Spieleentwicklung mit HTML-Canvas und Javascript

HTML: Hypertext Markup Language. Beschreibt den Inhalt und die Struktur einer Webseite.

CSS: Cascading Style Sheets. Definiert Aussehen, Layout und Gestaltung der Webseite.

Javascript: Programmiersprache, mit der zusätzliches Verhalten der Webseite festgelegt werden kann, z. B. dynamisches Ein- und Ausblenden oder Animieren von Elementen, asynchrones Nachladen von Inhalten oder Behandlung von User-Eingaben.

HTML-Grundlagen

Grundgerüst einer Webseite:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Titel der Seite</title>
  </head>
  <body>
    Inhalt der Seite
  </body>
</html>
```

Überschriften:

```
<h1>Überschrift 1. Ordnung</h1>
<h2>Überschrift 2. Ordnung</h2>
<h6>Überschrift 6. Ordnung</h6>
```

Textabsatz:

```
<p>Ein Absatz mit Text</p>
```

Ein Bild einbinden:

```
<img src="pfad/zum/bild.jpg" alt="Alternativtext für Screenreader" />
```

Allgemeine Block- und Inline-Elemente:

```
<div>Ich bin ein Block-Element</div>
<span>Ich bin ein Inline-Element</span>
```

Eine CSS-Datei einbinden (im <head>):

```
<link rel="stylesheet" href="pfad/zur/datei.css" type="text/css" />
```

Eine Javascript-Datei einbinden (im <head> oder <body>):

```
<script src="pfad/zur/datei.js"></script>
```

Javascript-Grundlagen

Eine Variable anlegen und Werte zuweisen:

```
const a = ‘Text’;     // Wert kann nicht verändert werden
let b = 17;           // Wert kann verändert werden
b = 38;
```

Text auf der Konsole ausgeben:

```
console.log(‘Ich bin ein String.’);
console.log(a);                  // Wert der Variable a ausgeben
console.log(‘Wert von a:’, a);  // Mehrere Parameter (Argumente)
```

Eine Funktion deklarieren und aufrufen:

```
function sum(a, b) {
  return a + b;
}
// oder:
const multiply = (a, b) ⇒ {
  return a * b;
};

const s = sum(3, 5);       // 8
const p = multiply(3, 5);  // 15
```

Bedingte Ausführung von Anweisungen:

```
if (age < 18) {
  console.log(‘Du bist zu jung!’);
} else {
  console.log(‘Bitte treten Sie ein!’);
}
```

Eine Liste (Array) von Werten Anlegen und Werte hinzufügen:

```
const liste = [17, 5, 12, -7, 8, 22, 13];
liste.push(9);
```

Zugriff auf Elemente eines Arrays:

```
console.log(liste[0]);     // 17
console.log(liste[3]);     // -7
liste[5] = 28;
```

Über jedes Element eines Arrays iterieren:

```
liste.forEach((element, index) ⇒ {
  console.log(‘An Index’, index, ‘steht:’, element);
});
```

Anlegen und Nutzen einer Klasse:

```
class Rectangle {
  constructor(width, height) {
    this.width = width;
    this.height = height;
  }

  area() {
    return this.width * this.height;
  }
}

const r1 = new Rectangle(5, 8);
const r2 = new Rectangle(3, 7);
console.log(r1.area());    // 40
console.log(r2.area());    // 21
```

Konzepte der Spieleentwicklung

Sprites: Eine Bilddatei, die mehrere Animationszustände eines Spielobjekts enthält und aus der nach Bedarf der passende Bildausschnitt zur Anzeige ausgewählt wird. Schnelle Abfolge verschiedener Teile des Sprites ergeben die Animation einer Bewegung.

Raycasting/Raytracing: Methode zur Erzeugung eines 2D-Bildes eines Blicks in eine 3D-Welt. Virtuelle Lichtstrahlen werden von der Kamera in die Szene hinein verfolgt, um zu ermitteln, wie der sichtbare Bildausschnitt gerade aussieht.

Game-Loop: Eine Schleife, die endlos läuft, User-Input entgegennimmt, der neuen Zustand des Spiels berechnet und die graphische Ausgabe daraufhin anpasst. Die Game-Loop gibt den „Herzschlag" des Spiels vor.

Physics Engine: Software-Modul zur Berechnung und Simulation eines möglichst realistischen Verhaltens von Objekten in einem Computerspiel auf Basis physikalischer Gesetze. Eine Physics Engine erlaubt bpsw. die Simulation von Gravitation, Kollisionen, Flugbahnen oder Elastizität.

Ein rotes Rechteck auf den Canvas zeichnen:

```
// HTML:
<canvas id="myCanvas"></canvas>

// Javascript:
window.addEventListener("load", function() {
  const canvas = document.getElementById("myCanvas");
  const ctx = canvas.getContext("2d");

  ctx.beginPath();
  ctx.fillStyle = "red";
  ctx.rect(10, 20, 400, 50);
  ctx.fill();
});
```

requestAnimationFrame: Weist den Browser an, die übergebene Funktion vor dem Nächsten Neuzeichnen des Viewport-Inhalts auszuführen. Die Häufigkeit der Aufrufe richtet sich nach der Refresh-Rate des Displays (z. B. 60 Hz).

Eine einfache Game-Loop in Javascript:

```
function animate() {
  // Neuberechnen des Zustands des Spiels
  // Auswerten des User-Inputs (Mausklicks, Tastatur-Eingaben etc.)
  // Graphische Ausgabe auf Basis des neuen Zustands

  requestAnimationFrame(animate);
}

animate();
```