



3 SZENEN UND TEAMARBEIT

Inhalt

- 3|1. Alles über die Verwendung von Szenen
- 3|2. Teamwork in der App-Entwicklung

3|1 Alles über die Verwendung von Szenen (engl. scene)

Bei unseren bisherigen Anwendungsbereichen hat eine einzige Benutzeroberfläche ausgereicht. Doch wie sieht es aus, wenn der Bildschirm zu klein ist, um alle Funktionen in das Haupt-Fenster (*Main-Scene*) zu bekommen?

Nehmen wir einen einfachen HTML-Editor als Beispiel, welcher über eine Vorschaufunktion verfügt. Man hat also eine Textbox, in der man folgende Zeilen eintragen könnte:

```
<html>
  <head>
    <title>Bsp</title>
  </head>
  <body>
    <table bg-color="red">
      <tr>
        <td>Zeile1 (1)</td>
        <td>Zeile1 (2)</td>
      </tr>
      <tr>
        <td>Zeile2 (1)</td>
        <td>Zeile2 (2)</td>
      </tr>
    </table>
  </body>
</html>
```

Anschließend soll dargestellt werden, wie der Code im Browser angezeigt werden würde. Dazu nutzen wir das HTML-Code-Element, da somit die Umwandlung selbstständig vom Programm durchgeführt wird.

Natürlich kann man das Eingabe- und Ausgabeelement auch nur in eine einzelne Szene verfrachten, jedoch ist es praktischer - zumindest bei der Anzeige - die volle Bildschirmgröße auszunutzen. Zu diesem Zweck erstellen wir einfach eine zweite Szene mit dem Namen **anzeige**. Dies kann man einfach über das Hinzufügen-Menü in Ares durchführen.

Szenen funktionieren wie Funktionen, d.h. man kann ihnen Werte übergeben.

Hier ein Beispiel:

```
//Funktion rechnen
void rechne(a,b)
{
    return(a+b);
}

//Aufruf der Funktion
var ergebnis=rechne(10,5);
```

Beim Aufrufen der Funktion **rechne** werden die Werte *10* und *5* übergeben und können somit in der Funktion verwendet werden. Über **return** lässt sich das Ergebnis zurückgeben

Ruf man nun eine Scene mit `pushscene` auf, so kann man Werte hinter den Namen mit einem Komma getrennt übergeben.

In der **anzeige**-Szene findet man den Wert `ArgFromPusher`. Dieser lässt sich jedoch auch anders benennen oder ergänzen. Weitere Werte müssen mit Komma getrennt werden.

Unser Programm übergibt also den Inhalt der Textbox an die zweite Scene. Dort setzen wir diesen übergebenen Text als Inhalt des HTML-Code-Elements.

So sehen dann die zwei Assistances aus:

Beginnen wir mit der **main-assistant.js**:

```
function MainAssistant(argFromPusher) {
}

MainAssistant.prototype = {
  setup: function() {
    Ares.setupSceneAssistant(this);
  },
  cleanup: function() {
    Ares.cleanupSceneAssistant(this);
  },
};
```

```
button1Tap: function(inSender, event) {
    var inhalt = this.$.textField1.getValue();
    Mojo.Log.error(inhalt);
    this.controller.stageController.pushScene('anzeige',
inhalt);
}
};
```

Und hier haben wir die **anzeige-assistant.js**:

```
function AnzeigeAssistant(argFromPusher) {
    inhalt = argFromPusher;
}

AnzeigeAssistant.prototype = {
    setup: function() {
        Ares.setupSceneAssistant(this);
    },
    cleanup: function() {
        Ares.cleanupSceneAssistant(this);
    },
    activate: function() {
        Mojo.Log.error(inhalt);
        this.$.html1.setContent(inhalt);
    }
};
```

3|2 Teamwork in der App-Entwicklung

Die Entwicklung von Anwendungen kann sehr komplex sein, vor allem bei kommerziellen Programmen. Während Hobby-Entwickler oft – und meist auch gerne – alleine arbeiten, arbeitet man in Entwicklerschmieden häufig im Team.

Auch wir von TamsPalm Dev entwickeln in Teamwork. Dabei ist es besonders wichtig, dass die anderen Teammitglieder wissen, was man selbst denn am Code geändert oder hinzugefügt hat. Gerade wenn man nicht am selben Standort arbeitet und kleine Meetings am Telefon oder über Skype abhält.

Sobald wir ein neues Projekt starten, sammeln wir zuerst alle Ideen und erstellen ein Konzept. Ggf. erstellen wir eine Mind-Map, um Ideen schnell hinzufügen oder ändern zu können.

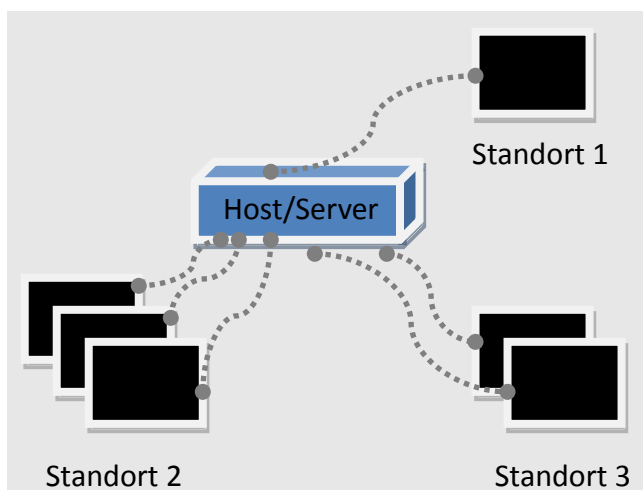
Ist das Konzept ausgearbeitet und die Idee ausgereift, besprechen wir noch einmal den Ablauf. Wir verteilen untereinander Aufgaben, wie beispielsweise die UI-Gestaltung oder die Programmierung verschiedene Programmelemente.

Notwendigkeiten für Teamwork

- ✓ genaueste Dokumentation der Änderungen
- ✓ regelmäßige Meetings
- ✓ klar definierte Aufgabenteilung
- ✓ ...

Ist dieser Schritt erledigt, beginnen wir mit der Entwicklung. Freilich ist es jedem selbst überlassen, wie er im Team arbeiten möchte. Doch ohne ein paar Regeln oder Anweisungen geht gar nichts. Beachten Sie hierfür die Info-Box *Notwendigkeiten für Teamwork*.

Das größte Problem in der Team-Entwicklung besteht darin, dass jeder Entwickler immer an der aktuellsten Version arbeiten muss, damit keine Inkompatibilitäten entstehen können. Nicht nur wenn man im Team arbeitet, sondern auch als „Alleinentwickler“ helfen Versionsverwaltungs-Systeme, wie zum Beispiel **Subversion (SVN)**, **Mecurial** oder **TortoiseHg**. Mit diesen Systemen ist es möglich, alle bis zur Fertigstellung der App entwickelte Revisionen zu verwalten. Wir selbst verwenden übrigens den TortoiseHg-Server, der frei erhältlich ist. Eine Versionsverwaltung funktioniert nach folgendem Prinzip:



Auf dem Server - das kann entweder ein echter Server, ein Client oder sogar nur Space (Dropbox eignet sich da besonders gut) sein – liegt der Source-Code der Anwendung. Auf den Server müssen alle Clients Zugriff haben. Jeder Client besitzt ein Arbeitsverzeichnis bzw. ein Clone von der Source, in dem der Entwickler arbeitet. Hat nun ein Entwickler Änderungen durchgeführt, muss er zuerst beim Server anfragen, ob in der Zwischenzeit Änderungen von anderen Entwicklern hochgeladen wurden. Ist dies der Fall, lädt der Client die Änderungen herunter und gleicht sie mit den eigenen Änderungen ab. Nun kann der Entwickler die Änderungen zum Server senden.

Nehmen wir als Beispiel mal einen Währungsumrechner. Ein Entwickler – nennen wir ihn **A** - erstellt nun eine Einstellungs-Szene und fügt einen *ListSelector* ein, damit der User eine Heimwährung festlegen kann. Sein Kollege – ihn nennen wir **B** - hat in der Zwischenzeit eine Hilfe-Szene erstellt, um dem User einen schnellen Weg zum Support zu bieten. B ist schneller als A fertig und lädt seine Änderungen hoch. Bevor A seine Änderungen hochlädt (= *comitted*), prüft er, ob sein Kollege eine Änderung gemacht hat. Da dies der Fall ist, lädt er die Änderungen seines Kollegen B in sein Arbeitsverzeichnis. Nun beschreibt A in seinem *Commit*, der bei jeder Änderung erstellen werden muss, was genau er hinzugefügt, geändert oder entfernt hat. Ist der Commit erstellt, lädt er ihn hoch. Da A zuvor die Änderungen von B heruntergeladen und mit seinen abgeglichen hat (= *mergen* = zusammenführen), ist auf dem Server nun die aktuellste Version vorhanden. Selbstverständlich kann man Änderungen bzw. Commits jederzeit rückgängig machen, falls Inkompatibilitäten auftreten. So kann jeder Entwickler unabhängig entwickeln und das Projekt kann schnell fertiggestellt werden.

Selbstverständlich kann man eine Versionsverwaltung auch dann verwenden, wenn man alleine arbeitet. So kann man seine Änderungen nämlich genau und einfach dokumentieren und im Fall der Fälle auf eine lauffähige Version zurückschalten.