Odisee
DE CO-HOGESCHOOL

Workshop Programming

with Three.js

Steven Ophalvens

29/11/2023

Odisee
DE CO-HOGESCHOOL

CONTENTS

1. Introduction
2. Project
3. Three.js – Getting started
4. Building a world
5. Adding interaction
6. More reading

Odisee
DE CO-HOGESCHOOL

1. Introduction

Who am I

- Steven Ophalvens – Web & Mobile courses

Who are you?

- IT-background?
  - Yes / No
  - What?

Odisee
DE CO-HOGESCHOOL

2. Project

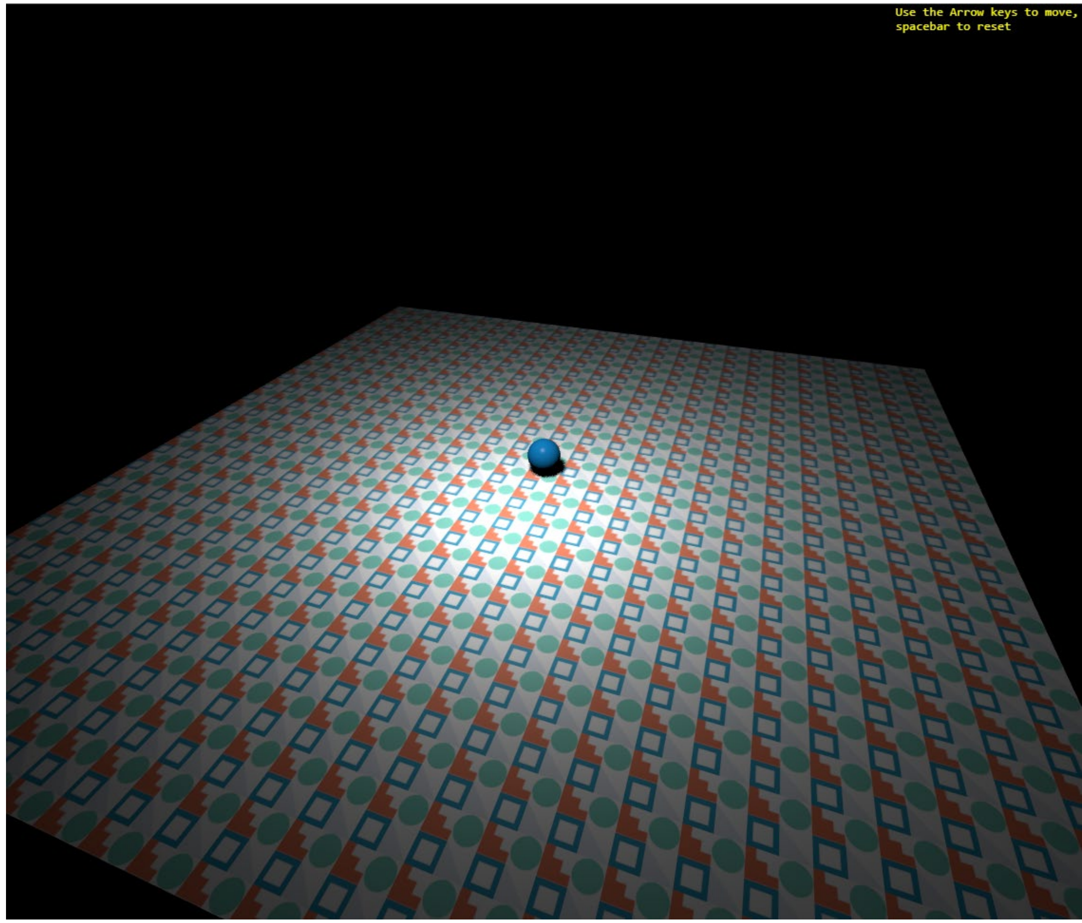
What we'll build

Where to get this presentation

Where to get the starter project

What we'll build

- A little 3D-scene with interaction (not yet a game!)
- Runs in a browser

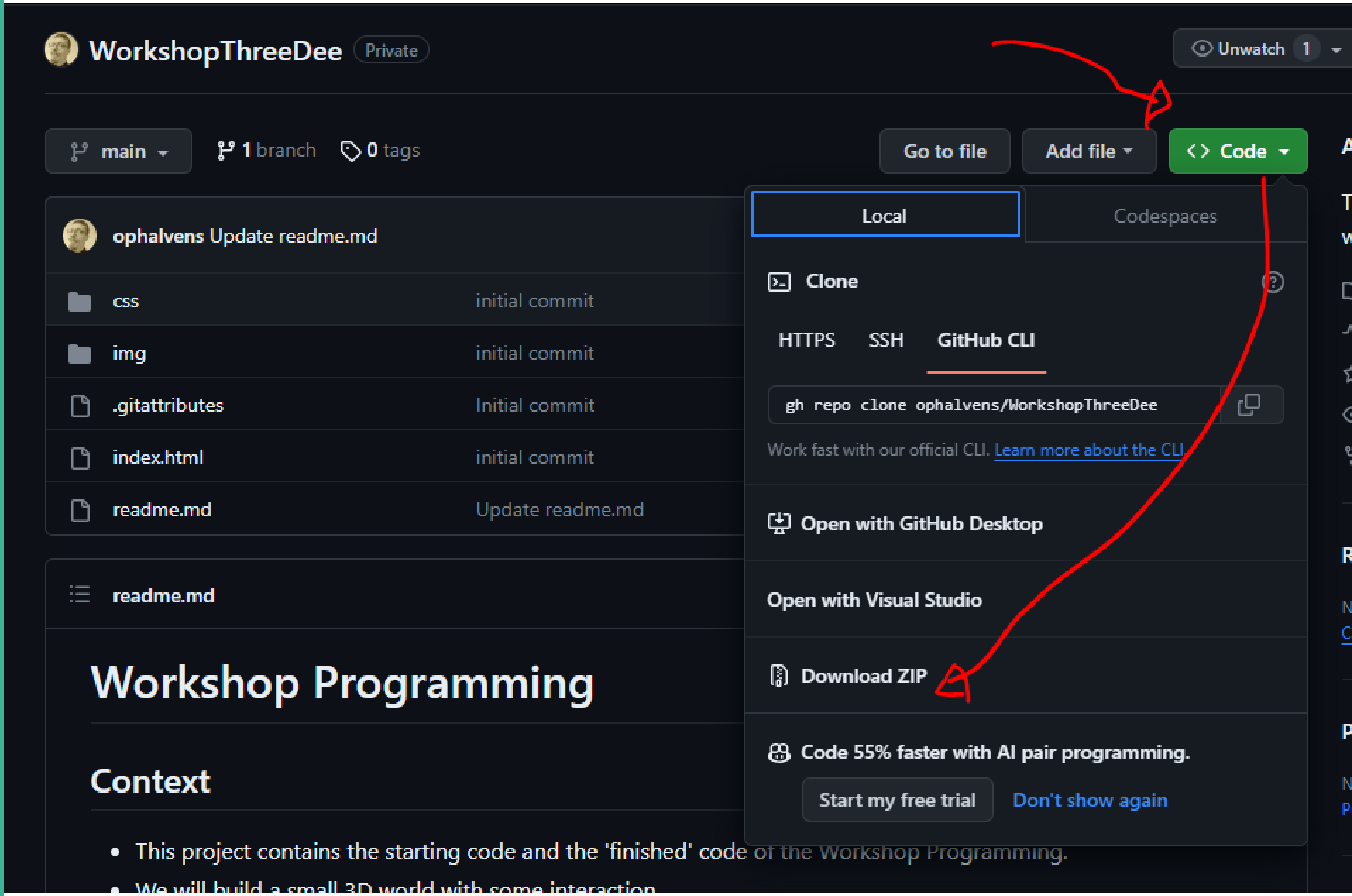


Use the Arrow keys to move,
spacebar to reset

https://github.com/ophalvens/WorkshopThreeDee

Where to get the source material

https://github.com/ophalvens/WorkshopThreeDee



```
gh repo clone ophalvens/WorkshopThreeDee
```

Work fast with our official CLI. Learn more about the CLI.

ophalvens Update readme.md

css	initial commit
img	initial commit
.gitattributes	Initial commit
index.html	initial commit
readme.md	Update readme.md

readme.md

Workshop Programming

Context

- This project contains the starting code and the 'finished' code of the Workshop Programming.
- We will build a small 3D world with some interaction

Where to get this presentation

- This presentation can be found in the source-code at presentation/Workshop_Programming_Threejs.pdf

3. Three.js

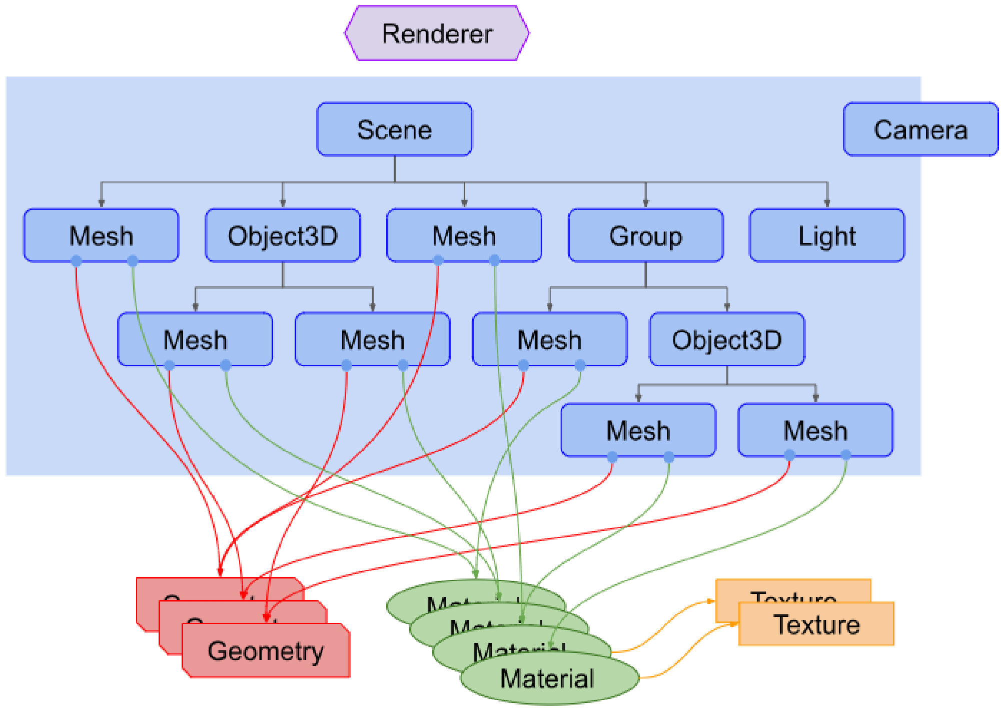
What is it?

3D in the browser

- HTML → describes a webpage
- CSS → descibes how a webpage looks (colors, size, etc)
- Javascript → scripting language
  - Three.js → Javascript library to make 3D easier (easier != easy)

Three.js

- Scene analogy to a Movie-scene:
  - Scene : where the actors and props are
  - Lights : without it, you'll only see black
  - Camera : shows the world from its viewpoint (a certain place, in a certain direction)
  - Meshes (or 'actors') : what can be seen in the scene



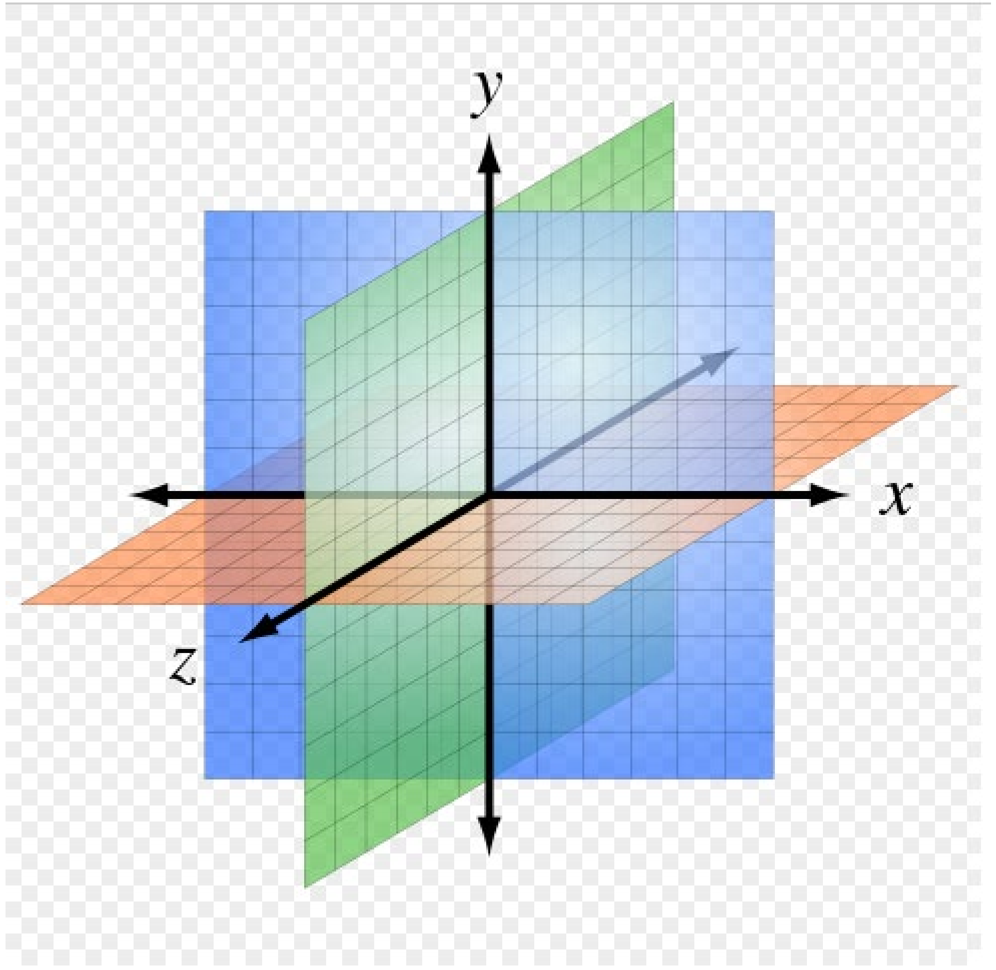
source: https://threejs.org/manual/#en/fundamentals

3D Coordinates

In this workshop these axis mean :

- x:0, y: 0, z: 0 -> origin / middle of the scene
- X-axis : horizontal axis : *left* is negative, *right* is positive
- Y-axis : *up* is positive, *down* is negative
- Z-axis : *away from the screen/camera* : negative, *towards or beyond the screen/camera* : positive

(at least until the camera itself changes position)



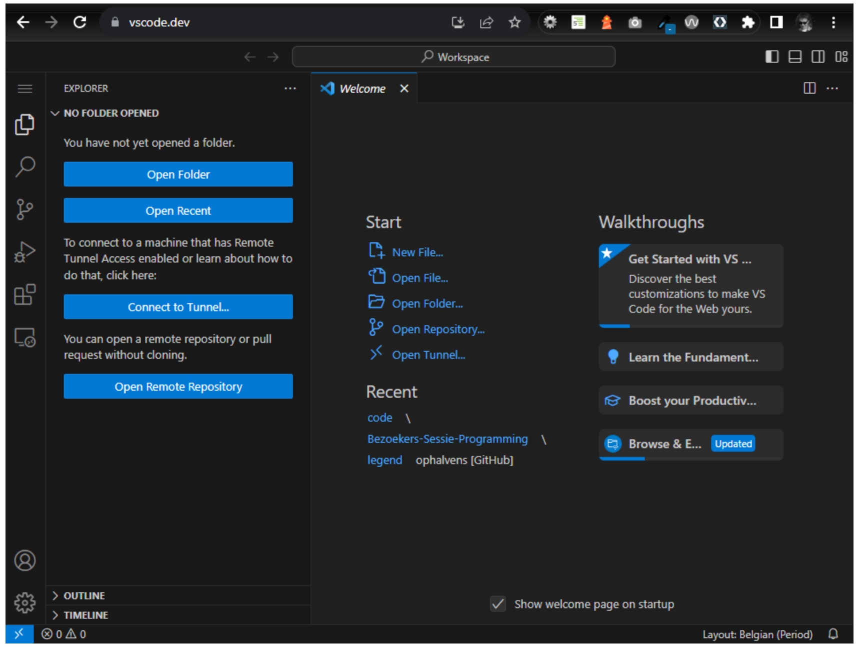
Source: https://nl.m.wikipedia.org/wiki/Bestand:3D_coordinate_system.svg

Odisee
DE CO-HOGESCHOOL

Getting started

Get the source code

- Download the starter code
- Extract the code → remember where you extract it!
- Go to https://vscode.dev



You have not yet opened a folder.

To connect to a machine that has Remote Tunnel Access enabled or learn about how to do that, click here:

You can open a remote repository or pull request without cloning.

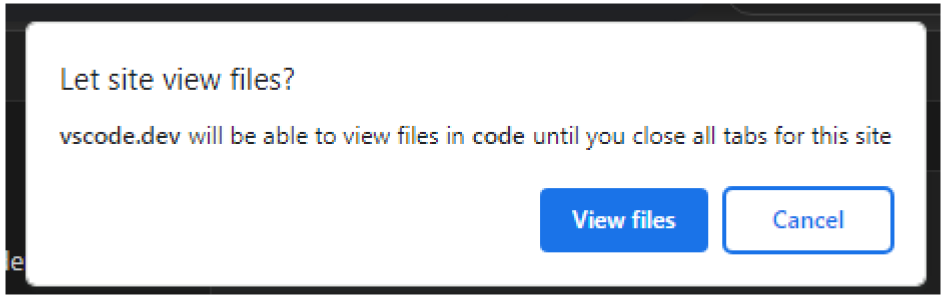
code \
Bezoekers-Sessie-Programming \
legend ophalvens [GitHub]

Discover the best customizations to make VS Code for the Web yours.

✓ Show welcome page on startup

Open the code

- Open Folder
- Remember, where did you extract the code?

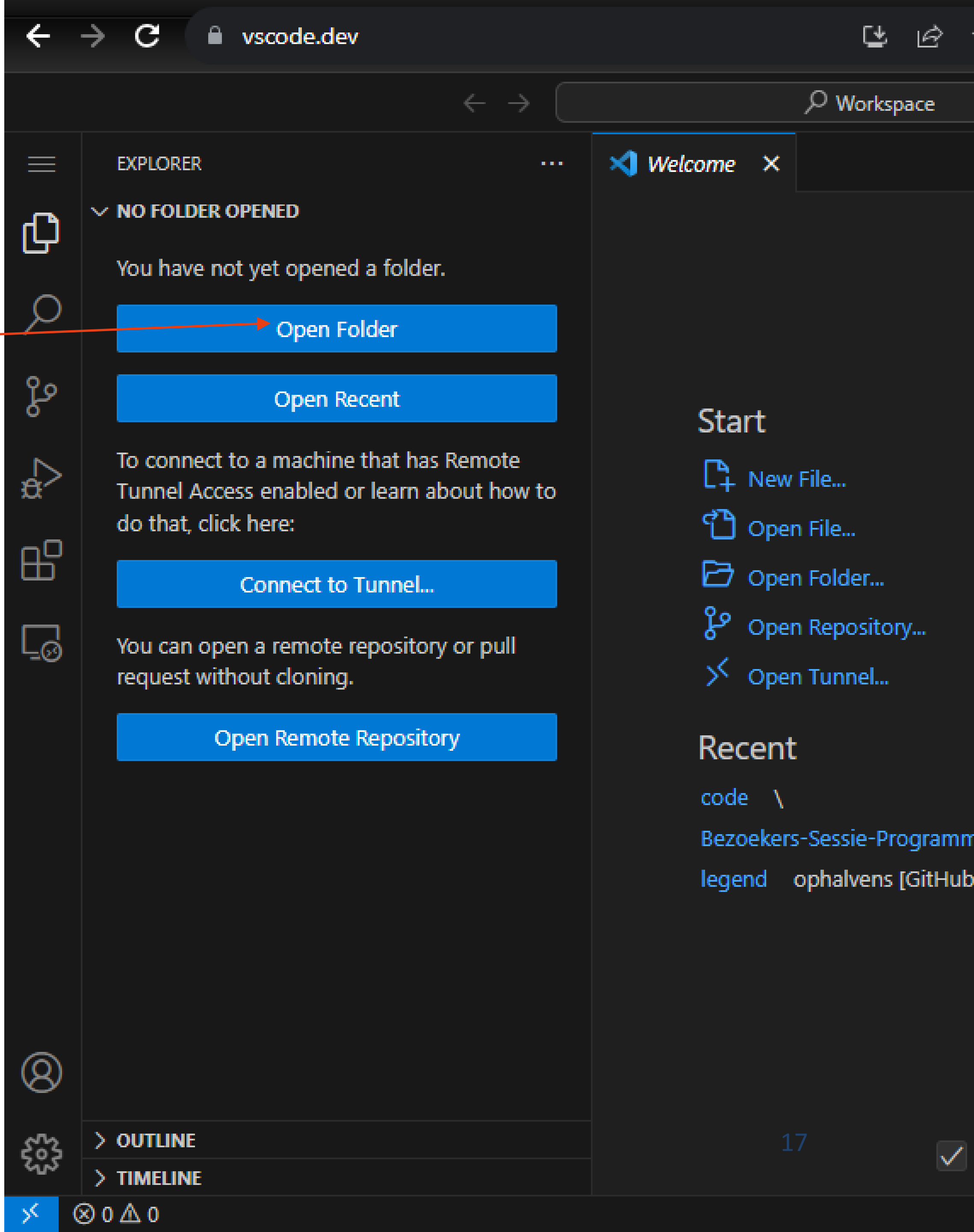


Let site view files?

vscode.dev will be able to view files in code until you close all tabs for this site

- Choose 'View files'

Odisee
DE CO-HOGESCHOOL



vscode.dev

EXPLORER

NO FOLDER OPENED

You have not yet opened a folder.

To connect to a machine that has Remote Tunnel Access enabled or learn about how to do that, click here:

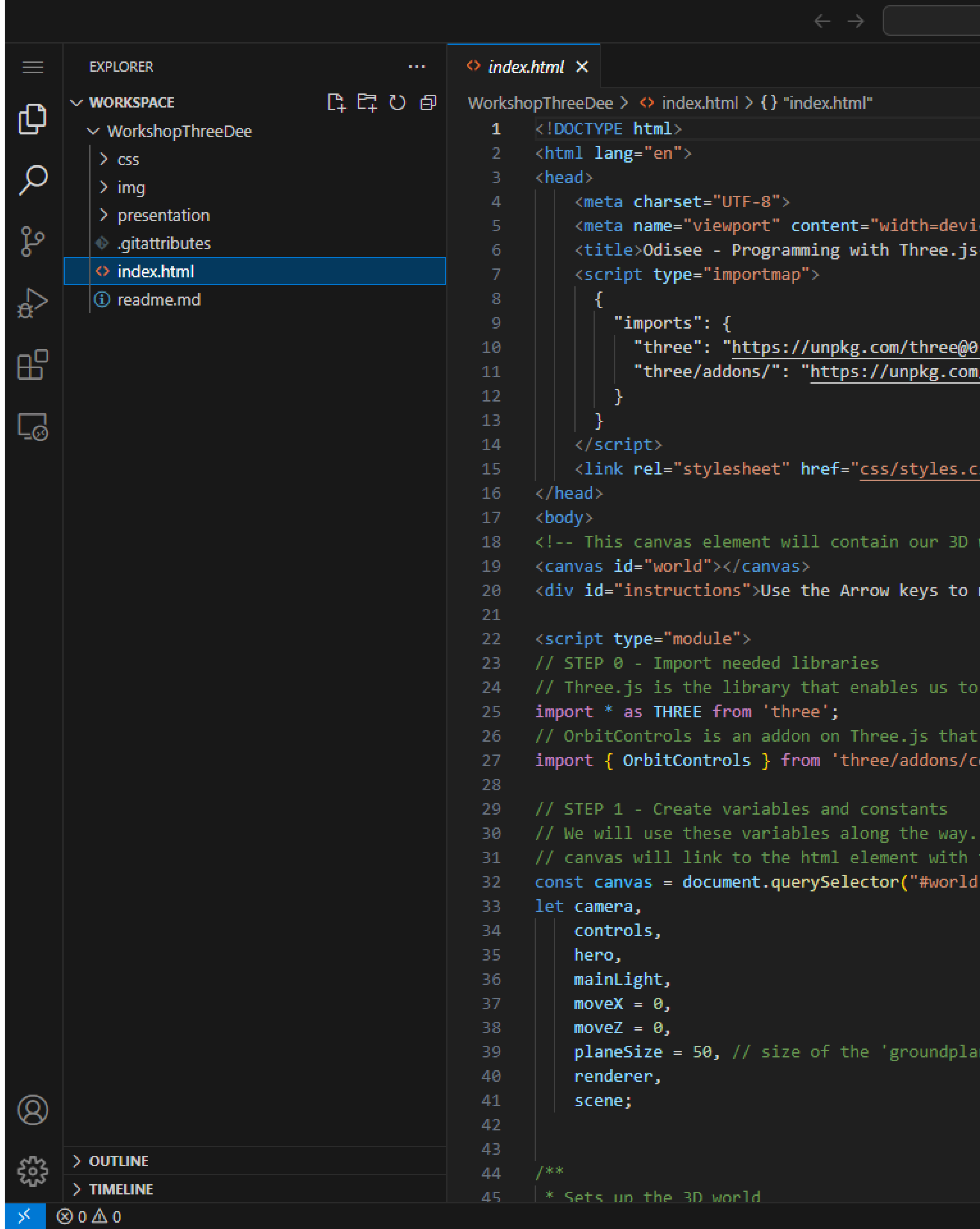
You can open a remote repository or pull request without cloning.

Project structure

- index.html
  - html elements (mainly a*canvas* and a *div*)
  - Javascript (*)
    - → this is where we will work with
- css (limited styling of the webpage)
- presentation (contains this presentation)

(*) the script is embedded because of the limitations of viewing this on a local machine with no rights

Odisee
DE CO-HOGESCHOOL



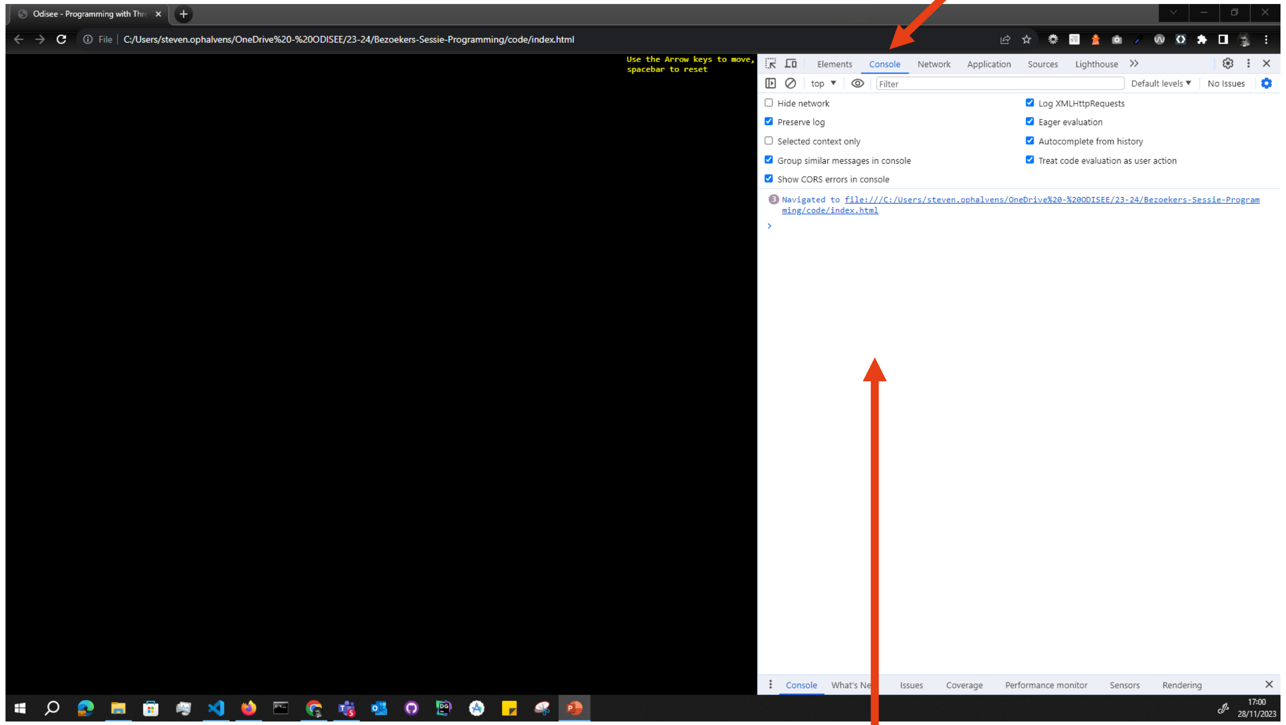
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=devi
    <title>Odisee - Programming with Three.js
    <script type="importmap">
      {
        "imports": {
          "three": "https://unpkg.com/three@0
          "three/addons/": "https://unpkg.com
        }
      }
    </script>
    <link rel="stylesheet" href="css/styles.c
</head>
<body>
<!-- This canvas element will contain our 3D
<canvas id="world"></canvas>
<div id="instructions">Use the Arrow keys to

<script type="module">
// STEP 0 - Import needed libraries
// Three.js is the library that enables us to
import * as THREE from 'three';
// OrbitControls is an addon on Three.js that
import { OrbitControls } from 'three/addons/c

// STEP 1 - Create variables and constants
// We will use these variables along the way.
// canvas will link to the html element with
const canvas = document.querySelector("#world
let camera,
    controls,
    hero,
    mainLight,
    moveX = 0,
    moveZ = 0,
    planeSize = 50, // size of the 'groundpla
    renderer,
    scene;


/**
 * Sets up the 3D world
```

How we will work

- A lot is given in the source code, but not everything
- We will further implement what is given
- Final code will be available at the end
- Run the code in the browser by opening index.html in the browser
  - Right-click on index.html > open with > Choose a browser
  - Double click on index.html

Testing the code

Choose **Console**


Odisee - Programming with Thr

File | C:/Users/steven.ophalvens/OneDrive%20-%20ODISEE/23-24/Bezoekers-Sessie-Programming/code/index.html

```
Use the Arrow keys to move,
spacebar to reset
```

- [ ] Hide network
- [x] Preserve log
- [ ] Selected context only
- [x] Group similar messages in console
- [x] Show CORS errors in console
- [x] Log XMLHttpRequests
- [x] Eager evaluation
- [x] Autocomplete from history
- [x] Treat code evaluation as user action

```
3 Navigated to file:///C:/Users/steven.ophalvens/OneDrive%20-%20ODISEE/23-24/Bezoekers-Sessie-Program
ming/code/index.html
>
```

This is the console

Open DEV-tools

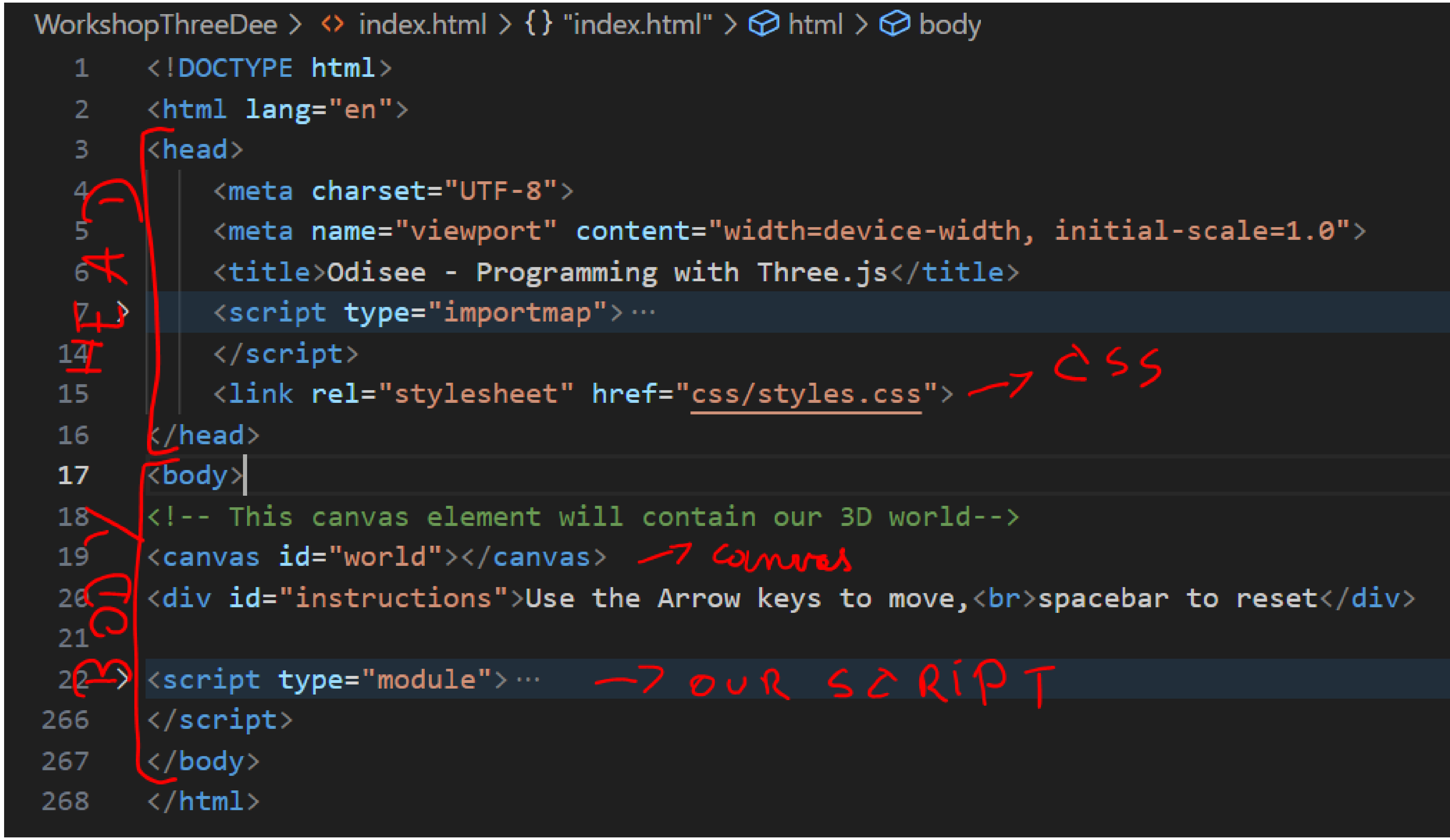
- F12 → this opens the DEV-tools
- Choose *Console*
  - *Errors & debugging*
- Update in your code? Refresh the page to see the changes (F5)

Odisee
DE CO-HOGESCHOOL

4. Building a world

HTML – some notable parts

- Head
  - Script THREE
  - Link css
- Body
  - Canvas
  - Div
  - script

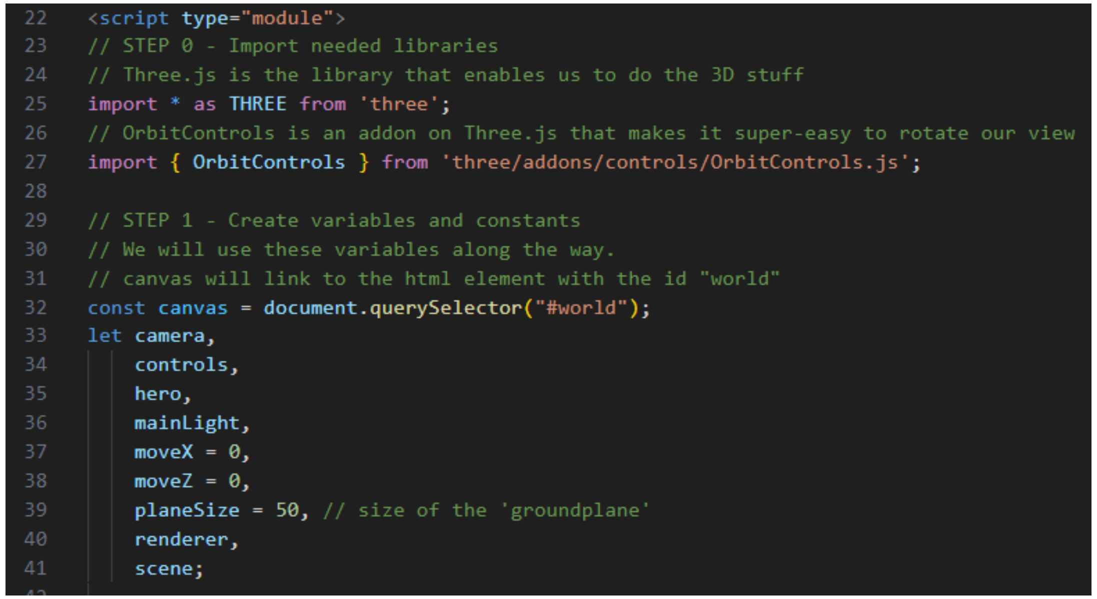

WorkshopThreeDee > index.html > {} "index.html" > html > body
HEAD
CSS
canvas
BODY
OUR SCRIPT

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Odisee - Programming with Three.js</title>
    <script type="importmap">···
    </script>
    <link rel="stylesheet" href="css/styles.css">
</head>
<body>
<!-- This canvas element will contain our 3D world-->
<canvas id="world"></canvas>
<div id="instructions">Use the Arrow keys to move,<br>spacebar to reset</div>

<script type="module">···
</script>
</body>
</html>
```

Odisee
DE CO-HOGESCHOOL

In the code (1)

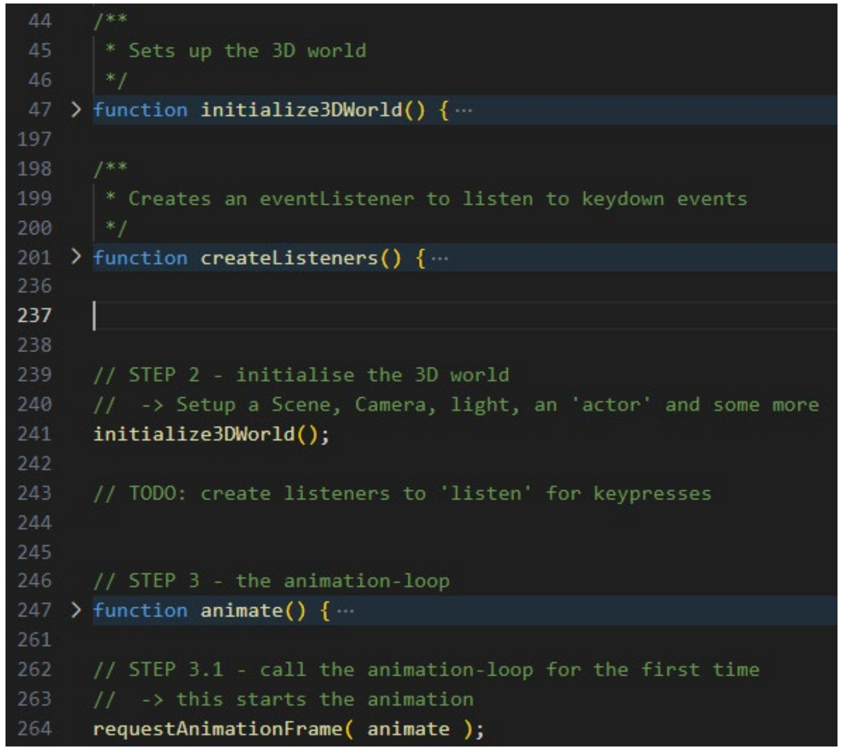
- import THREE → makes it available
- Import OrbitControls → to easily move the camera
- Several variables (let & const). Some are already used; some you will need in your logic.



```
<script type="module">
// STEP 0 - Import needed libraries
// Three.js is the library that enables us to do the 3D stuff
import * as THREE from 'three';
// OrbitControls is an addon on Three.js that makes it super-easy to rotate our view
import { OrbitControls } from 'three/addons/controls/OrbitControls.js';

// STEP 1 - Create variables and constants
// We will use these variables along the way.
// canvas will link to the html element with the id "world"
const canvas = document.querySelector("#world");
let camera,
    controls,
    hero,
    mainLight,
    moveX = 0,
    moveZ = 0,
    planeSize = 50, // size of the 'groundplane'
    renderer,
    scene;
```

In the code (2)

- initialize3DWorld()
  - Is a function
  - Is a block of code that can be called and executed
  - called on line 241 in this screenshot
- animate()
  - Is a function that is called about 60/second
  - First time called on line 264



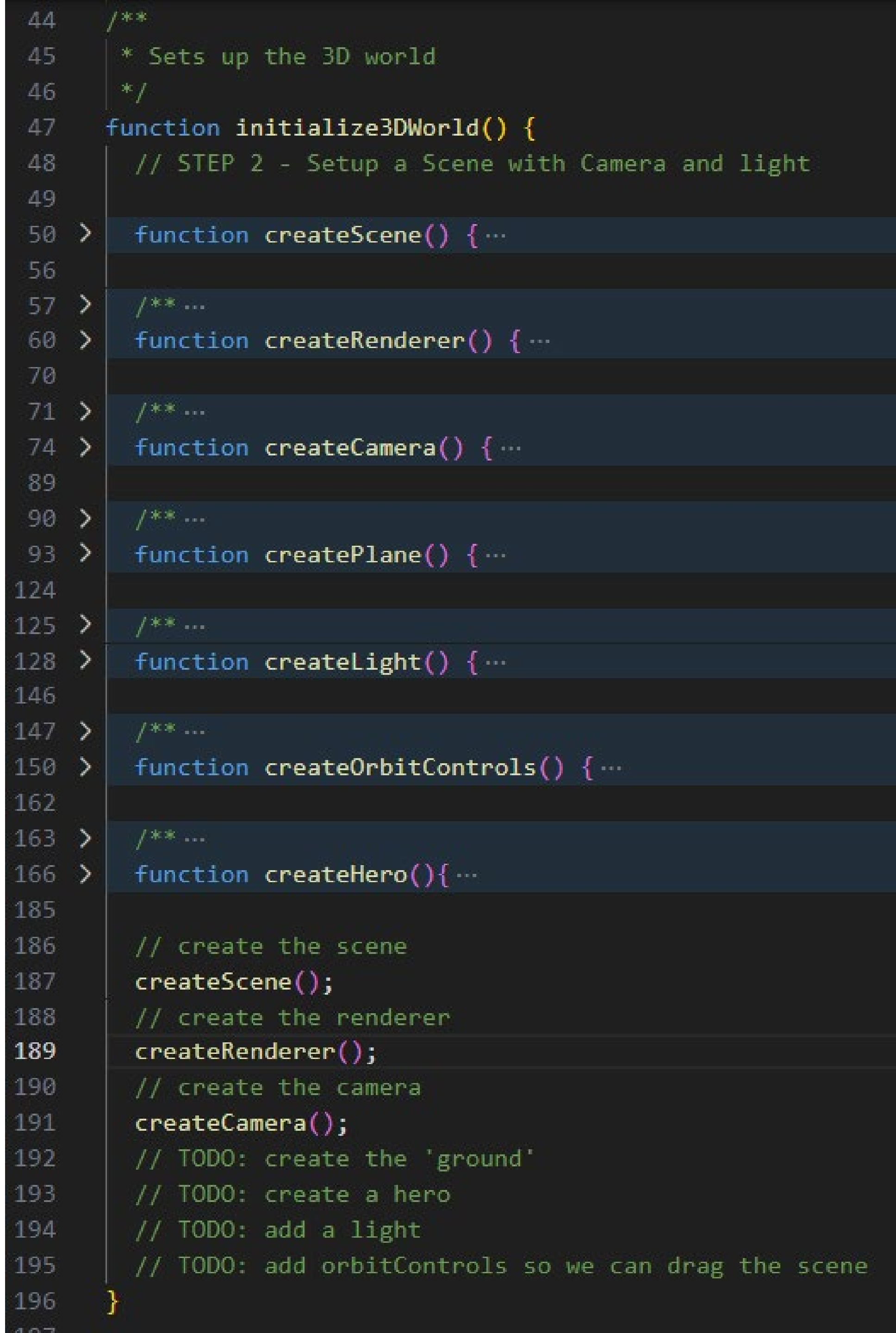
```
 44   /**
 45    * Sets up the 3D world
 46    */
 47 > function initialize3DWorld() {···
197
198   /**
199    * Creates an eventListener to listen to keydown events
200    */
201 > function createListeners() {···
236
237   |
238
239   // STEP 2 - initialise the 3D world
240   //  -> Setup a Scene, Camera, light, an 'actor' and some more
241   initialize3DWorld();
242
243   // TODO: create listeners to 'listen' for keypresses
244
245
246   // STEP 3 - the animation-loop
247 > function animate() {···
261
262   // STEP 3.1 - call the animation-loop for the first time
263   //  -> this starts the animation
264   requestAnimationFrame( animate );
```

Odisee
DE CO-HOGESCHOOL

What’s in initialize3DWorld?

- Again, some more functions
- Some of these are called:
  - createScene() → line 187
  - createRenderer() → line 189
  - createCamera() → line 191

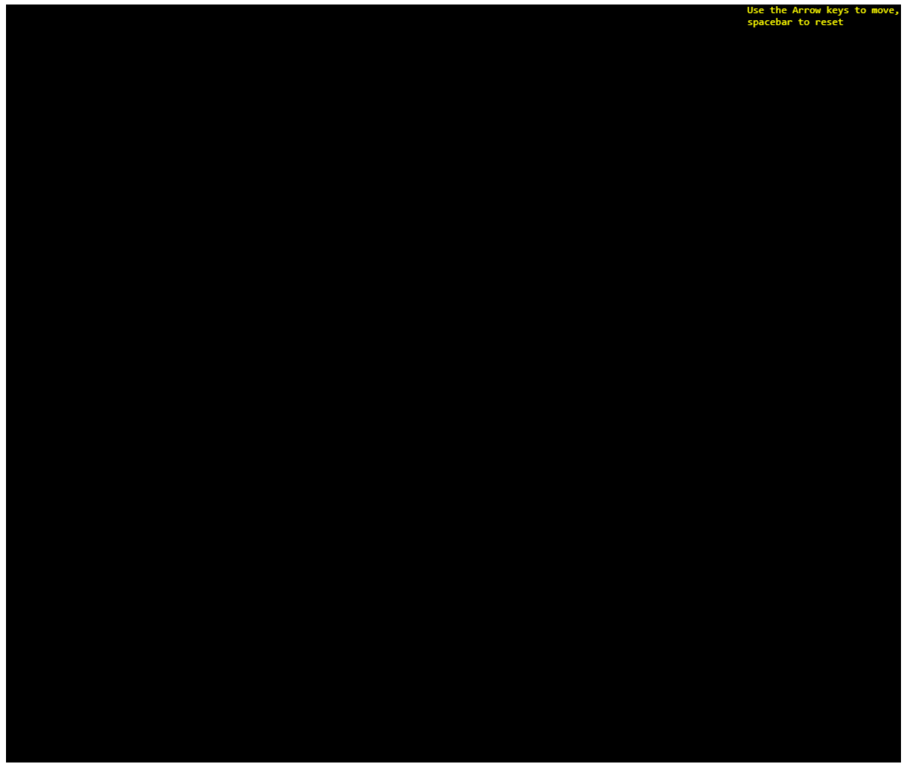
Odisee
DE CO-HOGESCHOOL



```
 44  /**
 45   * Sets up the 3D world
 46   */
 47  function initialize3DWorld() {
 48    // STEP 2 - Setup a Scene with Camera and light
 49
 50 >  function createScene() {···
 56
 57 >  /**···
 60 >  function createRenderer() {···
 70
 71 >  /**···
 74 >  function createCamera() {···
 89
 90 >  /**···
 93 >  function createPlane() {···
124
125 >  /**···
128 >  function createLight() {···
146
147 >  /**···
150 >  function createOrbitControls() {···
162
163 >  /**···
166 >  function createHero(){···
185
186    // create the scene
187    createScene();
188    // create the renderer
189    createRenderer();
190    // create the camera
191    createCamera();
192    // TODO: create the 'ground'
193    // TODO: create a hero
194    // TODO: add a light
195    // TODO: add orbitControls so we can drag the scene
196  }
```

What does it do?

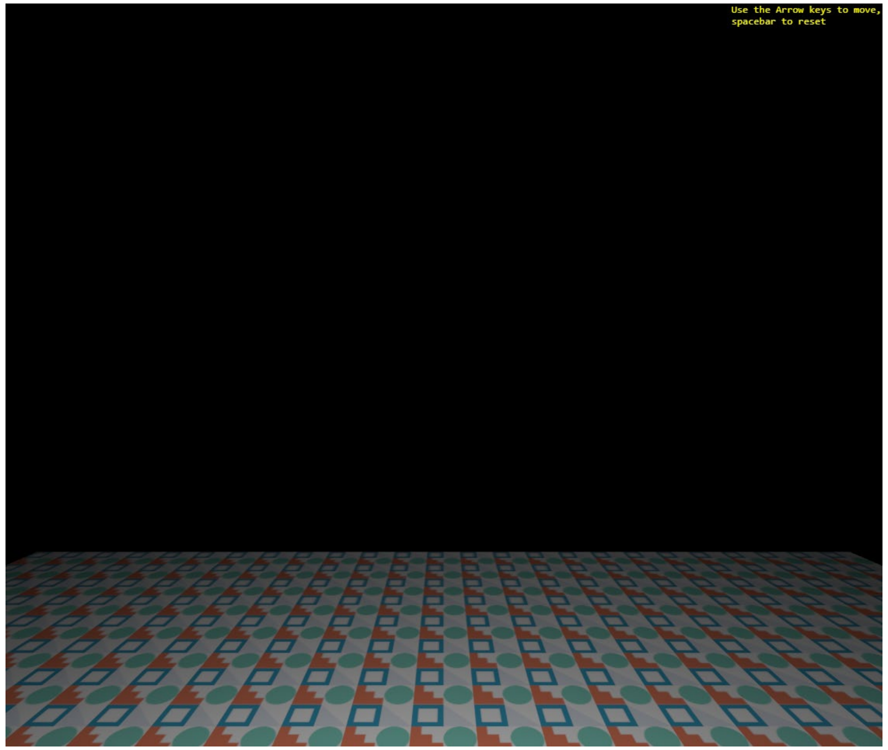
- Shows a black screen …
- No errors
- Still missing some things :
  - Add the ‘hero’, the ‘ground’ and a light



```
Use the Arrow keys to move,
spacebar to reset
```

What does it do?

- Still missing some things:
  - The hero (ball/sphere) is out of view
  - No interaction, a bit static
- Make is less static : add the orbitControls

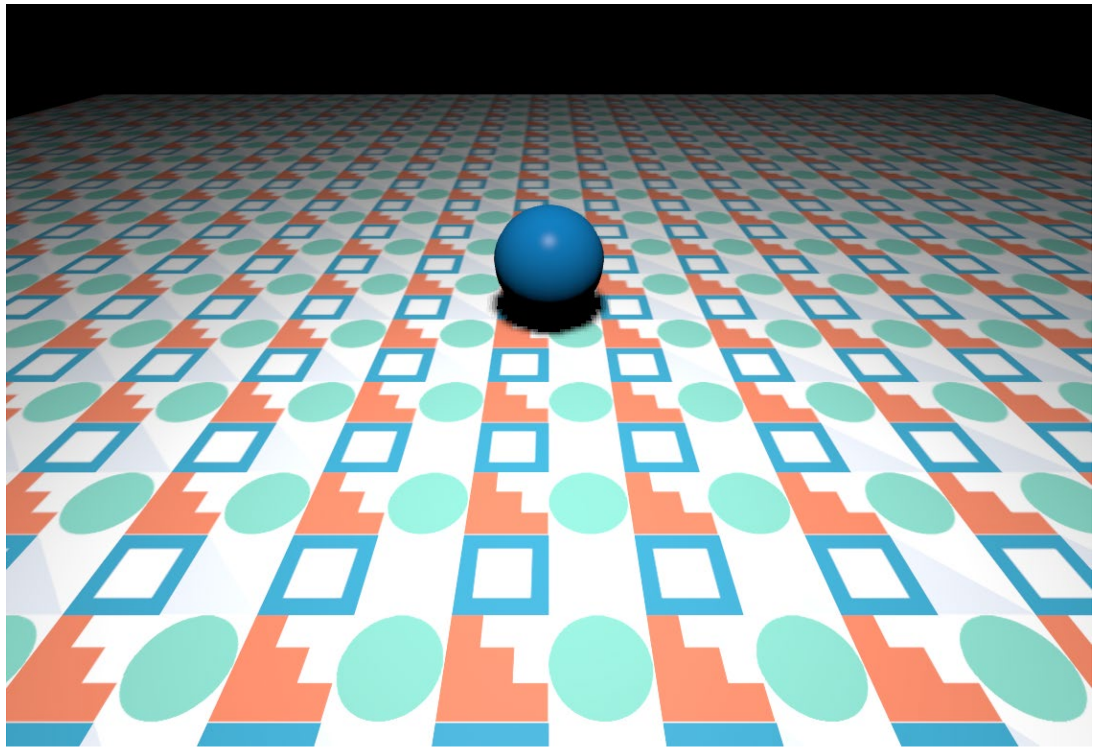


```
Use the Arrow keys to move,
spacebar to reset
```

Odisee
DE CO-HOGESCHOOL

What does it do?

- We see our ball, shadows and all!
- Try the following in the 3D view:
  - Click and drag
  - Scroll with your mouse-roller
  - Pinch-to-zoom on the trackpad



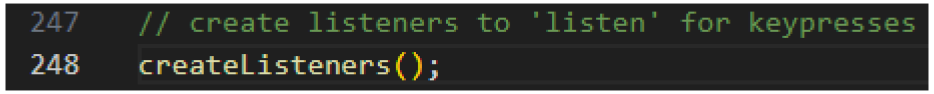
5. Adding interaction

Let’s move it!

Odisee
DE CO-HOGESCHOOL

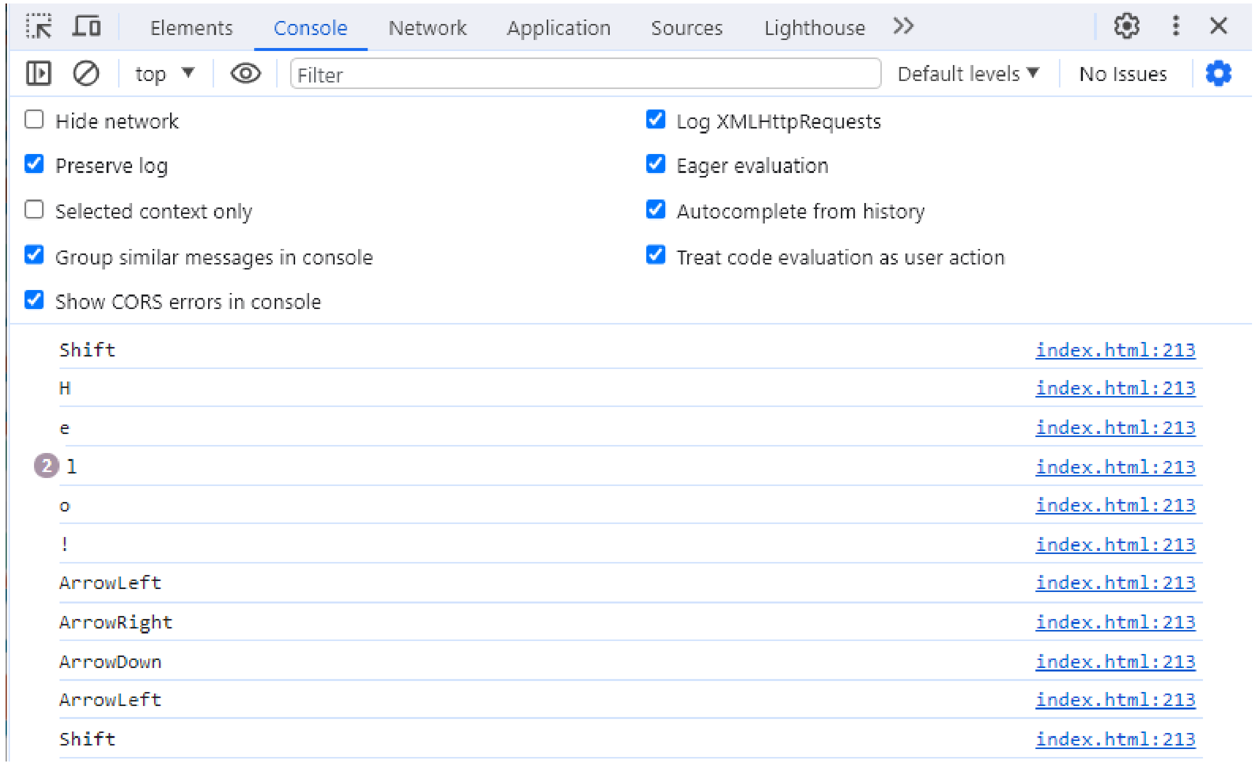
And now we listen

- Call the createListeners function



```
// create listeners to 'listen' for keypresses
createListeners();
```

- And see in the console (F12) what happens when you type something
- Your output could be something like this



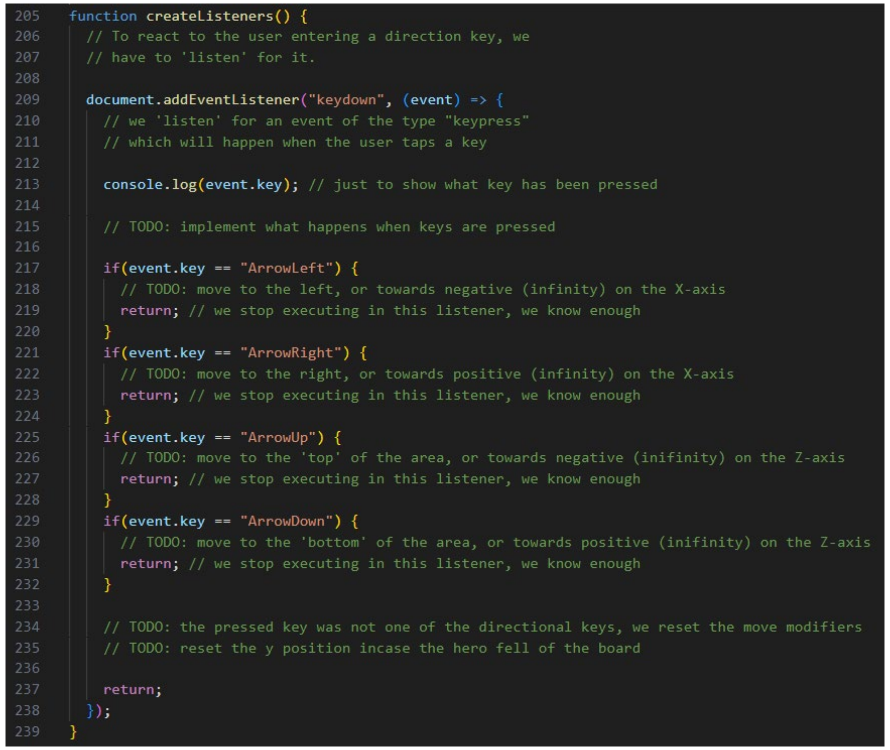
- [ ] Hide network
- [x] Preserve log
- [ ] Selected context only
- [x] Group similar messages in console
- [x] Show CORS errors in console
- [x] Log XMLHttpRequests
- [x] Eager evaluation
- [x] Autocomplete from history
- [x] Treat code evaluation as user action

```
   Shift                                    index.html:213
   H                                        index.html:213
   e                                        index.html:213
 2 l                                        index.html:213
   o                                        index.html:213
   !                                        index.html:213
   ArrowLeft                                index.html:213
   ArrowRight                               index.html:213
   ArrowDown                                index.html:213
   ArrowLeft                                index.html:213
   Shift                                    index.html:213
```

Odisee DE CO-HOGESCHOOL

What happened?

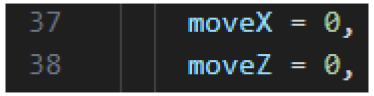
- We “listen” to the keydown event
- To see what key is entered, we log event.key (line 213)
- Remark the available stubs to react to the different arrow keys.
- IF the entered key equals the “ArrowLeft”, execute that code block



```
205 function createListeners() {
206   // To react to the user entering a direction key, we
207   // have to 'listen' for it.
208
209   document.addEventListener("keydown", (event) => {
210     // we 'listen' for an event of the type "keypress"
211     // which will happen when the user taps a key
212
213     console.log(event.key); // just to show what key has been pressed
214
215     // TODO: implement what happens when keys are pressed
216
217     if(event.key == "ArrowLeft") {
218       // TODO: move to the left, or towards negative (infinity) on the X-axis
219       return; // we stop executing in this listener, we know enough
220     }
221     if(event.key == "ArrowRight") {
222       // TODO: move to the right, or towards positive (infinity) on the X-axis
223       return; // we stop executing in this listener, we know enough
224     }
225     if(event.key == "ArrowUp") {
226       // TODO: move to the 'top' of the area, or towards negative (inifinity) on the Z-axis
227       return; // we stop executing in this listener, we know enough
228     }
229     if(event.key == "ArrowDown") {
230       // TODO: move to the 'bottom' of the area, or towards positive (inifinity) on the Z-axis
231       return; // we stop executing in this listener, we know enough
232     }
233
234     // TODO: the pressed key was not one of the directional keys, we reset the move modifiers
235     // TODO: reset the y position incase the hero fell of the board
236
237     return;
238   });
239 }
```

moveX and moveZ

- Remember the variables moveX and moveZ from before?
- We will use them to track the directions :
  - When the left arrow is pressed : moveX = -1 (move to the left)
  - When the right arrow is pressed : moveX = 1 (move to the right)
  - When the up arrow is pressed : moveZ = -1 (move away)
  - When the down arrow is pressed : moveZ = 1 (com forward)

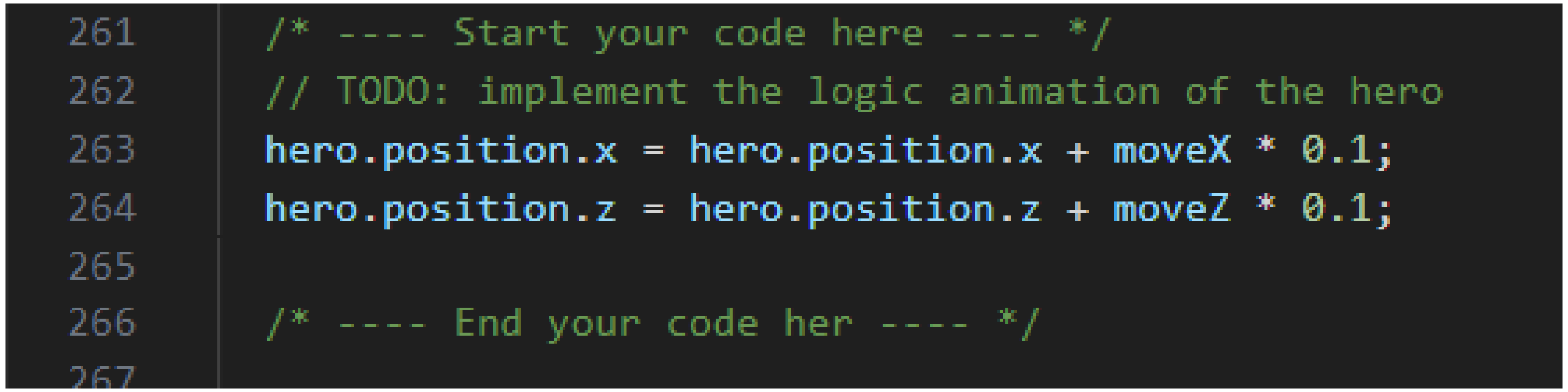


```
    moveX = 0,
    moveZ = 0,
```

Odisee
DE CO-HOGESCHOOL

moveX and moveZ

- Implement the change in position of the hero, based on moveX and moveZ
- In animate(), which is executed +/- 60 times / second

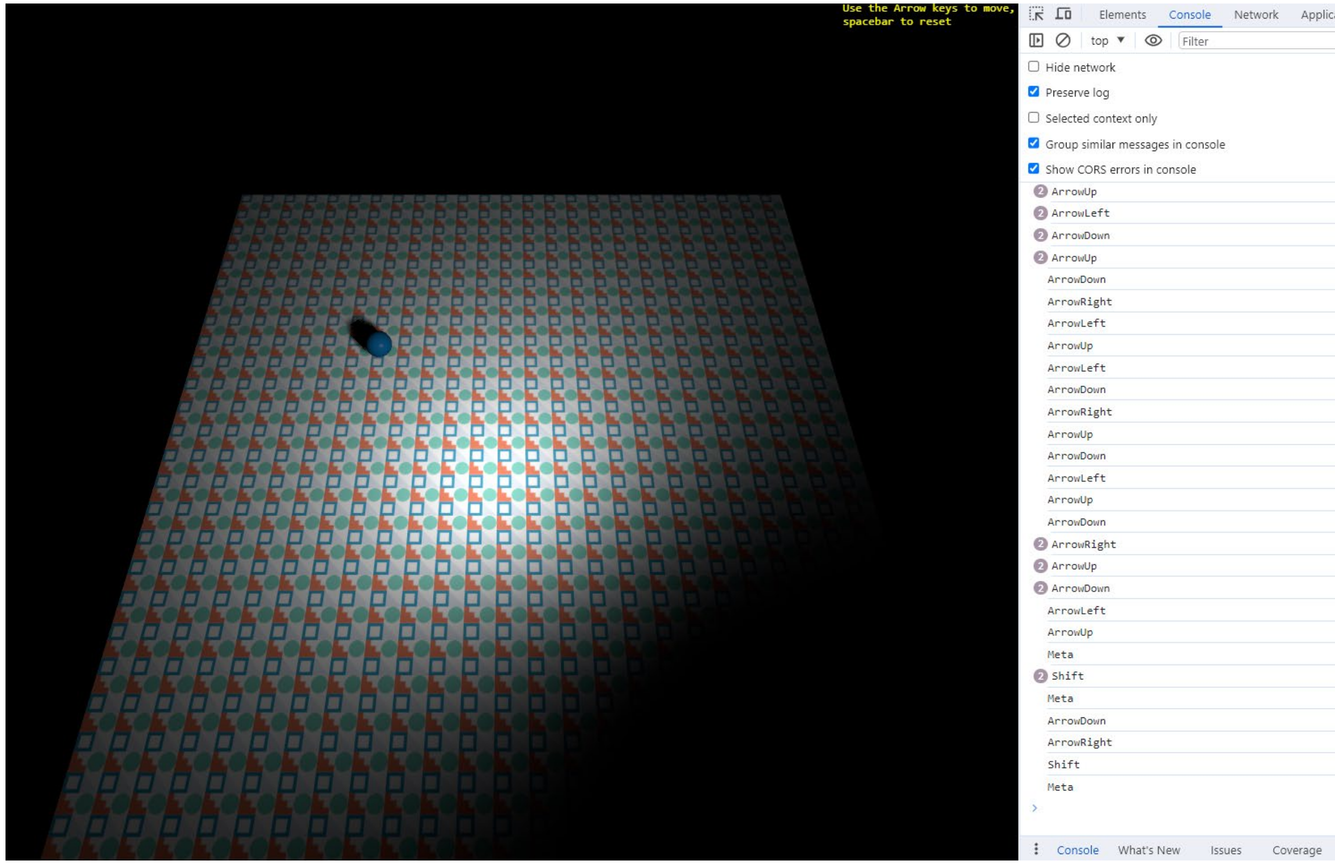


```
/* ---- Start your code here ---- */
// TODO: implement the logic animation of the hero
hero.position.x = hero.position.x + moveX * 0.1;
hero.position.z = hero.position.z + moveZ * 0.1;

/* ---- End your code her ---- */
```

Odisee
DE CO-HOGESCHOOL

Result

- This kind of works …
  but it’s not really nice :
  - No way to stop
  - No way to go in 8 directions
- Any idea how to stop the movement?



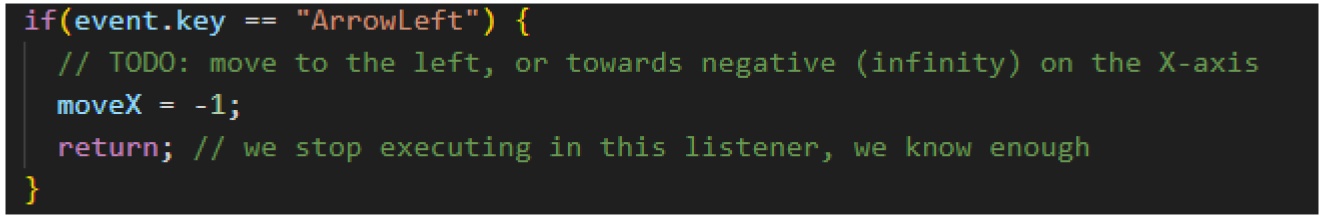
```
Use the Arrow keys to move,
spacebar to reset
```

☐ Hide network
☑ Preserve log
☐ Selected context only
☑ Group similar messages in console
☑ Show CORS errors in console

```
2 ArrowUp
2 ArrowLeft
2 ArrowDown
2 ArrowUp
  ArrowDown
  ArrowRight
  ArrowLeft
  ArrowUp
  ArrowLeft
  ArrowDown
  ArrowRight
  ArrowUp
  ArrowDown
  ArrowLeft
  ArrowUp
  ArrowDown
2 ArrowRight
2 ArrowUp
2 ArrowDown
  ArrowLeft
  ArrowUp
  Meta
2 Shift
  Meta
  ArrowDown
  ArrowRight
  Shift
  Meta
>
```

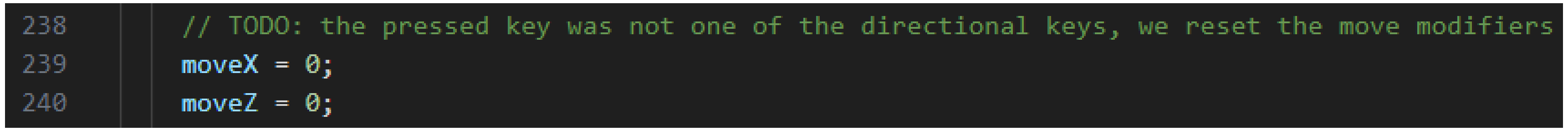
Let’s stop the action!

- If any key other than an arrow key is pressed, set moveX and moveZ to 0
- Because we return in each ‘arrow-code-block’, we leave the containing function and no more code from that function will be executed after a return statement has been executed.



```
if(event.key == "ArrowLeft") {
  // TODO: move to the left, or towards negative (infinity) on the X-axis
  moveX = -1;
  return; // we stop executing in this listener, we know enough
}
```

- Try this :



```
    // TODO: the pressed key was not one of the directional keys, we reset the move modifiers
    moveX = 0;
    moveZ = 0;
```

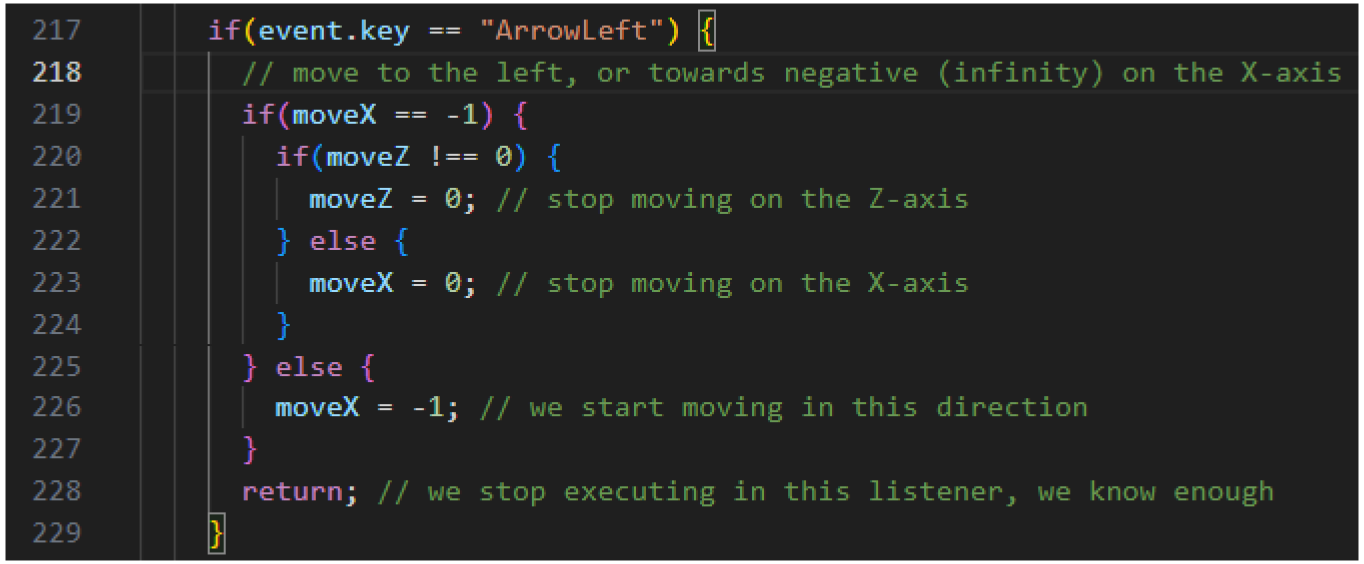
It stops!

- Ok, but how can we have the ball move in 8 directions?
- When a directional key is pressed : consider what needs to happen for both X and Z. Let’s take the LeftArrow
  - If moveX was anything but -1 → make it -1
  - If moveX was equal to -1
    - If in that case moveZ was not equal to 0 (-1 or 1) → the ball was moving on the Z axis, stop that → make moveZ equal to 0
    - If in that case moveZ was equal to 0 (not moving on the Z axis) → stop moving on the X axis (moveX was already -1 and only moving to the left) → make moveX equal to 0

Odisee
DE CO-HOGESCHOOL

And now in code …

- This works!
  But only for the left arrow key
- On a left arrow key, the ball can
  - Move left
  - Move diagonally left
  - Stop moving if it was moving left
- Can you work out the other 3 directions?
  If you missed this, you can take a look at [the code up to here]()



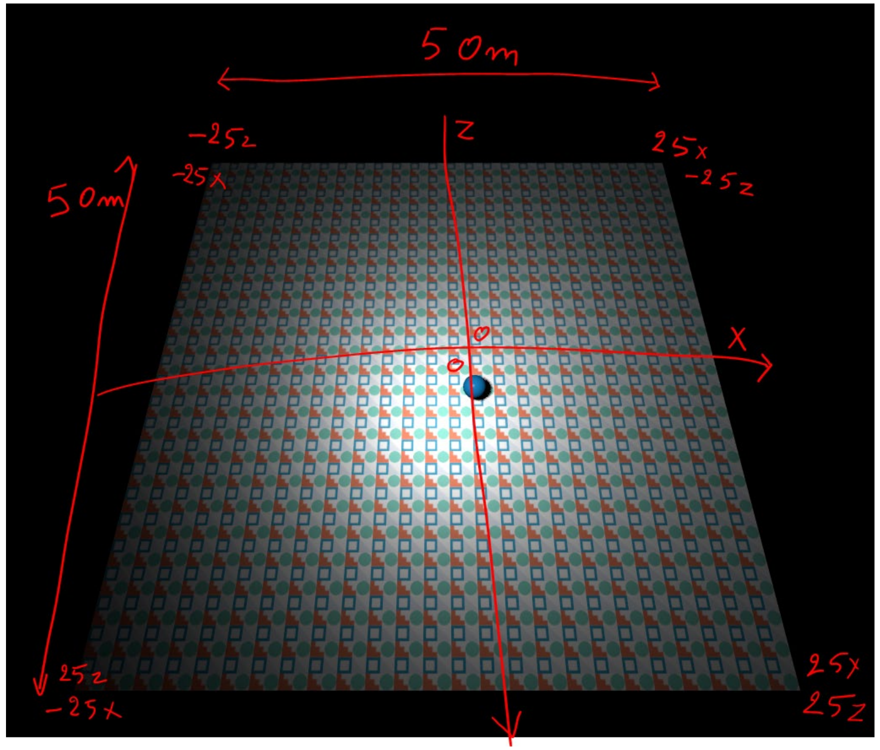
```
if(event.key == "ArrowLeft") {
  // move to the left, or towards negative (infinity) on the X-axis
  if(moveX == -1) {
    if(moveZ !== 0) {
      moveZ = 0; // stop moving on the Z-axis
    } else {
      moveX = 0; // stop moving on the X-axis
    }
  } else {
    moveX = -1; // we start moving in this direction
  }
  return; // we stop executing in this listener, we know enough
}
```

Let’s go overboard!

- What happens when the ball reaches the side of the plane?
  Nothing

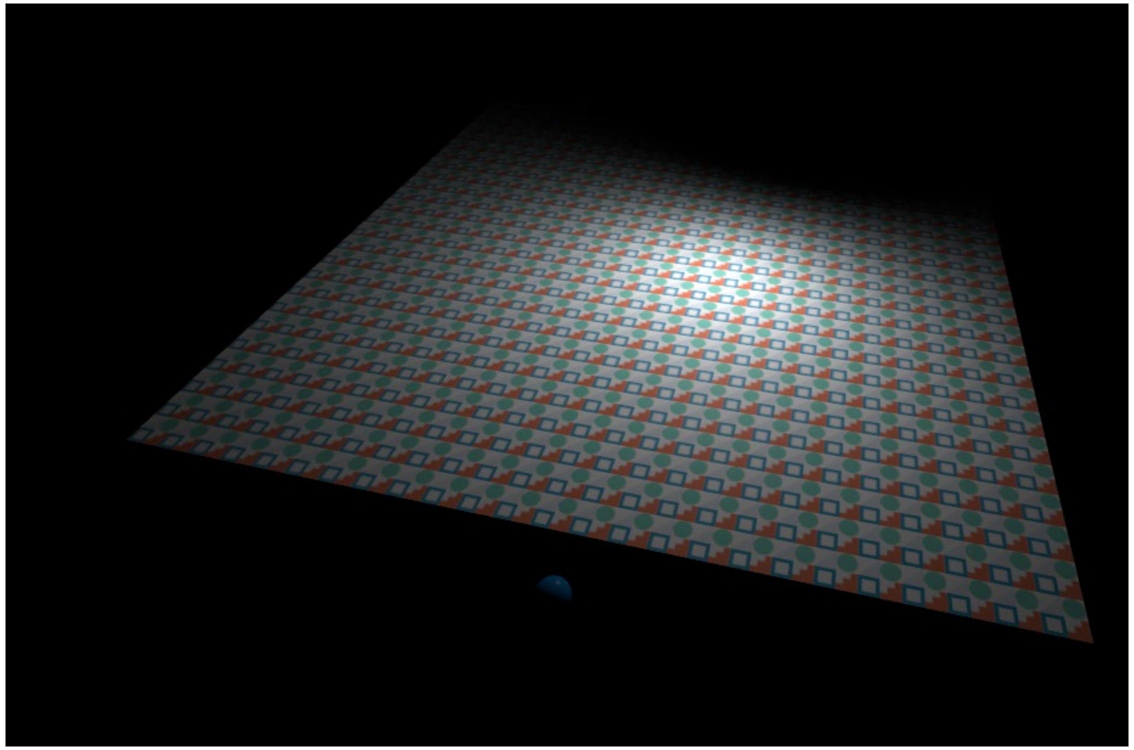
Let's go overboard!

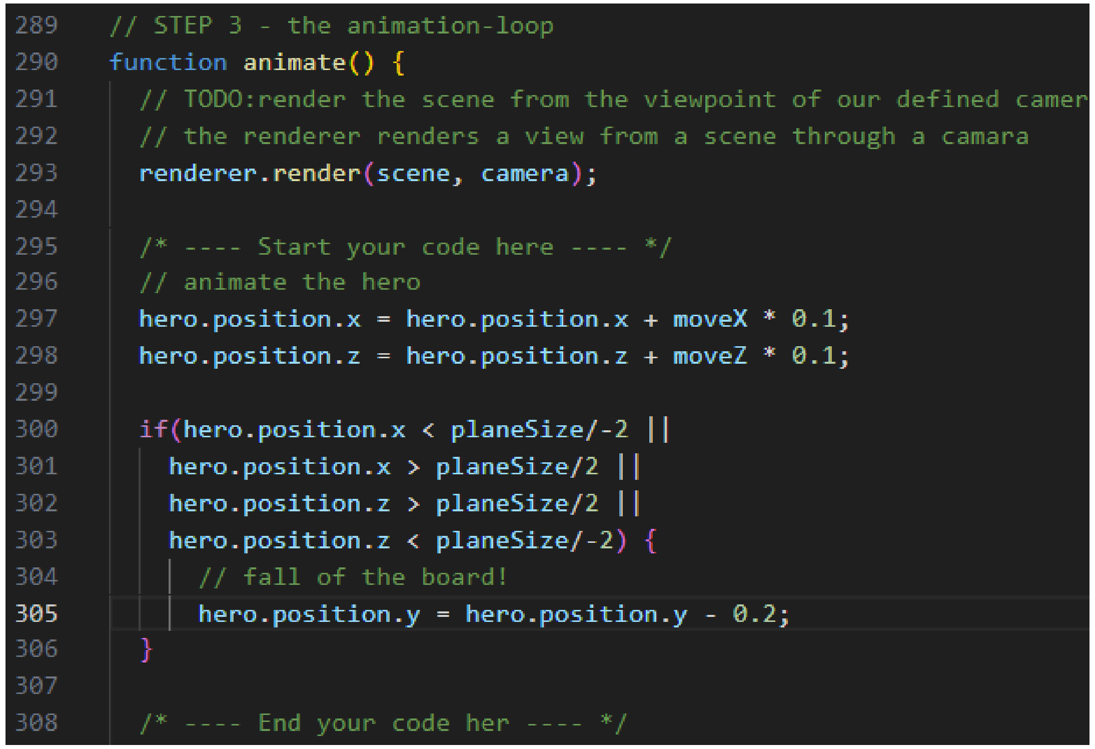
- What are the limits of the plane?
  Half the planeSize, both positive and negative on the X and Z axis!
  - Half = /2
  - Negative half = /-2
- When to test? After changing!



Odisee
DE CO-HOGESCHOOL

Let’s go overboard!

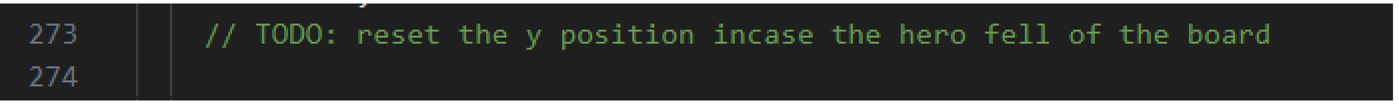




```
// STEP 3 - the animation-loop
function animate() {
  // TODO:render the scene from the viewpoint of our defined camer
  // the renderer renders a view from a scene through a camara
  renderer.render(scene, camera);

  /* ---- Start your code here ---- */
  // animate the hero
  hero.position.x = hero.position.x + moveX * 0.1;
  hero.position.z = hero.position.z + moveZ * 0.1;

  if(hero.position.x < planeSize/-2 ||
    hero.position.x > planeSize/2 ||
    hero.position.z > planeSize/2 ||
    hero.position.z < planeSize/-2) {
      // fall of the board!
      hero.position.y = hero.position.y - 0.2;
  }

  /* ---- End your code her ---- */
```

Hey, where is the hero!

- After the hero falls of the board, put him back!
- Several options, but we’ll do it on user action : after a pressed key



```
    // TODO: reset the y position incase the hero fell of the board
```

- How do we know that the hero fell off the board? Negative y position!
  Can you do it on yourself?

- When you are finished : press any key BUT an arrow key after the hero fell off the board

6. More reading

Learn more

Some interesting resources

- Three.js – the manual
- Three.js – the library

- Finished code

Final thoughts?

- Too hard / too easy ?
- Too fast / too slow ?
- Interesting / boring ?
- ???