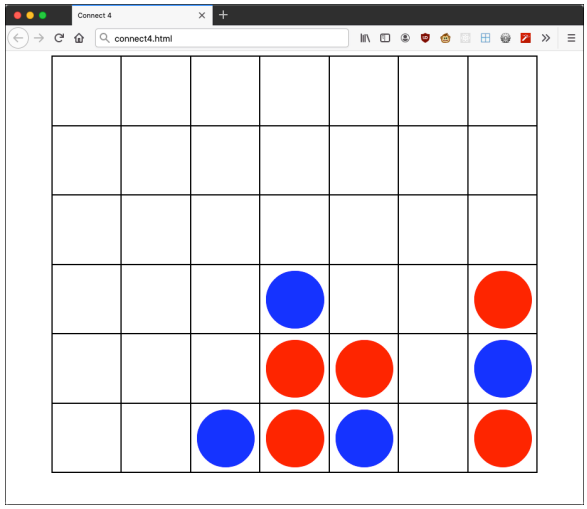
WBE-Praktikum 8

Spiel im Browser

Aufgabe 1: Vier gewinnt (Miniprojekt)



„**Vier gewinnt** (englisch: **Connect Four** oder **Captain's mistress**) ist ein Zweipersonen-Strategiespiel mit dem Ziel, als Erster vier der eigenen Spielsteine in eine Linie zu bringen. [...] **Regeln:** Das Spiel wird auf einem senkrecht stehenden hohlen Spielbrett gespielt, in das die Spieler abwechselnd ihre Spielsteine fallen lassen. Das Spielbrett besteht aus sieben Spalten (senkrecht) und sechs Reihen (waagerecht). Jeder Spieler besitzt 21 gleichfarbige Spielsteine. Wenn ein Spieler einen Spielstein in eine Spalte fallen lässt, besetzt dieser den untersten freien Platz der Spalte. Gewinner ist der Spieler, der es als erster schafft, vier oder mehr seiner Spielsteine waagerecht, senkrecht oder diagonal in eine Linie zu bringen. [...]" (Wikipedia)

Das Erzeugen der Grafik im Browserfenster (also dem Viewport) kann auf unterschiedliche Weise umgesetzt werden: HTML&CSS, Canvas oder SVG.

Aus Gründen der Einfachheit wählen wir eine möglichst einfache Darstellung mit HTML und CSS. In der Datei *connect4.html* finden Sie ein Beispiel, wie das gehen könnte. Das CSS ist zunächst mit im HTML-Code, sollte später aber in eine separate CSS-Datei ausgelagert werden. Im Beispielcode sind nur zwei Reihen à sieben Feldern vorgesehen. Bis auf das erste Feld sind aber alle ausgeblendet.

Miniprojekt

Das Miniprojekt, das in mehreren Praktikumsaufgaben in der zweiten Semesterhälfte in WBE durchgeführt wird, ist neben den Praktikumsabgaben der zweite praktische Leistungsnachweis während des Semesters. Es kann in Zweierteams durchgeführt werden und soll schliesslich auf GitHub Pages zusammen mit einer kurzen Dokumentation verfügbar gemacht werden. Es wird am Ende des Semesters bewertet und fliesst damit in die Semesterbewertung ein.

Aufgaben:

- Sehen Sie sich die Ausgabe im Browser-Fenster an. Welchem HTML-Element entspricht der blaue Kreis und welche CSS-Eigenschaften werden für seine Darstellung verwendet? Wie wird erreicht, dass sich die Grösse des Spielfelds an der Grösse des Viewports orientiert?
- Entfernen Sie die CSS-Angaben, welche dafür sorgen, dass nur das erste Feld angezeigt wird. Da *div* ein Blockelement ist, werden die Felder nun untereinander dargestellt.
- Passen Sie das CSS so an, dass zwei Zeilen à sieben Feldern angezeigt werden. Am besten lösen Sie das mit fliessenden Boxen (*float*). Wenn Sie Flexbox oder CSS-Grid kennen, können Sie auch eine von diesen Varianten verwenden – in diesem einfachen Beispiel sind alle drei Varianten nicht sehr aufwändig.

Jetzt haben wir erst zwei Zeilen des Spielfelds umgesetzt. Um nun zu sechst Zeilen zu kommen, müssten wir noch ziemlich viele *div*-Elemente mit der Klasse *field* hinzufügen. Da wir das aber vermeiden wollen, gehen wir zu DOM-Scripting über und erzeugen das Spielfeld per JavaScript.

Im Unterricht wurde eine Funktion *elt* eingeführt, welche das Einfügen von HTML-Elementen ins DOM vereinfacht.[1] Hier ist eine erweiterte Version der Funktion, in der auch Attribute gesetzt werden können:

```
function elt (type, attrs, ...children) {
  let node = document.createElement(type)
  for (a in attrs) {
    node.setAttribute(a, attrs[a])
  }
  for (let child of children) {
    if (typeof child != "string") node.appendChild(child)
    else node.appendChild(document.createTextNode(child))
  }
  return node
}
```

- Machen Sie noch einmal klar, wie *elt* funktioniert. Welche Argumente werden beim Aufruf angegeben? Was ist der Rückgabewert? Schlagen Sie unbekannte Funktionen oder Methoden in einer Dokumentation nach.[2]
- Entfernen Sie die *div*-Elemente mit der Klasse *field* aus dem HTML-Code. Schreiben Sie JavaScript-Code, welcher beim Laden des Dokuments sechs Zeilen mit je sieben leeren Feldern erzeugt.
- Ergänzen Sie Ihr Script so, dass in einigen der Felder eine rote oder blaue Spielfigur eingefügt wird, am besten gesteuert durch den Zufallszahlengenerator, Aufruf: `Math.random()`.

[1] Es ist klar, dass *elt* kein sehr aussagekräftiger Name für die Funktion ist. Da aber mit dem Einfügen von HTML-Strukturen ins DOM meist viele Aufrufe der Funktion verbunden sind, wurde ein möglichst kurzer Name gewählt.

[2] Zum Beispiel: https://devdocs.io/dom/

Der *body*-Teil des HTML-Dokuments dürfte jetzt recht übersichtlich sein. In Ihrem Script muss dann noch die Funktion *showBoard* implementiert (und *elt* eingefügt) werden:

```
<body>

  <div class="board"></div>

  <script>
    showBoard()
  </script>

</body>
```

Wir können nun das Spielfeld mit Hilfe eines Scripts relativ einfach generieren. Wenn wir das Spiel aber spielbar machen wollen, müssen wir auf Klick-Ereignisse reagieren und dabei immer wieder auf den aktuellen Spielzustand zugreifen können: Ist das dritte Feld in der zweiten Spalte schon belegt und wenn ja durch welche Farbe? Das ist relativ aufwändig zu ermitteln, da der aktuelle Spielzustand ausschliesslich im DOM repräsentiert ist.

Aufgabe 2: Trennung Model/View (Miniprojekt)

In dieser Aufgabe soll der Spielzustand (Model) von der Darstellung im Browser (View, im DOM repräsentiert) getrennt werden. Wir machen es zunächst einfach und repräsentieren das Spielfeld als Array von sechs Zeilen und jede Zeile als Array von 7 Strings: der leere String steht für ein leeres Feld, "r" und "b" für ein rot bzw. blau belegtes Feld. Zum Beispiel:

```
let state = {
  board: [
    [ '', '', '', '', '', '', '' ],
    [ '', '', '', '', '', '', '' ],
    [ '', '', '', '', '', '', '' ],
    [ '', '', '', '', '', '', '' ],
    [ '', 'b', '', '', '', '', '' ],
    [ '', 'r', '', 'b', '', '', '' ]
  ]
}
```

Ein leeres Spielfeld könnte man dann übrigens auch so anlegen:

```
let state = = Array(6).fill('').map(el => Array(7).fill(''))
```

- Schreiben Sie nun eine Funktion *showBoard*, welche das komplette Board für den aktuellen Spielstand ausgibt, also die entsprechenden *div*-Elemente ins DOM einfügt.
- Ergänzen Sie Ihr Script um einen Timer, der jede Sekunde ein Feld zufällig auswählt und dieses – ebenfalls zufällig – löscht oder mit einem roten oder blauen Spielstein belegt.