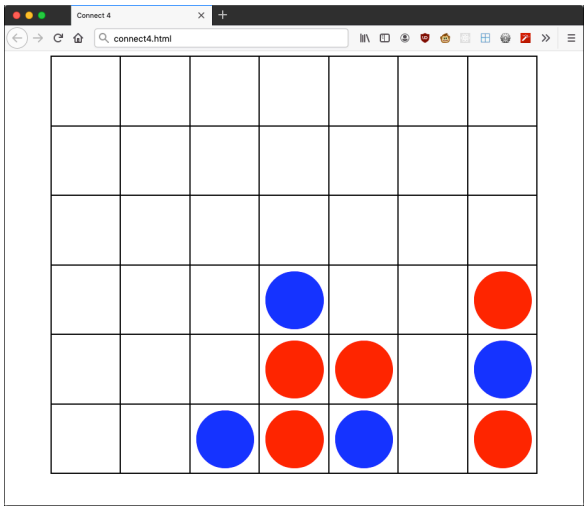
WBE-Praktikum 8

Spiel im Browser

Aufgabe 1: Vier gewinnt (Miniprojekt)

Mit der Implementierung des Spiels **Vier gewinnt** wurde im ersten Praktikum des Semesters begonnen. Dort wurde gezeigt, dass die Grafik im Browserfenster (also dem Viewport) auf unterschiedliche Weise umgesetzt werden kann: HTML&CSS, Canvas oder SVG.



Anstelle der statischen Grafik (Praktikum 1) soll in dieser und folgenden Praktikumslektionen die Ausgabe nun dynamisch von JavaScript aus erfolgen. **Da es ums vor allem um DOM-Scripting geht, verwenden wir dazu die Variante HTML und CSS.**[1]

Miniprojekt

Das Miniprojekt, das in mehreren Praktikumsaufgaben in der ersten Semesterwoche begonnen und in der zweiten Semesterhälfte fortgesetzt 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. Zum Miniprojekt gibt es keine laufenden Abgaben, es wird erst gegen Ende des Semesters abgegeben.

[1] Es ist natürlich durchaus möglich, später noch eine schönere Oberfläche mit Canvas oder SVG zu bauen. Für das aktuelle Praktikum und vor allem auch die Versuche mit dem Web-Framework in späteren Praktikumsaufgaben, ist es aber von Vorteil, zunächst mit der HTML&CSS-Variante zu arbeiten.

Diese und die folgenden Aufgaben basieren auf der Lösung der HTML&CSS-Variante zur Ausgabe des Spielfelds aus Praktikum 1. Die Anzeige des Spielfelds mit HTML und CSS benötigt ziemlich viel HTML-Code mit vielen «leeren» *div*-Elementen. Nun 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.[2] Hier ist eine erweiterte Version der Funktion, in der auch Attribute gesetzt werden können:

```
function elt (type, attrs, ...children) {
  let node = document.createElement(type)
  Object.keys(attrs).forEach(key => {
    node.setAttribute(key, attrs[key])
  })
  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.[3]
- 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()`.

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>
```

[2] 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.

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

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.