XML Aufbau

Medieninformatik II

Abenteuerspiel – On the wood way!

Das folgende Dokument beschreibt den strukturierten Aufbau einer XML Datei für A.E.Flash (Adventure Engine Flash). Voraussetzung zum Vertehen ist ein Verständnis für die grundlegende Funktionsweise von XML und dem zugehörigen Syntax.

Eine XML Datei mit einer Spielgeschichte für die „Adventure Engine Flash“ hat IMMER folgenden Aufbau:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<story>
      <config>...</config>
      <locations>...</locations>
</story>
```

Element: `story`

Das Element `story` beherbergt alle spielentscheidenden Elemente, es ist das root Element der XML Datei. Innerhalb dieses Elements befinden sich genau:

- 1 × `config`
- 1 × `locations`

Element: `config`

Wichtig bei dem Aufbau des `config` Elements ist, die **Reihenfolge muss immer die gleiche sein**. Der Aufbau des Elements sieht wie folgt aus :

```
<config>
      <startLocation>location.id</startLocation>
      <errorPickUp>pickUp Fehlermeldung</errorPickUp>
      <errorUse>use Fehlermeldung</errorUse>
      <errorUseWith>use with Fehlermeldung</errorUseWith>
      <errorOpen>open Fehlermeldung</errorOpen>
      <errorClose>close Fehlermeldung</errorClose>
</config>
```

Bei dem Element `startLocation` wird die `id` der `location` eingetragen, bei der das Spiel gestartet wird. Näheres zu Locations, siehe `location`. Die kursiv dargestellten Werte sind beliebig anpassbar

und repräsentieren die Standardfehlermeldungen im Spiel. Wird beispielsweise versucht ein Gegenstand aufzuheben, der nicht aufhebbar ist, wird die Fehlermeldung ausgegeben, die in `errorPickUp` definiert ist.

Element: `locations`

Hierbei handelt es sich um den Container für alle im Spiel existierenden `location` Elemente. Dies sind im weitesten Sinne unterschiedliche Räume oder Umgebungen, weshalb beliebig viele von ihnen existieren können, mindestens jedoch ein Element. Der Aufbau ist relativ simpel, ausführlicher wird es im `location` Abschnitt:

```
<locations>
      <location>...</location>
      ...
</locations>
```

Element: `location`

Da es im Spiel mehrere Orte oder Regionen geben kann, müssen diese selbstverständlich innerhalb der XML Datei deklariert werden. Dies geschieht innerhalb des Container-Elements `locations`. Für jeden eigenständigen Bildschirm bzw. Ort wird eine eigene `location` angelegt. Der Aufbau sieht in etwa wie folgt aus:

```
<location id="id" name="name" background="file.swf">
      <description>
            description
      </description>
      <hitboxes>
            <hitbox>...</hitbox>
      </hitboxes>
      <worlditems>
            <worlditem>...</worlditem>
      </worlditems>
      <events>
            <xyz></xyz>
      </events>
</location>
```

Für jede `location` innerhalb des Spieles müssen die folgenden Attribute/Elemente gesetzt werden:

id : hierbei handelt es sich um einen eindeutigen Bezeichner der `location`, der zur Identifikation dient.

name : ein Name für die `location`.

background : das Hintergrundbild für die `location`. Muss entweder eine JPEG oder eine SWF Datei sein.

description: eine Beschreibung für die `location`. Diese wird in der MessageBox angezeigt.

Weiterhin müssen in jedem `location` Element die Elemente `hitboxes`, `worlditems` und `events` vorhanden sein, auch dann, wenn diese keine Kinderelemente aufweisen. Auf die `hitboxes`, `worlditems` und `events` wird im folgenden näher eingegangen.Wie der Aufbau zeigt, gibt es also für jeden Ort (`location`) unterschiedliche Gegenstände (`worlditems`) und Regionen (`hitboxes`) und dazugehörige Aktionen (`events`).

Element: `hitbox`

Innerhalb des Spieles kann es mehrere Regionen geben, die jedoch keinen Gegenstand im spieltechnischen Sinne darstellen, beispielsweise einen Lichtschalter. Ein Lichtschalter stellt zwar physikalisch gesehen einen Gegenstand dar, aber dieser kann normalerweise nicht aufgenommen werden oder mit anderen Gegenständen zusammen genutzt werden. Um dennoch Interaktion zu gewährleisten, lässt sich eine `hitbox` verwenden.

HitBoxen kommen immer nur in dem Container-Element `hitboxes` vor. Hier können beliebig viele definiert werden. Prinzipieller Aufbau von `hitbox` Elementen:

```
<hitboxes>
      <hitbox id="id" x="#" y="#" width="#" height="#" visible="1/0"
       active="1/0">
            <name>name</name>
            <description>description</description>
      </hitbox>
      ...
</hitboxes>
```

Für jede `hitbox` müssen folgende Attribute/Elemente gesetzt werden:

id : hierbei handelt es sich um einen eindeutigen Bezeichner der `hitbox`, der zur Identifikation dient.

x : die X-Koordinate der Position der `hitbox` auf dem Bildschirm.

y : die Y-Koordinate der Position der `hitbox` auf dem Bildschirm.

width : die Breite der `hitbox` auf dem Bildschirm.

height : die Höhe der `hitbox` auf dem Bildschirm.

visible : Sichtbarkeit der `hitbox` auf dem Bildschirm; kann nachträglich geändert werden, siehe dazu `events`.

active : Interaktionsfähigkeit der `hitbox` auf dem Bildschirm; kann nachträglich geändert werden, siehe dazu `events`.

name : ein Name für die `hitbox`. Dieser wird beim Zeigen auf die `hitbox` am oberen Bildschirmrand ausgegeben.

description: eine Beschreibung für die `hitbox`. Diese wird in der MessageBox angezeigt.

Element: `worlditem`

Neben den Regionen, die durch eine `hitbox` definiert werden, gibt es im Spiel noch die Möglichkeit Gegenstände zu platzieren, die später vom Spieler aufgenommen und modifiziert bzw. Manipuliert werden können. Genau hierfür gibt es mehrere `worlditems`. Diese werden immer nur im Container-Element `worlditems` definiert. Prinzipieller Aufbau von `worlditem` Elementen:

```
<worlditems>
      <worlditem id="id" x="#" y="#" picture="file.swf"
       inventory="file.swf" visible="1/0" take="1/0" active="1/0">
            <name>name</name>
            <description>description</description>
      </worlditem>
      ...
</worlditems>
```

Für jedes `worlditem` müssen folgende Attribute/Elemente gesetzt werden:

id : hierbei handelt es sich um einen eindeutigen Bezeichner des `worlditem`, der zur Identifikation dient.

x : die X-Koordinate der Position des `worlditem` auf dem Bildschirm.

y : die Y-Koordinate der Position des `worlditem` auf dem Bildschirm.

picture : Grafik des `worlditem` auf dem Bildschirm; muss eine JPG oder SWF Datei sein.

inventory : Grafik des `worlditem` im Inventar; muss eine JPG oder SWF Datei sein. Die Größe der Grafik muss hierbei 80x80 Pixel betragen.

visible : Sichtbarkeit des `worlditem` auf dem Bildschirm; kann nachträglich geändert werden, siehe dazu `events`.

active : Interaktionsfähigkeit des `worlditem` auf dem Bildschirm; kann nachträglich geändert werden, siehe dazu `events`.

take : Aufnahmemöglichkeit des `worlditem` auf dem Bildschirm

name : ein Name für die `hitbox`. Dieser wird beim Zeigen auf die `hitbox` am oberen Bildschirmrand ausgegeben.

description: eine Beschreibung für die `hitbox`. Diese wird in der MessageBox angezeigt.

Element: `events`

Um eine wirkliche, modulare Konfiguration des Spiels zu gewährleisten, ist es notwendig, dass unterschiedlichste Aktionen zu unterschiedlichen Zeitpunkten für diverse Gegenstände/Regionen ausgeführt werden. Speziell hierfür gibt es die `events`. Um ein `event` aufzurufen, bedarf es der Deklaration des Aufrufs innerhalb eines `worlditems` oder einer `hitbox`. Beispiel für eine `hitbox`:

```
<hitbox id="id" ... eventClick="id1" ...>...</hitbox>
```

oder aber in einem `worlditem`:

```
<worlditem id="id" ... eventUse="id2" eventClick="id3">...</worlditem>
```

Insgesamt gibt es 6 verschiedene Möglichkeiten, zu denen ein `event` auftreten kann:

eventClick : wird aufgerufen, wenn auf ein Gegenstand/Region geklickt wird.

eventUse : wird aufgerufen, wenn der Befehl „use“ auf einen Gegenstand/Region angewandt wird.

eventUseWith : wird aufgerufen, wenn der Befehl „use with“ auf einen Gegenstand/Region angewandt wird.

eventPickUp : wird aufgerufen, wenn der Befehl „pick up“ auf einen Gegenstand/Region angewandt wird.

eventOpen : wird aufgerufen, wenn der Befehl „open“ auf einen Gegenstand/Region angewandt wird.

eventClose : wird aufgerufen, wenn der Befehl „close“ auf einen Gegenstand/Region angewandt wird.

eventLookAt : wird aufgerufen, wenn der Befehl „look at“ auf einen Gegenstand/Region angewandt wird.

Wie man dem obigen Beispiel entnehmen kann, entspricht der Aufruf eines `events` einem Attribut

des Elements `hitbox` oder `worlditem`. Der Wert des Attributs entspricht dabei der `id` des aufzurufenden `events`, welches innerhalb des `events` Elements für jede `location` deklariert wird:

```
<events>
      <id1>
            <goto>location.id</goto>
      </id1>
      <id2>
            <playSound item="file.swf" />
            <print>message</print>
      </id2>
      <id3>
            <playSound item="file.swf" />
            <print>message</print>
      </id3>
</events>
```

Wie man an dem obigen Beispiel erkennen kann, sind events wiederum nur XML Elemente. Wird beispielsweise bei einer `hitbox` ein Click Ereignis abgefangen, dass auf das Event `gotoOther` verweist, sieht dies folgendermaßen aus:

```
<hitbox id="id" ... eventClick="gotoOther" ...>...</hitbox>
```

Innerhalb der jeweiligen `location` wird nun im `events` Zweig nach einem `event` gesucht, dessen Knotenbezeichnung mit der aufzurufenden `id` übereinstimmt. In diesem Fall also z.B.:

```
<events>
      <gotoOther>eventActions</gotoOther>
</events>
```

Das gleiche Prinzip gilt selbstverständlich auch für `worlditems`. Innerhalb einer event Deklaration kann es nun verschiedene, durchzuführende Aktionen geben, wie beispielsweise das Entfernen eines Gegenstandes aus dem Inventar, das Bewegen zu einem anderen Ort usw. Hierzu siehe auch `eventActions`.

eventActions:

Zum Manipulieren der Spielumgebung, sei es das verändern des Inventars oder generell von Gegenständen und/oder Regionen muss es innerhalb von events Aktionen geben, die genau diesen Anspruch erfüllen. Nachfolgend alle Aktionen, die im Spiel möglich sind. Wichtig ist hierbei auch die Reihenfolge, in der die Aktionen notiert werden. Wenn beispielsweise für einen Gegenstand die Aktion „Gegenstand löschen“ vor der Aktion „Gegenstand aufnehmen“ ausgeführt wird, verursacht dies natürlich einen Fehler oder ein Ausführen der Aktionen findet nicht statt.

eventAction: `addInventoryItem`

Syntax:

```
<addInventoryItem location="location.id">worlditem.id</addInventoryItem>
```

Dieser Befehl fügt den per `worlditem.id` angegebenen Gegenstand zum Inventar hinzu. Zusätzlich zur `id` des `worlditems` muss noch die `location` angegeben werden, in der das `worlditem` definiert wurde. Dies geschieht durch die `id` der `location`.

eventAction: `goto`

Syntax:

```
<goto>location.id</goto>
```

Dieser Befehl sorgt dafür, dass der Spieler den Ort wechselt. Der neue Ort, bzw. die neue `location` wird durch die id der `location` spezifiziert.

eventAction: `playSound`

Syntax:

```
<playSound>file.swf</playSound>
```

Dieser Befehl sorgt dafür, dass ein Ton abgespielt wird. Dieser Ton muss als SWF Datei spezifiziert werden. In der SWF Datei muss dazu lediglich eine WAVE oder MP3 Datei eingebunden werden.

eventAction: `print`

Syntax:

```
<print>message</print>
```

Dieser Befehl sorgt dafür, dass in der MessageBox eine Nachricht ausgegeben wird. Die Nachricht selber (`message`) wird als Zeichenkette angegeben.

eventAction: `removeHitBox`

Syntax:

```
<removeHitBox location="location.id">hitbox.id</removeHitBox>
```

Dieser Befehl sorgt dafür, dass eine Region (`hitbox`) aus dem kompletten Spiel entfernt wird. Dazu muss lediglich die eindeutige `id` der Region (`hitbox`) angegeben werden und die dazugehörige `location` (per `id`), in der die zu löschende Region definiert ist.

eventAction: `removeInventoryItem`

Syntax:

```
<removeInventoryItem>worlditem.id</removeInventoryItem>
```

Dieser Befehl sorgt dafür, dass ein Gegenstand aus dem Inventar des Spielers entfernt wird. Dazu muss lediglich die eindeutige `id` des Gegenstandes (`worlditem`) angegeben werden.

eventAction: `removeWorldItem`

Syntax:

```
<removeWorldItem location="location.id">worlditem.id</removeWorldItem>
```

Dieser Befehl sorgt dafür, dass ein Gegenstand aus dem kompletten Spiel entfernt wird. Dazu muss lediglich die eindeutige `id` des Gegenstandes (`worlditem`) angegeben werden und die dazugehörige `location` (per `id`), in der der zu löschende Gegenstand definiert ist.

eventAction: `replaceInventoryItem`

Syntax:

```
<replaceInventoryItem location="(new)location.id"
oldLocation="(old)location.id" oldItem="(old)worlditem.id">
            (new)worlditem.id
</replaceInventoryItem>
```

Dieser Befehl sorgt dafür, dass ein Gegenstand aus dem Inventar durch einen anderen ersetzt wird. Dafür muss die `id` des alten Gegenstandes (`(old)worlditem.id`) und die `location id` des zu ersetzenden Gegenstandes (`(old)location.id`) angegeben werden. Zusätzlich wird die `id` des neuen Gegenstandes (`(new)worlditem.id`) und die `location id` des neuen Gegenstandes (`(new)location.id`) angegeben.

eventAction: `requireHitBoxActive`

Syntax:

```
<requireHitBoxActive location="location.id"
falseMSG="message">hitbox.id</requireHitBoxActive>
```

Dieser Befehl ermöglicht es zu prüfen, ob die Eigenschaft active einer `hitbox` auf 1 gesetzt ist. Hiermit kann man also beispielsweise abfragen, ob eine Tür aufgeschlossen ist. Wichtig hierbei ist zu wissen, dass die Verarbeitung weiterer Aktionen abgebrochen wird (sollten sie nach dieser Aktion folgen) sobald die Eigenschaft `active` der `hitbox` ungleich 1 ist. Beispiel:

```
<requireHitBoxActive location="location.id" falseMSG="hbox nicht
aktiv!">hitbox.id</requireHitBoxActive>
<print>hallo</print>
```

Ist die `hitbox.id` nicht aktiv, wird der Befehl `print` nicht mehr ausgeführt. Das Attribut `location` verweist hierbei per `id` auf die `location`, in der die `hitbox` definiert ist. Das Attribut `falseMSG` wird dann in der MessageBox ausgegeben, wenn die Eigenschaft `active` der `hitbox` ungleich 1 ist.

eventAction: `requireHitBoxInactive`

Syntax:

```
<requireHitBoxInactive location="location.id"
falseMSG="message">hitbox.id</requireHitBoxInactive>
```

Dieser Befehl ermöglicht es zu prüfen, ob die Eigenschaft `active` einer `hitbox` auf 0 gesetzt ist. Es ist also das genaue Gegenstück zu der Aktion `requireHitBoxActive`.

eventAction: `requireInventoryItem`

Syntax:

```
<requireInventoryItem falseMSG="message">worlitem.id</requireInventoryItem>
```

Dieser Befehl ermöglicht es zu prüfen, ob sich ein Gegenstand im Inventar des Spielers befindet. Der zu überprüfende Gegenstand (`worlditem`) wird hierbei per `id` spezifiziert. Hier gilt selbiges wie bei allen `require` Aktionen: ist die Bedingung nicht erfüllt, wird die weitere Verarbeitung von Aktionen abgebrochen.

eventAction: `requireInventoryItemUseWith`

Syntax:

```
<requireInventoryItemUseWith
falseMSG="message">worlditem.id</requireInventoryItemUseWith>
```

Diese Aktion ermöglicht es festzulegen, welcher Gegenstand mit welchem zusammen funktioniert. Der funktionierende Gegenstand (`worlditem`) wird hierbei per `id` angegeben.

Beispiel für die Anwendung dieser Aktion:

```
...
<location id="halle">
      <hitboxes>
            <hitbox id="tuer" ... eventUseWith="oeffneTuer">...</hitbox>
      </hitboxes>
      <worlditems>...</worlditems>
      <events>
            <oeffneTuer>
                  <requireInventoryItemUseWith falseMSG="falscher
Schluessel!">key</requireInventoryItemUseWith>
            </oeffneTuer>
      </events>
</location>
...
```

In diesem Beispiel wird davon ausgegangen, dass der Spieler einen Gegenstand (`worlditem`) mit der `id` = `key` im Inventar hat. Zieht er diesen nun auf die Tuer, wird das `eventUseWith` Event

ausgelöst, woraufhin der `oeffneTuer` Teilbaum abgearbeitet wird. Hier wird nun geprüft, ob der aktuell verwendete Gegenstand (der auf die Tür gezogen wird) die `id` = `key` besitzt. Ist dies nicht der Fall, wird die `falseMSG` an die MessageBox ausgegeben. Hier gilt selbiges wie bei allen `require` Aktionen: ist die Bedingung nicht erfüllt, wird die weitere Verarbeitung von Aktionen abgebrochen.

eventAction: `requireWorldItemActive`

Syntax:

```
<requireWorldItemActive location="location.id"
falseMSG="message">worlditem.id</requireWorldItemActive>
```

Dieser Befehl ermöglicht es zu prüfen, ob die Eigenschaft `active` eines `worlditem` auf 1 gesetzt ist. Hier gilt selbiges wie bei allen `require` Aktionen: ist die Bedingung nicht erfüllt, wird die weitere Verarbeitung von Aktionen abgebrochen. Das Attribut `location` verweist hierbei per `id` auf die `location`, in der das `worlditem` definiert ist. Das Attribut `falseMSG` wird dann in der MessageBox ausgegeben, wenn die Eigenschaft `active` des `worlditem` ungleich 1 ist.

eventAction: `requireWorldItemInactive`

Syntax:

```
<requireWorldItemInactive location="location.id"
falseMSG="message">worlditem.id</requireWorldItemInactive>
```

Dieser Befehl ermöglicht es zu prüfen, ob die Eigenschaft `active` eines `worlditem` auf 0 gesetzt ist. Es ist also das genaue Gegenstück zu der Aktion `requireWorldItemActive`.

eventAction: `requireWorldItemClose`

Syntax:

```
<requireWorldItemClose location="location.id"
falseMSG="message">worlditem.id</requireWorldItemClose>
```

Dieser Befehl ermöglicht es zu prüfen, ob die Eigenschaft `open` eines `worlditem` auf 0 gesetzt ist. Hier gilt selbiges wie bei allen `require` Aktionen: ist die Bedingung nicht erfüllt, wird die weitere Verarbeitung von Aktionen abgebrochen. Das Attribut `location` verweist hierbei per `id` auf die `location`, in der das `worlditem` definiert ist. Das Attribut `falseMSG` wird dann in der MessageBox ausgegeben, wenn die Eigenschaft `open` des `worlditem` ungleich 0 ist.

eventAction: `requireWorldItemOpen`

Syntax:

```
<requireWorldItemOpen location="location.id"
falseMSG="message">worlditem.id</requireWorldItemOpen>
```

Dieser Befehl ermöglicht es zu prüfen, ob die Eigenschaft `open` eines `worlditem` auf 1 gesetzt ist. Es ist also das genaue Gegenstück zu der Aktion `requireWorldItemActive`.

eventAction: `toggleHitBoxActive`

Syntax:

```
<toggleHitBoxActive location="location.id"
newValue="1/0">hitbox.id</toggleHitBoxActive>
```

Dieser Befehl ermöglicht es die Eigenschaft `active` einer `hitbox` zu setzen. Dazu muss die `hitbox` per `id` und die `location`, in der sich die `hitbox` befindet spezifiziert werden (ebenfalls per `id`). Die Eigenschaft `active` kann dabei auf zweierlei weise gesetzt werden. Entweder gibt man per Attribut `newValue` den neuen Wert an (`1` oder `0`) oder man invertiert den vorhandenen Wert, indem man den Parameter weglässt. War die Eigenschaft vor dem Aufruf also `1`, so ist sie danach `0` und umgekehrt.

eventAction: `toggleHitBoxVisible`

Syntax:

```
<toggleHitBoxVisible location="location.id"
newValue="1/0">hitbox.id</toggleHitBoxVisible>
```

Dieser Befehl ermöglicht es die Eigenschaft `visible` einer `hitbox` zu setzen. Dazu muss die `hitbox` per `id` und die `location`, in der sich die `hitbox` befindet spezifiziert werden (ebenfalls per `id`). Die Eigenschaft `visible` kann dabei auf zweierlei weise gesetzt werden. Entweder gibt man per Attribut `newValue` den neuen Wert an (`1` oder `0`) oder man invertiert den vorhandenen Wert, indem man den Parameter weglässt. War die Eigenschaft vor dem Aufruf also `1`, so ist sie danach `0` und umgekehrt.

eventAction: `toggleWorldItemActive`

Syntax:

```
<toggleWorldItemActive location="location.id"
newValue="1/0">worlditem.id</toggleWorldItemActive>
```

Dieser Befehl ermöglicht es die Eigenschaft `active` eines `worlditem` zu setzen. Dazu muss das `worlditem` per `id` und die `location`, in der sich das `worlditem` befindet spezifiziert werden (ebenfalls per `id`). Die Eigenschaft `active` kann dabei auf zweierlei weise gesetzt werden. Entweder gibt man per Attribut `newValue` den neuen Wert an (`1` oder `0`) oder man invertiert den vorhandenen Wert, indem man den Parameter weglässt. War die Eigenschaft vor dem Aufruf also `1`, so ist sie danach `0` und umgekehrt.

eventAction: `toggleWorldItemOpen`

Syntax:

```
<toggleWorldItemOpen location="location.id"
newValue="1/0">worlditem.id</toggleWorldItemOpen>
```

Dieser Befehl ermöglicht es die Eigenschaft `open` eines `worlditem` zu setzen. Dazu muss das `worlditem` per `id` und die `location`, in der sich das `worlditem` befindet spezifiziert werden (ebenfalls per `id`). Die Eigenschaft `open` kann dabei auf zweierlei weise gesetzt werden. Entweder gibt man per Attribut `newValue` den neuen Wert an (`1` oder `0`) oder man invertiert den vorhandenen Wert, indem man den Parameter weglässt. War die Eigenschaft vor dem Aufruf also `1`, so ist sie danach `0` und umgekehrt.

eventAction: `toggleWorldItemVisible`

Syntax:

```
<toggleWorldItemVisible location="location.id"
newValue="1/0">worlditem.id</toggleWorldItemVisible>
```

Dieser Befehl ermöglicht es die Eigenschaft `visible` eines `worlditem` zu setzen. Dazu muss das `worlditem` per `id` und die `location`, in der sich das `worlditem` befindet spezifiziert werden (ebenfalls per `id`). Die Eigenschaft `visible` kann dabei auf zweierlei weise gesetzt werden. Entweder gibt man per Attribut `newValue` den neuen Wert an (`1` oder `0`) oder man invertiert den vorhandenen Wert, indem man den Parameter weglässt. War die Eigenschaft vor dem Aufruf also `1`, so ist sie danach `0` und umgekehrt.