WBE-Praktikum 14

Abschluss

Keine neuen Aufgaben ☺

Nur eine Anregung für diejenigen, welche das Miniprojekt noch erweitern möchten.

Sie können die Gelegenheit nutzen, sich die eine oder Praktikumsaufgabe noch einmal vorzunehmen oder noch nicht gelöste Aufgaben zu bearbeiten.

Den letztmöglichen Termin für Praktikumsabgaben erfahren Sie in den Lektionen, da dies von Klasse zu Klasse abhängig von den Terminen der Unterrichts- und Praktikumslektionen unterschiedlich geregelt ist.

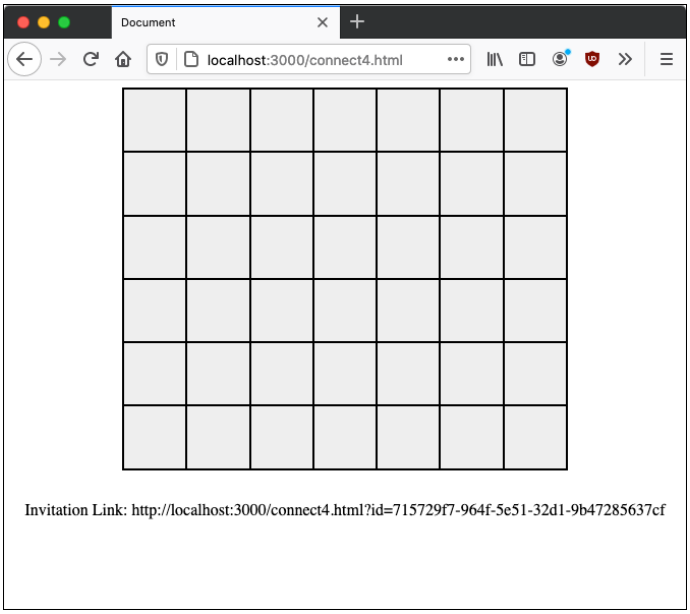
Wenn Sie noch eine Herausforderung suchen: Versuchen Sie, das Beispiel «Vier gewinnt» in der Multiplayer-Variante mit Synchronisation über den Server mit unserer Bibliothek SuiWeb zu lösen. Mehr dazu auf den folgenden Seiten.

Anregung: Multiplayer-Variante von «Vier gewinnt»

Fakultative Zusatzaufgabe

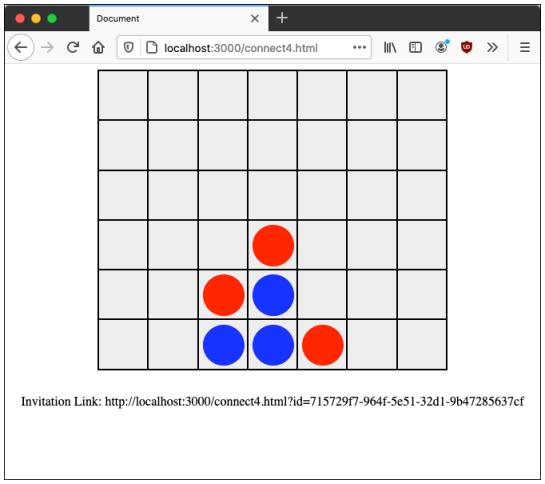
In dieser Aufgabe wird beschrieben, wie Sie «Vier gewinnt» in einer Multiplayer-Variante mit Synchronisation über den Server mit unserer Bibliothek SuiWeb lösen können. Das ist sicher nicht ganz einfach. Da die abgegebene Version «standalone» funktionieren soll, ist diese Variante nicht für die Abgabe gedacht. Zum Vorgehen:

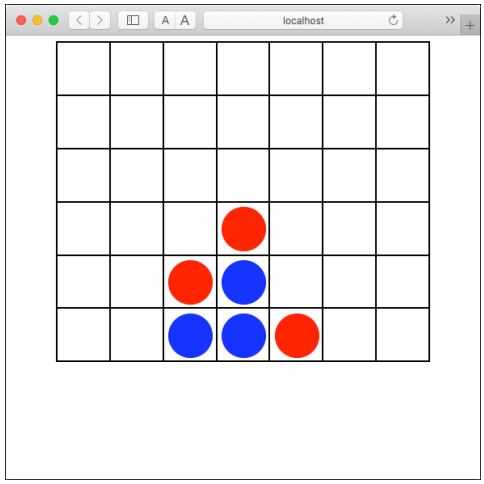
Ziel ist, dass zwei Spieler in eigenen Browsern gegeneinander spielen können. Die Kommunikation läuft dabei über einen Server. Die Idee ist, dass nach dem Aufruf des Spiels ein Link eingeblendet wird, unter dem ein Mitspieler am Spiel teilnehmen kann (s. Bild).



Damit das nicht zu kompliziert wird, gehen wir folgendermassen vor:

- Als Server verwenden wir unseren REST Service mit Express.js aus einem früheren Praktikum. Wir speichern einfach den gesamten Spielzustand (Variable *state*) auf dem Server.
- Unser REST-Service speichert beliebige Datenstrukturen unter einer *id*, welche nach einem POST-Request automatisch generiert und zurückgegeben wird. Unter dieser *id* kann dann in der Folge auf die gespeicherten Daten zugegriffen werden.
- Damit wir keine Probleme wegen *Cross Origin Requests* bekommen, laden wir die HTML- und CSS-Dateien vom gleichen Server, der auch die REST-API zur Verfügung stellt.
- Wir verwenden keine Echtzeitkommunikation mit dem Server. Stattdessen holen wir in regelmässigen Abständen (zum Beispiel alle zwei Sekunden) den aktuellen Zustand vom Server. Diese als *Polling* bezeichnete Technik führt natürlich zu kleinen Verzögerungen beim Abgleich des Zustands, ist aber relativ einfach zu implementieren. Ein paar Hinweise zu alternativen Techniken finden Sie am Ende dieser Praktikumsbeschreibung.
- Da im Zustand auch angegeben ist, ob rot oder blau den nächsten Zug hat, kann die Eingabe anhand dieser Information jeweils für einen der Mitspieler gesperrt werden. Mit einem grauen Hintergrund wird angezeigt, dass aktuell keine Eingabe möglich ist:





Aufgaben:

- Zunächst zum Server. Wie schon im früheren Praktikum verwenden wir *Express* für den Server. Um auch statische Dateien wie *connect4.html* und *styles.css* übertragen zu können ist eine zusätzliche Mittelware nötig.
- Nun zum Client. Implementieren Sie eine Funktion *initGame* für die Initialisierung. Zunächst wird überprüft, ob die URL eine *id* angehängt hat. Ist das der Fall handelt es sich um den eingeladenen Spieler: die eigene Farbe (*session.me*) wird von rot auf blau umgestellt. Falls nicht, muss das Spiel auf dem Server erst noch angelegt werden (*postState*).
- Implementieren Sie die Funktion *postState*: Zunächst wird der komplette Zustand mit POST an den Server geschickt. Dieser antwortet mit einer *id*. Diese wird in einer Variablen gespeichert und zur Generierung einer Einladungs-URL verwendet, die auf der Seite unter dem Spielfeld angezeigt wird (*document.body.appendChild*).

- Ergänzen Sie die Ereignisbehandlung (*attachEventHandler*) so, dass Mausklicks nur ausgewertet werden, wenn man selber am Zug ist.
- Ergänzen Sie die Ausgabe des Spielfelds (*showBoard*) so, dass die Klasse *inactive* gesetzt oder gelöscht wird, je nachdem ob man selber am Zug ist oder nicht. Hinweis: DOM-Knoten haben ein Attribut *classList* mit Methoden *add* und *remove*.
- Implementieren Sie die Methode *getState*. Sie holt den Zustand vom Server und ersetzt damit den bisherigen Zustand. Anschliessend wird das Spielfeld neu ausgegeben. Diese Methode wird in regelmässigen Intervallen aufgerufen und sorgt dafür, dass beide Spieler den gleichen Spielstand angezeigt bekommen.
- Implementieren Sie die Methode *putState*. Sie sendet den aktuellen Zustand an den Server. Ergänzen Sie die Ereignisbehandlung so, dass *putState* immer nach dem Einfügen eines Spielsteins aufgerufen wird.

Nun sollte alles beieinander sein und das Spiel müsste in zwei Browsern spielbar sein. Probieren Sie es aus. Natürlich könnte das Spiel noch in verschiedener Hinsicht verbessert werden:[1]

- Match aus mehreren Spielen und zählen, wer wie oft gewonnen hat
- Spiel aufgeben und neues Spiel starten
- Eingabe von (Spitz-) Namen der Spieler
- Zufällige Auswahl wer den ersten Zug hat
- Liste von Teilnehmern auf dem Server, die nach Spielpartnern suchen
- Animation beim Einfügen von Spielsteinen
- Visuelles Feedback, wenn man in eine Spalte geklickt hat, die bereits voll ist oder wenn man klickt obwohl man nicht am Zug ist

Ergänzende Hinweise

Das regelmässige Abfragen eines Servers, ob neue Informationen vorliegen, wird wie oben beschrieben als **Polling** bezeichnet. Da HTTP-Anfragen vom Client initiiert werden, ist dies zunächst die naheliegende Variante. Nachteile: unnötiger Netzwerkverkehr, unnötige Serveraktivität. Vorteil: einfach zu implementieren.

Eine Variante wird als **Long Polling** bezeichnet: vom Client kommt wie gehabt eine HTTP-Anfrage, der Server wartet aber mit der Antwort, bis neue Informationen vorliegen oder ein vordefiniertes Zeitintervall abgelaufen ist. Eine weitere Alternative sind **WebSockets**: hier wird eine TCP-Verbindung zwischen Client und Server aufgebaut, über die beide Seiten Informationen senden und empfangen können.

Einige Hinweise zur Vertiefung:

- Seite zu WebSockets des Mozilla Developer Networks:
  https://developer.mozilla.org/de/docs/WebSockets
- Socket.io ist eine Bibliothek zur Echtzeitkommunikation, verwendet WebSockets, Long Polling als Fallback, wenn WebSocket-Verbindung nicht zustande kommt
  https://socket.io/docs/
- Faye ist ein „publish-subscribe" Nachrichtensystem
  https://faye.jcoglan.com

[1] Die Ergebnisse können Sie uns gerne zur Verfügung stellen (auch wenn das Semester dann bereits vorbei ist). Vielleicht ergeben sich daraus Anregungen für kleine Projekte oder weitere Praktikumsaufgaben, bzw. verbesserte Musterlösungen.