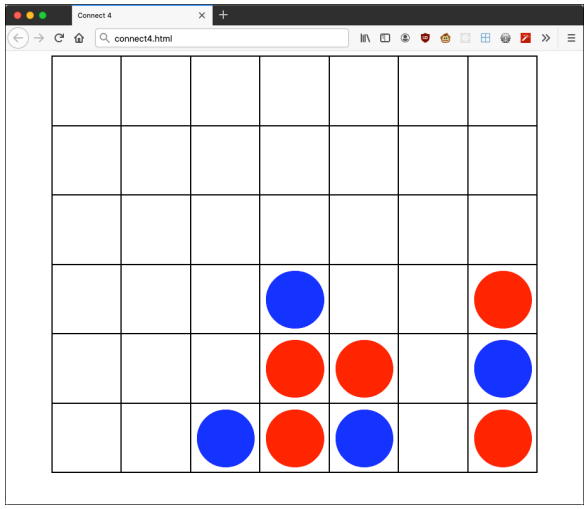
WBE-Praktikum 9

Spiele im Browser

Aufgabe 1: Ereignisse (Miniprojekt)



Die Arbeit an dem im letzten Praktikum begonnenen Spiel «Vier gewinnt» soll nun fortgesetzt werden. Es geht nun darum, das Spiel interaktiv zu machen, das heisst auf Klick-Ereignisse zu reagieren.

Aufgaben:

- Erstellen Sie eine Kopie des letzten Projektstands und passen Sie das Script so an, dass mit einem leeren Spielfeld begonnen wird: also keine vorbelegten Felder und kein Timer, der Felder zufällig belegt oder wieder leert.
- 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.

Wenn Sie den Zustand so angelegt haben wie im letzten Praktikum vorgeschlagen, können Sie das Zustandsobjekt einfach durch ein weiteres Attribut ergänzen, das angibt, ob rot oder blau am Zug ist.

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.

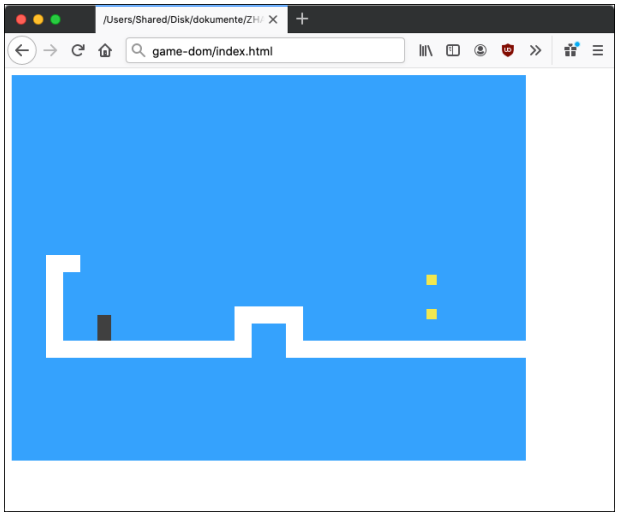
- Passen Sie Ihr Script so an, dass beim Klick auf ein Feld zunächst die Spalte dieses Felds ermittelt wird und die Spielfigur im untersten freien Feld dieser Spalte platziert wird. Wenn die Spalte bereits voll ist, wird der Klick ignoriert.
- Ergänzen Sie die Seite um einen Button «Neues Spiel», welcher das Spielfeld leert.
- Ergänzen Sie das Script um die Angabe, ob rot oder blau den nächsten Zug hat.

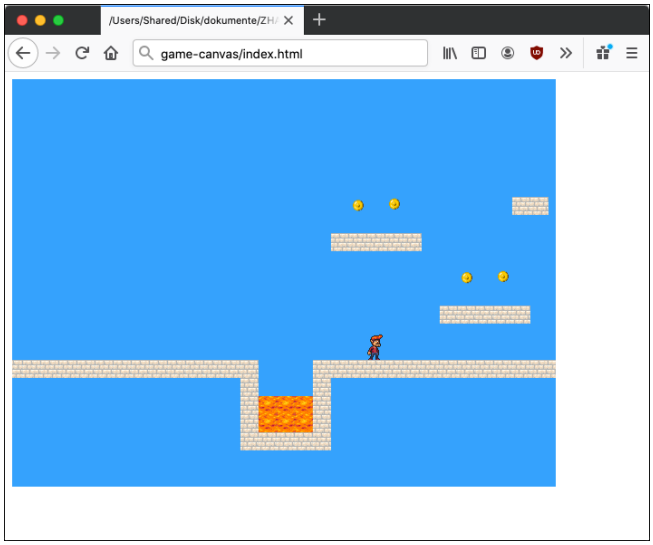
Aufgabe 2: DOM und Canvas (fakultativ)

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 ☺.

Aufgabe 3: Funktionen dekorieren (fakultativ)

Hier ist eine rekursive Variante der *fibonacci*-Funktion:

```
let fibonacci = (n) => {
  if (n < 2) return n
  else return fibonacci(n-1) + fibonacci(n-2)
}
```

Sie soll nun noch um Features erweitert werden, ohne den Programmcode der Funktion anzupassen. Um zum Beispiel eine Protokollierung der Aufrufe an die Funktion anzuhängen, könnte folgender Aufruf verwendet werden:

```
fibonacci = trace(fibonacci, "fibonacci")
```

Das zweite Argument mit dem Funktionsnamen dient nur dazu, ein gut lesbares Protokoll zu erhalten. Die Funktion *trace* erhält eine Funktion als Argument, modifiziert sie und gibt die modifizierte Funktion zurück. Sie könnte so aufgebaut sein:

```
function trace (func, funcname) {
  return (...args) => {
    console.log(funcname + "(" + args.toString() + ")")
    return _____________________________
  }
}
```

Aufgaben:

- Ergänzen Sie Ihr Script um die *trace*-Funktion. Der *return*-Wert der Rückgabefunktion fehlt noch. Was muss zurückgegeben werden, dass die Funktion abgesehen vom Protokollieren wie gewohnt arbeitet, also weiterhin den Funktionswert liefert?
- Probieren Sie, ob das Protokollieren funktioniert, indem Sie zum Beispiel *fibonacci(5)* berechnen.
- Inwiefern ist die neue Funktion *fibonacci* eine Closure?
- Nun ist *fibonacci* keine pure Funktion mehr, denn ausser dem Berechnen des Funktionswerts haben wir nun Seiteneffekte (*console.log*). Da solche Seiteneffekte das Testen erschweren, kommentieren Sie die Zeile mit dem *trace*-Aufruf wieder aus.
- In *memo.js* finden Sie einen weiteren Dekorierer: *memo*. Dieser dient dazu, alle Zwischenresultate der modifizierten Funktion in einem Cache (hier eine *Map*) zwischenzuspeichern. Sehen Sie sich die Funktion *memo* an. Es genügt, wenn Sie ungefähr verstehen, wie *memo* arbeitet.
- Kopieren Sie *memo* ebenfalls in Ihr Script und modifizieren Sie *fibonacci* nun damit. Ergänzen Sie Ihr Script um eine Zeitmessung: *fibonacci(30)* mit und ohne Memoizer.

Die letzten Beispiele sollen zeigen, dass JavaScript – speziell was die Möglichkeiten von Funktionen angeht – eine sehr mächtige Programmiersprache ist. Wir befinden uns hier bereits im Bereich der Funktionalen Programmierung, ein Programmierparadigma, welches in den Wahlfächern PSPP und FUP ausführlicher angesehen wird.