Memory

Wir programmieren zusammen ein Memory.

Spielanleitung

Memory wird mit einer geraden Anzahl an Karten gespielt.
Diese haben jeweils eine farbige Vorderseite und eine graue Rückseite.
Jeweils zwei passen zueinander.
Alle Karten werden vermischt und umgedreht auf einem Tisch ausgebreitet, so dass man die Farben nicht sieht.
Ein*e Spieler*in dreht zwei Kärtchen nacheinander um.
Wenn die beiden zueinander passen, darf er/sie noch einmal 2 umdrehen.
Wenn die Karten aber nicht zusammen passen ist die nächste person dran.

Gewonnen hat, wer am meisten Paare gefunden hat.

Vorbereitung

Erstelle auf dem Desktop (oder einem Ordner deiner Wahl) ein Order namens: "memory"
In diesem erstellst du folgende Dateien:

- index.html
- index.js
- style.css

index.html

Im `index.html` ist die Struktur unseres Spieles gespeichert.

Wikipedia definiert HTML so:

> Die Hypertext Markup Language (HTML) ist eine textbasierte Auszeichnungssprache zur Strukturierung elektronischer Dokumente
> wie Texte mit Hyperlinks, Bildern und anderen Inhalten. HTML-Dokumente sind die Grundlage des World Wide Web und
> werden von Webbrowsern dargestellt.

Die Elemente die im Browser angezeigt werden sollen, werden in den sogenannten Tags definiert.
Tags welche Inhalt haben können wie z.B. Knöpfe, Überschriften oder Paragraphen, werden geöffnet (`<tag>`) und geschlossen (`</tag>`).

Beispiel:

```
<h1>Ich bin eine Überschrift!</h1>
<h2>Ich bin eine kleinere Überschrift!</h2>
<p>Ich bin ein Paragraph<p>
<button>Ich bin ein Knopf</button>
```

Kopiere nun folgenden Code in dein `index.html`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./style.css">
    <title>Memory</title>
</head>
<body>
    <h1 id="player">Spieler*in 1 ist am Zug.</h1>
    <h2 id="score">0 : 0</h2>
    <div id="game"></div>
</body>
<script src="./index.js"></script>
</html>
```

Wenn du dein `index.html` nun im browser öffnest, solltest du eine Überschrift die sagt wer am Zug ist sehen, so wie der Punktestand.

style.css

Damit unser spiel auch gut aussieht, haben wir CSS. Mit CSS kannst du das Aussehen von deinem Spiel bestimmen. Dafür kannst du Regeln für das darstellen vom HTML machen - so genannte rules.
Diese sind folgendermassen aufgebaut:

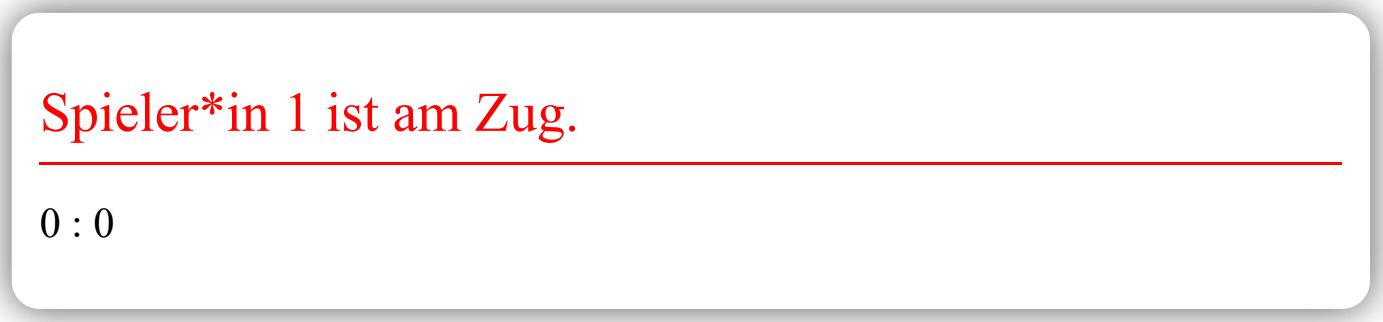
```
[Selektor] {
    [Eigenschaft]: [Wert];
    [Eigenschaft]: [Wert];
    ...
}
```

Ein Selektor wäre z.B `h1`. Damit würde dann die Regel auf alle grossen Überschriften zutreffen.
Ein Beispiel fü eine Eigenschaft ist die Farbe: `color: red`

Also wenn wir alle grossen Überschriften rot haben wollen müssen wir folgende Regel schreiben:

```
h1 {
    color: red;
}
```

Dann sieht unser html so aus:



Nun kannst du folgendes CSS in den `style.css` kopieren:

```
#game {
    /*
    Stelle die Memory-Karten in einem Raster dar.
    */
    display: grid;
    grid-template-columns: 1fr 1fr 1fr 1fr;
    gap: 1rem;
}

#game div {
    /*
    Mache die Memory-Karten Kartenartig
    */
    min-width: 2rem;
    min-height: 10rem;
    background-color: rgb(225, 225, 225);
    border-radius: 0.5rem;
}
```

index.js

Im `index.js` ist die Logik für das Spiel gespeichert. Wir benötigen es, damit die Memory-Karten gemischt werden oder um die Punkte an zu zeigen.

Wikipedia definiert JavaScript so:

> JavaScript (kurz JS) ist eine Skriptsprache, die ursprünglich 1995 von Netscape für dynamisches HTML in Webbrowsern entwickelt
> wurde, um Benutzerinteraktionen auszuwerten, Inhalte zu verändern, nachzuladen oder zu generieren und so die Möglichkeiten von HTML
> zu erweitern. Heute wird JavaScript auch außerhalb von Browsern angewendet, etwa auf Servern und in Mikrocontrollern.

Nun kannst du folgenden Code in dein `index.js` einfügen:

```
const COLORS = ["maroon", "red", "purple", "fuchsia", "lime", "black", "yellow",
"navy"]

// in dieser Variable werden die umgedrehten Karten gespeichert
let selected = []

// alle Farben noch einmal hinzufügen
COLORS.push(...COLORS)
// Farben mischen
COLORS.sort((a, b) => 0.5 - Math.random())

for (let i = 0; i < COLORS.length; i++) {
    const color = COLORS[i];

    // neues div (leeres Element) erstellen
    const div = document.createElement("div")

    // div ins html laden
    document.getElementById("game").appendChild(div)

    //Todo 00
}

// Todo 01


// Todo 02
```

Wenn du nun `index.html` in einem browser deiner Wahl öffnest, solltest du 16 Memory-Karten sehen.
Falls das nicht der fall sein sollte, überprüfe ob alle Dokumente im selben Ordner
sind und ob alle imports im HTML hast (für JavaScript und CSS).

Das Spiel

Karten aufdecken

Damit wir das Memory überhaupt spielen können, muss man die Karten umdrehen können.
Dafür müssen wir, wenn die Karte angeklickt wird, die richtige Farbe zu den Styles hinzufügen.

Dies kannst den `//ToDo 00` im `index.js` Kommentar mit folgendem Code ersetzen:

```
// definieren was auf Knopfdruck passiert
div.onclick = (event) => {
    //ToDo 00
}
```

In diesem Codestück weist du der `onclick` eigenschaft eine Funktion zu.
Da dieseFunktion keinen Namen braucht brauchen wir die sogenannte Lambda schreibweise.
In den Klammern, definieren wir die Parameter (werte) die der Methode mitgegeben werden.
In diesem fall ist das das Klickevent. Mit dem Pfeil weisen wir auf die Aktionen die
Ausgeführt werden sollen. Diese sind inzwischen geschweiften Klammern.

Innerhalb der geschweiften Klammern müssen wir nun die Hintergrundfarbe einstellen.
Dafür hat das `div` die Property `style`. Diese können wir mit einem Punkt
wie gefolgt anwählen: `div.style`. Um die hintergrundfarbe zu ändern müssen wir die
Property `backgroundColor` der Property `style` verändern. Wir setzen sie auf die Farbe des `div`'s mit
folgendem Code:

```
div.style.backgroundColor = color
```

Nun sollten sich die Memory-Karten, wenn du im Browser auf sie klickst, sich "umdrehen" bzw. jeweils ihre Farbe anzeigen.

Karten verdecken

Damit man nicht in einem Zug alle Karten aufdecken kann, müssen wir die Anzahl von
Karten die man aufdecken darf auf 2 begrenzen. Dafür stellen wir den "Aufdeck-Code"
in ein If-Statement. In diesem wird dann geprüft, ob weniger als 2 Karten umgedreht sind.
Damit wir wissen wie viele Karten überhaupt umgedreht sind, fügen wir die Karte beim
Aufdecken dem `selected` array hinzu.

Das machen wir mit folgendem Code:

```
if (selected.length < 2) {
    // Farbe anzeigen
    div.style.backgroundColor = color
    // zu den umgedrehten Karten hinzufügen
    selected.push(div)
}
```

Wenn du nun probierst alle Karten aufzudecken, geht das nicht. Nach 2 Karten ist Fertig.
Da mit sich die Karten wieder zudecken, braucht es noch ein wenig mehr Logik.
Nämlich müssen wir wenn jeweils 2 Karten aufgedeckt sind, diese wieder zudecken.
Damit man aber sieht welche Farbe die 2te Karte hat, brauchen wir eine kleine Verzögerung.

Damit der Code leserlich bleibt, lagern wir die allgemeine Spiellogik in eine eigene Funktion
aus. Ersetze `ToDo 01` mit folgendem Code:

```
// in dieser Funktion ist die Spiellogik
function game() {
    // schauen ob 2 Karten umgedreht sind
    if (selected.length === 2) {
        // eine halbe Sekunde warten
```

```
        setTimeout(() => {
            // Todo 01
        }, 500)
    }
}
```

Rufe nun die Funktion innerhalb vom `onclick` auf das sollte dann so aus sehen:

```
div.onclick = (event) => {
    // Code nur ausführen, wenn noch nicht 2 Karten umgedreht sind
    if (selected.length < 2 && !selected.includes(div)) {
        // Farbe anzeigen
        div.style.backgroundColor = color
        // zu den umgedrehten Karten hinzufügen
        selected.push(div)
        game()
    }
}
```

Damit die Karten dann wirklich umgedreht werden, müssen wir innerhalb der `game` Funktion beim ToDo die Farben der div's in der `selected` liste noch zurück setzten und diese dan anschliessend leeren. Das kannst du mit folgendem Code machen:

```
// Karten verdecken
selected.forEach((div) => {
    div.style = ""
})
// Variable leeren
selected = []
```

Korrekte Paare aus dem Spiel entfernen

Um zu erkenn en ob zwei gleiche Karten aufgedeckt wurden, müssen wir nur überprüfen, ob deren Farbe die selbe ist. Das können wir ganz einfach mit einem If-Statement machen. Wir schauen ob die Hintergrundfarbe des div's an position 0 des `selected` Arrays die selbe ist wie die des div's an position 1. das kannst du mit folgendem If-Statement machen:

```
// überprüfen, ob es zwei gleiche Karten sind
if (selected[0].style.backgroundColor === selected[1].style.backgroundColor) {

} else {

}
```

Verschiebe nun das umdrehen der Karten in den `else` Teil des If-Statements.
Im `if` Teil, kannst du nun den code fürs entfernen einfügen:

```
// Karten deaktivieren
.forEach((div) => {
    // hintergrund Farbe der Karte auf transparent
    div.style.backgroundColor = "transparent"
    // onclick entfernen
    div.onclick = null;
})
```

Nun sollten korrekte paare verschwinden. Probier's doch mal aus!

Punkte zählen

Damit wir die Punkte zählen können, müssen wir 3 Dinge speichern: die Punkte von Player 1, die Punkte von Player 2 und wer von beiden am Zug ist. Das machen wir mit folgenden Variablen.

```
let isPlayerOne = true;
let pointsPlayerOne = 0;
let pointsPlayerTwo = 0;
```

Diese kannst du am anfangs des JavaScript files initialisieren.

Jedes Mal, wenn nun ein Player 2 gleiche Karten aufdeckt, rechnen wir die entsprechende Variable + 1:

```
if (isPlayerOne) {
    pointsPlayerOne += 1
} else {
    pointsPlayerTwo += 1
}
```

Nun müssen wir jeweils noch den Player wechseln. Das machen wir in dem wir `isPlayerOne` auf das Gegenteil von `isPlayerOne` setzen:

```
// Player wechseln
isPlayerOne = !isPlayerOne
```

Das machen wir jedes mal wenn ein Player 2 unterschiedliche Karten aufgedeckt hat.

Display aktualisieren

Damit wir sehen wie der punktestand ist und auch wer am Zug ist, müssen wir das HTML mit den richtigen Daten aktualisieren. Das machen wir in einer eigenen Funktion, damit man diese immer wen etwas geändert hat aufrufen kann.
Im HTML sind jeweils schon Überschrifts-Tag's für den aktuellen Player und den score vorhanden. Diese kannst du mit deren id selektieren.

Um den richtigen spieler anzuzeigen, kannst du eine neue Variable initialisieren und mit Hilfe von einem If-Statement den korrekten Player hineinschreiben. Hier der code:

```
function updateScreen() {
    let player = ""

    if (isPlayerOne) {
        player = "1"
    } else {
        player = "2"
    }

    // Elemente updaten
    document.getElementById("player").innerText = "Spieler*in " + player + " ist
am Zug."
    document.getElementById("score").innerText = pointsPlayerOne + ":" +
pointsPlayerTwo
}
```

Nun musst du noch die `updateScreen` Funktion nach dem updaten der Punkte aufrufen.

Jetzt kannst du dein fertiges Memory ausprobieren.

Zusatz

Wenn du noch Zeit hast kannst du noch weitere Features programmieren.
Hier einige Vorschläge:

- Knopf um ein neues Spiel zu starten
- Bilder statt Farben
- Variable Anzahl von Spieler*innen