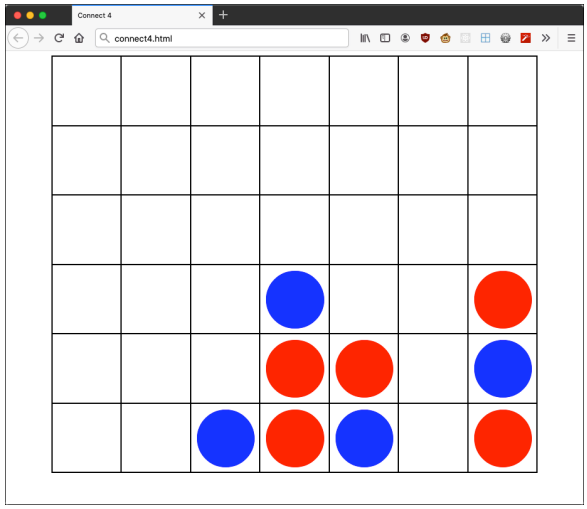
**WBE-Praktikum 9**

# Spiele im Browser

## Aufgabe 1: Vier gewinnt




„**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.

**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. Das ist sowohl mit fliessenden Boxen (*float*) als auch mit Flexbox oder CSS-Grid möglich. Empfehlung: Flexbox (die anderen Varianten auch noch auszuprobieren wäre sicher eine gute Übung).
- 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.

Das Anlegen neuer HTML-Elemente kann einfach mit der im Unterricht besprochenen Funktion elt erledigt werden. Am besten erweitern wir sie so, dass auch gleich 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
}
```

Ein *div*-Element mit Klasse *field*, kann dann so eingefügt werden:
`board.appendChild(elt("div", {"class": "field"}))`

- 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()`.
- Wenn wir nun noch einen Event Handler ergänzen, der beim Klick auf das Spielfeld abwechselnd rote und blaue Scheiben auf dem Spielfeld platziert, wäre das Spiel bereits einsetzbar. Ein Nachteil ist aber, dass der aktuelle Spielstand dann ausschliesslich im DOM repräsentiert ist. Das macht Erweiterungen kompliziert. Stellen Sie sich vor sie möchten eine Funktion schreiben, welche erkennt, ob ein Spieler bereits gewonnen hat. Daher: Definieren Sie eine Datenstruktur, die den aktuellen Spielstand repräsentiert.
- 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 so, dass beim Klick auf ein Feld eine Spielfigur (Klasse *piece*) in das Feld eingefügt wird. Die Farbe soll bei jedem Klick wechseln: einmal rot, einmal blau.

Das Spiel ist nun spielbar. Es gibt aber noch in verschiedener Hinsicht Verbesserungspotential. So gibt es noch keinen Schutz, dass ein belegtes Feld nicht erneut belegt wird. Ausserdem fallen die Spielsteine im Originalspiel jeweils nach unten, bis sie auf ein belegtes Feld treffen.

**Ergänzungen (fakultativ):**

- Neue Spielfiguren werden eingefügt, indem auf eine Spalte geklickt wird. Sie werden dann auf in dieser Spalte auf den von unten gesehen ersten freien Platz gelegt (fallen nach unten).

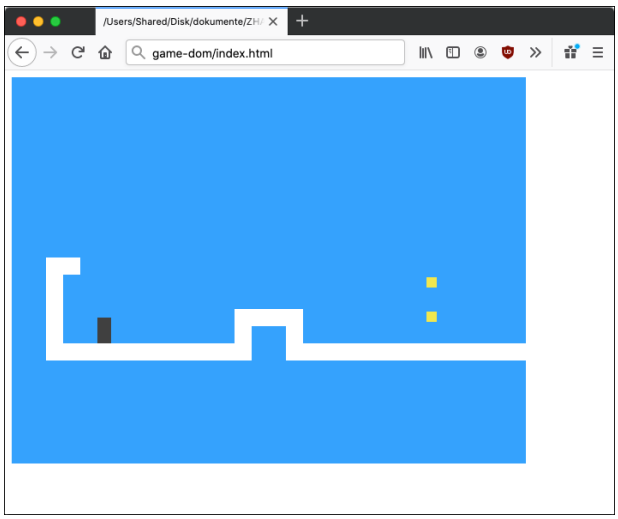
- Es wird angezeigt, ob rot oder blau am Zug ist.
- Es gibt einen Button «Neues Spiel», der ein neues, leeres Spielfeld anzeigt.
- Es wird erkannt, wenn eine Spalte voll ist.
- Das ist mit etwas Aufwand verbunden: Es wird erkannt, wenn ein Spieler gewonnen hat, und eine entsprechende Meldung ausgegeben. Neue Spielfiguren können dann nicht mehr ins Spielfeld eingefügt werden.

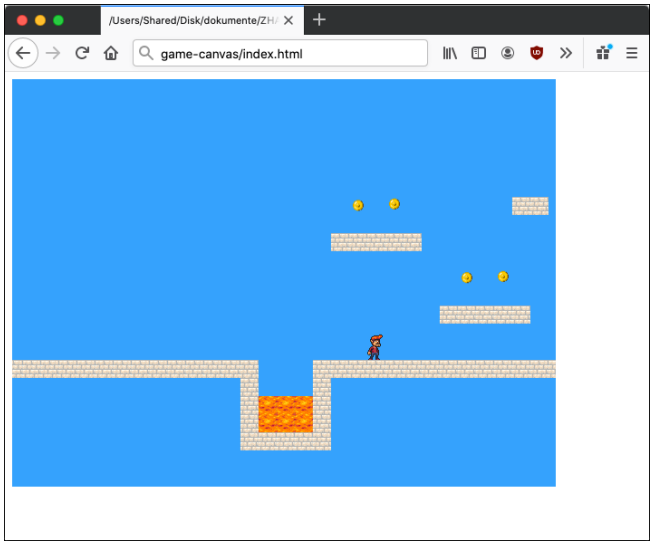
## Aufgabe 2: DOM und Canvas (fakultativ)

Im Verzeichnis *game-dom* finden Sie ein kleines «Jump-and-run»-Spiel. Es ist ein Beispiel aus dem Buch *Eloquent JavaScript (Third Edition)* von Marijn Haverbeke und im Kapitel 16: *Project – A Platform Game* näher beschrieben.

https://eloquentjavascript.net

Diese Version des Spiels verwendet das HTML und CSS für die Darstellung. Im Vergleich zu unseren bisherigen Beispielen ist der Code relativ umfangreich. Eine Beschreibung der einzelnen Funktionen finden Sie im genannten Buchkapitel.




Eine etwas verschönerte Grafik ist mit der Canvas-API möglich. Diese Variante des Spiels finden Sie unter *game-canvas* und die zugehörige Beschreibung der Anpassungen in Kapitel 17 des genannten Buchs. Probieren Sie das Spiel einmal aus.

Grössere Änderungen am Spiel wären viel zu aufwändig für eine Praktikumslektion. Bei Interesse lesen Sie die beiden Kapitel des Buchs. Übungsaufgaben finden Sie bei Bedarf ebenfalls dort.

Ansonsten können Sie sich auch einfach einen Überblick über die Implementierung des Spiels verschaffen, die eine oder andere kleine Änderung vornehmen, Levels anpassen oder komplett neue Levels erzeugen. Wir sind gespannt auf neue Spielvarianten ☺.