

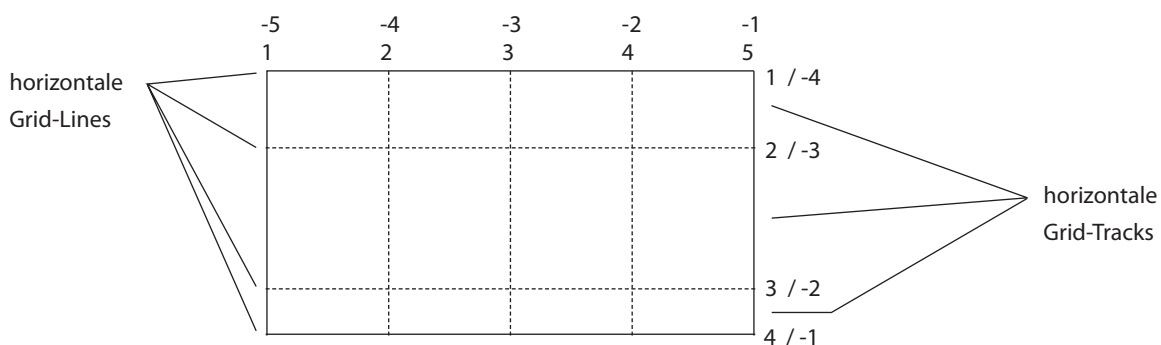
Responsive Layouts mit CSS-Grid

CSS Grid ist ein zweidimensionales Layoutsystem, das in einem HTML-Blockelement ein Tabellenraster aufzieht. Es funktioniert wie CSS-Flexbox mit einem Elternelement und ein oder mehreren Kindelementen. Alle Grid-Elemente müssen Blockelemente sein.

Es ist möglich, dass ein Grid-Element als Flex-Container bestimmt wird und somit die beiden Layouttechnologien Flexbox und Grid sinnvoll miteinander kombiniert werden.

1. Grundbegriffe

- » **Grid:** beschreibt die Technologie im Allgemeinen
- » **Grid-Container:** Ein Elternelement, welches als Grid-Container bestimmt wird. In der Regel enthält ein Grid-Container mehrere Kindelemente, mit denen ein Raster gebildet wird. Wenn man für die komplette Webseite ein Layoutraster benötigt, verwendet man das Element body als Grid-Container. Die CSS-Regel lautet: `display: grid;`
- » **Grid-Line:** Die Rasterlinien sind grundsätzlich nummeriert beginnend mit 1 links oben oder mit -1 rechts unten.
- » **Grid-Tracks:** bezeichnet eine Reihe oder Spalte im Raster, also den Zwischenraum zwischen den Grid-Lines.
- » **Grid-Items** und **Grid-Area:** Die Rasterlemente in einem CSS-Grid, die durch die horizontalen und vertikalen Grid-Lines entstehen. Die Rasterelemente können einzelne Felder sein oder sich als Grid-Area (Rasterbereich) über mehrere Felder erstrecken.
- » **Grid-Template:** Mit Templates wird das Raster grundsätzlich definiert. Mit „Rows“ werden die Zeilen bestimmt, mit „Columns“ die Spalten.



Nummerierung eines CSS Grids.

Die Benennung der Gridlines ist möglich, jedoch nicht notwendig und wird daher im weiteren Verlauf nicht weiter vertieft.

Erstellung mit CSS

Grundsätzlich wird das Raster im Elternelement definiert. Das bedeutet, dass die Anzahl der Zeilen und Spalten im Grid-Container bestimmt sein müssen. Die Größe und Positionierung der Kindelemente ist eine Eigenschaft des jeweiligen Kindelements. Dabei ist es möglich, den Start- und Endpunkt so zu wählen, dass das Kindelement über mehrere Spalten und Zeilen reicht.

Die eigentliche Herausforderung stellt dabei die Größenangabe der Felder mit absoluten oder relativen Werten dar, damit sich das Grid an die Gegebenheiten der Viewports anpassen kann.

2. Das Raster im Grid-Container anlegen

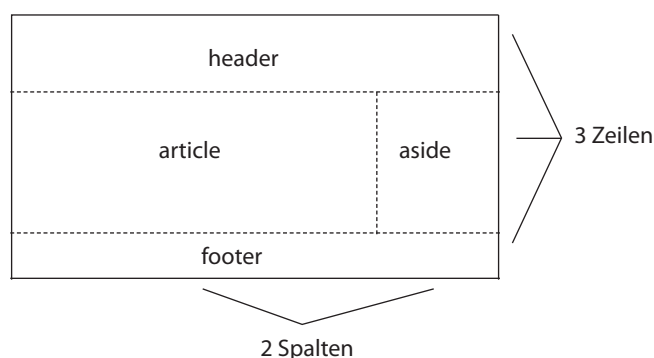
Die Bestimmung des Rasters wird mit der Regel `display: grid;` eingeleitet. Danach erfolgt die Definition der Zeilen und Spalten. Da sich das Layout in der Regel über den ganzen Viewport erstreckt, macht es in Sinn, das HTML-Element `body` als Grid-Container zu nehmen. Grundsätzlich kann jedes Blockelement als Grid-Container verwendet werden.

Wichtig: Bevor Sie anfangen, ein Grid-Raster zu bestimmen, sollten Sie genau wissen, wie Ihr Raster aussehen soll. Wieviele Zeilen und Spalten benötigen Sie? Welche Grid-Tracks haben eine feste oder variable Größe? Im ersten Beispiel wird ein einfaches fluides Raster vorgestellt:

HTML-Code

```
01. <body>
02   <header> </header>
03   <main> </main>
04   <aside> </aside>
05   <footer> </footer>
06 </body>
```

schematische Darstellung



CSS-Code für die Erstellung eines Rasters mit drei Zeilen und zwei Spalten. Jeder einzelne Wert steht für eine Zeile bzw. Spalte und wird mit einem Leerzeichen getrennt:

```
01. body{
02   display: grid;
03   grid-template-rows: 200px 1fr 100px;
04   grid-template-columns: 80% 20%;
05 }
```

Sie können die Größe der Spalten und Zeilen mit absoluten Werten (z. B. Pixel) und relativen Werten (z. B. Prozent, fr) angeben. Damit legen Sie fest, welche Grid-Tracks eine feste Breite bzw. Höhe haben oder sich variabel an den Viewport anpassen. Der relative Wert „auto“ bedeutet „so groß wie der Inhalt“.

Nachdem im Grid-Container ein Raster definiert wurde, können Sie dem Grid-Container weitere Eigenschaften zuweisen. Die wichtigsten Eigenschaften im Überblick:

Relative Größenangaben für responsive Layouts

vh = Viewport-Höhe
vw = Viewport-Breite
fr = Bruchteil des verfügbaren Platzes

Eigenschaft

`grid-template-columns`
`grid-template-rows`
`grid-template-areas`
`grid-column-gap`, `grid-row-gap`
`grid-auto-flow`
`justify-items`
`align-items`

Beschreibung

relativer oder absoluter Wert je Spalte
relativer oder absoluter Wert je Zeile
Benennung der Rasterfelder
Breite der Grid-Lines
Ausrichtung der Befüllung (horizontal oder vertikal)
Grid-Items an der horizontalen Achse ausrichten
Grid-Items an der vertikalen Achse ausrichten

Rasterfelder und Rasterbereiche

Sobald man das Raster angelegt hat, werden die Grid-Items im Raster positioniert und dimensioniert. Dafür kann man ein Grid-Item von Rasterlinie zu Rasterlinie erstrecken lassen oder einen Rasterbereich zuweisen. Ein Rasterbereich erhält im Grid-Container einen eigenen Namen und erstreckt sich über ein oder mehrere Rasterfelder. Dabei muss jedes Rasterfeld genannt werden, Rasterfelder ohne Namen werden mit einem „.“ (Punkt) gekennzeichnet. Rasterfelder, die zu einem Rasterbereich zusammengefasst werden sollen, bekommen jeweils den gleichen Namen.

Die Vorgehensweise mit Rasterbereichen ermöglicht eine nachvollziehbare Darstellung des Rasters im Code.

In unserem einfachen fluiden Raster werden alle Rasterfelder benannt und dem CSS-Code die Eigenschaft `grid-template-areas` hinzugefügt:

```
01. body{
02     display: grid;
03     grid-template-rows: 200px 1fr 100px;
04     grid-template-columns: 80% 20%;
05     grid-template-areas:
06     "header header"
      "main aside"
      "footer footer";
07 }
```

3. Kindelemente: Grid-Items

Grid-Items können im Raster anhand von Grid-Lines oder Grid-Tracks positioniert werden.

Noch einfacher und nachvollziehbarer ist die Positionierung anhand von Rasterbereichen. Die Zuweisung zu einem Rasterbereich erfolgt mit der Eigenschaft `grid-area` und erhält als Wert die Bezeichnung der areas aus dem Grid-Container. Wenn mehrere Rasterfelder die Bezeichnung tragen, erstreckt sich das Grid-Item über mehrere Rasterfelder.

Die Kindelemente unseres einfachen fluiden Rasters werden im Raster positioniert und gleichzeitig dimensioniert:

CSS-Code für Grid-Items

```
08. header{
09     grid-area: header;
10 }

11 main{
12     grid-area: main;
13 }

14 aside{
15     grid-area: aside;
16 }

17 footer{
18     grid-area: footer;
19 }
```

Kindelemente im Elternelement ausrichten

In den bisherigen Beispielen erstrecken sich die Kindelemente immer über die volle Fläche eines Rasterfeldes, obwohl vielleicht weniger Inhalt enthalten ist. Sowohl für das Elternelement als auch für die einzelnen Kindelemente kann dieses Verhalten verändert werden.

Ausrichtung im Elternelement bestimmen:

Die Befehle für das Eltern-Element lauten `justify-items` für das horizontale Verhalten, und `align-items` für das vertikale Verhalten. Es stehen jeweils die Werte `start`, `end`, `center` und `stretch` zur Auswahl. `stretch` ist der Standardwert und selbstverständlich können `justify-items` und `align-items` kombiniert werden.

Individuelle Ausrichtung der Kindelemente

Wenn lediglich die Ausrichtung in einem einzelnen Rasterfeld verändert werden soll, stehen dafür die Befehle `align-self` und `justify-self` zur Verfügung. Auch für diese Eigenschaften existieren die Werte `start`, `end`, `center` und `stretch`. Beispiel:

```
footer {  
    align-self: center;  
}
```

Zusammenfassung

Der Grid-Container wird in einem HTML-Blockelement definiert – häufig schon direkt im `body` – und ist das Elternelement für ein oder mehrere Kindelemente, die sogenannten Grid-Items.

Die Vorgehensweise mit Rasterbereichen ermöglicht eine nachvollziehbare Schreibweise in der Codeansicht. Der fertige Code unseres einfachen fluiden Rasters verbirgt sich im QR-Code (nachdem die Grid-Items noch jeweils eine andere Hintergrundfarbe erhalten haben, damit man sie erkennen kann). Außerdem hat `body` noch eine Mindesthöhe von 100vh (100% Viewport-Höhe) erhalten, damit sich der Main-Bereich über die ganze Viewport-Höhe erstreckt. Der Footer bleibt damit immer am unteren Viewport-Rand und verhält sich wie ein „Sticky-Footer“.



Das „einfache fluide Raster“ als
komplette HTML-Datei

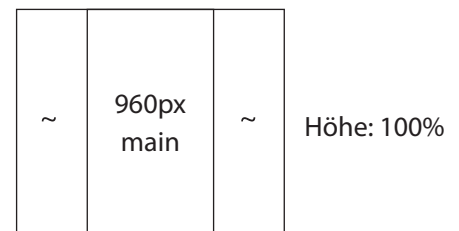
Übungsaufgabe Grid (ohne Media Queries)

1. Grid-Spalten

Gegeben ist die HTML-Datei „grid01.html“ mit einem Bereich `main`.

Erstellen Sie ein dreispaltiges Grid, dessen

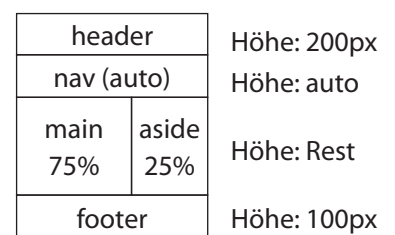
- » mittlere Spalte 960px breit ist und dessen
 - » äußere Spalten gleich breit sind und die restliche Fläche ausfüllen.
- Ergänzen Sie dafür den CSS-Code entsprechend.



2.1 Grid-Spalten und -Zeilen

Gegeben ist die HTML-Datei „grid02.html“ mit den Bereichen `header`, `nav`, `main`, `aside` und `footer`.

Ergänzen Sie den CSS-Code, um das nebenstehende Schema eines Weblayouts darzustellen.

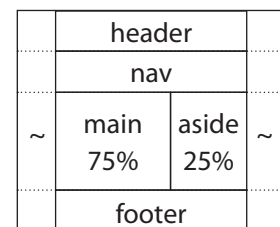


Gesamtbreite: 100%

2.2 Grid zentrieren

Verwenden Sie die fertige HTML-Datei aus Übungsaufgabe 2.1.

Ergänzen Sie das Grid um zwei weitere äußere Spalten, die den restlichen Platz gleichmäßig aufteilen. Die Gesamtbreite von `main` und `aside` beträgt 960px.



Gesamtbreite: 960px

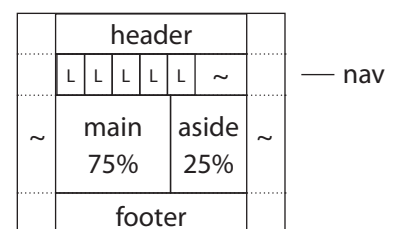
2.3 Grid und Flexbox

Ersetzen Sie im Bereich `nav` das Wort „Nav“ durch eine Navigationsleiste mit fünf Link-Items. Verwenden Sie dafür eine unsortierte Liste. Als Link nehmen Sie „#“. Beispiel:

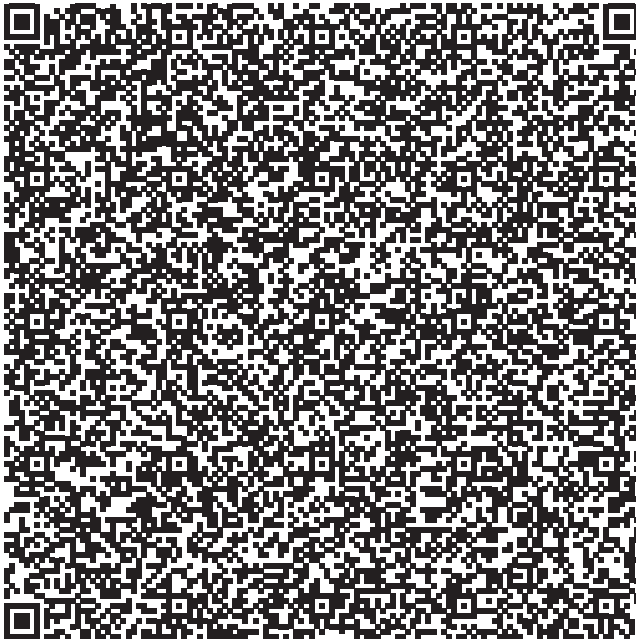
```
<ul>
<li> <a href="#">Link</a> </li>
<li> <a href="#">Link</a> </li> </ul>
```

Definieren Sie die Navigationsleiste als Flexbox, damit die Link-Items nebeneinander stehen und innerhalb von `nav` linksbündig positioniert sind.

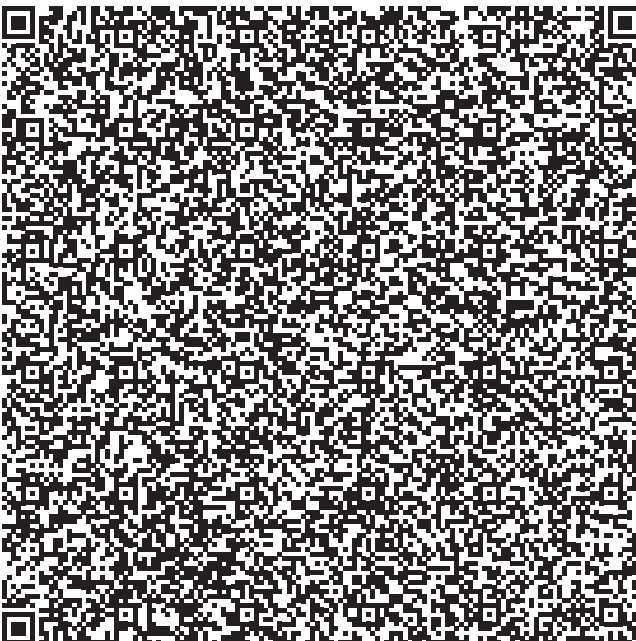
Ergänzen Sie den CSS-Code mit dem CSS-Anweisungen aus der Datei „grid23-styles.css“.



Anhang 1 – Übungsdateien



grid10.html



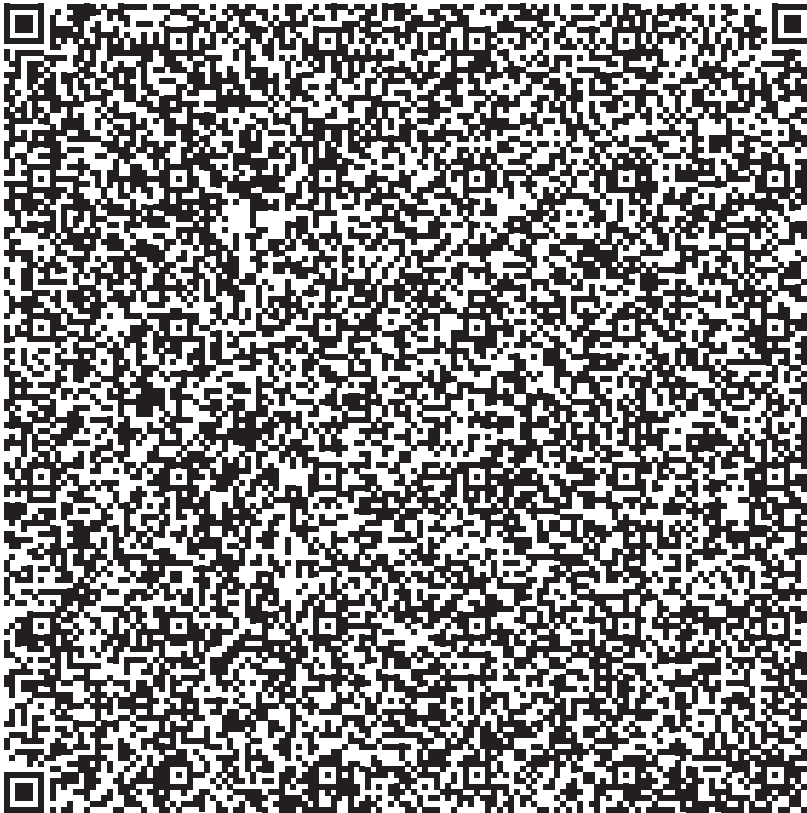
grid02.html

Anhang 1 – Übungsdateien

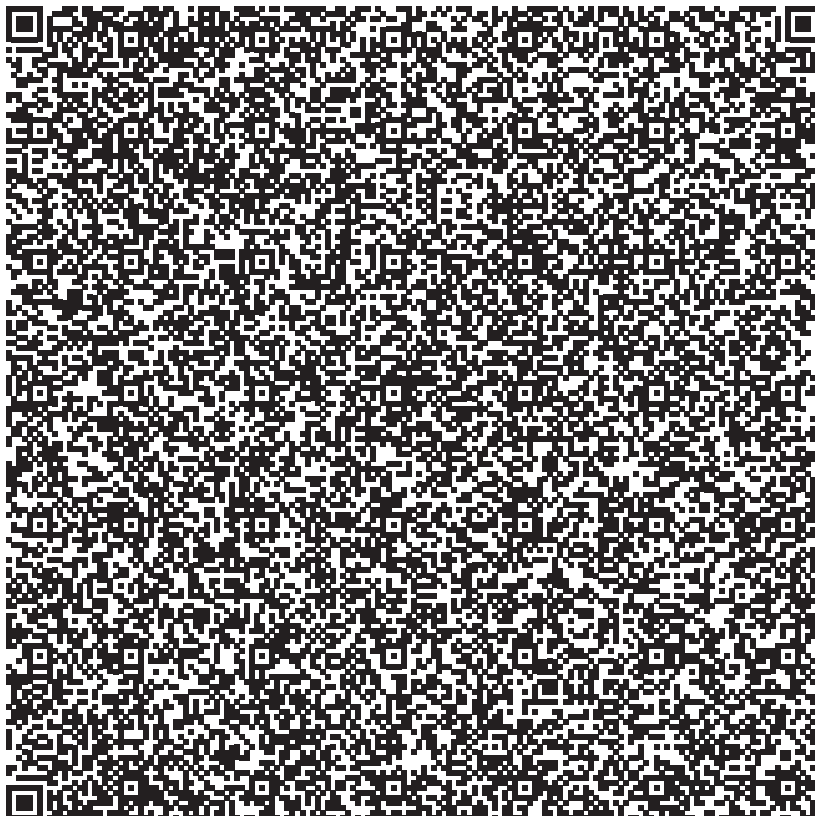


`grid23-styles.css`

Anhang 2 – Lösungsvorschläge

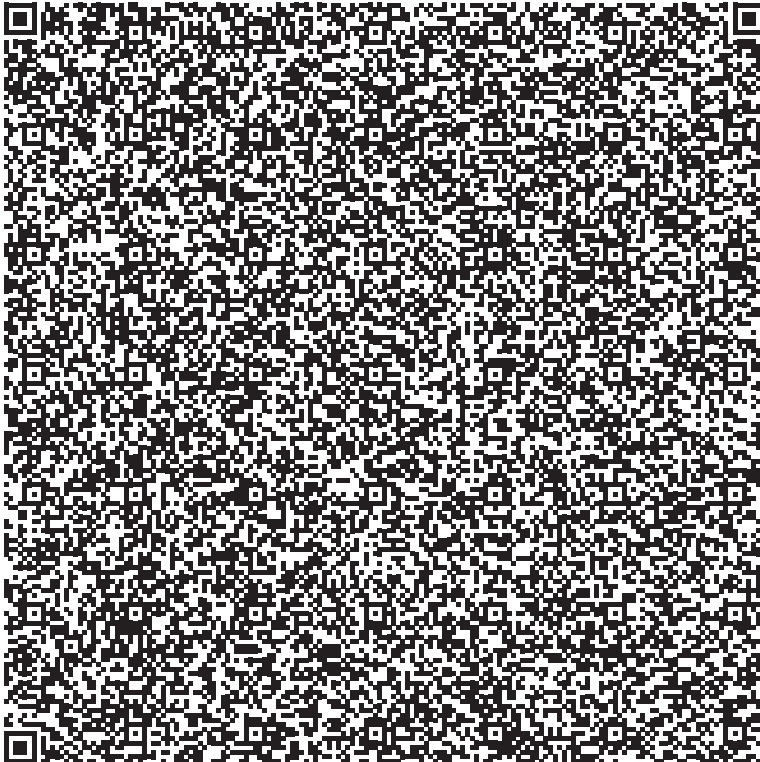


[grid10_LV.html](#)

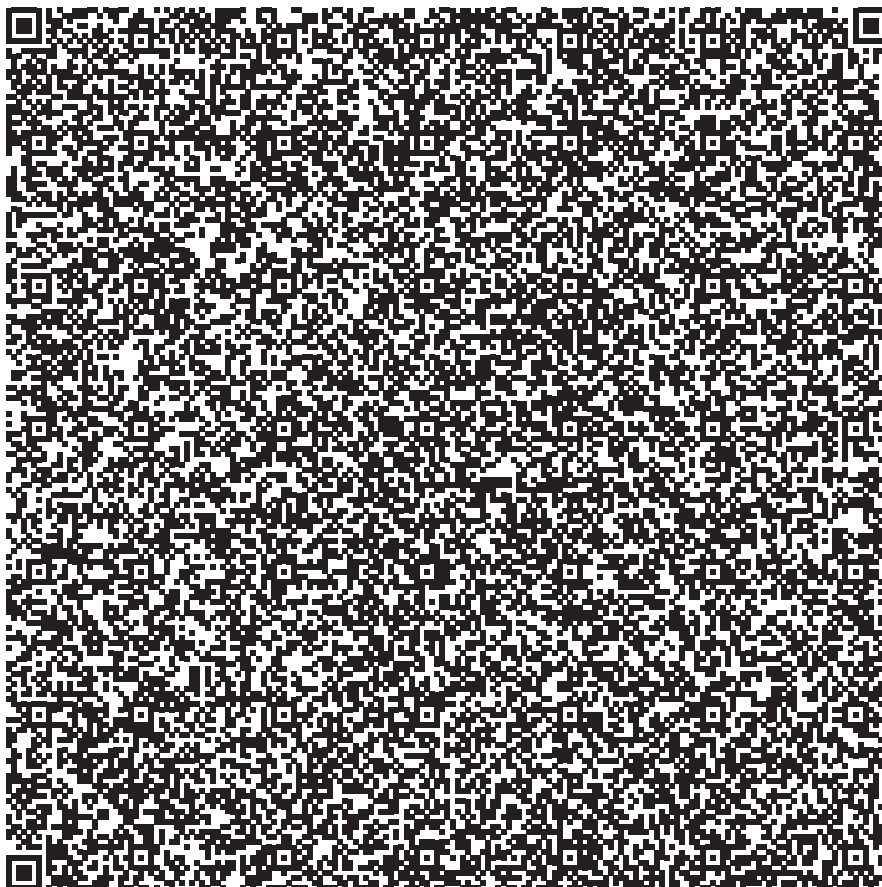


[grid21_LV.html](#)

Anhang 2 – Lösungsvorschläge



[grid22_LV.html](#)



[grid23_LV.html](#)