker om de kanon- en kogelfunctionaliteit te beheren en uitbreiden.

### Het pijltje

```
document.addEventListener('keydown', (event) => this.handleInput(event));
```

- **document.addEventListener('keydown', (event) => this.handleInput(event));**  
Dit is een eventlistener die luistert naar toetsaanslagen op het hele document. Wanneer een toets wordt ingedrukt, wordt de handleInput-methode van het kanon (in dit geval this) aangeroepen.

#### Uitleg van de syntaxis:

- **document:** Het gehele HTML-document waarop de eventlistener wordt toegevoegd.
- **addEventListener('keydown', (event) => this.handleInput(event));**: Een methode die luistert naar de keydown-gebeurtenis (toetsaanslagen).
- **(event)**: Een callback-functie die een event parameter als argument accepteert, waarin details van de toetsaanslag zijn opgeslagen.
- **this.handleInput(event)**: De handleInput-methode wordt aangeroepen met het event object als parameter. In een klassiek objectgebaseerd patroon verwijst this naar het huidige object (bijvoorbeeld de klasse Cannon).

In dit specifieke geval:

- this verwijst naar de instantie van het kanonobject.
- De methode handleInput(event) bevat de logica voor het verwerken van de toetsinput (zoals beweging van het kanon of het afvuren van een kogel).

### 2.1.3 Three.js code

In de onderstaande code wordt gebruik gemaakt van Three.js voor het renderen van 3D-omgevingen. Hier zijn de specifieke gedeelten van de code die Three.js gebruiken:

#### 1. Scene, Camera en Renderer:

```

const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
const renderer = new THREE.WebGLRenderer();

```

Deze secties creëren de basis van een 3D-scene, inclusief een camera en een renderer om de scene te renderen.

#### 2. Meshes (Kanon, Kogels):

```

const cannonGeometry = new THREE.BoxGeometry(1, 1, 1);
const cannonMaterial = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
const cannon = new THREE.Mesh(cannonGeometry, cannonMaterial);
scene.add(cannon);
const bulletGeometry = new THREE.BoxGeometry(0.2, 0.5, 0.2);

```

```
const bulletMaterial = new THREE.MeshBasicMaterial({ color: 0xffff00 });
let bullets = [];
```

Hier worden de geometrieën en materialen voor het kanon en de kogels gemaakt, en nieuwe Mesh-objecten worden toegevoegd aan de scene.

### 3. Event Listener:

```
document.addEventListener('keydown', function(event) {
  if (event.code === 'Space') {
    const newBullet = new THREE.Mesh(bulletGeometry, bulletMaterial);
    newBullet.position.set(cannon.position.x, cannon.position.y + 1, 0);
    newBullet.visible = true;
    bullets.push(newBullet);
    scene.add(newBullet);
  }
});
```

Dit voegt een nieuwe kogel toe aan de scene wanneer de spatiebalk wordt ingedrukt.

### 4. Rendering de Scene:

```
renderer.render(scene, camera);
```

Deze lijn zorgt ervoor dat de scene wordt gerenderd door de renderer met behulp van de huidige camera-positie.

De gehele opzet van deze code maakt gebruik van Three.js om een interactieve 3D-scene te creëren waarin een kanon kogels kan afvuren die omhoog bewegen en verwijderd worden wanneer ze uit beeld zijn.

## 2.1.4 Uitleg Three.js gebruikte onderdelen

Laten we elk van deze onderdelen van Three.js uitleggen:

### 1. THREE.Scene()

- **Beschrijving:** THREE.Scene() creëert een lege 3D-scene waarin alle objecten, zoals meshes, lichten en camera's, worden geplaatst.
- **Gebruik:**

```
const scene = new THREE.Scene();
```

Dit creëert een nieuwe scene object waarin je al je 3D-objecten kunt toevoegen.

### 2. THREE.PerspectiveCamera(fov, aspect, near, far)

- **Beschrijving:** THREE.PerspectiveCamera() creëert een camera met een perspectiefisch zicht. Dit geeft diepte aan het 3D-beeld.
- **Parameters:**
  - fov (Field of View): De hoek van het zicht in graden (meestal tussen 45 en 75 graden).
  - aspect: De verhouding tussen breedte en hoogte van het scherm (meestal window.innerWidth / window.innerHeight).
  - near: De afstand van de nabijste zichtbare punt.
  - far: De afstand van het verste zichtbare punt.
- **Gebruik:**

```
const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
```

Dit stelt een camera in met een zichtveld van 75 graden.

### 3. THREE.WebGLRenderer()

- **Beschrijving:** THREE.WebGLRenderer() genereert de rendering engine die de scene in 3D kan renderen met behulp van WebGL.
- **Gebruik:**

```
const renderer = new THREE.WebGLRenderer();
```

Dit zorgt voor het opbouwen van het canvas en het renderen van de scene naar dat canvas.

### 4. THREE.BoxGeometry(width, height, depth)

- **Beschrijving:** THREE.BoxGeometry() maakt een rechthoekige geometrie (kubus) met de opgegeven breedte, hoogte en diepte.
- **Gebruik:**

```
const bulletGeometry = new THREE.BoxGeometry(0.2, 0.5, 0.2);
```

Dit creëert een doosvormige geometrie voor de kogels met de opgegeven afmetingen.

### 5. THREE.MeshBasicMaterial()

- **Beschrijving:** THREE.MeshBasicMaterial() creëert een materiaal dat eenvoudig kleur of kleurverloop toevoegt aan een Mesh. Dit is een eenvoudige weergave zonder extra effecten zoals belichting of schaduw.
- **Gebruik:**

```
const bulletMaterial = new THREE.MeshBasicMaterial({ color: 0xffff00 });
```

Dit geeft een eenvoudige gele kleur aan de kogel.

### 6. THREE.Mesh(geometry, material)

- **Beschrijving:** THREE.Mesh() maakt een Mesh object dat een geometrie (zoals THREE.BoxGeometry) combineert met een materiaal (zoals THREE.MeshBasicMaterial). Dit zorgt ervoor dat de geometrie kan worden weergegeven in de 3D-scene.
- **Gebruik:**

```
const cannon = new THREE.Mesh(cannonGeometry, cannonMaterial);
```

Dit maakt een Mesh (het kanon) door de geometrie van de kubus te combineren met een eenvoudig groen materiaal.

### 7. renderer.render(scene, camera)

- **Beschrijving:** renderer.render(scene, camera) zorgt ervoor dat de scene wordt gerenderd door de opgegeven camera. Dit is het proces waarbij de 3D-scene wordt omgezet naar een 2D-beeld op het scherm.
- **Gebruik:**

```
renderer.render(scene, camera);
```

Dit geeft het visuele output van de scene weer.

### 8. bullets.push(newBullet)

- **Beschrijving:** push() voegt een nieuw element toe aan een array. In dit geval wordt een nieuwe Mesh toegevoegd aan het array van kogels (bullets).
- **Gebruik:**

```
bullets.push(newBullet);
```

Dit voegt een nieuwe Mesh (de kogel) toe aan het array van kogels.

### 9. scene.add(object)

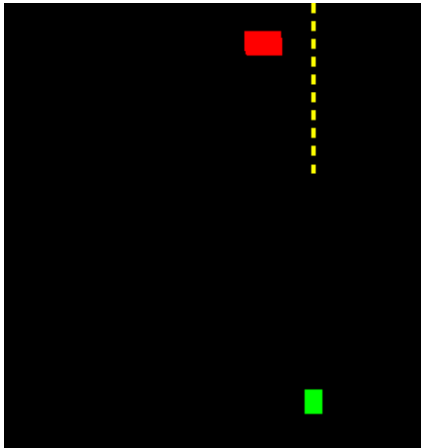
- **Beschrijving:** scene.add(object) voegt een object (zoals een Mesh, licht of camera) toe aan de scene.

```
scene.add(cannon);
```

Dit voegt het kanon toe aan de 3D-scene, zodat het zichtbaar is in de rendering.

Met deze functies en methoden kun je een volledige 3D-scene bouwen en manipuleren in Three.js.

## 2.2 Schieten met een eenvoudig kanon op een vliegend object



Code:

```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Space Invaders</title>
  <style>
    body { margin: 0; overflow: hidden; }
    canvas { display: block; }
  </style>
</head>
<body>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>
  <script>
    // Scene, Camera en Renderer instellen
    const scene = new THREE.Scene();
    const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
    const renderer = new THREE.WebGLRenderer();
    renderer.setSize(window.innerWidth, window.innerHeight);
    document.body.appendChild(renderer.domElement);

    // Lichten toevoegen
    const light = new THREE.DirectionalLight(0xffffff, 1);
    light.position.set(1, 1, 1).normalize();
    scene.add(light);

    // Kanon maken
    const cannonGeometry = new THREE.BoxGeometry(1, 1, 1);
    const cannonMaterial = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
    const cannon = new THREE.Mesh(cannonGeometry, cannonMaterial);
    cannon.position.set(0, -10, 0);
    scene.add(cannon);

    // Ruimteschip maken
    const shipGeometry = new THREE.BoxGeometry(2, 1, 1);
    const shipMaterial = new THREE.MeshBasicMaterial({ color: 0xff0000 });
    const ship = new THREE.Mesh(shipGeometry, shipMaterial);
    ship.position.set(-20, 10, 0);
    scene.add(ship);
```

```

// Kogel maken
const bulletGeometry = new THREE.BoxGeometry(0.2, 0.5, 0.2);
const bulletMaterial = new THREE.MeshBasicMaterial({ color: 0xffff00 });
let bullets = []; // Array om kogels bij te houden
let bulletSpeed = 0.5; // Snelheid van de kogel

// Camera positie
camera.position.z = 30;

// Eventlisteners voor toetsenbordinput
document.addEventListener('keydown', function(event) {
  if (event.key === 'z' || event.key === 'Z') {
    cannon.position.x -= 0.5; // Beweeg kanon naar links
  }
  if (event.key === 'm' || event.key === 'M') {
    cannon.position.x += 0.5; // Beweeg kanon naar rechts
  }
  if (event.code === 'Space') {
    // Maak een nieuwe kogel met een kleine tussenruimte
    if (bullets.length === 0 || bullets[bullets.length - 1].position.y > cannon.position.y + 1.2) {
      const bullet = new THREE.Mesh(bulletGeometry, bulletMaterial);
      bullet.position.set(cannon.position.x, cannon.position.y + 1, 0); // Kogel positie boven het kanon
      bullet.visible = true; // Maak de kogel zichtbaar
      scene.add(bullet);
      bullets.push(bullet); // Voeg de kogel toe aan de array
    }
  }
});

// Functie om ruimteschip te vernietigen
function destroyShip() {
  scene.remove(ship);
  const explosionGeometry = new THREE.SphereGeometry(1.5, 32, 32);
  const explosionMaterial = new THREE.MeshBasicMaterial({ color: 0xff0000 });
  const explosion = new THREE.Mesh(explosionGeometry, explosionMaterial);
  explosion.position.copy(ship.position);
  scene.add(explosion);

  setTimeout(() => {
    scene.remove(explosion);
  }, 500); // Explosie verdwijnt na een halve seconde
}

// Animatie loop
function animate() {
  requestAnimationFrame(animate);

  // Beweeg het ruimteschip van links naar rechts
  ship.position.x += 0.1;
  if (ship.position.x > 20) {
    ship.position.x = -20; // Verander richting als het ruimteschip de rand bereikt
  }

  // Beweeg de kogels omhoog
  for (let i = 0; i < bullets.length; i++) {
    const bullet = bullets[i];
    if (bullet) {
      bullet.position.y += bulletSpeed;
      if (bullet.position.y > 15) {
        scene.remove(bullet);
      }
    }
  }
}

```

```

        bullets.splice(i, 1); // Verwijder de kogel als het buiten beeld is
        i--; // Pas index aan omdat een element verwijderd is
    }
    // Controleer op botsing met het ruimteschip
    if (bullet.position.distanceTo(ship.position) < 1.5) {
        destroyShip();
        scene.remove(bullet);
        bullets.splice(i, 1); // Verwijder de kogel
        i--; // Pas index aan
    }
}

renderer.render(scene, camera);
}

animate();
</script>
</body>
</html>

```

De formule die wordt gebruikt om de afstand tussen de kogel en het ruimteschip te berekenen is de **afstand-formule** tussen twee 3D-punten. Toepassing van de formule:

```

if (bullet.position.distanceTo(ship.position) < 1.5) {
    destroyShip();
    scene.remove(bullet);
    bullets.splice(i, 1); // Verwijder de kogel
    i--; // Pas index aan
}

```

#### Formule:

De formule `bullet.position.distanceTo(ship.position)` berekent de afstand tussen de positie van de kogel en de positie van het ruimteschip. De formule is als volgt:

$$\text{afstand} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Hierbij:

- $x_1, y_1, z_1$  zijn de coördinaten van de kogel.
- $x_2, y_2, z_2$  zijn de coördinaten van het ruimteschip.

De afstand wordt dan vergeleken met een vaste waarde van 1.5. Als de afstand kleiner is dan 1.5, wordt er een botsing gedetecteerd.

In de loop worden alle kogels doorgelopen en wordt deze afstand voor elke kogel gecontroleerd, zoals hieronder:

```
bullet.position.distanceTo(ship.position) < 1.5
```

Als deze afstand kleiner is dan 1.5, wordt het ruimteschip vernietigd en de kogel verwijderd.

In de code zie je geen expliciete wiskundige formule. Dat komt omdat de afstandsberekening verborgen zit in de implementatie van de methode `distanceTo()` in de Three.js-bibliotheek. Deze methode voert de formule voor de afstand tussen twee 3D-punten uit.



Wanneer het ruimteschip wordt geraakt door een kogel, gebeurt het volgende:

1. **Vernietigen van het ruimteschip:** Het ruimteschip wordt verwijderd uit de scène en vervangen door een explosie.
2. **Explosie toevoegen:** Er wordt een nieuwe THREE.Mesh met een sferische geometrie en een rode materiaalkleur toegevoegd om de explosie weer te geven.
3. **Verwijderen van de kogel:** De getroffen kogel wordt verwijderd uit de scène en uit de bullets array.

**Code die verantwoordelijk is voor deze acties:**

De code die verantwoordelijk is voor het vernietigen van het ruimteschip en het toevoegen van de explosie, evenals het verwijderen van de kogel, bevindt zich in de destroyShip()-functie:

```
function destroyShip() {  
  scene.remove(ship); // Ruimteschip verwijderen  
  const explosionGeometry = new THREE.SphereGeometry(1.5, 32, 32);  
  const explosionMaterial = new THREE.MeshBasicMaterial({ color: 0xff0000 });  
  const explosion = new THREE.Mesh(explosionGeometry, explosionMaterial);  
  explosion.position.copy(ship.position); // Explosion positie volgen die van het ruimteschip  
  scene.add(explosion);  
  
  setTimeout(() => {  
    scene.remove(explosion); // Explosie verwijderen na een halve seconde  
  }, 500);  
}
```

Daarnaast wordt de getroffen kogel verwijderd uit de scène en uit de bullets array:

```
scene.remove(bullet);  
bullets.splice(i, 1); // Verwijder de kogel  
i--; // Pas index aan
```

Deze stappen zorgen ervoor dat het ruimteschip wordt vervangen door een explosie en dat de kogel verwijderd wordt zodra de botsing plaatsvindt.

## 2.3 spel uitbreiding

### 2.3.1 Elk spel heeft zijn begin en zijn einde

Een eigenschap van een spelletje is dat het een begin en een einde moet hebben. Verder moet er gemonitord kunnen worden in bijvoorbeeld de tijd, hoeveel schoten er zijn gelost en aangegeven kunnen worden wanneer het spel geëindigd is en hoeveel punten je uiteindelijk hebt gewonnen of hoeveel ruimte schepen je hebt neergeschoten.

Verder hoort er ook een leuk verhaaltje bij.

### 2.3.2 Het bijbehorend verhaaltje

Het universum is in beroering. Een invasie van kwaadaardige ruimteschepen nadert de aarde met de dreiging van vernietiging. Jij, de laatste hoop van de mensheid, bestijgt het krachtige kanon dat de bescherming biedt tegen deze dreiging. Je missie is helder: bescherm de aarde tegen deze buitenaardse aanvallen en schiet alles neer wat zich aan de horizon aandient.

#### Het Spel: Ruimteverdediging

In dit intensieve ruimtetijdavontuur is het jouw taak om golven van vijandige schepen en dreigende bommen af te weren. Het spel heeft een vast begin en een definitief einde: je zult doorgaan totdat alle schepen neergeschoten zijn, of totdat je kanon vernietigd wordt door een bom.

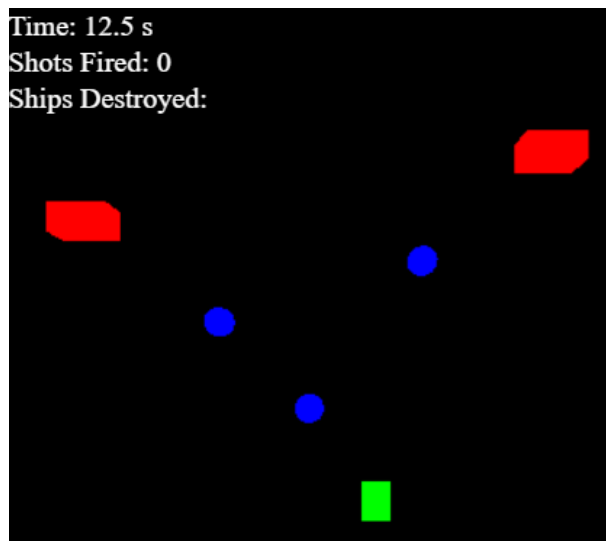
#### Spel-eigenschappen:

1. **Beginscherm en Introductie:** Het spel begint wanneer je de "M" toets indrukt om het spel te starten. Zodra de strijd losbarst, verschijnt een boodschap waarin je wordt verwelkomd en gevraagd wordt om je doel te bereiken.
2. **Tijd en Statistieken:** Terwijl de tijd voortgaat, houdt het spel je op de hoogte van hoe lang je al speelt, hoeveel schoten er zijn afgevuurd en hoeveel vijandige schepen je hebt neergeschoten. Elke vernietigde vijand verhoogt je score en draagt bij aan de missie om de aarde te beschermen.
3. **De Eindstrijd:** Wanneer alle vijandige schepen zijn vernietigd of je kanon het niet langer aankan, wordt het spel beëindigd. Een scherm met de boodschap van overwinning of nederlaag verschijnt, samen met je totale score en de statistieken van je prestaties.
4. **Motivatie en Avontuur:** Terwijl je door de niveaus vordert, worden de vijanden uitdagender en dodelijker, met meer bommen en geavanceerdere tactieken. Elk succes geeft je de voldoening dat je een stap dichterbij de bescherming van de aarde komt.
5. **Herhalingswaarde:** Het spel moedigt je aan om steeds weer opnieuw te spelen om een hogere score te behalen en sneller te leren hoe je beter kunt navigeren door de dreiging van de vijanden.

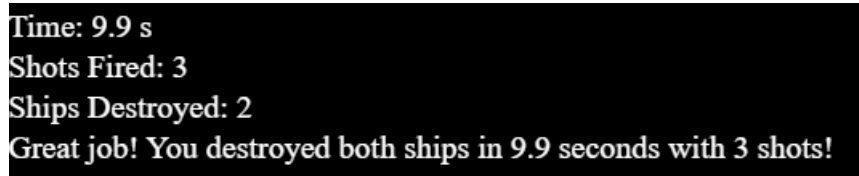
Met dit avontuur in gedachten, moet je nu snel reageren, nauwkeurig schieten en slimme beslissingen nemen om de overwinning te behalen en te overleven in de strijd tegen de buitenaardse dreiging. Succes, ruimteheld!

Dit spel is een bijna compleet spel met twee ruimtevaartuigen die (blauwe) bommen gooien, waarmee geprobeerd wordt om het kanon uit te schakelen. Het kanon kan de ruimte schepen neerschieten, maar dit wordt bemoeilijkt omdat er slechts 1 schot per keer afgeschoten kan worden. Pas wanneer de kogel van het scherm verdwijnt kun je weer schieten. Het kanon kan met de toetsen "M" en "Z" verplaatst worden om de bommen te ontwijken.

Hoe het spelletje eruit ziet tijdens het gevecht:



Het einde van het spelletje:



Verder is er ook geluid toegevoegd. Hoe dat werkt wordt hierna uitgelegd.

#### Code:

```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Space Invaders</title>
  <style>
    body { margin: 0; overflow: hidden; }
    canvas { display: block; }
  </style>
</head>
<body>
  <script src="https://cdn.jsdelivr.net/npm/three.js@r128/three.min.js"></script>
  <script>
    // Geluidsbestanden instellen
    const shotSound = new Audio('shot.wav');
    const explosionSound = new Audio('explosion.wav');
    const gameOverSound = new Audio('gameover.mp3');

    // Scene, Camera en Renderer instellen
    const scene = new THREE.Scene();
    const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
    const renderer = new THREE.WebGLRenderer();
    renderer.setSize(window.innerWidth, window.innerHeight);
    document.body.appendChild(renderer.domElement);

    // Timer, aantal schoten en aantal vernietigde schepen HTML toevoegen
    const timerDisplay = document.createElement('div');
    const shotsDisplay = document.createElement('div');
```

```

const shipsDisplay = document.createElement('div');
const messageDisplay = document.createElement('div');
timerDisplay.style.position = shotsDisplay.style.position = shipsDisplay.style.position =
messageDisplay.style.position = 'absolute';
timerDisplay.style.top = '10px';
shotsDisplay.style.top = '30px';
shipsDisplay.style.top = '50px';
messageDisplay.style.top = '70px';
timerDisplay.style.left = shotsDisplay.style.left = shipsDisplay.style.left = messageDisplay.style.left = '10px';
timerDisplay.style.color = shotsDisplay.style.color = shipsDisplay.style.color = messageDisplay.style.color =
'white';
document.body.appendChild(timerDisplay);
document.body.appendChild(shotsDisplay);
document.body.appendChild(shipsDisplay);
document.body.appendChild(messageDisplay);

// Starttijd, aantal schoten en aantal vernietigde schepen bijhouden
const startTime = Date.now();
let shotsFired = 0;
let shipsDestroyed = 0;
let gameOver = false;

// Variabelen voor snelheid en bomkans
let shipSpeed = 0.1; // Snelheid van ruimteschepen
let bombDropChance = 0.01; // Kans dat een bom wordt gedropt

// Lichten toevoegen
const light = new THREE.DirectionalLight(0xffffff, 1);
light.position.set(1, 1, 1).normalize();
scene.add(light);

// Kanon maken
const cannonGeometry = new THREE.BoxGeometry(1, 1, 1);
const cannonMaterial = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
const cannon = new THREE.Mesh(cannonGeometry, cannonMaterial);
cannon.position.set(0, -10, 0);
scene.add(cannon);

// Ruimteschepen maken
const shipGeometry = new THREE.BoxGeometry(2, 1, 1);
const shipMaterial = new THREE.MeshBasicMaterial({ color: 0xff0000 });
const ship1 = new THREE.Mesh(shipGeometry, shipMaterial);
ship1.position.set(-20, 10, 0);
scene.add(ship1);

const ship2 = new THREE.Mesh(shipGeometry, shipMaterial);
ship2.position.set(20, 15, 0);
scene.add(ship2);

// Kogel maken
const bulletGeometry = new THREE.BoxGeometry(0.2, 0.5, 0.2);
const bulletMaterial = new THREE.MeshBasicMaterial({ color: 0xffff00 });
let bullet = new THREE.Mesh(bulletGeometry, bulletMaterial);
bullet.visible = false; // Kogel is in eerste instantie onzichtbaar
scene.add(bullet);

// Bom maken
const bombGeometry = new THREE.SphereGeometry(0.5, 32, 32);
const bombMaterial = new THREE.MeshBasicMaterial({ color: 0x0000ff });
const bombs = [];

```

```

// Variabelen voor beweging en status
let ship1Direction = 1;
let ship2Direction = -1;
let bulletFired = false;
let bulletSpeed = 0.5;
let ship1Destroyed = false;
let ship2Destroyed = false;

// Camera positie
camera.position.z = 30;

// Eventlistener voor toetsenbordinput
document.addEventListener('keydown', function(event) {
  if (event.key === 'z' || event.key === 'Z') {
    cannon.position.x -= 0.5;
  }
  if (event.key === 'm' || event.key === 'M') {
    cannon.position.x += 0.5;
  }
  if (event.code === 'Space' && !bulletFired) {
    bullet.position.set(cannon.position.x, cannon.position.y + 1, 0);
    bullet.visible = true;
    bulletFired = true;
    shotsFired++;
    shotSound.play(); // Speel het schotgeluid
  }
});

// Functie om een ruimteschip te vernietigen
function destroyShip(ship) {
  scene.remove(ship);
  const explosionGeometry = new THREE.SphereGeometry(1.5, 32, 32);
  const explosionMaterial = new THREE.MeshBasicMaterial({ color: 0xff0000 });
  const explosion = new THREE.Mesh(explosionGeometry, explosionMaterial);
  explosion.position.copy(ship.position);
  scene.add(explosion);

  explosionSound.play(); // Speel het explosiegeluid

  setTimeout(() => {
    scene.remove(explosion);
  }, 500);
}

// Functie om het kanon te vernietigen
function destroyCannon() {
  scene.remove(cannon);
  const explosionGeometry = new THREE.SphereGeometry(2, 32, 32);
  const explosionMaterial = new THREE.MeshBasicMaterial({ color: 0xff0000 });
  const explosion = new THREE.Mesh(explosionGeometry, explosionMaterial);
  explosion.position.copy(cannon.position);
  scene.add(explosion);

  explosionSound.play(); // Speel het explosiegeluid

  setTimeout(() => {
    gameOver = true;
    gameOverSound.play(); // Speel het game-over geluid
  }, 500);
}

```

```

// Animatielus
function animate() {
  if (gameOver) return;

  requestAnimationFrame(animate);

  const elapsedTime = ((Date.now() - startTime) / 1000).toFixed(1);
  timerDisplay.textContent = `Time: ${elapsedTime} s`;
  shotsDisplay.textContent = `Shots Fired: ${shotsFired}`;
  shipsDisplay.textContent = `Ships Destroyed: ${shipsDestroyed}`;

  if (ship1Destroyed && ship2Destroyed && !gameOver) {
    messageDisplay.textContent = `Great job! You destroyed both ships in ${elapsedTime} seconds with ${shotsFired} shots!`;
    gameOver = true;
    gameOverSound.play();
    return;
  }

  // Beweging van ruimteschepen
  if (!ship1Destroyed) {
    ship1.position.x += shipSpeed * ship1Direction;
    if (ship1.position.x > 20 || ship1.position.x < -20) {
      ship1Direction *= -1;
    }

    // Ruimteschip laat bom vallen
    if (Math.random() < bombDropChance) {
      const bomb = new THREE.Mesh(bombGeometry, bombMaterial);
      bomb.position.set(ship1.position.x, ship1.position.y - 1, 0);
      bombs.push(bomb);
      scene.add(bomb);
    }
  }

  if (!ship2Destroyed) {
    ship2.position.x += shipSpeed * ship2Direction;
    if (ship2.position.x > 20 || ship2.position.x < -20) {
      ship2Direction *= -1;
    }

    // Ruimteschip laat bom vallen
    if (Math.random() < bombDropChance) {
      const bomb = new THREE.Mesh(bombGeometry, bombMaterial);
      bomb.position.set(ship2.position.x, ship2.position.y - 1, 0);
      bombs.push(bomb);
      scene.add(bomb);
    }
  }

  // Beweging van kogels
  if (bulletFired) {
    bullet.position.y += bulletSpeed;
    if (bullet.position.y > 15) {
      bullet.visible = false;
      bulletFired = false;
    }

    if (!ship1Destroyed && bullet.position.distanceTo(ship1.position) < 1.5) {
      destroyShip(ship1);
    }
  }
}

```

```

        ship1Destroyed = true;
        shipsDestroyed++;
        bullet.visible = false;
        bulletFired = false;
    }
    if (!ship2Destroyed && bullet.position.distanceTo(ship2.position) < 1.5) {
        destroyShip(ship2);
        ship2Destroyed = true;
        shipsDestroyed++;
        bullet.visible = false;
        bulletFired = false;
    }
}
// Beweging van bommen
for (let i = bombs.length - 1; i >= 0; i--) {
    bombs[i].position.y -= 0.1; // Beweging van de bom naar beneden
    if (bombs[i].position.y < cannon.position.y + 1 && bombs[i].position.distanceTo(cannon.position) < 1.5) {
        destroyCannon();
        scene.remove(bombs[i]);
        bombs.splice(i, 1);
    }
    if (bombs[i].position.y < -15) {
        scene.remove(bombs[i]);
        bombs.splice(i, 1);
    }
}
}
renderer.render(scene, camera);
}

animate();
</script>
</body>
</html>

```

### 2.3.3 Toevoegen van geluid

Een spelletje zonder geluid is als een schilderij zonder kleur.

Er zijn verschillende websites waar je geluidsfragmenten zoals explosies of schietgeluiden kunt downloaden. Hier zijn een aantal suggesties:

#### Gratis en rechtenvrije opties:

1. [Freesound](#)
  - Een grote community-gebaseerde database van geluiden die vaak gratis zijn onder een Creative Commons-licentie. Controleer de licentie voor commercieel gebruik.
2. [Zapsplat](#)
  - Biedt gratis geluidseffecten, waaronder explosies en schietgeluiden. Ze vragen wel om een vermelding als je de gratis optie gebruikt.
3. [SoundBible](#)
  - Een collectie van gratis geluidseffecten, vaak zonder copyright of onder een Creative Commons-licentie.

Zorg ervoor dat je bij het downloaden altijd controleert of het gebruik van de geluidsfragmenten voldoet aan de licentievoorwaarden, vooral als je ze commercieel wilt gebruiken.

## Voorbeeld

Met dit voorbeeld kun je shoot en Explode geluiden laten horen. Je zal in de directory dan de geluidsfragmenten **shot.wav** en **explosion.wav** moeten toevoegen. Het voordeel van dit voorbeeld is dat het geluidsfragment niet helemaal afgespeeld hoeft te worden, waardoor je continu kunt blijven “schieten”.

Shoot

Explode

```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Unlimited Sound Effects</title>
</head>
<body>
  <button id="shoot">Shoot</button>
  <button id="explode">Explode</button>

  <script>
    // Geluidsbestanden instellen
    const shotSoundPath = 'shot.wav';
    const explosionSoundPath = 'explosion.wav';

    // Functie om een nieuw audiobestand te maken en af te spelen
    function playSound(soundPath) {
      const sound = new Audio(soundPath);
      sound.play();
    }

    // Eventlistener voor de "Shoot"-knop
    document.getElementById('shoot').addEventListener('click', () => {
      playSound(shotSoundPath);
    });

    // Eventlistener voor de "Explode"-knop
    document.getElementById('explode').addEventListener('click', () => {
      playSound(explosionSoundPath);
    });
  </script>
</body>
</html>
```

## 2.4 De tank

De tank kan heen en weer bewegen, de loop kan omhoog en omlaag bewegen en tijdens het schieten wordt er geluid geproduceerd.





## code

```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Improved Cannon</title>
  <style>
    body { margin: 0; overflow: hidden; }
    canvas { display: block; background: #87CEEB; }
  </style>
</head>
<body>
  <canvas id="gameCanvas"></canvas>

  <script>
    const canvas = document.getElementById("gameCanvas");
    const ctx = canvas.getContext("2d");
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;

    // Kanon eigenschappen
    const cannonBaseWidth = 80;
    const cannonBaseHeight = 20;
    const cannonBarrelLength = 120;
    const cannonBarrelWidth = 15;
    let cannonX = canvas.width / 2 - cannonBaseWidth / 2;
    let cannonAngle = -Math.PI / 2; // Richting omhoog

    // Kogel eigenschappen
    const bulletWidth = 5;
    const bulletHeight = 10;
    let bullets = [];

    // Schotgeluid
    function playShotSound() {
      const shotSound = new Audio('shot.wav');
      shotSound.play();
    }

    // Game loop
    function update() {
      ctx.clearRect(0, 0, canvas.width, canvas.height);

      // Teken het kanon
      drawCannon();

      // Teken kogels
      ctx.fillStyle = "#FFF000";
      for (let i = 0; i < bullets.length; i++) {
        const bullet = bullets[i];
        bullet.x += bullet.speedX;
        bullet.y += bullet.speedY;
        ctx.fillRect(bullet.x, bullet.y, bulletWidth, bulletHeight);

        // Verwijder kogel als deze buiten het scherm komt
        if (bullet.y < 0 || bullet.y > canvas.height || bullet.x < 0 || bullet.x > canvas.width) {
          bullets.splice(i, 1);
          i--;
        }
      }
    }
  </script>
</body>
</html>
```

```

    }
}

requestAnimationFrame(update);
}

// Functie om het kanon te tekenen
function drawCannon() {
    // Teken kanonlichaam
    ctx.fillStyle = "#008000";
    ctx.fillRect(cannonX, canvas.height - cannonBaseHeight, cannonBaseWidth, cannonBaseHeight);

    // Teken kanonloop
    ctx.save();
    ctx.translate(cannonX + cannonBaseWidth / 2, canvas.height - cannonBaseHeight); // Verplaats naar het draaipunt
    ctx.rotate(cannonAngle); // Draai het kanon
    ctx.fillStyle = "#404040";
    ctx.fillRect(0, -cannonBarrelWidth / 2, cannonBarrelLength, cannonBarrelWidth);
    ctx.restore();

    // Teken wielen
    ctx.fillStyle = "#000000";
    ctx.beginPath();
    ctx.arc(cannonX + 20, canvas.height - 10, 10, 0, Math.PI * 2);
    ctx.fill();
    ctx.beginPath();
    ctx.arc(cannonX + cannonBaseWidth - 20, canvas.height - 10, 10, 0, Math.PI * 2);
    ctx.fill();
}

// Kogel afvuren
function shoot() {
    playShotSound();

    // Bereken uitgang van de loop
    const barrelEndX = cannonX + cannonBaseWidth / 2 + cannonBarrelLength * Math.cos(cannonAngle);
    const barrelEndY = canvas.height - cannonBaseHeight + cannonBarrelLength * Math.sin(cannonAngle);

    // Snelheid van de kogel
    const speed = 7;
    const speedX = speed * Math.cos(cannonAngle);
    const speedY = speed * Math.sin(cannonAngle);

    bullets.push({ x: barrelEndX, y: barrelEndY, speedX, speedY });
}

// Toetsenbordcontrole
document.addEventListener("keydown", function (event) {
    if (event.key === "ArrowLeft") {
        cannonX -= 10; // Verplaats kanon naar links
    }
    if (event.key === "ArrowRight") {
        cannonX += 10; // Verplaats kanon naar rechts
    }
    if (event.key === "ArrowUp") {
        cannonAngle -= 0.1; // Verhoog hoek (naar boven)
    }
    if (event.key === "ArrowDown") {
        cannonAngle += 0.1; // Verlaag hoek (naar beneden)
    }
});

```

```

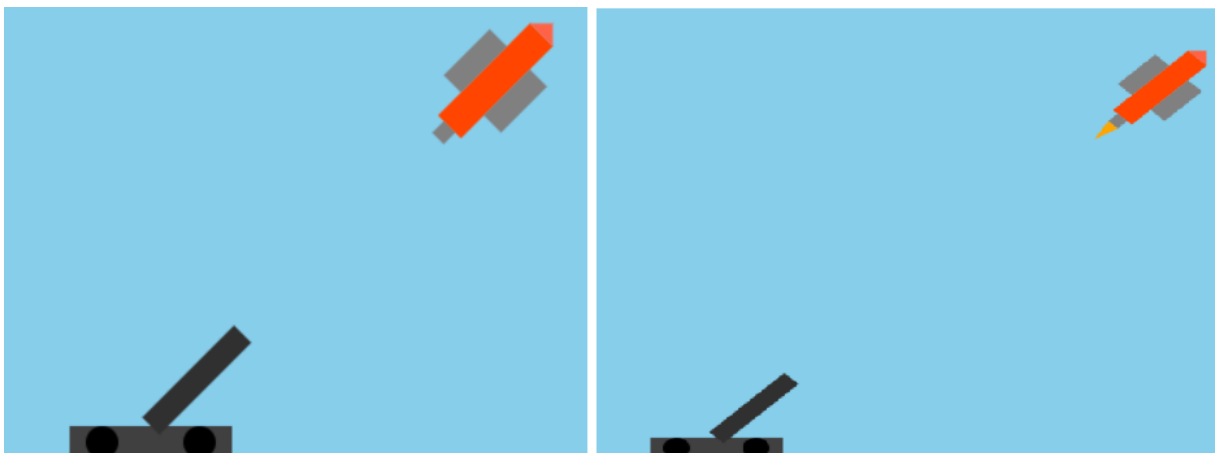
    }
    if (event.key === " " || event.code === "Space") {
        shoot(); // Schiet een kogel
    }
    });

    // Start de game loop
    update();
</script>
</body>
</html>

```

## 2.5 kruisraketten i.p.v. kogels

In plaats van kogels zou de tank ook een kruisraket kunnen afschieten. In de lucht ontbrandt de raketmotor waardoor de raket in een versnelling komt (zie het rechter plaatje).



### Code:

```

<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Realistic Cruise Missile</title>
    <style>
        body { margin: 0; overflow: hidden; }
        canvas { display: block; background: #87CEEB; }
    </style>
</head>
<body>
    <canvas id="gameCanvas"></canvas>

    <script>
        const canvas = document.getElementById("gameCanvas");
        const ctx = canvas.getContext("2d");
        canvas.width = window.innerWidth;
        canvas.height = window.innerHeight;

        // Raket eigenschappen
        const missileWidth = 80;
        const missileHeight = 20;
        const wingWidth = 15;
        const missileInitialSpeed = 5;
        const missileAcceleration = 0.5;
    </script>

```

```

let missiles = [];

// Kanon eigenschappen
const launcherWidth = 100;
const launcherHeight = 20;
const launcherBarrelLength = 80;
const launcherBarrelWidth = 15;
let launcherX = canvas.width / 2 - launcherWidth / 2;
let launcherAngle = -Math.PI / 4;

// Geluidseffecten
const launchSound = new Audio('launch.wav');
const ignitionSound = new Audio('ignition.wav');

// Game loop
function update() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  // Teken de lanceerinrichting
  drawLauncher();

  // Teken raketten
  for (let i = 0; i < missiles.length; i++) {
    const missile = missiles[i];

    if (missile.ignited) {
      missile.speedX += missileAcceleration * Math.cos(missile.angle);
      missile.speedY += missileAcceleration * Math.sin(missile.angle);
    }

    missile.x += missile.speedX;
    missile.y += missile.speedY;

    drawMissile(missile);

    if (
      missile.y < 0 ||
      missile.y > canvas.height ||
      missile.x < 0 ||
      missile.x > canvas.width
    ) {
      missiles.splice(i, 1);
      i--;
    }
  }

  requestAnimationFrame(update);
}

function drawLauncher() {
  ctx.fillStyle = "#404040";
  ctx.fillRect(launcherX, canvas.height - launcherHeight, launcherWidth, launcherHeight);

  ctx.save();
  ctx.translate(launcherX + launcherWidth / 2, canvas.height - launcherHeight);
  ctx.rotate(launcherAngle);
  ctx.fillStyle = "#303030";
  ctx.fillRect(0, -launcherBarrelWidth / 2, launcherBarrelLength, launcherBarrelWidth);
  ctx.restore();
}

```

```

    ctx.fillStyle = "#000000";
    ctx.beginPath();
    ctx.arc(launcherX + 20, canvas.height - 10, 10, 0, Math.PI * 2);
    ctx.fill();
    ctx.beginPath();
    ctx.arc(launcherX + launcherWidth - 20, canvas.height - 10, 10, 0, Math.PI * 2);
    ctx.fill();
}

function drawMissile(missile) {
    ctx.save();
    ctx.translate(missile.x, missile.y);
    ctx.rotate(missile.angle);

    // Teken hoofdstructuur
    ctx.fillStyle = "#FF4500";
    ctx.fillRect(-missileWidth / 2, -missileHeight / 2, missileWidth, missileHeight);

    // Teken neus
    ctx.fillStyle = "#FF6347";
    ctx.beginPath();
    ctx.moveTo(missileWidth / 2, -missileHeight / 2);
    ctx.lineTo(missileWidth / 2 + 10, 0);
    ctx.lineTo(missileWidth / 2, missileHeight / 2);
    ctx.fill();

    // Teken vleugels
    ctx.fillStyle = "#808080";
    ctx.fillRect(-missileWidth / 4, -missileHeight / 2 - wingWidth, missileWidth / 2, wingWidth);
    ctx.fillRect(-missileWidth / 4, missileHeight / 2, missileWidth / 2, wingWidth);

    // Teken staart
    ctx.fillStyle = "#808080";
    ctx.fillRect(-missileWidth / 2 - 10, -missileHeight / 4, 10, missileHeight / 2);

    // Teken motorvlammen
    if (missile.ignited) {
        ctx.fillStyle = "#FFA500";
        ctx.beginPath();
        ctx.moveTo(-missileWidth / 2 - 10, -missileHeight / 4);
        ctx.lineTo(-missileWidth / 2 - 30, 0);
        ctx.lineTo(-missileWidth / 2 - 10, missileHeight / 4);
        ctx.fill();
    }

    ctx.restore();
}

function launchMissile() {
    launchSound.play();

    const barrelEndX = launcherX + launcherWidth / 2 + launcherBarrelLength * Math.cos(launcherAngle);
    const barrelEndY = canvas.height - launcherHeight + launcherBarrelLength * Math.sin(launcherAngle);

    const newMissile = {
        x: barrelEndX,
        y: barrelEndY,
        speedX: missileInitialSpeed * Math.cos(launcherAngle),
        speedY: missileInitialSpeed * Math.sin(launcherAngle),
        angle: launcherAngle,
    };
}

```

```

        ignited: false
    };

    missiles.push(newMissile);

    setTimeout(() => {
        newMissile.ignited = true;
        ignitionSound.play();
    }, 1000);
}

document.addEventListener("keydown", function (event) {
    if (event.key === "ArrowLeft") launcherX -= 10;
    if (event.key === "ArrowRight") launcherX += 10;
    if (event.key === "ArrowUp") launcherAngle -= 0.1;
    if (event.key === "ArrowDown") launcherAngle += 0.1;
    if (event.key === " " || event.code === "Space") launchMissile();
});

update();
</script>
</body>
</html>

```

### 3. Quiz game

Een spel zonder geweld

Einstein

### Einstein's Time Travel Adventure

Wat is de relativiteitstheorie van Einstein?

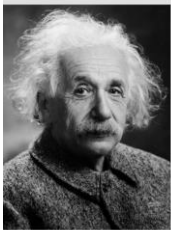
Het is een theorie over zwaartekracht.

Het zegt dat tijd relatief is en afhangt van snelheid en zwaartekracht.

Het verklaart hoe licht zich gedraagt in vacuüm.

**Einstein's Time Travel Adventure**

Je hebt het spel voltooid! Gefeliciteerd!



**Code:**

```

<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Einstein's Time Travel Adventure</title>
    <style>
        body { font-family: Arial, sans-serif; background-color: #e0e0e0; }
        .container { text-align: center; margin-top: 50px; }
        .question { font-size: 24px; margin: 20px; }
        .answer-buttons { margin-top: 20px; }
        button { padding: 10px 20px; margin: 10px; font-size: 18px; cursor: pointer; }
    </style>

```

```

    .result { font-size: 28px; color: green; }
    .error { font-size: 28px; color: red; }
    .image { margin-top: 30px; }
</style>
</head>
<body>
  <div class="container">
    <h1>Einstein's Time Travel Adventure</h1>
    <div id="question-container">
      <div class="question" id="question">Vraag wordt hier weergegeven...</div>
      <div class="answer-buttons">
        <button onclick="checkAnswer(0)">Antwoord 1</button>
        <button onclick="checkAnswer(1)">Antwoord 2</button>
        <button onclick="checkAnswer(2)">Antwoord 3</button>
      </div>
    </div>
    <div id="result"></div>
    <div class="image">
      <img id="einstein-image" src="" alt="Albert Einstein" width="300" style="display:none;">
    </div>
  </div>

  <script>
    const questions = [
      {
        question: "Wat is de relativiteitstheorie van Einstein?",
        answers: [
          "Het is een theorie over zwaartekracht.",
          "Het zegt dat tijd relatief is en afhangt van snelheid en zwaartekracht.",
          "Het verklaart hoe licht zich gedraagt in vacuüm."
        ],
        correctAnswer: 1
      },
      {
        question: "Wat is het foto-elektrische effect?",
        answers: [
          "De beweging van elektronen door licht.",
          "Het fenomeen waarbij licht elektronen uit een materiaal kan slaan.",
          "De snelheid van licht in een medium."
        ],
        correctAnswer: 1
      },
      {
        question: "Hoe beïnvloedt de snelheid de tijd volgens Einstein?",
        answers: [
          "Hoe sneller je beweegt, hoe sneller de tijd voor je gaat.",
          "Hoe sneller je beweegt, hoe trager de tijd voor jou gaat.",
          "Snelheid heeft geen effect op de tijd."
        ],
        correctAnswer: 1
      }
    ];

    let currentQuestionIndex = 0;

    function showQuestion() {
      const question = questions[currentQuestionIndex];
      document.getElementById("question").innerText = question.question;
      const buttons = document.querySelectorAll(".answer-buttons button");
      question.answers.forEach((answer, index) => {

```

```

        buttons[index].innerText = answer;
    });
}

function checkAnswer(selected) {
    const correctAnswer = questions[currentQuestionIndex].correctAnswer;
    const resultContainer = document.getElementById("result");
    const einsteinImage = document.getElementById("einstein-image");

    if (selected === correctAnswer) {
        resultContainer.innerText = "Correct! Je hebt een stap verder gezet in de tijd!";
        resultContainer.classList.add("result");
        resultContainer.classList.remove("error");
        einsteinImage.src =
"https://upload.wikimedia.org/wikipedia/commons/d/d3/Einstein_1921_portrait2.jpg"; // Voeg hier een
afbeelding van Einstein toe
        einsteinImage.style.display = "block";
    } else {
        resultContainer.innerText = "Helaas, dat is incorrect! Probeer het nog eens.";
        resultContainer.classList.add("error");
        resultContainer.classList.remove("result");
        einsteinImage.style.display = "none";
    }

    // Ga naar de volgende vraag of herstart het spel
    currentQuestionIndex++;
    if (currentQuestionIndex < questions.length) {
        setTimeout(() => {
            showQuestion();
            resultContainer.innerText = "";
            einsteinImage.style.display = "none";
        }, 1500);
    } else {
        setTimeout(() => {
            resultContainer.innerText = "Je hebt het spel voltooid! Gefeliciteerd!";
            document.getElementById("question-container").style.display = "none";
        }, 1500);
    }
}

// Start het spel
showQuestion();
</script>
</body>
</html>

```



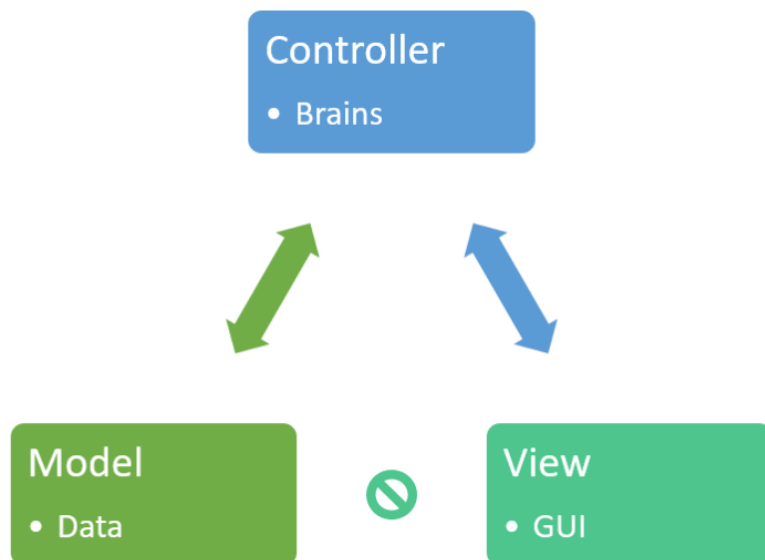
## 4.0 Het MVC-Architectuurpatroon

In de wereld van applicatie- en webontwikkeling is het **Model-View-Controller (MVC)**-patroon een van de meest populaire en krachtige ontwerppatronen. Dit patroon helpt ontwikkelaars om complexe applicaties op te splitsen in drie duidelijke componenten, elk met een specifieke verantwoordelijkheid. Laten we eens kijken naar hoe dit patroon werkt en waarom het zo waardevol is.

### 4.1 Wat is MVC?

Het Model-View-Controller (MVC) patroon is een gestructureerde manier om applicaties te bouwen door de verantwoordelijkheden te scheiden in drie duidelijke lagen:

1. **Model:** Dit is de laag die verantwoordelijk is voor het beheren en manipuleren van gegevens. Het model communiceert vaak met een database of API's om gegevens op te halen of bij te werken. Dit kan variëren van eenvoudige lokale opslag in browsers tot complexe interacties met externe datastores.
2. **View:** De view is verantwoordelijk voor het presenteren van de gegevens aan de gebruiker. Dit kan grafische gebruikersinterfaces zijn, maar ook meer geavanceerde weergaves zoals dashboards, grafieken, of rapporten. De view communiceert rechtstreeks met de gebruiker en biedt interactiemogelijkheden zoals knoppen, formulieren en andere visuele elementen.
3. **Controller:** De controller fungeert als de hersenen van de applicatie. Het ontvangt input van de view en vertaalt deze naar een verzoek aan het model om gegevens op te halen of te bewerken. Daarna retourneert de controller de bijgewerkte gegevens terug naar de view om de gebruiker van relevante feedback te voorzien.



Modelweergave Controller Ontwerppatroon: Scheiding van zorgen

## 4.2 De MVC-architectuur toegelicht

### Hoe het werkt

De specialiteit van MVC is dat het de applicatielogica isoleert van de gebruikersinterfacelaag. De Verwerkingsverantwoordelijke ontvangt vervolgens verzoeken van een gebeurtenis en verwerkt het Model om de gegevens voor te bereiden die nodig zijn voor de weergave. De weergave maakt gebruik van deze gegevens en genereert uitvoer die de gebruiker kan bekijken.

### 4.3 Waarom gebruiken ontwikkelaars MVC?

Ontwikkelaars geven om MVC omdat het een goed gestructureerd raamwerk biedt dat efficiëntie, schaalbaarheid en onderhoudbaarheid bevordert. Laten we eens kijken naar de voordelen van dit patroon:

#### 1. Scheiding van Zorgen:

Door de code te scheiden in drie duidelijke componenten, zorgt MVC ervoor dat elke laag een specifieke verantwoordelijkheid heeft. Dit betekent dat je code eenvoudiger kunt lezen, begrijpen en onderhouden. Het zorgt ervoor dat wijzigingen in de ene laag geen invloed hebben op de andere, wat compatibiliteit garandeert.

#### 2. Los gekoppeld:

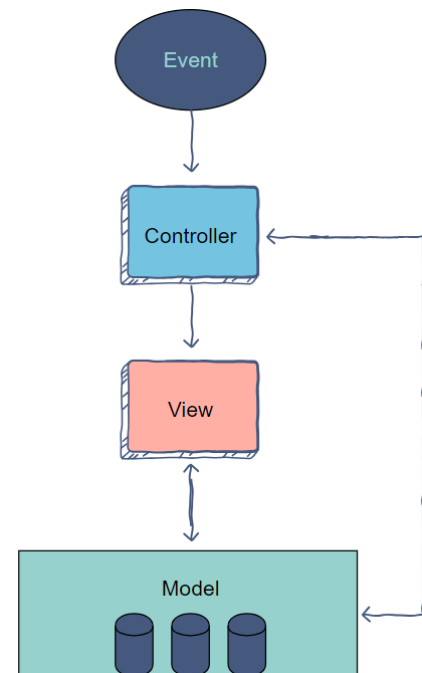
Elk component werkt onafhankelijk van de andere, wat helpt bij het beheren van complexiteit. Dit maakt het gemakkelijk om uitbreidingen te doen zonder bestaande codebases te beïnvloeden.

#### 3. Testbaarheid:

Omdat elke laag afzonderlijk kan worden getest, stroomlijnt MVC de ontwikkelingscyclus. Dit helpt bugs eerder op te sporen en de kwaliteit van de software te verbeteren.

#### 4. Herbruikbaarheid:

De code in het model kan herbruikbaar worden gemaakt zonder aanpassingen, wat ontwikkelaars helpt om efficiënter te werken. Dit geldt ook voor de view en controller, die beiden als losse componenten kunnen worden ingezet in andere applicaties.



## 4.4 Hoe wordt MVC toegepast?

In moderne webapplicaties en Single Page Applications (SPA's) wordt MVC vaak gecombineerd met frameworks zoals **Ruby on Rails**, **ASP.NET MVC**, **Laravel**, **Vue.js**, **Angular** en **React**. Deze frameworks bieden tools en best practices om de scheiding tussen model, view en controller te vereenvoudigen.

### Praktische Toepassing:

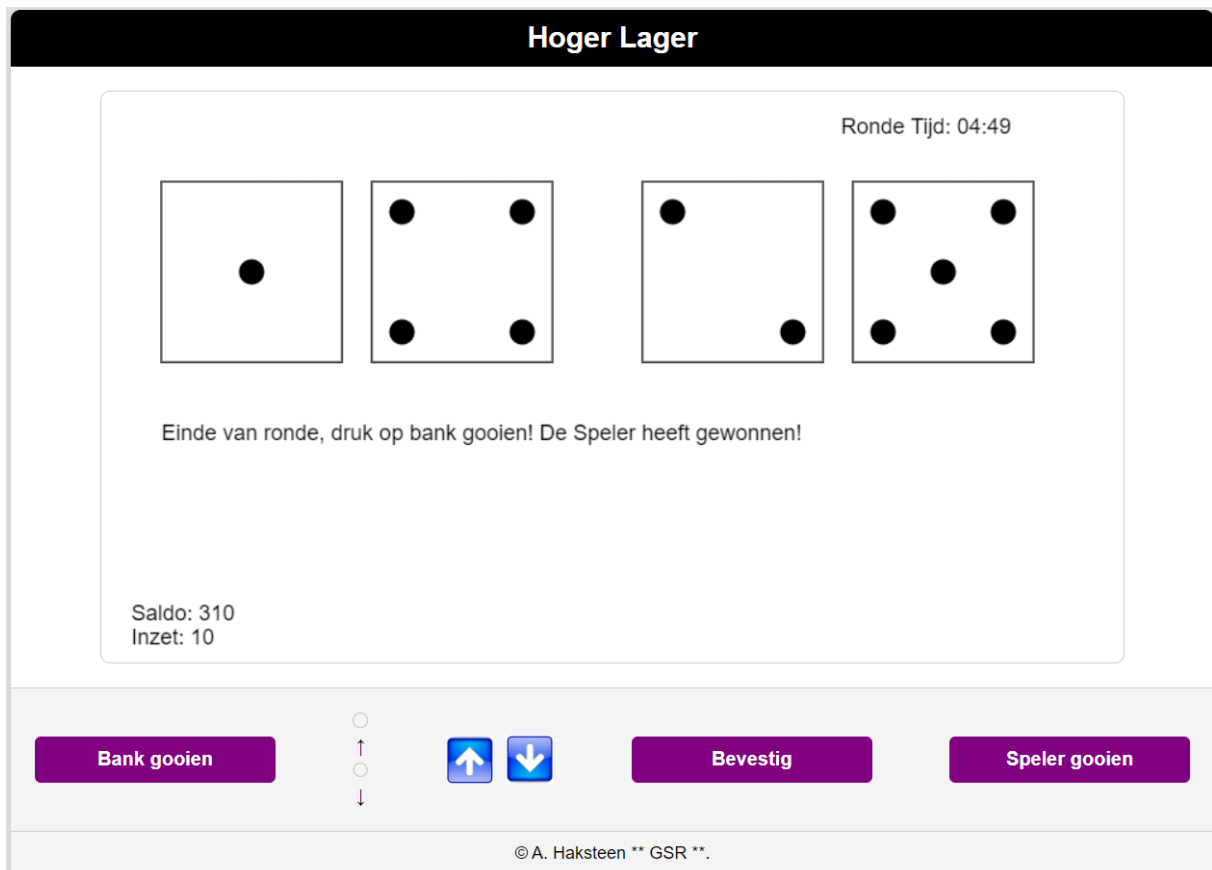
- **Model:** Hier bevindt zich de bedrijfslogica, data-opslag en -verwerking. Het zorgt ervoor dat gegevens correct en efficiënt worden beheerd.

- **View:** De visuele kant van de applicatie waarin gebruikers hun interacties kunnen uitvoeren, zoals klikken, typen en bladeren.
- **Controller:** De verbindende laag die ervoor zorgt dat invoer uit de view wordt omgezet naar specifieke verzoeken aan het model, en vervolgens de reactie van het model terug naar de view brengt.

Het MVC-patroon blijft een waardevol hulpmiddel in applicatieontwikkeling, vooral omdat het de complexiteit van moderne applicaties beheersbaar maakt. Door de scheiding van verantwoordelijkheden kunnen ontwikkelaars efficiënter werken, beter uitbreidingen implementeren en eenvoudiger onderhoud uitvoeren. Elk van de componenten draagt bij aan een betere architectuur die past bij zowel eenvoudige als complexe applicaties.

## 4.5 MVC voorbeeld: Spelletje hoger-lager

De bank gooit een getal (hier 5), de speler gokt of hij hoger of lager zal gooien. De speler gokt op hoger. De speler kan ook zijn inzet verhogen. Daarna bevestigt de speler de inzet en doet een worp (7). De uitkomst is inderdaad hoger ( $7 > 5$ ) waardoor zijn saldo met 10 is verhoogd van 300 naar 310.



**Code:**

```
<!DOCTYPE html>
<html lang="nl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Hoger Lager</title>
  <style>
    /* Algemene stijlen */
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: Arial, sans-serif;
      background-color: #e0e0e0;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      min-height: 100vh;
    }

    main {
      background-color: white;
      width: 95%;
      max-width: 1000px;
      border-radius: 8px;
      box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
      overflow: hidden;
    }

    header {
      background-color: black;
      color: white;
      text-align: center;
      padding: 10px 20px;
    }

    header h1 {
      font-size: 1.5rem;
    }

    /* Canvas sectie */
    .canvas-container {
      display: flex;
      justify-content: center;
      padding: 20px;
    }

    canvas {
      border: 1px solid #ccc;
      border-radius: 8px;
    }

    /* Navigatie sectie */
    nav {
      display: flex;
      flex-wrap: wrap;
      justify-content: space-between;
      align-items: center;
      padding: 20px;
      gap: 10px;
      background-color: #f4f4f4;
```

```

    border-top: 1px solid #ccc;
  }

  nav button {
    flex: 1;
    max-width: 200px;
    padding: 10px;
    font-size: 1rem;
    font-weight: bold;
    color: white;
    background-color: purple;
    border: none;
    border-radius: 5px;
    cursor: pointer;
  }

  nav button:hover {
    background-color: #6a0dad;
  }

  .radio-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 5px;
  }

  .radio-container input[type="radio"] {
    cursor: pointer;
  }

  .img-container {
    display: flex;
    gap: 10px;
  }

  .img-container img {
    width: 40px;
    height: 40px;
    cursor: pointer;
  }

  footer {
    text-align: center;
    padding: 10px;
    background-color: #f4f4f4;
    border-top: 1px solid #ccc;
    font-size: 0.9rem;
  }
</style>
</head>
<body>
<main>
  <header>
    <h1>Hoger Lager</h1>
  </header>
  <div class="canvas-container">
    <canvas id="canvas" width="850" height="475">
      Uw browser ondersteunt het HTML5-element canvas niet. Update uw browser of uw besturingssysteem voor alle
      mogelijkheden!
    </canvas>
  </div>
  <nav>
    <button id="bankGooien">Bank gooien</button>
    <div class="radio-container">
      <input type="radio" value="hoger" name="verwachting" id="hoger"> &uarr; /*pijl up*/
      <input type="radio" value="lager" name="verwachting" id="lager"> &darr; /* pijl down */
    </div>
  </nav>

```

```

</div>
<div class="img-container">
  
  
</div>
<button id="bevestigen">Bevestig</button>
<button id="spelerGooien">Speler gooien</button>
</nav>
<footer>&copy; A. Haksteen ** GSR **</footer>
</main>
<script>
"use strict";
window.onload = startHogerLager;
function startHogerLager(){
  var controller = new ControllerHogerLager();
}
</script>
<script>
"use strict";

class ControllerHogerLager {
  constructor(){
    this.view = new ViewHogerLager();
    this.bank = new BankHogerLager();
    this.speler = new SpelerHogerLager();
    this.dobbelsteen1Speler = new DobbelsteenHogerLager();
    this.dobbelsteen2Speler = new DobbelsteenHogerLager();
    this.dobbelsteen1Bank = new DobbelsteenHogerLager();
    this.dobbelsteen2Bank = new DobbelsteenHogerLager();
    this.klok = new KlokHogerLager();
    this.spel = new SpelHogerLager();

    this.setupEventListeners();

    this.view.toonMededeling("Welkom");
    this.view.toonDobbelstenenBank();
    this.view.toonDobbelstenenSpeler();

    document.getElementById("hoger").disabled = true; /* */
    document.getElementById("lager").disabled = true; /* */

    this.view.toonInzet(this.speler.getInzet());
    this.view.toonSaldo(this.speler.getSaldo());

    this.klok.setKlok(5,0);
    this.view.toonKlok("Ronde Tijd: ", this.klok.getMinuut(), this.klok.getSeconden());
    console.log("class ControllerHogerLager Constructor");
  }

  setupEventListeners() {
    document.getElementById("bankGooien").addEventListener("click", () => this.reageerOpBankGooien());
    document.getElementById("spelerGooien").addEventListener("click", () => this.reageerOpSpelerGooien());
    document.getElementById("inzetVerlagen").addEventListener("click", () => this.reageerOpInzetVerlagen());
    document.getElementById("inzetVerhogen").addEventListener("click", () => this.reageerOpInzetVerhogen());
    document.getElementById("hoger").addEventListener("click", () => this.reageerOpVoorspelHoger());
    document.getElementById("lager").addEventListener("click", () => this.reageerOpVoorspelLager());
    document.getElementById("bevestigen").addEventListener("click", () => this.reageerOpBevestigen());
    console.log("class ControllerHogerLager_setupEventListeners");
  }

  reageerOpBankGooien() {
    console.log("class ControllerHogerLager_reageerOpBankGooien");
    if (this.spel.getToestand() == 1) {
      this.view.resetDobbelstenen();
      this.dobbelsteen1Bank.Rol();
      this.dobbelsteen2Bank.Rol();
      this.view.toonDobbelstenenBank(this.dobbelsteen1Bank.getWaarde(), this.dobbelsteen2Bank.getWaarde());
    }
  }
}

```

```

        this.bank.setScore(this.dobbelsteen1Bank.getWaarde(), this.dobbelsteen2Bank.getWaarde());
        this.speler.resetInzet();
        this.view.toonInzet(this.speler.getInzet());
        this.view.toonSaldo(this.speler.getSaldo());
        this.spel.volgendeToestand();
        this.view.toonMededeling(""); /* toegevoegd */
        this.startKlok();
    } else if (this.spel.getToestand() != 1) {
        this.view.toonMededeling("U heeft al gegoid!");
    }
}

reageerOpBevestigen() {
    console.log("class ControllerHogerLager reageerOpBevestigen");
    if (this.spel.getToestand() == 3) {
        this.view.toonMededeling("Bevestigd!");
        this.spel.volgendeToestand();
    } else if (this.spel.getToestand() != 3) {
        this.view.toonMededeling("U heeft nog niet alles gedaan!");
    }
}

reageerOpSpelerGooien() {
    console.log("class ControllerHogerLager reageerOpSpelergooien");
    if (this.spel.getToestand() == 4) {
        this.dobbelsteen1Speler.Rol();
        this.dobbelsteen2Speler.Rol();
        this.view.toonDobbelstenenSpeler(this.dobbelsteen1Speler.getWaarde(), this.dobbelsteen2Speler.getWaarde());
        this.speler.setScore(this.dobbelsteen1Speler.getWaarde(), this.dobbelsteen2Speler.getWaarde());
        this.spel.bepaalWinnaar(this.speler.getVoorspelling(), this.bank.getScore(), this.speler.getScore());
        this.updateSaldo();
        this.view.toonInzet(this.speler.getInzet());
        this.view.toonSaldo(this.speler.getSaldo());

        if (this.speler.getSaldo() >= 500 || this.speler.getSaldo() == 0) {
            this.eindeSpel();
        } else if (this.speler.getSaldo() > 0 && this.speler.getSaldo() < 500) {
            this.view.toonMededeling("Einde van ronde, druk op bank gooien! De $" + this.spel.getWinnaar() + " heeft gewonnen!");
            this.spel.resetToestand();
            this.view.resetVerwachting();
        }
        clearInterval(this.intervalklok);
    } else if (this.spel.getToestand() != 4) {
        this.view.toonMededeling("FOUT! U heeft nog niet alles gedaan, doe dit eerst!");
    }
}

reageerOpInzetVerlagen() {
    console.log("class ControllerHogerLager_reageerOpInzetVerlagen");
    if (this.spel.getToestand() == 3) {
        this.speler.verlaagInzet();
        let inzet = this.speler.getInzet();
        if (inzet < 10) {
            this.speler.setInzet(10);
            this.view.toonMededeling("Die inzet kan niet lager worden dan 10!");
        } else {
            this.view.toonInzet(this.speler.getInzet());
            this.view.toonSaldo(this.speler.getSaldo());
            this.view.toonMededeling("Druk op Bevestigen als u voldoende u inzet heeft gewijzigd!");
        }
    } else if (this.spel.getToestand() != 3) {
        this.view.toonMededeling("U heeft nog niet alles gedaan!");
    }
}

reageerOpInzetVerhogen() {

```

```

        console.log("class ControllerHogerLager_reageerOpInzetVerhogen");
        if (this.spel.getToestand() == 3) {
            this.speler.verhoogInzet();
            let inzet = this.speler.getInzet();
            let verschil = 500 - this.speler.getSaldo();
            if (inzet > verschil || inzet > this.speler.getSaldo() && verschil > 0) {
                this.speler.setInzet(this.speler.getInzet() - 10);
                this.view.toonInzet(this.speler.getInzet());
                this.view.toonSaldo(this.speler.getSaldo());
                this.view.toonMededeling("De inzet kan niet hoger dan het saldo of hoger dan het verschil met 500 zijn!");
            } else {
                this.view.toonInzet(this.speler.getInzet());
                this.view.toonSaldo(this.speler.getSaldo());
                this.view.toonMededeling("Druk op Bevestigen als u voldoende u inzet heeft gewijzigd!");
            }
        } else if (this.spel.getToestand() != 3) {
            this.view.toonMededeling("U heeft nog niet alles gedaan!");
        }
    }

    reageerOpVoorspelHoger() {
        console.log("class ControllerHogerLager_reageerOpVoorspelHoger");
        if (this.spel.getToestand() < 4) {
            this.speler.voorspellingHoger();
            if (this.spel.getToestand() < 3) {this.spel.volgendeToestand();}
        }
    }

    reageerOpVoorspelLager() {
        console.log("class ControllerHogerLager_reageerOpVoorspelLager");
        if (this.spel.getToestand() < 4) {
            this.speler.voorspellingLager();
            if (this.spel.getToestand() < 3) {this.spel.volgendeToestand();}
        }
    }

    startKlok() {
        console.log("class ControllerHogerLager_startKlok");
        this.klok.setKlok(4, 59);
        this.intervalklok = setInterval(() => {
            this.view.toonKlok("Ronde Tijd: ", this.klok.getMinuut(), this.klok.getSeconden());
            this.klok.reset();
        }, 1000);
    }

    updateSaldo() {
        console.log("class ControllerHogerLager_updateSaldo");
        let newSaldo;
        if (this.spel.getWinnaar() == "Bank") {
            newSaldo = this.speler.getSaldo() - this.speler.getInzet();
        } else if (this.spel.getWinnaar() == "Speler") {
            newSaldo = this.speler.getSaldo() + this.speler.getInzet();
        }
        this.speler.setSaldo(newSaldo);
    }

    eindeSpel() {
        console.log("class ControllerHogerLager_eindeSpel");
        this.view.toonMededeling(this.speler.getSaldo() >= 500 ? "Het spel is afgelopen! U heeft gewonnen! Druk op Bank Gooien voor een nieuw spel!" : "Het spel is afgelopen! U heeft VERLOREN! Druk op Bank Gooien voor een nieuwe spel!");
        this.speler.setSaldo(300);
        this.speler.setInzet(10);
        this.view.toonInzet(this.speler.getInzet());
        this.view.toonSaldo(this.speler.getSaldo());
        this.view.resetVerwachting();
        this.spel.resetToestand();
    }
}

```



```

}
</script>
<script>
class SpelHogerLager{
    constructor(){
        this.winnaar = "";
        this.toestand = 1;
        console.log("class SpelHogerLager_constructor");
    }
    bepaalWinnaar(voorspelling, scoreBank, scoreSpeler){
        console.log("class SpelHogerLager bepaalWinnaar");
        if(voorspelling == "Hoger"){
            if(scoreSpeler > scoreBank){
                this.winnaar = "Speler";
            }
            else if(scoreSpeler < scoreBank){
                this.winnaar = "Bank";
            }
            else if(scoreSpeler == scoreBank){
                this.winnaar = "Bank";
            }
        }
        else if(voorspelling == "Lager"){
            if(scoreSpeler > scoreBank){
                this.winnaar = "Bank";
            }
            else if(scoreSpeler < scoreBank){
                this.winnaar = "Speler";
            }
            else if(scoreSpeler == scoreBank){
                this.winnaar = "Bank";
            }
        }
    }
    getWinnaar(){
        console.log("class SpelHogerLager_getWinnaar");
        return this.winnaar;
    }
    volgendeToestand(){
        console.log("class SpelHogerLager_volgendeToestand");
        this.toestand += 1;
    }
    resetToestand(){
        console.log("class SpelHogerLager_resetToestand");
        this.toestand = 1;
    }
    getToestand(){
        console.log("class SpelHogerLager_getToestand");
        return this.toestand;
    }
}
</script>
<script>
"use strict";

class SpelerHogerLager {
    constructor(){
        this.Saldo = 300;
        this.Inzet = 10;
        this.Score = 0;
        this.Voorspelling = "";
        console.log("class SpelerHogerLager_constructor");
    }
    getSaldo(){
        console.log("class SpelerHogerLager_getSaldo");
        return this.Saldo;
    }
}

```

```

        setSaldo(newSaldo){
        console.log("class SpelerHogerLager_setSaldo");
            this.Saldo = newSaldo;
        }

        getInzet(){
        console.log("class SpelerHogerLager_getInzet");
            return this.Inzet;
        }

        setInzet(newInzet){
        console.log("class SpelerHogerLager_setInzet");
            this.Inzet = newInzet;
        }

        verhoogInzet(){
        console.log("class SpelerHogerLager_verhoogInzet");
            this.Inzet += 10;
        }

        verlaagInzet(){
        console.log("class SpelerHogerLager_verlaagInzet");
            this.Inzet -= 10;
        }

        resetInzet(){
        console.log("class SpelerHogerLager_resetInzet");
            this.Inzet = 10;
        }

        setScore(dobb1speler, dobb2speler){
        console.log("class SpelerHogerLager_setScore_speler");
            this.Score = dobb1speler + dobb2speler;
        }

        getScore(){
        console.log("class SpelerHogerLager_getScore_speler");
            return this.Score;
        }

        voorspellingHoger(){
            this.Voorspelling = "Hoger";
            console.log("class SpelerHogerLager_voorspellingHoger");
        }

        voorspellingLager(){
            this.Voorspelling = "Lager";
            console.log("class SpelerHogerLager_voorspellingLager");
        }

        getVoorspelling(){
            console.log("class SpelerHogerLager_getVoorspelling");
            return this.Voorspelling;
        }
    }
</script>
<script>
"use strict";

class BankHogerLager
{
    constructor(){
        this.Score;
        console.log("class BankHogerLager_constructor");
    }
    setScore(dobb1bank, dobb2bank){
        console.log("class BankHogerLager_setScore_bank");
    }
}

```

```

        this.Score = dobb1bank + dobb2bank;
        //console.log(dobb1bank + dobb2bank); /* */
        document.getElementById("hoger").disabled = false; /* */
        document.getElementById("lager").disabled = false; /* */
    }
    getScore(){
        console.log("class BankHogerLager_getScore_Bank");
        return this.Score;
    }
}
</script>
<script>
"use strict";

class DobbelsteenHogerLager
{
    constructor(){
        this.Waarde;
        console.log("class DobbelsteenHogerLager_constructor");
    }

    Rol(){
        console.log("class DobbelsteenHogerLager_Rol");
        this.Waarde = Math.floor(Math.random() * 6) + 1;
    }

    getWaarde(){
        console.log("class DobbelsteenHogerLager_getwaarde");
        return this.Waarde;
    }
}
</script>
<script>
"use strict";

class KlokHogerLager{
    constructor(){
        this.minuut;
        this.seconden;
        console.log("class KlokHogerLager_constructor");
    }
    setKlok(minuten, seconden){
        this.minuut = minuten;
        this.seconden = seconden;
    }
    getUur(){
        return this.uur;
    }
    getMinuut(){
        return this.minuut;
    }
    getSeconden(){
        return this.seconden;
    }
    reset(){
        this.seconden -= 1;
        if(this.seconden <= 0){
            this.minuut -= 1;
            this.seconden = 59;
        }
    }
}
</script>
<script>
"use strict";

class ViewHogerLager{

```

```

constructor(){
    //Canvas klaarzetten voor gebruik
    this.pen = document.getElementById("canvas").getContext("2d");
    console.log("class ViewHogerLager_constructor");
}

toonDobbelstenenBank(dob1,dob2) {
    var x, y;
    console.log("class ViewHogerLager_toonDobbelstenenBank");
    //tekenen dobbelstenen van de Bank
    this.pen.clearRect(50,75,150,150);
    this.pen.strokeRect(50,75,150,150);

    switch(dob1){
        case 1:
            this.toonOog(125, 150);
            break;

        case 2:
            this.toonOog( 75, 100);
            this.toonOog(175, 200);
            break;

        case 3:
            this.toonOog( 75, 100);
            this.toonOog(175, 200);
            this.toonOog(125, 150);
            break;

        case 4:
            this.toonOog( 75, 100);
            this.toonOog(175, 200);
            this.toonOog(175, 100);
            this.toonOog( 75, 200);
            break;

        case 5:
            this.toonOog( 75, 100);
            this.toonOog(175, 200);
            this.toonOog(175, 100);
            this.toonOog( 75, 200);
            this.toonOog(125, 150);
            break;

        case 6:
            this.toonOog( 75, 100);
            this.toonOog( 75, 150);
            this.toonOog(175, 200);
            this.toonOog(175, 150);
            this.toonOog(175, 100);
            this.toonOog( 75, 200);
            break;
    }

    this.pen.clearRect(225,75,150,150);
    this.pen.strokeRect(225,75,150,150);

    switch(dob2){
        case 1:
            this.toonOog(300, 150);
            break;

        case 2:
            this.toonOog(250, 100);
            this.toonOog(350, 200);
            break;

        case 3:
            this.toonOog(250, 100);
            this.toonOog(350, 200);
            this.toonOog(300, 150);
            break;

        case 4:
            this.toonOog(250, 100);
    }

```

```

        this.toonOog(350, 200);
        this.toonOog(350, 100);
        this.toonOog(250, 200);
        break;
    case 5:
        this.toonOog(250, 100);
        this.toonOog(350, 200);
        this.toonOog(350, 100);
        this.toonOog(250, 200);
        this.toonOog(300, 150);
        break;
    case 6:
        this.toonOog(250, 100);
        this.toonOog(250, 150);
        this.toonOog(350, 200);
        this.toonOog(350, 150);
        this.toonOog(350, 100);
        this.toonOog(250, 200);
        break;
    }
    this.pen.clearRect(450,75,150,150);
    this.pen.clearRect(625,75,150,150);
}

toonDobbelstenenSpeler(dob3, dob4){
    var x, y;
    console.log("class ViewHogerLager toonDobbelstenenSpeler");
    // tekenen dobbelstenen van de Speler
    this.pen.clearRect(450,75,150,150);
    this.pen.strokeRect(450,75,150,150);

    switch(dob3){
        case 1:
            this.toonOog(525, 150);
            break;
        case 2:
            this.toonOog(475, 100);
            this.toonOog(575, 200);
            break;
        case 3:
            this.toonOog(475, 100);
            this.toonOog(575, 200);
            this.toonOog(525, 150);
            break;
        case 4:
            this.toonOog(475, 100);
            this.toonOog(575, 200);
            this.toonOog(575, 100);
            this.toonOog(475, 200);
            break;
        case 5:
            this.toonOog(475, 100);
            this.toonOog(575, 200);
            this.toonOog(575, 100);
            this.toonOog(475, 200);
            this.toonOog(525, 150);
            break;
        case 6:
            this.toonOog(475, 100);
            this.toonOog(475, 150);
            this.toonOog(575, 200);
            this.toonOog(575, 150);
            this.toonOog(575, 100);
            this.toonOog(475, 200);
            break;
    }
}

```

```

        this.pen.clearRect(625,75,150,150);
        this.pen.strokeRect(625,75,150,150);

        switch(dob4){
            case 1:
                this.toonOog(700, 150);
                break;

            case 2:
                this.toonOog(650, 100);
                this.toonOog(750, 200);
                break;

            case 3:
                this.toonOog(650, 100);
                this.toonOog(750, 200);
                this.toonOog(700, 150);
                break;

            case 4:
                this.toonOog(650, 100);
                this.toonOog(750, 200);
                this.toonOog(750, 100);
                this.toonOog(650, 200);
                break;

            case 5:
                this.toonOog(650, 100);
                this.toonOog(750, 200);
                this.toonOog(750, 100);
                this.toonOog(650, 200);
                this.toonOog(700, 150);
                break;

            case 6:
                this.toonOog(650, 100);
                this.toonOog(650, 150);
                this.toonOog(750, 200);
                this.toonOog(750, 150);
                this.toonOog(750, 100);
                this.toonOog(650, 200);
                break;
        }
    }

    toonOog(x, y){
        console.log("class ViewHogerLager_toonOog");
        this.pen.beginPath();
        this.pen.arc(x, y, 10, 0, 2*Math.PI);
        this.pen.fill();
        this.pen.stroke();
    }

    toonInzet(inzet) {
        console.log("class ViewHogerLager_toonInzet");
        this.pen.clearRect(25,420,150,50);
        this.pen.fillText("Inzet: " + inzet,25,460);
    }

    toonSaldo(saldo) {
        console.log("class ViewHogerLager_toonDobbelstenenBank");
        this.pen.fillText("Saldo: "+ saldo, 25,440);
    }

    //gebruik toonmededeling("") om de mededeling weg te halen
    toonMededeling(mededeling) {
        console.log("class ViewHogerLager_toonmededeling");
        this.pen.clearRect(50,250,850,50);
        this.pen.font = "18px sans-serif";
        this.pen.fillText(mededeling,50,290);
    }
}

```

```

resetVerwachting(){
    console.log("class ViewHogerLager_resetVerwachting");
    document.getElementById("hoger").checked = false;
    document.getElementById("lager").checked = false;
    document.getElementById("hoger").disabled = true;
    document.getElementById("lager").disabled = true;
}
resetDobbelstenen(){
    console.log("class ViewHogerLager_resetDobbelstenen");
    this.pen.clearRect(50,75,150,150);
    this.pen.clearRect(225,75,150,150);
    this.pen.clearRect(450,75,150,150);
    this.pen.clearRect(625,75,150,150);
}
toonKlok(tekst, minuut, seconden){
    console.log("class ViewHogerLager_toonKlok");
    this.pen.clearRect(605,15,175,45);
    this.pen.font = "18 px Corbel";
    if(minuut < 10 && seconden > 10){
        this.pen.fillText(tekst + "0" + minuut + ":" + seconden, 615,35);
    }
    else if(minuut < 10 && seconden < 10){
        this.pen.fillText(tekst + "0" + minuut + ":" + "0" + seconden, 615,35);
    }
    else if(minuut > 10 && seconden < 10){
        this.pen.fillText(tekst + minuut + ":" + "0" + seconden, 615,35);
    }
    else if(minuut >= 10 && seconden >= 10){
        this.pen.fillText(tekst + minuut + ":" + seconden, 615, 35);
    }
}
}
</script>
</body>
</html>

```

**NOTE!!** Vergeet niet de onderstaande plaatjes met de bijbehorende naam in dezelfde directory te zetten als de software.



up.jpg



down.jpg

## 4.6 MVC voorbeeld: Vier op een rij

Vul je naam en de naam van de speler in. Start het spel. Er wordt aangegeven wie aan de beurt is. Klik met de muis in de kolom waar je de munt wilt hebben. Wanneer er 4 op een rij wordt gedetecteerd wijst het programma de winnaar aan!



### Code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>vier op een rij</title>
    <!-- <link href="vieropenrij.css" rel="Stylesheet" /> -->

    <style>

*
{
  margin:0px;
  padding:0px;
}

body
{
  font-family:Arial;
  font-size:20px;
  background-color:Gray;
}

h1
{
  text-align:center;
  margin-bottom:20px;
}

#startscherm
{
  display:block;
}
```



```

#speelveldscreen
{
    display:block;
}

#wrapper
{
    background-color:White;
    width:780px;

    margin:20px auto;
    padding:10px 0px 5px 0px;
}

header
{
    margin-top:20px;
    margin-left:20px;
}

#speelveld
{
    width:364px;
    height:312px;
    border: 1px solid #666666;
    margin-right:auto;
    margin-left:auto;
    margin-bottom:20px;
}

#speler1
{
    float:left;
    width:200px;
    text-align:center;
}

#speler2
{
    float:right;
    width:200px;
    text-align:center;
}

#namenveld div
{
    padding: 20px;
}

#namenveld
{
    width:500px;

    margin-right:auto;
    margin-left:auto;
    margin-bottom:20px;
}

.buttonswrapper{
    text-align:center;
}

.klein
{
    font-size:60%;
    color:red;
}

```

```

}
.fiche
{
    height:50px;
    background: url(leeg.jpg);
    background-size:100% 100%;
    background-repeat:no-repeat;
    width:50px;
    border:1px solid #666666;
    margin:20px auto;
}

.fiche r
{
    height:50px;
    background: url(ROOD.jpg);
    background-size:100% 100%;
    background-repeat:no-repeat;
    width:50px;
    border:1px solid #666666;
    margin:20px auto;
}

.fiche_o
{
    height:50px;
    background: url(ORANJE.jpg);
    background-size:100% 100%;
    background-repeat:no-repeat;
    width:50px;
    border:1px solid #666666;
    margin:20px auto;
}

#speelveld div
{
    height:50px;
    background: url(../images/spel/leeg.jpg);
    background-size:100% 100%;
    background-repeat:no-repeat;
    width:50px;
    float: left;
    border:1px solid #666666;
}

button
{
    margin:0px 60px;
}

#mededeling {
    width: 350px;
    margin:20px auto;
    border-color:red;
    border-width: thin;
    border-style:solid;
    padding:10px;
    text-align:center;
    color:red;
}

footer
{
    padding:10px;
    text-align:right;
    font-size:0.5em;
}

```

```

        font-style:italic;
        color:red;
    }

    </style>

    <!-- <script type="text/javascript" src="Controller.js"> </script> */
    <script type="text/javascript" src="View.js"> </script>
    <script type="text/javascript" src="Speler.js"> </script>
    <script type="text/javascript" src="VierOpEenRij.js"> </script>
    <script type="text/javascript" src="app.js"> </script> -->

    <script>

'use strict'

class Controller
{
    constructor()
    {
        var controller = this;
        this.view = new View();
        this.speler1 = new Speler("ROOD");
        this.speler2 = new Speler("ORANJE");
        this.spel = new VierOpEenRij(this.speler1);

        document.querySelector('#speelveld').addEventListener('click',
function(e){controller.reageerKlikOpSpeelveld(e)});
        document.querySelectorAll('.buttonswrapper>button')[0].addEventListener('click',
function(e){controller.reageerKlikOpNieuwSpel()});
        document.querySelectorAll('.buttonswrapper>button')[1].addEventListener('click',
function(e){controller.reageerKlikOpStart()});
        document.querySelectorAll('.buttonswrapper>button')[2].addEventListener('click',
function(e){controller.reageerKlikOpStoppen()});

        this.view.toonSpeelveldschermb();
    }

    reageerKlikOpNieuwSpel()
    {
        var fiches = this.spel.getFiches();

        this.spel.reset();
        this.view.toonFiches(fiches);

        this.view.toonMededeling(this.spel.getSpelerAanDeBeurt().getNaam() + " is aan de beurt");

        this.spel.maakSpelActief();
        this.view.verbergNavigatie();
    }

    reageerKlikOpStoppen()
    {
        //alert("controller_reageerKlikOpStoppen");
        this.view.toonStartschermb();
        this.view.resetNamen();
    }

    reageerKlikOpStart()
    {
        var naam1 = document.getElementById("naam1").value;
        var naam2 = document.getElementById("naam2").value;

        if(naam1!=naam2 && naam1!="" && naam2 != "")
        {
            this.spel.reset();

```

```

        //update view
        var fiches = this.spel.getFiches();
        this.view.toonFiches(fiches);

        this.speler1.resetScore();
        this.speler2.resetScore();

        this.speler1.setNaam(naam1);
        this.speler2.setNaam(naam2);

        //this.view.toonSpeelveldschermb();
        this.view.verbergNavigatie();
        this.view.toonSpelers(this.speler1, this.speler2);

        this.view.toonMededeling(this.spel.getSpelerAanDeBeurt().getNaam() + " is aan de beurt");

        this.spel.maakSpelActief();
    }
}

reageerKlikOpSpeelveld(event)
{
    var actieveSpeler,passieveSpeler;

    if (this.spel.isSpelActief())
    {
        var kolomNummer = this.view.getGeklikteKolom(event);

        var hetMag = this.spel.magZet(kolomNummer,this.spel.getSpelerAanDeBeurt().getKleur());

        if(hetMag === true)
        {
            var winnaar= this.spel.isWinnaar(this.spel.getSpelerAanDeBeurt().getKleur());

            if(winnaar === true)
            {
                this.spel.spelerAanDeBeurt.verhoogScore();
                this.view.toonMededeling(this.spel.getSpelerAanDeBeurt().getNaam()+"
heeft gewonnen!!!");

                this.view.toonSpelers(this.speler1, this.speler2);
                this.view.toonNavigatie();
            }

            var gelijkspel = this.spel.isGelijkspel();

            if(gelijkspel == true)
            {
                this.view.toonMededeling("gelijkspel!");
                this.view.toonNavigatie();
            }

            //update view
            var fiches = this.spel.getFiches();
            this.view.toonFiches(fiches);

            if(this.spel.getSpelerAanDeBeurt().getNaam() === this.speler1.getNaam()){
                this.spel.spelerAanDeBeurt = this.speler2;
            }
            else
            {
                this.spel.spelerAanDeBeurt = this.speler1;
            }

            if (this.spel.isSpelActief())
            {

```



```

        //alert("View_toonSpeelveldschem");
        document.querySelector("#speelveldschem").style.display = "block";
        document.querySelector("#startscherm").style.display = "none";
    }

    toonStartschem()
    {

        document.querySelector("#startscherm").style.display = "none";
        document.querySelector("#startscherm").style.display = "block";
        //alert("View_toonSpeelveldschem");
    }

    toonMededeling(mededeling)
    {

        //alert("View_toonMededeling");
        var mededelingDiv = document.querySelector("#mededeling");

        mededelingDiv.innerHTML = mededeling;
    }

    getGeklikteKolom(event)
    {

        var bron = event.target;
        var i = this.vakjesDivs.length;

        //this.vakjesDivs = document.querySelectorAll("#speelveld>div");

        while (this.vakjesDivs[i] !== bron && i > -1)
        {i--;
        }

        if (i > -1 )
        {
            console.log(i%7)
            return i%7;
        } else
        {return -1;}
    }

    toonNavigatie()
    {

        //alert("View_toonNavigatie");
        document.querySelector("#spelknoppen").style.visibility = "visible";
    }

    verbergNavigatie()
    {

        //alert("View_verbergNavigatie");
        document.querySelector("#spelknoppen").style.visibility = "hidden";
    }

    toonFiches(fiches)
    {

        //alert("View_toonFiches");
        for( var rij = 0; rij < 6; rij++)
        {
            for (var kolom = 0; kolom < 7; kolom++)
            {
                var positie = rij * 7 + kolom;
                this.vakjesDivs[positie].style.backgroundImage = "url('"+fiches[rij][kolom]+".jpg')";
            }
        }
    }
} </script>

```

```

<script> 'use strict'

class Speler
{
    constructor(kleur){
        //alert("Speler");
        this.kleur = kleur;
        this.score = 0;
        this.naam = "";
    }

    resetScore()
    {
        //alert("Speler_resetScore");
        this.score = 0;
    }

    getNaam()
    {
        //alert("Speler_getNaam");
        return this.naam;
    }

    setNaam(naam)
    {
        //alert("Speler_setNaam");
        this.naam = naam;
    }

    getScore()
    {
        //alert("Speler_getScore");
        return this.score;
    }

    verhoogScore()
    {
        //alert("Speler_verhoogScore");
        this.score++;
    }

    getKleur()
    {
        //alert("Speler_getKleur");
        return this.kleur;
    }
} </script>
<script> 'use strict'

class VierOpEenRij
{
    constructor(spelerAanDeBeurt){
        //alert("VierOpEenRij");

        this.vakjes = new Array(6);
        this.spelActief = false;
        this.spelerAanDeBeurt = spelerAanDeBeurt;

        for( var i = 0; i < 6; i++)
        {
            this.vakjes[i]=new Array(7);
        }
        // this.reset();
    }
}

```

```

    }

    maakSpelActief()
    {
        //alert("VierOpEenRij_maakSpelActief");
        this.spelActief = true;
    }

    isSpelActief()
    {
        //alert("VierOpEenRij_isSpelActief");
        return this.spelActief;
    }

    isGelijkspel()
    {
        //alert("VierOpEenRij_isGelijkspel");
        var gelijk=true;
        for(var rij=0;rij<6;rij++)
        {
            for (var kolom = 0; kolom < 7; kolom++)
            {
                if (this.vakjes[rij][kolom] == "LEEG")
                {
                    gelijk=false;
                }
            }
        }

        if(gelijk==true)
        {
            this.spelActief=false;
        }

        return gelijk;
    }

    isWinnaar(kleur)
    {
        var winnaar = false;
        //alert("VierOpEenRij_isWinnaar");
        //Controle Horizontaal
        for(var rij = 0; rij < 6; rij++)
        {
            for (var kolom = 0; kolom <= 3; kolom++)
            {
                if ((this.vakjes[rij][kolom] == kleur && this.vakjes[rij][kolom + 1] == kleur &&
this.vakjes[rij][kolom + 2] == kleur && this.vakjes[rij][kolom + 3]) == kleur)
                {
                    this.vakjes[rij][kolom] = "GROEN";
                    this.vakjes[rij][kolom+1] = "GROEN";
                    this.vakjes[rij][kolom+2] = "GROEN";
                    this.vakjes[rij][kolom+3] = "GROEN";
                    this.spelActief = false;
                    winnaar = true;
                }
            }
        }

        //Controle Verticaal
        for(var kolom = 0; kolom < 7; kolom++)
        {
            for (var rij = 0; rij <= 2; rij++)
            {
                if ((this.vakjes[rij][kolom] == kleur && this.vakjes[rij+1][kolom] == kleur &&
this.vakjes[rij+2][kolom] == kleur && this.vakjes[rij+3][kolom]) == kleur)
                {

```



```

        this.vakjes[rij][kolom] = "GROEN";
        this.vakjes[rij+1][kolom] = "GROEN";
        this.vakjes[rij+2][kolom] = "GROEN";
        this.vakjes[rij+3][kolom] = "GROEN";
        this.spelActief = false;
        winnaar = true;
    }
}

//Controle Diagonaal (hoog naar laag)
for (var kolom = 0; kolom <= 3; kolom++)
{
    for (var rij = 0; rij <= 2; rij++)
    {
        if (this.vakjes[rij][kolom] == kleur && this.vakjes[rij + 1][kolom + 1] == kleur &&
this.vakjes[rij + 2][kolom + 2] == kleur && this.vakjes[rij + 3][kolom + 3] == kleur)
        {
            this.vakjes[rij][kolom]="GROEN";
            this.vakjes[rij + 1][kolom + 1]="GROEN";
            this.vakjes[rij + 2][kolom + 2]="GROEN";
            this.vakjes[rij + 3][kolom + 3]="GROEN";
            this.spelActief=false;
            winnaar = true;
        }
    }
}

//Controle Diagonaal (laag naar hoog)
for (var kolom = 0; kolom <= 3; kolom++)
{
    for (var rij = 5; rij >= 3; rij--)
    {
        if (this.vakjes[rij][kolom] == kleur && this.vakjes[rij - 1][kolom + 1] == kleur &&
this.vakjes[rij - 2][kolom + 2] == kleur && this.vakjes[rij - 3][kolom + 3] == kleur)
        {
            this.vakjes[rij][kolom] = "GROEN";
            this.vakjes[rij - 1][kolom + 1] = "GROEN";
            this.vakjes[rij - 2][kolom + 2] = "GROEN";
            this.vakjes[rij - 3][kolom + 3] = "GROEN";
            this.spelActief = false;
            winnaar = true;
        }
    }
}

return winnaar;
}

reset()
{
    document.getElementById("naam1").readOnly = true;
    document.getElementById("naam2").readOnly = true;
    //alert("VierOpEenRij_reset");
    for( var rij = 0; rij < 6; rij++)
    {
        for (var kolom = 0; kolom < 7; kolom++)
        {
            this.vakjes[rij][kolom] = "LEEG";
        }
    }
}

getFiches()
{
    //alert("VierOpEenRij_getFiches");

```

```

        return this.vakjes;
    }

    getSpelerAanDeBeurt()
    {
        //alert("VierOpEenRij_getSpelerAanDeBeurt");
        return this.spelerAanDeBeurt;
    }

    magZet(kolom, kleur)
    {
        //alert("VierOpEenRij_magZet");
        for(var rij = 5; rij >= 0; rij--)
        {
            if(this.vakjes[rij][kolom] === "LEEG")
            {
                this.vakjes[rij][kolom] = kleur;

                return true;
            }
        }

        return false;
    }
} </script>
<script> "use strict"

window.onload=startApp;

function startApp()
{
    //alert("Nieuw_app");
    var myController = new Controller();
} </script>

</head>
<body>
    <div id='speelveldscherm'>
        <div id="wrapper">
            <header>
                
                <h1>Vier op een rij <sup><span
class='klein'>SuperLite<span></sup> </h1>
            </header>

            <div id='speler1'>
                <div>Speler_1</div>
                <input type="text" id="naam1" size ="10" />
                <div class='fiche_r'></div>
                <div>0</div>
            </div>

            <div id='speler2'>
                <div>Speler_2</div>
                <input type="text" id="naam2" size ="10" />
                <div class='fiche_o'></div>
                <div>0</div>
            </div>
            <div id="speelveld">

                <div></div>
                <div></div>
                <div></div>
                <div></div>
                <div></div>
                <div></div>
            </div>
        </div>
    </div>

```

```

<div></div>

<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>

<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>

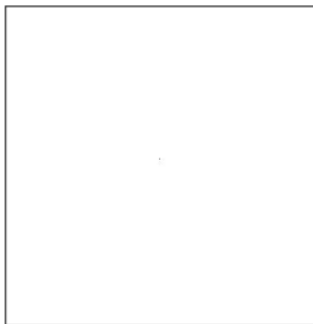
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>

<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>

</div>
<div id='mededeling'>
    Vul jullie namen in, start daarna het spel
</div>
<div id="spelknoppen" class="buttonswrapper">
    <button>nieuw spel</button>
    <button>spel starten</button>
    <button>stoppen</button>
</div>
<footer>copyright Tony Haksteen </footer>
</div>
</body>
</html>

```

**NOTE!!** Vergeet niet de onderstaande plaatjes met de bijbehorende naam in dezelfde directory te zetten als de software.



LEEG.jpg



ROOD.jpg

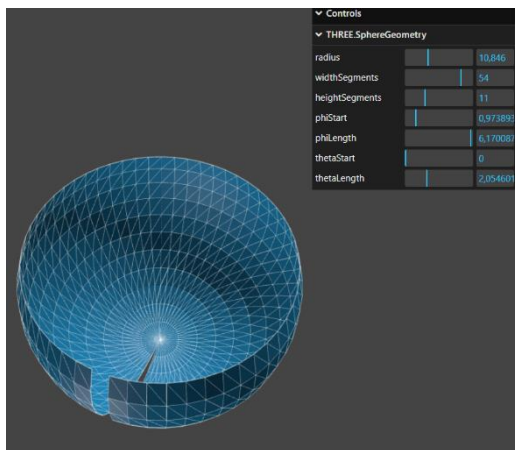


ORANJE.jpg



GROEN.jpg

## 4.7 Manipulatie van een draaiende bal



Als laatste nog een link naar: [Three.js Geometry Browser](#)

Waarin je een draaiende bal kunt manipuleren.