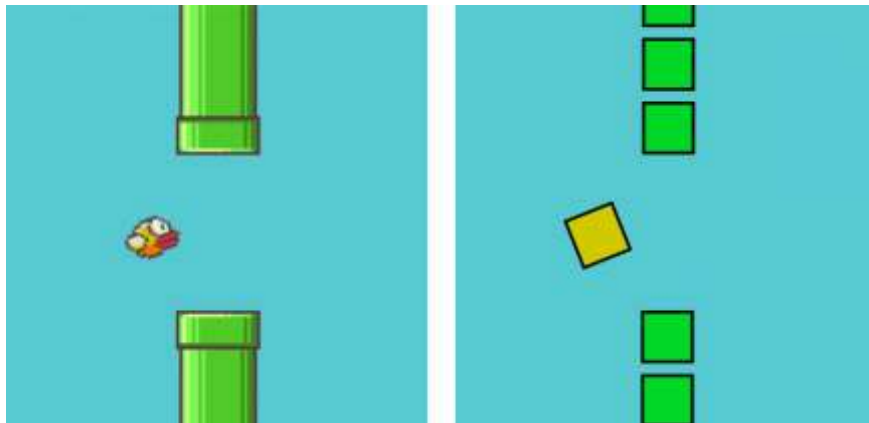


# Flappy Bird



Flappy Bird is een leuke game met eenvoudig te begrijpen mechanica. In deze opdracht leer je hoe je deze game maakt in HTML5 en JavaScript. We gaan een eenvoudige versie van Flappy Bird maken in slechts 65 lijnen Javascript met functies uit Phaser. Phaser is een gratis verzameling functies om games te maken in een webbrowser.

Als je wilt weten hoe het spel dat we gaan maken eruit ziet, klik dan [hier](#). Druk op de spatiebalk om te springen.

We gebruiken Visual Studio Code om te programmeren. Als dit programma nog niet geïnstalleerd is, kan je het hier downloaden: <https://code.visualstudio.com/>

De belangrijkste commando's in Visual Studio Code zijn:

Toets	Menu	Waarvoor
<b>CTRL+N</b>	File > New File	Maak een nieuw bestand
<b>CTRL+S</b>	File > Save	Bewaar een bestand
<b>CTRL+K CTRL+O</b>	File > Open Folder	Bekijk de inhoud van een map
<b>CTRL+K S</b>	File > Save All	Alle bestanden bewaren

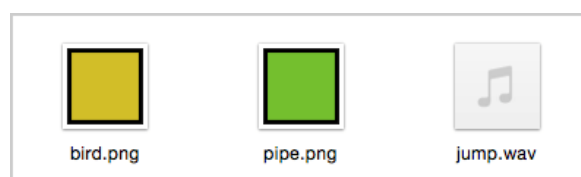
## Start

Start Visual Studio Code als je dat nog niet gedaan hebt.

Om te starten met de opdracht ga je naar de map **JavaScript/Opdrachten/FlappyBird** (via **File > Open Folder**).

Hier zie je

- phaser.min.js, de Phaser functies versie 2.4.3.
- index.html, waar het spel zal worden weergegeven.
- main.js, een bestand waarin we al onze code te schrijven.
- assets/, een map met 2 foto's en een geluidseffect.



## Een leeg project

We gaan het lege project invullen. Open het bestand index.html en voeg deze code.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title> Flappy Bird Clone </title>
    <script type="text/javascript" src="phaser.min.js"></script>
    <script type="text/javascript" src="main.js"></script>
    <script type="text/javascript" src="game.js"></script>
  </head>
  <body>
  </body>
</html>
```

Deze HTML zorgt ervoor dat de 3 Javascript bestanden worden geladen.

In het bestand game.js bevat straks de drie belangrijkste functies van een spel: preload, create en update.

Functie	Wat doet het?
preload()	Deze functie zorgt ervoor dat alle plaatjes en geluiden, als die in het spel zitten, geladen worden. Ze krijgen een plekje in het geheugen van de computer zodat deze er snel bij kan als een plaatje getoond moet worden of een geluid moet worden afgespeeld.
create()	Met deze functie worden alle onderdelen van het spel op het scherm geplaatst en wordt ingesteld dat het spel reageert op je toetsenbord of muis.
update()	Alle code in deze functie wordt 60 keer per seconde uitgevoerd. Dat noemen we ook wel de <i>framerate</i> . Er wordt 60 keer per seconde bekeken of je op een toets gedrukt hebt, en wat er dan moet gebeuren. Als de held van het spel ergens tegenaan botst, moet daar ook zo snel mogelijk iets mee gedaan worden. Wat dat is, staat in andere functies. Bijvoorbeeld 'game over' of 'beweeg naar links of rechts'.

Jij moet nu de code programmeren voor deze drie functies.

## De vogel

We gaan nu eerst ervoor zorgen dat de vogel in het spel zal springen zodra je op je spatiebalk drukt.

De code is voorzien van commentaar, dus je zou de code moeten kunnen begrijpen. Om het wat overzichtelijk te houden is zijn wat Phaser-regels en status weggehaald. Maar je moet ze in jou code wel laten staan.

Ten eerste voer je de functies preload (), create () en update () in.

```
function preload() {
  // Load the bird sprite
  game.load.image('bird', 'assets/bird.png');
}

function create () {
  // Change the background color of the game to blue
  game.stage.backgroundColor = '#71c5cf';
}
```

```
// Set the physics system
game.physics.startSystem(Phaser.Physics.ARCADE);

// Display the bird at the position x=100 and y=245
this.bird = game.add.sprite(100, 245, 'bird');

// Add physics to the bird
// Needed for: movements, gravity, collisions, etc.
game.physics.arcade.enable(this.bird);

// Add gravity to the bird to make it fall
this.bird.body.gravity.y = 1000;

// Call the 'jump' function when the spacekey is hit
var spaceKey = game.input.keyboard.addKey(
    Phaser.Keyboard.SPACEBAR);
spaceKey.onDown.add(this.jump, this);
}

function update() {
    // If the bird is out of the screen (too high or too low)
    // Call the 'restartGame' function
    if (this.bird.y < 0 || this.bird.y > 490)
        this.restartGame();
}
```

En net onder deze code toe te voegen je deze twee nieuwe functies toe.


```
// Make the bird jump
function jump() {
    // Add a vertical velocity to the bird
    this.bird.body.velocity.y = -350;
}
```


```
// Restart the game
function restartGame() {
    // Start the 'main' state, which restarts the game
    game.state.start('main');
}
```

## Testen

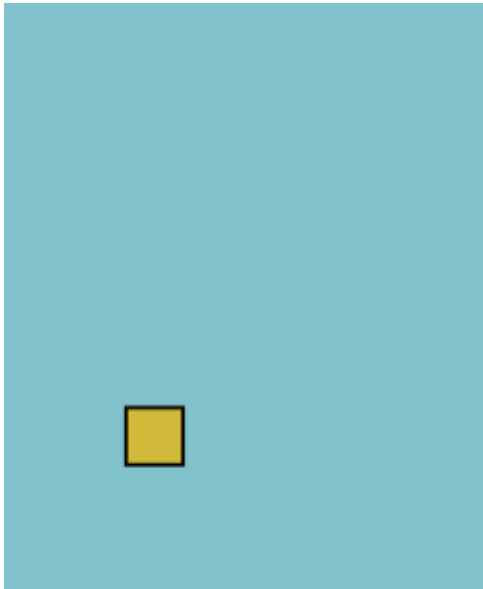
Om je werk te testen hebben we een webserver nodig. Druk op CTRL+P en type

```
ext install ritwickdey.liveserver
```

Druk op ENTER om de installatie van de webserver te starten. Als het klaar is, zie je rechtsonder in je scherm: . Klik hierop en een browser wordt gestart om je pagina te bekijken.

LET OP: Je ziet  alleen wanneer je in Visual Studio Code een folder hebt geopend. Je opent een folder met CTRL-K + CTRL-O (dus CTRL indrukken en dan op O en K drukken). Het kan ook via het menu: File > Open Folder. De folder die je kiest is de bron voor de webserver.

Als het goed is zie je dit op je scherm.



## De pijpen

Een Flappy Bird spel zonder obstakels (de groene leidingen) is niet echt interessant, dus laten we dat veranderen.

Ten eerste, laden we de pijp sprite in de functie **preload()**.

```
game.load.image('pipe', 'assets/pipe.png');
```

Aangezien we gaan een heleboel pijpen te behandelen in het spel, is het makkelijker om een Phaser functie genaamd "group" te gebruiken. De groep bevat alle leidingen die je ziet in het spel. Om de groep te maken voegen je dit toe in de functie **create()**.

```
// Create an empty group  
this.pipes = game.add.group();
```

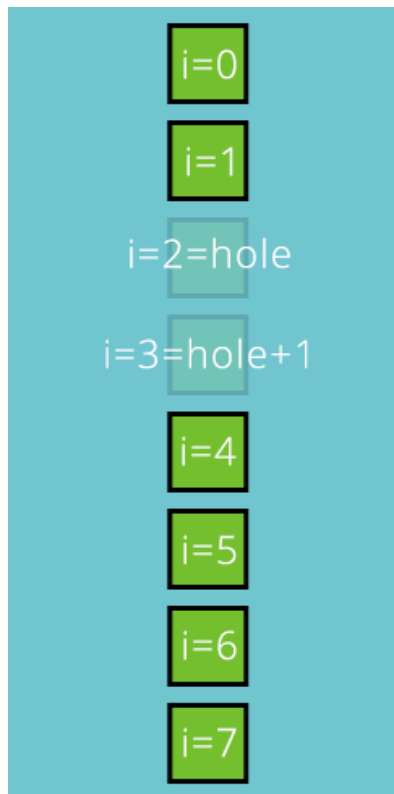
Nu moeten we een nieuwe functie om een pijp in het spel toe te voegen. We kunnen dat doen met een nieuwe functie.

```
function addOnePipe(x, y) {  
    // Create a pipe at the position x and y  
    var pipe = game.add.sprite(x, y, 'pipe');  
  
    // Add the pipe to our previously created group  
    this.pipes.add(pipe);  
  
    // Enable physics on the pipe  
    game.physics.arcade.enable(pipe);  
  
    // Add velocity to the pipe to make it move left  
    pipe.body.velocity.x = -200;  
  
    // Automatically kill the pipe when it's no longer visible  
    pipe.checkWorldBounds = true;  
    pipe.outOfBoundsKill = true;  
}
```

De vorige functie creëert een pijp, maar we moeten 6 pijpen in een rij ergens weer te geven met een gat in het midden. Dus maken we een nieuwe functie die precies dat doet.

```
function addRowOfPipes() {  
    // Randomly pick a number between 1 and 5  
    // This will be the hole position  
    var hole = Math.floor(Math.random() * 5) + 1;  
  
    // Add the 6 pipes  
    // With one big hole at position 'hole' and 'hole + 1'  
    for (var i = 0; i < 8; i++)  
        if (i !== hole && i !== hole + 1)  
            this.addOnePipe(400, i * 60 + 10);  
}
```

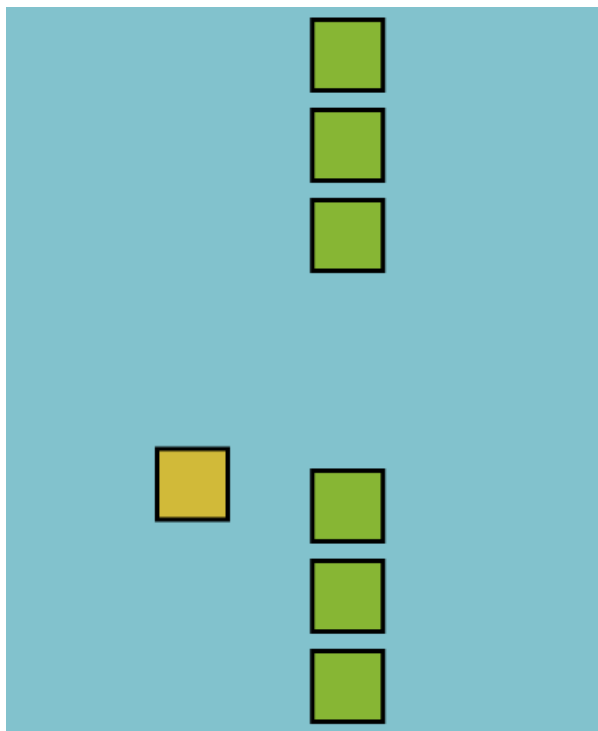
De volgende afbeelding maakt duidelijk wat er gebeurt als hole gelijk is aan 2.



Om daadwerkelijk buizen toe te voegen in ons spel moeten we de functie **addRowOfPipes()** elke 1,5 seconde aanroepen. We kunnen dit doen door het toevoegen van een timer in de functie **create()**.

```
this.timer = game.time.events.loop(1500, this.addRowOfPipes, this);
```

Nu kan je het bestand opslaan en de code testen. Het begint langzaam al op een echte game te lijken.



## Scoren en Botsingen

Het laatste wat we nodig hebben om het spel te beëindigen is het toevoegen van een score en de afhandelen van botsingen. En dit is heel gemakkelijk te doen.

Voeg het volgende stukje toe in de functie **create()** om de score in de linkerbovenhoek weergegeven.

```
this.score = 0;
this.labelScore = game.add.text(20, 20, "0",
    { font: "30px Arial", fill: "#ffffff" });
```

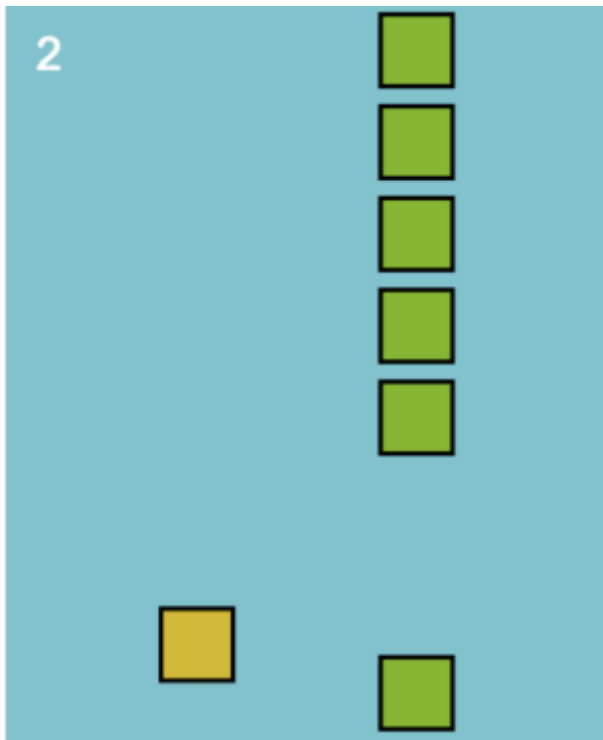
In de functie **addRowOfPipes ()**, zet je de volgende code om de score te verhogen met 1 wanneer nieuwe leidingen worden gemaakt.

```
this.score += 1;
this.labelScore.text = this.score;
```

Vervolgens voegen we deze lijn in de functie **update()** aan **restartGame()** elke keer dat de vogel in botsing komt met een pijp.

```
game.physics.arcade.overlap(
    this.bird, this.pipes, this.restartGame, null, this);
```

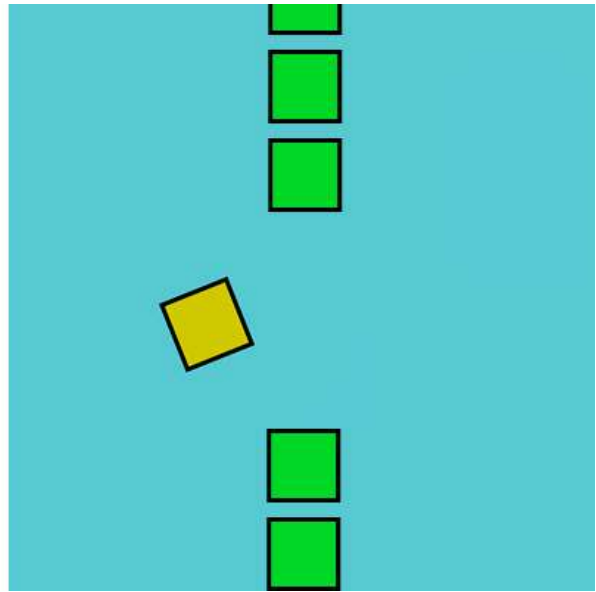
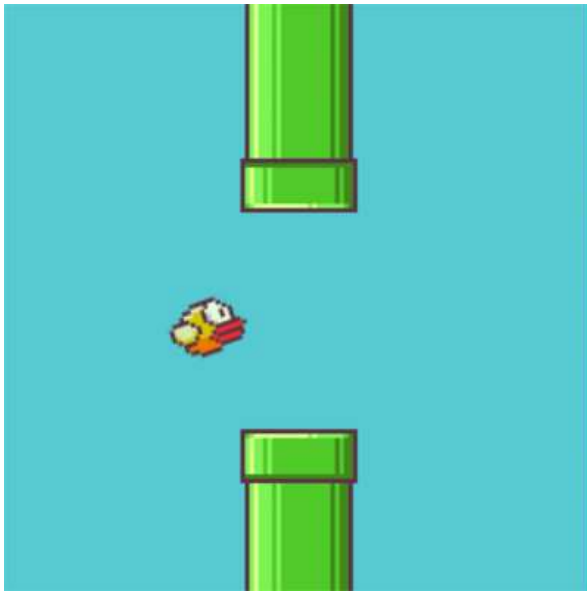
En we zijn klaar! Gefeliciteerd, je hebt nu een Flappy Bird kloon in HTML5.



## Ok, en nu?

Het spel werkt, maar het is een beetje saai. In het volgende deel van de opdracht zie je hoe we het beter kunnen maken door het toevoegen van geluiden en animaties.

## Flappy Bird in HTML5 Met Phaser - Deel 2

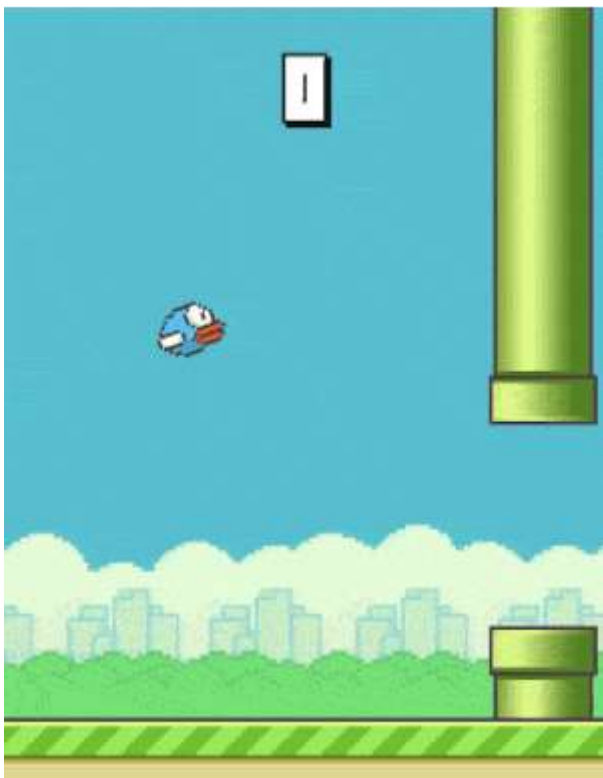


In het eerste deel van deze tutorial hebben we een eenvoudige Flappy Bird. Het was leuk, maar nogal saai om te spelen. In dit deel zullen we zien hoe om animaties en geluiden toe te voegen. We zullen niet de basis van het spel te veranderen, maar we maken het spel wel ietsje leuker.

Open het bestand `main.js` die we gemaakt hebben in het vorige deel.

### Voeg een vlieg-animatie toe

De vogel beweegt op en neer, maar wel op een nogal saaie manier. Laten we dat verbeteren door het toevoegen van een aantal animaties, zoals in het originele spel.





In het originele spel zie je dat de vogel naar beneden draait, en als de vogel springt, draait het weer een beetje omhoog.

De eerste is eenvoudig. We hoeven alleen maar om dit in de functie `update()` toe te voegen.

```
if (this.bird.angle < 20)
    this.bird.angle += 1;
```

Voor de tweede beweging, zou je gewoon `this.bird.angle = -20` in de code kunnen zetten. Maar als je de hoek zo maar verandert ziet het er gek uit. In plaats daarvan veranderen we de hoek van de vogel in een korte tijd. We kunnen dit doen door het maken van een animatie in de functie `jump()` functie.

```
// Create an animation on the bird
var animation = game.add.tween(this.bird);

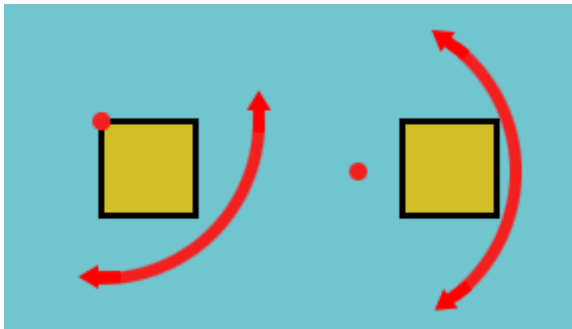
// Change the angle of the bird to -20° in 100 milliseconds
animation.to({angle: -20}, 100);

// And start the animation
animation.start();
```

Je kunt zelfs, als je wilt, deze code schrijven als een enkele regel.

```
game.add.tween(this.bird).to({angle: -20}, 100).start();
```

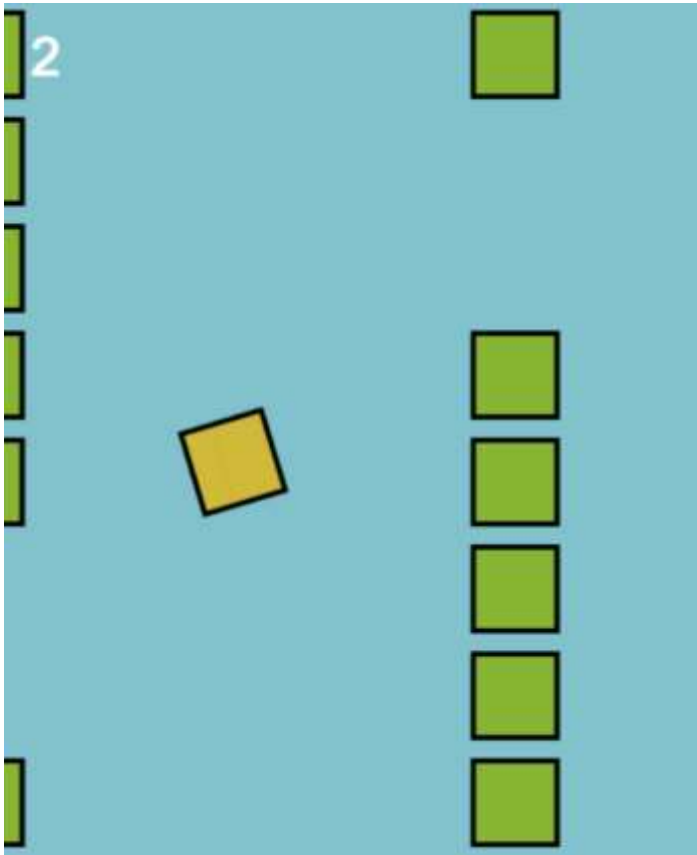
Als je het spel nu test, merk je dat de vogel niet draait als het origineel Flappy Bird. Het draait als de tekening aan de linkerkant, en we willen dat het lijkt op de beweging aan de rechterkant.



We moeten het middelpunt van de draaiing, de rode stip, veranderen. We noemen dit draaipunt ook wel “anchor” in het Engels. Dus voegen we deze lijn van code in de functie `create()`.

```
// Move the anchor to the left and downward
this.bird.anchor.setTo(-0.2, 0.5);
```

Als je het spel nu test, moet de animatie zien er een stuk beter uitzien.



### Het einde toevoegen

Tot nu toe, als Flappy Bird dood gaat, starten we het spel gelijk opnieuw. Maar in plaats daarvan gaan we nu de vogel van het scherm laten vallen.

Ten eerste, updaten we deze regel code in de functie `update()` om `hitPipe()` aan te roepen in plaats van `restartGame()` wanneer de vogel een pijp raakt.

```
game.physics.arcade.overlap(  
    this.bird, this.pipes, this.hitPipe, null, this);
```

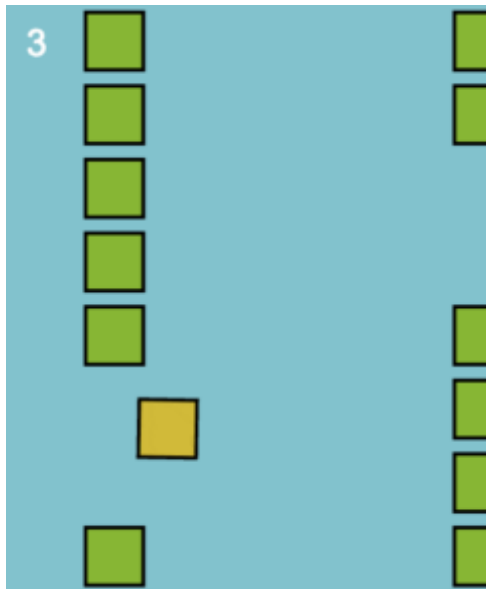
Nu maken we de nieuwe functie `hitPipe ()`.

```
function hitPipe() {  
    // If the bird has already hit a pipe, do nothing  
    // It means the bird is already falling off the screen  
    if (this.bird.alive == false)  
        return;  
  
    // Set the alive property of the bird to false  
    this.bird.alive = false;  
  
    // Prevent new pipes from appearing  
    game.time.events.remove(this.timer);  
  
    // Go through all the pipes, and stop their movement  
    this.pipes.forEach(function(p) {  
        p.body.velocity.x = 0;  
    }, this);  
}
```

Nog een laatste ding. We willen niet in staat zijn om de vogel te laten springen als het dood is. Zodat geeft we de sprong () door toevoeging van deze 2 lijnen bij het begin van de functie.

```
if (this.bird.alive == false)
    return;
```

En we zijn klaar voegen animaties.



### Geluid toevoegen

Het toevoegen van geluid is super eenvoudig met Phaser.

We beginnen met het laden van de sprong geluid in de functie preload().

```
game.load.audio('jump', 'assets/jump.wav');
```

Nu voegen we het geluid in het spel toe door de volgende regel in de functie create().

```
this.jumpSound = game.add.audio('jump');
```

Tenslotte voegen we deze lijn in de functie jump() om het geluid effect daadwerkelijk te spelen.

```
this.jumpSound.play();
```

En dat is het, we hebben nu animaties en geluiden in het spel!