Praktikum 0: Clear Screen

In diesem Praktikum werden wir den WebGL2-Bildschirm in Java-Script mit einer konstanten Farbe füllen. Dazu müssen Sie ständig Dokumentation bereit halten und diese auch selber lesen.

Die WebGL2 Dokumentation finden Sie unter anderem hier: https://developer.mozilla.org/en-US/docs/Web/API/WebGL2RenderingContext.

Meistens jedoch brauchen Sie die WebGL1 Dokumentation: https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/

1 Entwicklungsumgebung einrichten

a) Installieren Sie einen Browser, z.B. Firefox, Chrome oder Microsoft Edge!

b) Installieren Sie Visual Studio Code!

c) Öffnen Sie Visual Studio Code und installieren Sie dort die Extension “Live Server”! Diese Extension stellt einen Web-Server zur Verfügung.

d) Laden Sie das ZIP-Archiv `CG2Lab.zip` von Moodle auf Ihren Rechner!

e) Entpacken Sie das ZIP-Archiv in einen Ordner auf Ihrem Computer!

f) Öffnen Sie das Wurzelverzeichnis in Visual Studio Code! Sie sollten folgendes Bild erhalten:

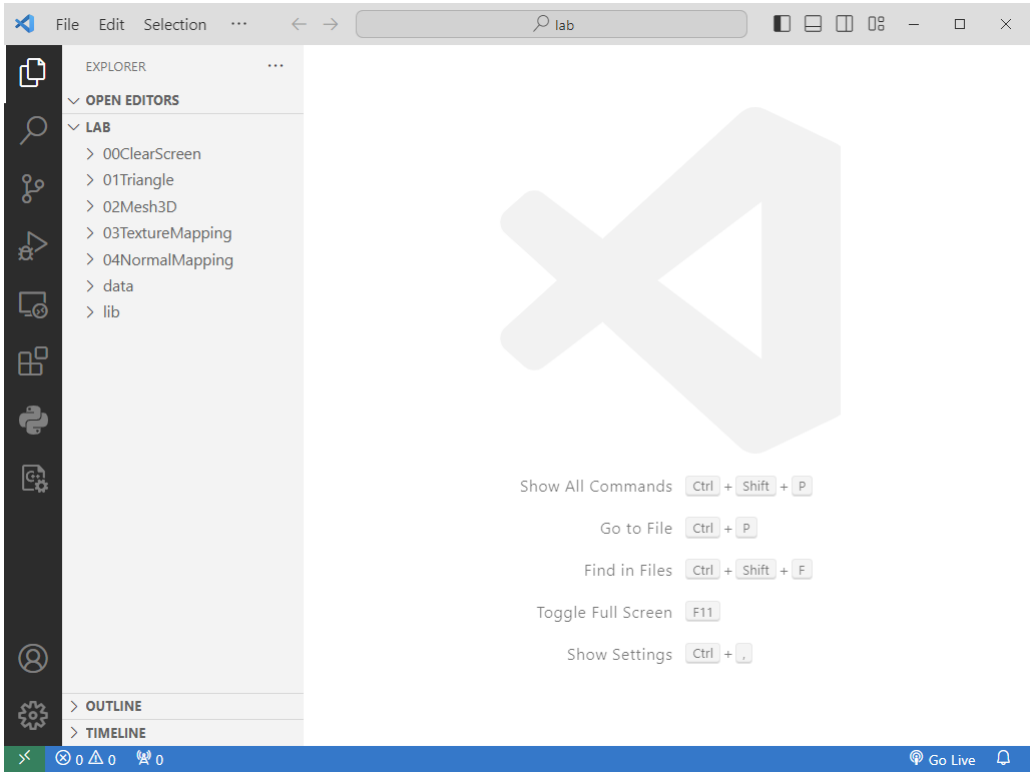

Abbildung 1: Bild vom Wurzelverzeichnis in Visual Studio Code

g) Starten Sie den Webserver “Live Server”, in dem Sie in Visual Studio Code unten rechts auf “Go Live” drücken.

Abbildung 2: Bild vom GoLive Button, den Sie in Visual Studio Code unten rechts finden sollten.

Es sollte sich nun Ihr Standard-Browser öffnen.

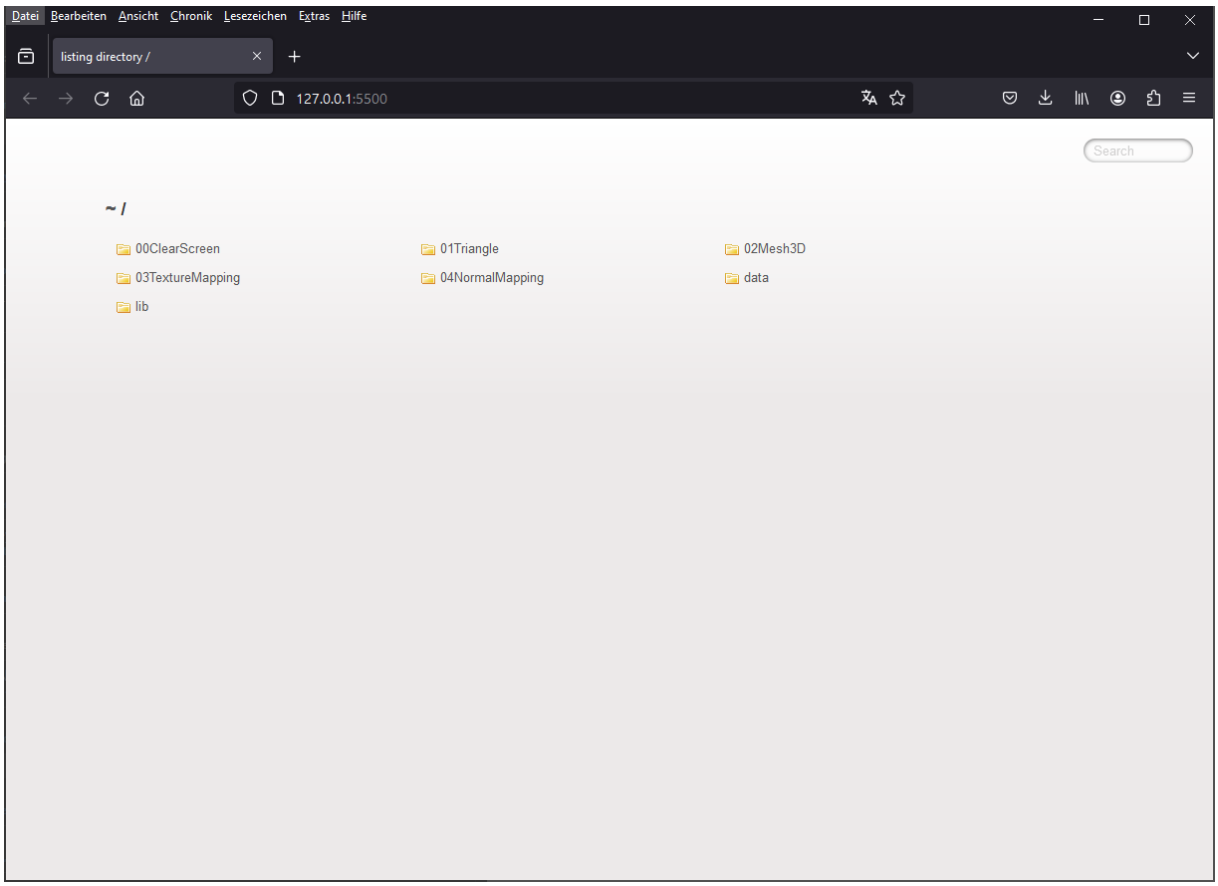

Abbildung 3: Bild vom Browser

Die Extension “Live Server” stellt auf Ihrem System einen Web-Server zur Verfügung. Dieser ist unter http://127.0.0.1:5500/ erreichbar. Der Server sieht alle Dateien des Verzeichnisses, das Sie in Visual Studio Code geöffnet haben.

h) Öffnen Sie `http://127.0.0.1:5500/Tutorials/00ClearScreen/ClearScreen.html` in Ihrem Browser! Sie sollten folgendes Bild erhalten:

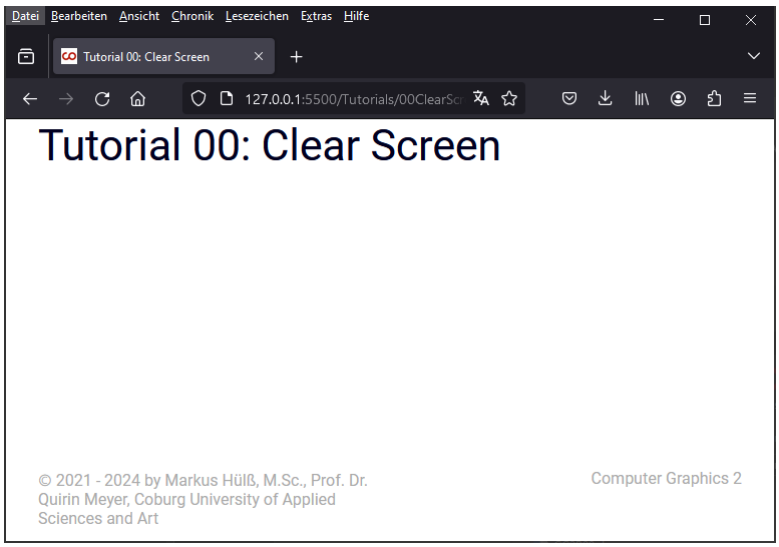

Abbildung 4: Browser nach dem Starten des LiveServers

2 Basis Struktur verstehen

a) Öffnen Sie die Datei `Tutorials/00ClearScreen/ClearScreen.html`! Dort finden Sie die HTML Seite von der aus eine Java-Script Datei geladen wird.

b) Öffnen Sie in Visual Studio Code die Datei `Tutorials/00ClearScreen/ClearScreen.js`!

```
1  //@ts-check
2  /**
3  * A class that clears the screen of a WebGL2 rendering context.
4  */
5  export class ClearScreen {
6  /**
7  * The sychronous constructor.
8  */
9  constructor()
10 {
11     let mCanvas = document.querySelector("#canvas");
12     /** @type {WebGL2RenderingContext} */
13     this.gl = mCanvas.getContext("webgl2");
14 }
15
16 /**
17 * Call this function explicitly after the synchronous constructor.
18 * After called from the outside, wait.
19 */
20 async setup()
21 {
```

```
22  }
23
24  /**
25  * Call this function to trigger rendering of the next frame.
26  * You must call it
27  * - Right after setup() has finished.
28  * - To issue the next frame, at the very bottom of your
29      draw()-method.
30  */
31  nextFrame()
32  {
33      requestAnimationFrame(this.draw.bind(this));
34  }
35
36  draw() {
37      this.gl.clearColor(1.0, 0.0, 0.0, 1.0);
38      this.gl.clear(this.gl.COLOR_BUFFER_BIT | this.gl.
            DEPTH_BUFFER_BIT);
39
40      // Issue the draw command as often as possible.
41      this.nextFrame();
42
43  }
44  }
45
46  let t = new ClearScreen();  // Create object.
47  let setupDone = t.setup();  // Let object create stuff.
48                              // asynchronously.
49  // setup returns immediately so you can do other stuff here.
50  await setupDone;            // ... but now want to make sure that
51                              // setup() is done...
52  t.nextFrame();              // ... because we want to draw the
        first
53                              // frame with nextFrame(), and that
                                   relies
54                              // on data created by the setup
                                   function.
```

c) **Fehler finden:** Verwenden Sie immer `//@ts-check` als erste Zeile in *jeder* JavaScript Datei. Es hilft Ihnen dabei Fehler bereits während des Programmierens zu finden! Fügen Sie dies ein

d) **Form:** Unser Java-Script-Programm hat immer die gleiche Form. Sie lehnt sich an die Form in Processing an. Es gibt eine `class`, die hier `ClearScreen` genannt wird. Deren Aufgabe ist es, auf einem HTML-Canvas `mCanvas` einen WebGL2 “Rendering Context” zu verwalten und dort Grafikkartenkommandos auszuführen.

e) **Member-Variablen** in Java-Script-Klassen werden mit `this.` geprefixt. Member-Variablen sind in allen Methoden der Klasse sichtbar. So speichern wir den WebGL2 Kontext in `this`

`.gl` ab. Dieser wird im `constructor` deklariert und definiert.

f) Allgemein werden Membervariablen in Methoden der Klasse definiert und deklariert. Das ist anders als in anderen Programmiersprachen wie C++ oder Java, in der Member-Variablen außerhalb von Methoden, aber innerhalb der Klasse, deklariert werden.

g) Um in Visual Studio Code mitzuteilen, dass `this.gl` ein WebGL2 Kontext ist, schreiben wir

```
/** @type {WebGL2RenderingContext} */
```

unmittelbar vor die Deklaration. So kann Visual Studio Code uns Code Vervollständigung anbieten.

h) Implementieren Sie den Konstruktor, so dass Sie einen WebGL2 Kontext also Membervariable `this.gl` erhalten!

i) Der `constructor` wird aufgerufen, sobald Sie ein Objekt der Klasse mit `new` erzeugen. Der Konstruktor kann nur synchrone Aufgaben abarbeiten. Da es aber im Web-Umfeld und auch bei und in CG2 viele asynchrone Aufgabe gibt, wie z.B. das Laden von 3D Modellen, Texture oder Shader-Dateien, und Java-Script keine asynchronen Konstruktoren unterstützt, lagern wir diese Aufgaben in die Methode `async setup()` aus. Sie als Programmierender müssen dann zuerst den Konstruktor mittels `new` aufrufen:

```
let t = new ClearScreen();
```

und anschließend explizit die asynchrone `setup`-Methode ausführen:

```
let setupDone = t.setup();  // Let object create stuff.
                            // asynchronously.
// setup returns immediately
// so you can do other stuff
// here.
await setupDone;            // ... but now want to make sure that
                            // setup() is done...
```

Da `setup` asynchron ausgeführt wird, können Sie in der Zeile darauf parallel zu den Anweisungen in der `setup` Methode, andere Anweisungen ausführen. Das ist sinnvoll, weil z.B. das Laden von Daten aus dem Internet lange dauert und Sie in der Zwischenzeit andere sinnvolle Aufgaben erledigen können. Dazu wird direkt nach dem aufruf von `setup` ein `Promise` names `setupDone` zurückgeliefert auf das sie zu einem späteren Zeitpunkt warten können.

Mit `await setupDone` können sie dann warten, dass alle Daten geladen sind.

Wenn Sie keine sinnvolle Aufgaben dazwischen ausführen wollen, können sich direkt warten:

```
1 await t.setup();
```

j) Instanziieren Sie ein Objekt der Klasse `ClearScreen`!

k) Anschließend müssen Sie das erste Frame zeichnen:

```
1 t.nextFrame()
```

Dies ruft die `draw`-Methode auf, allerdings verpackt in einem internen WebGL2-Synchronisationsmechanismus, der über `requestAnimationFrame` gesteuert wird:

```
1 requestAnimationFrame(this.draw.bind(this));
```

Dann ist da noch das `.bind(this)`, welches sicherstellt, dass der nächste Aufruf von `draw` wieder auf unserem Objekt ausgeführt wird. `requestAnimationFrame` würde sonst `draw` losgelöst vom seinem Objekt aufrufen und das gäbe dann nicht vorhersehbares Verhalten.

l) In der `draw`-Methode selber folgen dann die Aufgaben, die wir gleich bearbeiten werden. Wenn alle Zeichenkommandos eines Bildes ausgeführt sind, wollen wir gleich das nächste Bild zeichnen, um so flüssige Animationen sicherzustellen. Deshalb müssen Sie `draw` rekursiv selber aufrufen und zwar mittels `nextFrame`.

m) Implementieren Sie die `nextFrame`- Methode und rufen Sie diese an geeigneten Stelle auf!

3 Fragen

a) Wo wird das Java-Script `ClearScreen.js` geladen? Finden Sie die Zeile im HTML Code!

b) Wie starten Sie ihr WebGL Programm?

c) Was bewirkt `//@ts-check`? Wo muss es im Code stehen?

d) Wie erhalten Sie Code-Completion auf Klassen, wenn Sie sie nicht selber definiert haben?

e) Wie werden Membervariablen deklariert und definiert?

f) Was passiert im Konstruktor?

g) Warum benötigen wir eine separate `setup()`-Methode? Wie führen Sie diese aus? Was müssen Sie beachten?

h) Wie zeichnen Sie das erste Frame?

i) Wie stellen Sie als Programmierender sicher, dass eine kontinuierliche Animation abläuft?

j) Schreiben Sie Pseudo Code einer Abstrakten-Basisklasse in Java Script!

4 Bildschirm löschen

a) WebGL2 ist ein endlicher Zustandsautomat bzw. auf Englisch “Finte State-Machine”. In WebGL2 können Sie Zustände setzen. Diese bleiben dann erhalten bis Sie geändert werden. Ein Zustand ist z.B. die Hintergrundfarbe.

b) Setzen Sie in `draw` die Clear-Color auf Rot. Lesen Sie dazu die Dokumentation zu `clearColor()` in der WebGL1 Dokumentation.

Hier die Lösung:

```
1 draw() {
2     this.gl.clearColor(1.0, 0.0, 0.0, 1.0);
3     this.nextFrame();
4 }
```

Testen Sie Ihr Programm. Sie sollten folgende Ausgabe erhalten:

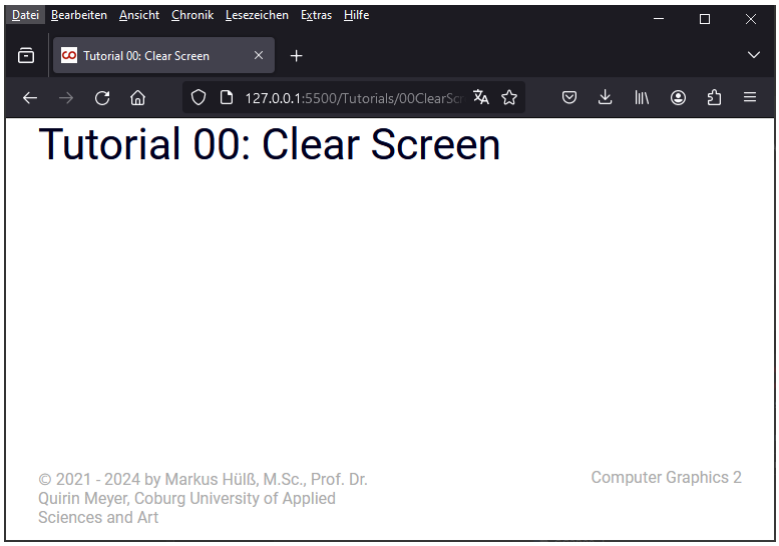

Abbildung 5: Bild nach dem Sie nur die Farbe gesetzt haben!

c) Bisher sieht man noch keinen roten Bildschirm. Sie müssen deshalb WebGL2 mitteilen, dass Sie den Hintergrundlöschen wollen! Die geschieht mit `this.gl.clear()`. Lesen Sie die Dokumentation dazu und löschen Sie den Buffer.

Hier die Lösung:

```
1 draw() {
2     this.gl.clearColor(1.0, 0.0, 0.0, 1.0);
3     this.gl.clear(this.gl.COLOR_BUFFER_BIT);
4     this.nextFrame();
```

```
5 }
```

Jetzt sollten Sie folgendes Bild erhalten:

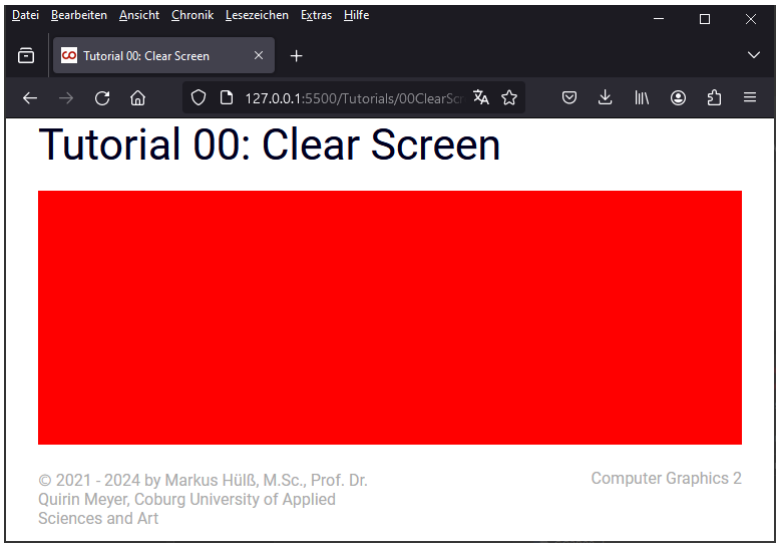

Abbildung 6: Endlich. Der Bildschirm ist rot!

d) Bauen Sie absichtlich einen Fehler ein z.B.

```
1 draw() {
2     this.gl.FEHLERclearColor(1.0, 0.0, 0.0, 1.0);
3     this.gl.clear(this.gl.COLOR_BUFFER_BIT);
4     this.nextFrame();
5 }
```

und schauen was im Browser passiert.

e) Jetzt drücken Sie F12 im Browser und Sie kommen in den Entwicklermodus. Das Entwickler-Werkzeuge-Fenster öffnet und sieht so aus:

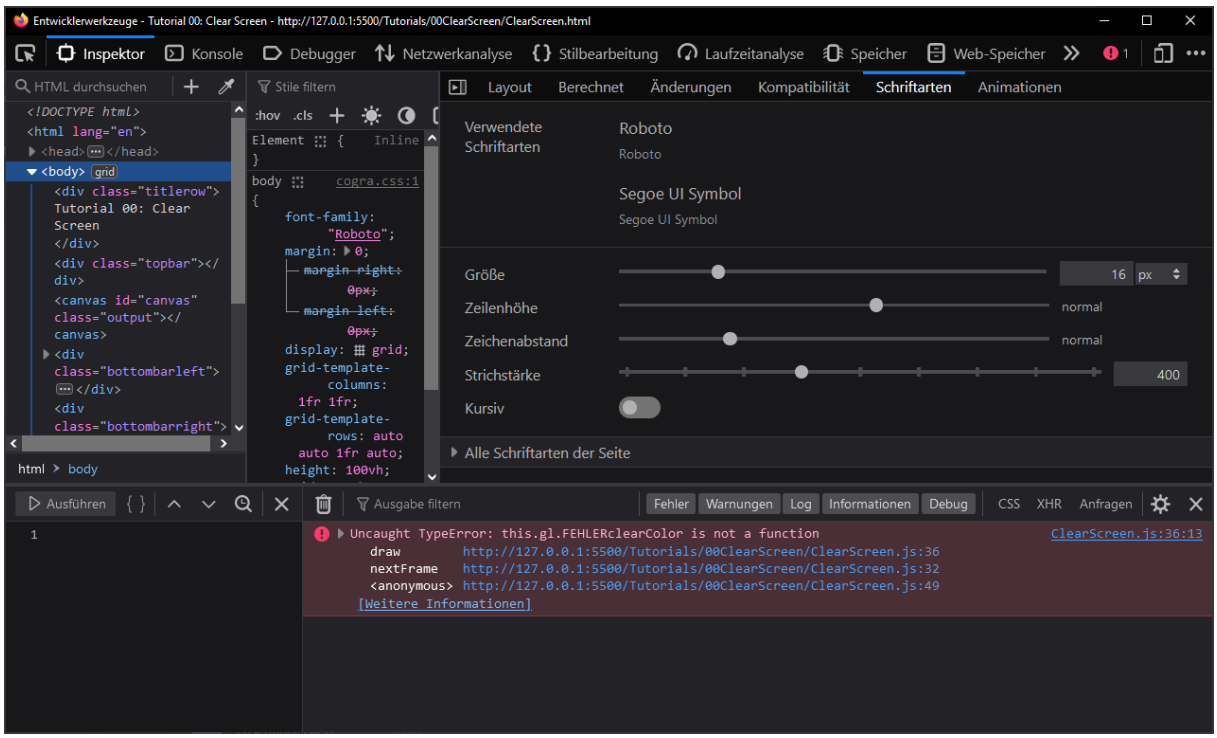

Abbildung 7: Das Fenster, das mit dem Entwicklermodus des Browsers geöffnet wird.

Wechseln Sie in den Debugger-Tab und Sie sollten folgendes Bild sehen:

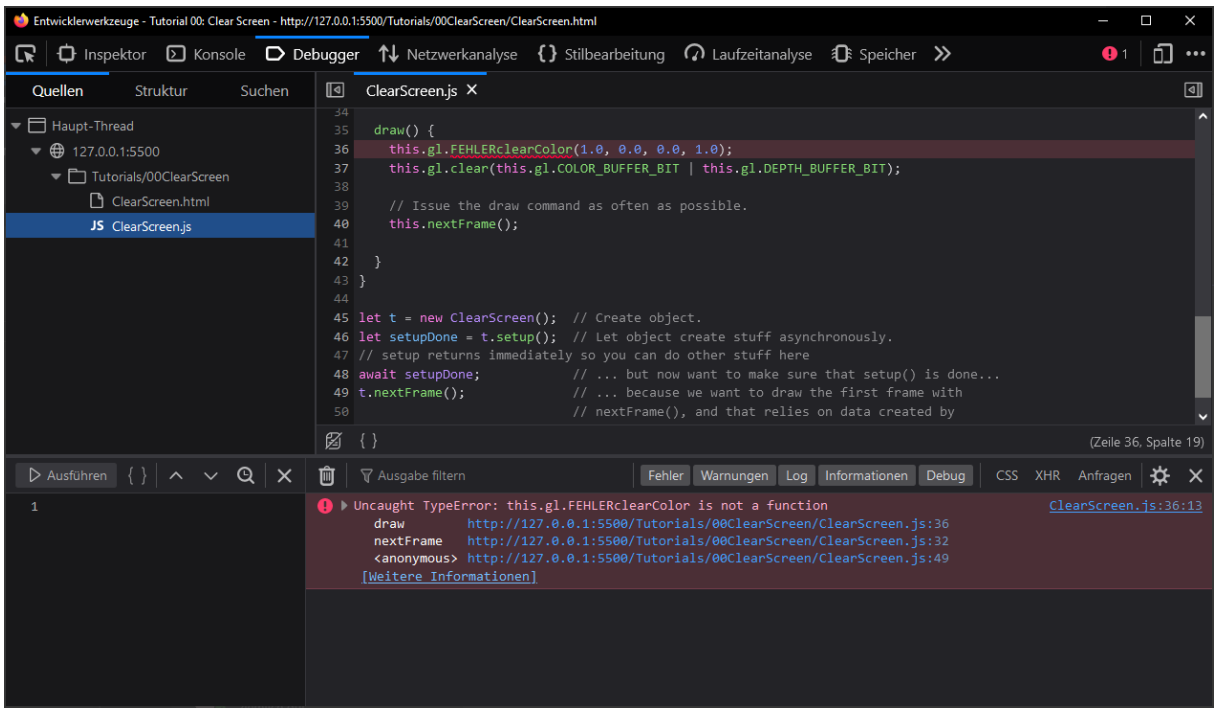

Abbildung 8: Der Debugger zeigt Java-Script Fehler an.

Unten sehen Sie nun den Programmierfehler und können ihn in Visual Studio Code beheben.

f) Nach Beheben des Fehlers in Visual Studio Code speichern Sie die geänderte Datei. Der Browser aktualisiert sich automatisch!

g) Nutzen Sie `this.gl.getParameter` um die Parameter die Clear-Farbe zu ermitteln und geben Sie diese auf der Console mittels `console.log` aus.

```
var clearColor = this.gl.getParameter(this.gl.COLOR_CLEAR_VALUE);
console.log(`${clearColor[0]}, ${clearColor[1]}, ${clearColor[2]},
    ${clearColor[3]} `);
```

Im Entwickler-Wergzeuge-Fenster Ihres Browsers, müssen Sie dazu in den Tab “Konsole” wechseln. Sie sollten dann folgende Ausgabe erhalten:

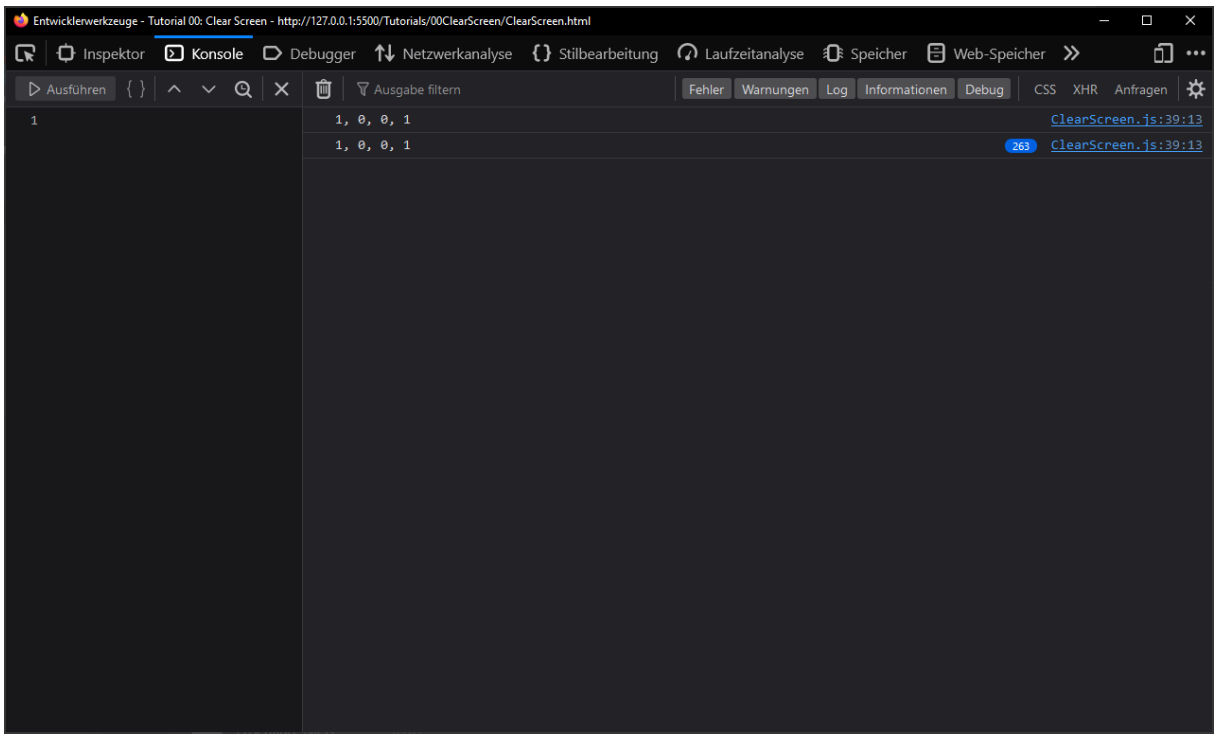

Abbildung 9: Ausgaben auf der Konsole sind hilfreich zum Debuggen.

5 Fragen

a) Definieren Sie eine Finite State-Machine? Nutzen Sie eine Internet Recherche dazu!

b) Setzen Sie die Hintergrundfarbe auf Blau!

c) In welchem Wertebereich liegen die Parameter von `this.gl.clear`? Was passiert, wenn Sie den Wertebereich nicht einhalten? Um welchen Datentyp handelt e sich?

d) Löschen Sie dem Depth-Buffer mittels `this.gl.clear()`!

e) Geben Sie den Tiefenwert mit dem der Tiefenpuffer gelöscht wird auf der Konsole aus!

f) Wie kann man den Tiefenwert setzen mit dem der Tiefenpuffer gelöscht wird?