Sterren Plukken met Phaser 3

In deze workshop gaan we een eenvoudig spelletje maken met Phaser (versie 3). Door het spel te maken leer je wat je met Javascript en Phaser kunt doen en je eigen spel maken voor op de computer.

Wat is Phaser?

De eerste vraag is misschien: wat is dat eigenlijk, Phaser?

Phaser is een verzameling functies, gemaakt in JavaScript, waarmee het heel makkelijk is om games te maken die in de browser werken. Omdat al die functies door andere mensen al gemaakt is, hoef je die zelf niet meer te bedenken. Als je meer wilt weten over Phaser ga je naar de website phaser.io.

Wat heb je nodig?

Om te beginnen heb je de volgende dingen nodig:

- Een **browser** (duh). Dit is bijvoorbeeld Chrome, Firefox of Edge.
- Een programma om code te schrijven. Wij gebruiken hiervoor **Visual Studio Code**.
- Een **webserver** om de bestanden van je spel naar de browser te sturen. Wij gebruiken het programma Fenix zodat je alles op je eigen computer kunt doen.

Browser

Een browser staat standaard al geïnstalleerd. Het maakt niet uit welke je gebruikt. Moderne browsers zijn prima geschikt voor spelletjes.

Visual Studio Code

Voor het schrijven van JavaScript code kan je notepad (of kladblok) gebruiken, maar erg prettig is dat niet. We gebruiken liever Visual Studio Code. Ook dit programma staat al geinstalleerd, maar je kunt het ook gratis downloaden op: https://code.visualstudio.com/

Webserver

Een webserver zorgt ervoor dat de browser het spel kan openen. Vroeger kon je die bestanden ook vaak direct vanaf de computer openen, maar de makers vonden dat niet zo veilig. Je kon er allerlei rare dingen mee doen, en virussen mee oplopen. Dat willen we natuurlijk niet!

Je kunt in Visual Studio Code ook een webserver gebruiken. Je installeert hem zo:

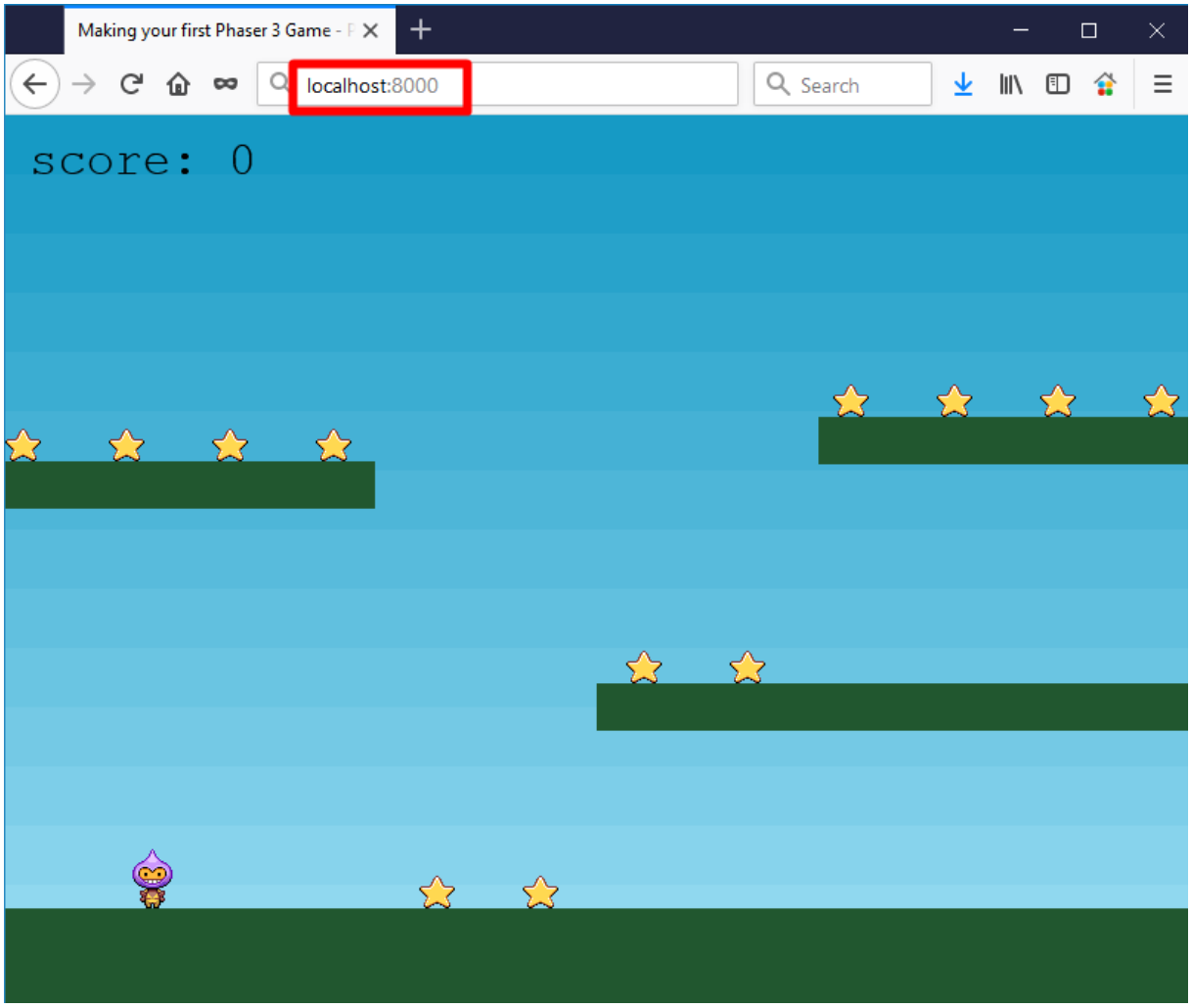
Druk in Visual Studio Code op CTRL+P en type

```
ext install ritwickdey.liveserver
```

Druk op ENTER om de installatie te starten. Als het klaar is, zie je rechtsonder in je scherm: 

.

Je spel kan je starten door op Go Live te klikken. Dan verschijnt het volgende scherm... als je klaar ben met programmeren!



Genoeg voorbereiding! Aan de slag.

De onderdelen van het spel

Voor het gemak hebben we map gemaakt waar al wat in staat. Start Visual Studio Code en kies **File > Open Folder**. Ga naar de map **\javascript\opdrachten\sterrenplukken**en kies **Select Folder**.

Hier zie je bestanden index.html en game.js. In index.html staat dit:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Phaser Game</title>
    <script src="//cdn.jsdelivr.net/npm/phaser@3.11.0/dist/phaser.js"></script>
    <style type="text/css">
        body {
            margin: 0;
        }
    </style>
</head>
<body>
<script type="text/javascript" src="game.js"></script>
</body>
</html>
```

Het bestand index.html is alleen een plek om spel te kunnen open in een browser. Het echte werk zit in het bestand game.js.

Maar dat bestand is nu nog een beetje leeg. Nou ja, niet helemaal.

```
var config = {
    type: Phaser.AUTO,
    width: 800,
    height: 600,
    physics: {
        default: 'arcade',
        arcade: {
            gravity: { y: 300 },
            debug: false
        }
    },
    scene: {
        preload: preload,
        create: create,
        update: update
    }
};

function preload ()
{
}
```

```
function create ()
{
}

function update ()
{
}
```

Om het spel te kunnen starten hebben we een paar instellingen nodig. Die instellingen slaan we op in de variabele **config.** Je ziet hier bijvoorbeeld dat het spel een vlak gebruikt van 800 pixels breed en 600 pixels hoog. Je kunt ook zelf kiezen hoeveel zwaartekracht er is in je spel[1].

Het spel heeft één scene en die scene bevat afbeeldingen die geladen moeten worden. Dat gebeurt in de functie **preload**. Dan wordt de scene opgebouwd en alles plaatjes krijgen een plekje in het spel. Daarvoor is de functie **create**. Als het spel begint gaat de scene bewegen natuurlijk. Een poppetje beweegt en er gebeurt wat op het scherm. Dat is het werk van de functie **update**.

De onderdelen laden

In het begin moeten we alle afbeeldingen laden. Alles wat we in de scene zien wordt opgehaald in de functie **preload**.

```
function preload ()
{
    this.load.image('sky', 'assets/sky.png');
    this.load.image('ground', 'assets/platform.png');
    this.load.image('star', 'assets/star.png');
    this.load.image('bomb', 'assets/bomb.png');
    this.load.spritesheet('dude',
        'assets/dude.png',
        { frameWidth: 32, frameHeight: 48 }
    );
}
```

Je ziet dat we lucht hebben, maar ook grond, sterren, en nog meer. Voor het laden gebruiken we de functie **this.load.image**. Deze functie komt uit Phaser en gebruikt twee parameters:

- De eerste is een tekst zodat we later in de code kunnen verwijzen naar bijvoorbeeld 'star' als we iets moeten doen met het de sterren in het spel.
- De tweede parameter verwijst naar het bestand, het plaatje, in de map assets.

[1] Zwaartekracht zorgt ervoor dat alles naar beneden gaat. Maar net als in de echte wereld valt een bal of poppetje niet ineens, met een vaste snelheid, naar beneden. Hoe langer je valt, hoe sneller het gaat.

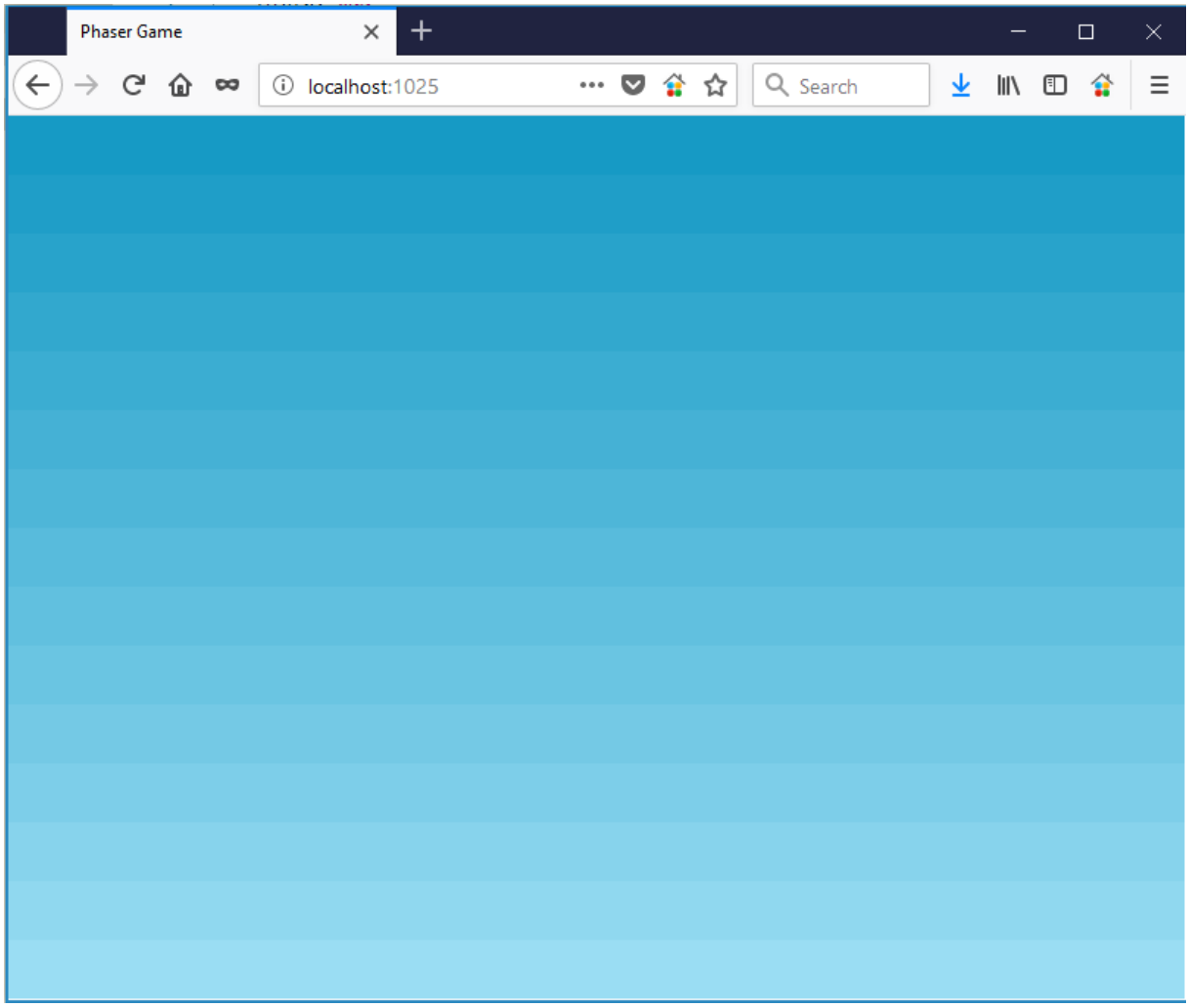
De onderdelen laten zien

Als we de onderdelen geladen hebben kunnen we op het scherm laten zien. Dat zit allemaal in de functie create. Zet er maar vast het volgende in.

```
function create ()
{
    this.add.image(400, 300, 'sky');
}
```

Misschien vraag je je af, waarom staat daar 400 en 300? Dat komt omdat in Phaser alle objecten in het midden beginnen. Als je 0,0 zou invullen als plek voor de lucht, dan zou je alleen de rechter onderkant zien. Het speelveld is 800 bij 600 pixels, dus begint de linker bovenhoek op 400, 300.

Kijk maar vast in de browser naar http://localhost:1025 om te zien of het goed gaat.



Dat ziet er nog wel een beetje leeg uit. Met de functie **this.add.image** kan je afbeeldingen toevoegen aan de scene. Het maakt Phaser niet uit wáár je de afbeelding neerzet. Als je het buiten het vlak van 800x600 zet zie je het niet, maar is het nog wel onderdeel van de scene. Die scene heeft geen grenzen en gaat dus oneindig door buiten het vlak dat je ziet.

We gaan wat grondvlakken toevoegen. Die vlakken zit in een groep om ze allemaal op dezelfde manier te gebruiken in het spel (als een plek om op de staan).

Voeg de volgende code toe aan de functie **create**

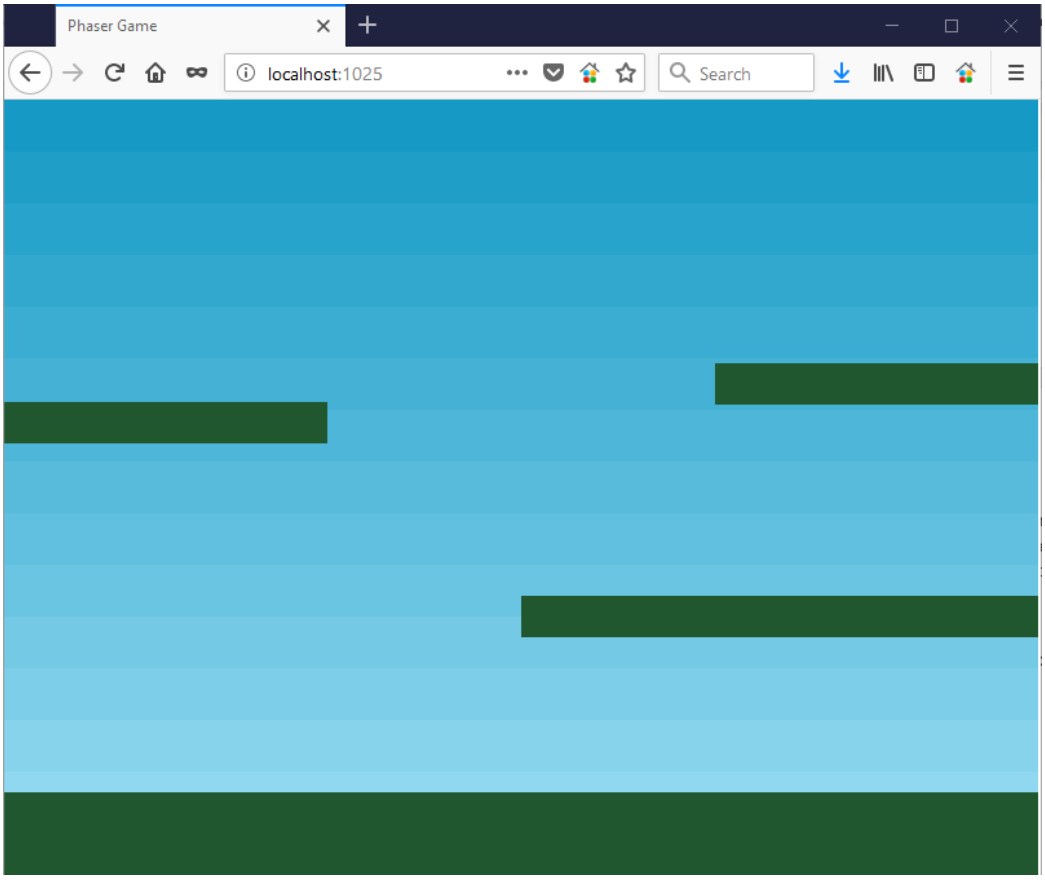
```
    platforms = this.physics.add.staticGroup();

    platforms.create(400, 568, 'ground').setScale(2).refreshBody();

    platforms.create(600, 400, 'ground');
    platforms.create(50, 250, 'ground');
    platforms.create(750, 220, 'ground');
```

De grondvlakken, in je code noemen we het platforms, vormen een statische groep. Phaser kent twee groepen: statische groepen en dynamische groepen. Een dynamische groep beweegt op je scherm en kan je sneller of langzamer laten bewegen. Een statische groep staat vast, beweegt dus niet, en heeft ook een vaste vorm. De grond is natuurlijk een goed voorbeeld van iets dat niet beweegt[2].

Voor de grond hebben we een simpel plaatje van 400x32 pixels groot. Maar omdat de grond op de bodem van de scene wat groter moet zijn voeg je dit toe: **setScale(2).refreshBody();**

Zo wordt het plaatje twee keer groter. Groot genoeg zodat de held van het spel straks niet zomaar naar beneden valt.

De scene ziet er nu zo uit.



[2] Ja, tenzij we een aardbeving in het spel programmeren. Maar vandaag doen we dat niet.

Leuk plaatje, maar waar is de held van het spel? Hiervoor voeg je de volgende code toe aan de functie **create**.

```
    player = this.physics.add.sprite(100, 450, 'dude');

    player.setBounce(0.2);
    player.setCollideWorldBounds(true);
    player.body.setGravityY(300);

    this.anims.create({
        key: 'left',
        frames: this.anims.generateFrameNumbers('dude', { start: 0, end: 3 }),
        frameRate: 10,
        repeat: -1
    });

    this.anims.create({
        key: 'turn',
        frames: [ { key: 'dude', frame: 4 } ],
        frameRate: 20
    });

    this.anims.create({
        key: 'right',
        frames: this.anims.generateFrameNumbers('dude', { start: 5, end: 8 }),
        frameRate: 10,
        repeat: -1
    });
```

In dit stukje code gebeuren twee dingen. Eerst voeg je de held toe aan het spel met de functie **this.physics.add.sprite**. Zie je dat je hier weer eerst de plek kiest (100, 450) en dan de naam van het plaatje ('dude')?

Nadat je het plaatje hebt toegevoegd staan er ook wat animaties bij. Onze held komt in het spel met een klein sprongetje. Daar zorgt player.setBounce(0.2) voor. Je kunt met dat getal 0.2 een beetje spelen om te zien wat het effect is.

Als je het plaatje van de held goed bekijkt, dan zie je dat er in één plaatje wel negen posities zijn bewaard. Zo'n positie noemen we een frame.

De animaties voor links ('left') gebruikt de frames 0, 1, 2 en 3. De animatie voor rechts gebruikt de frames 5, 6, 7 en 8.

Als je het spel nu opnieuw in de browser laadt zie je dat het poppetje door de bodem valt. Je weet dat Phaser dynamische en statische groepen heeft. De grondvlakken zijn statisch. Het poppetje is dynamisch, want die kan bewegen. Maar nu moeten we nog vertellen dat het poppetje wordt tegengehouden door de grond. Dat doen we via een zogenoemde Collider. Voeg deze regel toe aan de **create** functie.

```
    this.physics.add.collider(player, platforms);
```

De Collider is een magisch object. Het test tijdens het spel of twee object tegen elkaar botsen en stopt dan de beweging van het dynamische object. Nu blijft onze held met twee pootjes op de grond staan.

Bewegen

Nu moeten we het poppetje nog laten bewegen. Gelukkig helpt Phaser ook daarmee met deze simpele regel.

```
    cursors = this.input.keyboard.createCursorKeys();
```

Deze kan je toevoegen aan de **create** functie. We hebben nu gelijk controle over de vier pijltjestoetsen op je toetsenbord. Je moet nog wel vertellen welke kant de held op moet.

Hiervoor voeg je de volgende code toe aan de **update** functie.

```
    if (cursors.left.isDown)
    {
        player.setVelocityX(-160);

        player.anims.play('left', true);
    }
    else if (cursors.right.isDown)
    {
        player.setVelocityX(160);

        player.anims.play('right', true);
    }
    else
    {
        player.setVelocityX(0);

        player.anims.play('turn');
    }
```

Waarom moet dit in de **update** functie? Dat komt omdat deze functie tijdens het spel 60 keer per seconde wordt uitgevoerd. Zo vaak wordt dus gecontroleerd of je op een pijltjestoets drukt zodat het poppetje kan bewegen. Nu kan het poppetje naar links en naar rechts. Als de toets wordt los gelaten stopt hij gelijk. De functie **player.setVelocityX** doet dit en de parameter is de snelheid waarmee het poppetje rent. Een snelheid van 0 betekent natuurlijk stoppen.

Springen

Als het pijltje omhoog wordt ingedrukt moet het poppetje kunnen springen. Voeg de volgende code toe aan de **update** functie.

```
    if (cursors.up.isDown && player.body.touching.down)
    {
        player.setVelocityY(-330);
    }
```

Als je de toets los laat valt het poppetje vanzelf weer naar beneden. Probeer het getal van de functie **player.setVelocityY** zo te maken dat het poppetje van vlak naar vlak kan springen.

Vallende Sterren

Om onze held wat leuks te laten doen voegen we sterren toe aan de scene. In de **create** functie voeg je dit toe.

```
    stars = this.physics.add.group({
        key: 'star',
        repeat: 11,
        setXY: { x: 12, y: 0, stepX: 70 }
    });

    stars.children.iterate(function (child) {
        child.setBounceY(Phaser.Math.FloatBetween(0.4, 0.8));
    });
```

Dit lijkt op de manier waarop je de grond hebt toegevoegd aan het spel. Alleen is dit een dynamische groep. Je voegt 12 sterren toe. Hé, dat is gek. Er staat toch 11? Dat komt de functie **this.physics.add.group** al direct één keer wordt uitgevoerd. Daarna zal hij dat nog 11 keer herhalen. De sterren komen telkens 70 pixels verder naast elkaar te staan.

De sterren beginnen bovenaan en omdat het dynamische objecten zijn, en dus reageren op zwaartekracht, vallen ze naar beneden. De functie **child.setBounceY** zorgt voor een beetje trampoline effect.

Als je dit nu in de browser bekijkt zal je merken dat de sterren dwars door de grond zakken. Niks houdt ze tegen. Dat lossen we zo op.

```
this.physics.add.collider(stars, platforms);
```

Zet dit in de **create** functie zodat de sterren door de grond worden tegengehouden. Net als onze held.

Laten we gelijk maar instellen dat je controleert of het poppetje tegen een ster loopt.

```
this.physics.add.overlap(player, stars, collectStar, null, this);
```

Bedenk zelf maar waar je dit toevoegt. Dat moet inmiddels niet meer zo moeilijk zijn. In deze regel staat ook de naam van een functie **collectStar**. Maar die functie hebben we nog niet. Zet het volgende onderaan in je code.

```
function collectStar (player, star)
{
    star.disableBody(true, true);
}
```

Deze functie zorgt er nu alleen voor dat de ster waar je tegenaan loopt wordt uitgeschakeld en verborgen. Nu kan je dus met het poppetje sterren verzamelen.

Score bijhouden

Het zou wel leuk zijn als we ook de score zouden bijhouden zodat je ziet hoeveel sterren je hebt verzameld. Nu kan je nog makkelijk bij de sterren, maar dat maken we straks wat lastiger. Wacht maar af.

Eerst de score. Voeg hiervoor twee variabelen toe aan je code. Zet het maar boven de **create** functie. Let op: niet *in* een functie!

```
var score = 0;
var scoreText;
```

In de **create** functie geven we de scoreText een plekje op het scherm.

```
scoreText = this.add.text(16, 16, 'score: 0',
                    { fontSize: '32px', fill: '#000' });
```

De score komt zo linksboven te staan.

Als je dat hebt gedaan moeten we de **collectStar** functie uitbreiden. Elke keer als de held een ster plukt wordt er 10 bij de score opgeteld.

```
    score += 10;
    scoreText.setText('Score: ' + score);
```

De tekst op het scherm wordt ook bijgewerkt om de nieuwe score te laten zien. Bekijk het resultaat eens zodat je weet of je op de goede weg bent.

Moeilijker maken

Sterren plukken is natuurlijk leuk, maar niet heel spannend. We maken het lastiger door iedere keer na het ophalen van alle sterren een bommetje los te laten. Probeer de sterren opnieuw te verzamelen zonder geraakt te worden! Hoe meer bommetjes er rondspringen, hoe lastiger het wordt natuurlijk.

De bommetjes voeg je zo toe (in de **create** functie of had je dat zelf al bedacht).

```
    bombs = this.physics.add.group();

    this.physics.add.collider(bombs, platforms);

    this.physics.add.collider(player, bombs, hitBomb, null, this);
```

De bommetjes zijn ook weer een dynamische groep, want ze springen in het rond. Maar als de held tegen een bom botst is het… game over. De functie hitBomb zorgt daarvoor. Die moet je nog wel toevoegen. Zet dit onderaan je code.

```
function hitBomb (player, bomb)
{
    this.physics.pause();

    player.setTint(0xff0000);

    player.anims.play('turn');

    gameOver = true;
}
```

We zijn er bijna. De bommetjes moeten nog wel losgelaten worden zodra alle sterren zijn verzameld.

Dat is een stuk code die je in de **collectStar** functie moet zetten.

```
    if (stars.countActive(true) === 0)
    {
        stars.children.iterate(function (star) {
            star.enableBody(true, star.x, 0, true, true);
        });

        var x = (player.x < 400) ?
                Phaser.Math.Between(400, 800) : Phaser.Math.Between(0, 400);

        var bomb = bombs.create(x, 16, 'bomb');
        bomb.setBounce(1);
        bomb.setCollideWorldBounds(true);
        bomb.setVelocity(Phaser.Math.Between(-200, 200), 20);
        bomb.allowGravity = false;
    }
```

Met de functie **stars.Countactive(true)** kunnen we snel zien hoeveel sterren er nog zijn. Als ze allemaal zijn geplukt staat de teller op 0. We kunnen die sterren allemaal weer actief maken via de functie **star.enableBody(true, star.x, 0, true, true)**. Die functie voeren we uit op iedere ster in de groep.

Daarna maken we een bom via **bombs.create(x, 16, 'bomb')**. Voor de plek waar we de bom loslaten (de x positie) kijken we naar de plek van de speler. Het moet natuurlijk wel mogelijk zijn om de bom te ontwijken. Dat zou lastig zijn als hij precies boven je hoofd wordt losgelaten.

Oké, klaar. Spelen maar!

Opnieuw beginnen

Alhoewel game over nu echt wel game over is, zou het wel leuk zijn om opnieuw te kunnen starten. Voeg daarom dit stukje toe aan de functie hitBomb.

```
    var restartButton = this.add.text(150, 300,
      'Game Over! Click to   restart.', { fontSize:'32px', fill: '#f00' });
    restartButton.setInteractive();
    restartButton.on('pointerdown', () => { location.reload() } );
```

En verder?

Misschien heb je het spel gemaakt door alle code te knippen en te plakken. Dat levert snel resultaat op. Maar kijk ook eens goed naar de programmacode die je gemaakt hebt. Snap je wat elke regel doet? Experimenteer eens met wat getallen in het spel om te zien wat er gebeurt.

Misschien kan je het spel verder uitbreiden of een ander spel maken. Op https://phaser.io/learn/community-tutorials zie je wat er allemaal mogelijk is.