Aufgabe 1 - Minesweeper

In einer ersten Phase sollte man mit der Datenstruktur beginnen, sprich diese erst einmal definieren. Hierfür benötigt man ein SCRIPT-Tag, welches in der Kopfzeile der HTML-Datei definiert ist, mit folgendem Inhalt:

```
var field    = [];
var contents = [];
```

In der Tat werden zwei verschieden Listen gebraucht:

- In der ersten wird festgehalten ob ein Feld zugedeckt ist oder nicht.
  - Wenn ein Feld zugedeckt ist, wird eine Schaltfläche eingeblendet, ansonsten wird der Inhalt angezeigt.
- In der zweiten wird der Inhalt des Felder gespeichert.
  - In einem Feld kann eine Bombe enthalten sein, eine Zahl oder nichts.

Um die Felder sauber zu initialisieren, muss eine Funktion nach dem Laden des Dokumentes gestartet werden. Dazu wird das Attribut „onload" im BODY-Tag verwendet:

```
<body onload="initGame()">
```

Die Funktion selbst wird oben im SCRIPT-Tag definiert:

```
function initGame()
{
      ...
}
```

Zum einen muss die Funktion die beiden Variable **`field`** und **`content`** initialisieren:

```
// init field and content
for(var i=0; i<6*6; i++)
{
      // hide each field with a button
      field.push('b');
      // add an empty content
      contents.push('');
}
```

Da das Spielfeld 6x6 Felder groß ist, müssen insgesamt 6*6 Einträge in beide Listen gemacht werden. In die Liste **`field`** werden lauter „b" Zeichen eingefügt, um so an zu geben, dass alle Felder versteckt sein sollen. (Für Test-Zwecke kann

man diese Liste aber auch mit einem anderen Zeichen befüllen, damit nicht direkt Schaltflächen angezeigt werden.). Die zweite Liste wird mit 6*6 leeren Einträgen aufgefüllt. Dies ist wichtig, damit die Zellen existieren und später darauf zugegriffen werden kann.

Im Spiel sollen 3 Bomben zufällig platziert werden. Um sicher zu stellen, dass es auch wirklich drei sind und nicht, dass eine Bombe zufälliger weise in einem Feld platziert wird wo schon vorher eine Bombe war, benötigt man folgenden Code:

```
// get 3 different random nombers for the bombs
var bombs = [];
while(bombs.length<3)
{
      // generate a new random field index
      var rnd = Math.floor(Math.random()*6*6);
      // only add it, it this index is not yet on the list
      if(bombs.indexOf(rnd)==-1)
            bombs.push(rnd);
}
// set a bomb onto each of the selected fields
for(var i=0; i<bombs.length; i++)
      contents[bombs[i]]='b';
```

Erklärungen:

- Es wird eine leere Liste **bombs** erstellt.
- Solange die Liste nicht 3 Elemente enthält, wird eine Zufallszahl aus dem Bereich [0,36] generiert. Wenn diese nicht schon in der Liste **bombs** enthalten ist, so wird sie der Liste angehängt.
- Am Ende sollten 3 verschiedene Zahlen aus dem Raum [0,35] in der Liste enthalten sein, welche die Positionen der Bomben definieren.
- Nun wird an diese Positionen ein „b" in der Liste **contents** eingefügt.

Ab diesem Moment ist es interessant eine Methode zu schreiben, welche das Feld visualisiert, da man so überprüfen kann, dass die Bomben auch wirklich da sind!

Der ganze HTML Code der Datei sieht so aus (CSS Code auf der letzten Seite):

```
      <body onload="initGame()">
            <div id="area">
                  <p class="title">Minesweeper</p>
                  <div id="images">
                        <img id="smiley" src="ms-neutral-face.png">
                        <span id="nob">3</span>
                        <img src="clanbomber.png">
                  </div>
                  <div id="field"></div>
            </div>
      </body>
```

Das Feld selbst soll also im DIV-Tag mit der ID „field" angezeigt werden. Die dafür nötige Funktion ist folgende:

```
function displayField()
{
	// let's use a table
	var code = '<table id="buttons"><tr>';
	for(var i=0; i<field.length; i++)
	{
		//case of a hidden field --> button
		if(field[i]=='x')
			code+='<td><button></button></td>';
		// case of a bomb --> image
		else if (contents[i]=='b')
			code+='<td><img src="bomb_exp.png"></td>';
		// any other case (empty of number)
		else
			code+='<td>'+contents[i]+'</td>';
		// start a new row
		if(i % 6 == 5) code+='</tr><tr>';
	}
	// remove the trailing "<tr>"
	code=code.substring(0,code.lastIndexOf('<tr>'));
	// close the table
	code+='</table>';
	// output the newly generated HTML code
	document.getElementById('field').innerHTML=code;
}
```

Erklärungen:

- Das Feld wird in einer Tabelle angezeigt, d.h. Der entsprechende HTML Code muss erzeugt werden.
- Für jede Zelle müssen drei Fälle in Betracht gezogen werden:
  - Das Feld ist zugedeckt (**`field[i]=='b'`**): In diesem Fall muss eine Schaltfläche eingetragen werden.
  - Das Feld ist aufgedeckt und beinhalte eine Bombe (**`contents[i]=='b'`**): In diesem Fall muss das Bild der Bombe angezeigt werden.
  - In jedem anderen Fall wird einfach der Inhalt der Liste **`contents`** angezeigt.
- Würde man das Feld mit den Positionen anzeigen, so würde es wie folgt aussehen:

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

- Am Ende jeder Zeile muss also ein **</tr><tr>** eingefügt werden, um den Umbruch der Zeile zu erzwingen. Leider bleibt dann zu Schluss ein überschüssiges **<tr>** stehen, welches dann am Ende wieder herausgefiltert werden muss bevor das TABLE-Tage geschlossen werden kann.
- Am Ende wird der Erzeugte Code dann noch in das entsprechende DIV-Tag eingefügt.

Ruft man diese Methode am Ende der `initGame()` Methode auf, so wird das Feld mit Schaltflächen angezeigt. Wenn man die Zeile:

```
field.push('x');
```

in z.B.

```
field.push('c');
```

umwandelt, so wird der Inhalt angezeigt und man kann überprüfen, dass immer drei Bomben im Spielfeld platziert sind.

Als nächstes müssen die Zahlen, welche angeben wie viele Bomben um ein Feld herum liegen, berechnet, platziert und angezeigt werden. Dafür ist folgender Code nötig, der in der Methode **`initGame()`** vor dem Aufruf von **`displayField()`** platziert werden muss.

```
// count the value of bombs around each field
for(var i=0; i<field.length; i++)
{
      // do this only for empty fields
      if(contents[i]=='')
      {
            var count = 0;
            // above
            if((i % 6 > 0) && (i > 6)  && (contents[i-7]=='b')) count++;
            if(               (i > 5)  && (contents[i-6]=='b')) count++;
            if((i % 6 < 5) && (i > 5)  && (contents[i-5]=='b')) count++;

            // left & right
            if((i % 6 > 0)             && (contents[i-1]=='b')) count++;
            if((i % 6 < 5)             && (contents[i+1]=='b')) count++;

            // below
            if((i % 6 > 0) && (i < 30) && (contents[i+5]=='b')) count++;
            if((            (i < 30) && (contents[i+6]=='b')) count++;
            if((i % 6 < 5) && (i < 29) && (contents[i+7]=='b')) count++;

            if(count!=0) contents[i]=count;
      }
}
```

Erklärungen:

Die Grundidee ist die, dass für jedes Feld, welche keine Bombe enthält, geschaut wird, wie viele Bomben in dem umliegenden Felder vorhanden sind. Jede Zelle ist von maximal 6 anderen Zellen umgeben. Probleme bereiten nur die Zellen am Rand.

Als erstes ist es wichtig zu sehen, wie man die Positionen der umliegenden Zellen bestimmt. Dies erklärt sich am einfachsten in einer Tabelle an Hand eines Beispiels:

0	1	2	3	4	5
6	7	8	9	10	11
12	13	**14**	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

Nehmen wir die Zelle mit der Position 14, so ist die Rechnung, um von 14 auf die Position der umliegenden Zellen zu gelange die folgende:

	14-7	14-6	14-5		
	14-1	**14**	14+1		
	14+5	14+6	14+7		

Ersetze man nun 14 durch die Variabel **i**, so erhält man die generell gültige Formel:

	i-7	i-6	i-5		
	i-1	**i**	i+1		
	i+5	i+6	i+7		

Da nicht für jede Zelle 6 umliegende Zellen existieren, muss die Spezialfälle einzeln behandeln. Am einfachsten ist es, sich Fall für Fall an zu schauen.

Frage: Für welche Zellen existiert eine Zelle „oben links"?

Die Antwort findet sich in den hier grün markierten Zellen:

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

Wenn **i** die Position der Zelle darstellt, so werden diese Zellen wie folgt mathematisch beschrieben:

- Alle grünen Zellen haben einen Position welche größer als 6 ist: **`i > 6`**
- Teilt man die Position der grünen Zellen durch 6, so ergibt sich ein Rest der nicht Null ist: **`i % 6 > 0`**

  Hier, zur besseren Veranschaulichung, die gleiche Tabelle welche den Rest der Position geteilt durch 6 anzeigt:

0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5

Die Tabelle für alle Zellen welche eine darüber liegende Zelle besitzen, sieht dnan so aus:

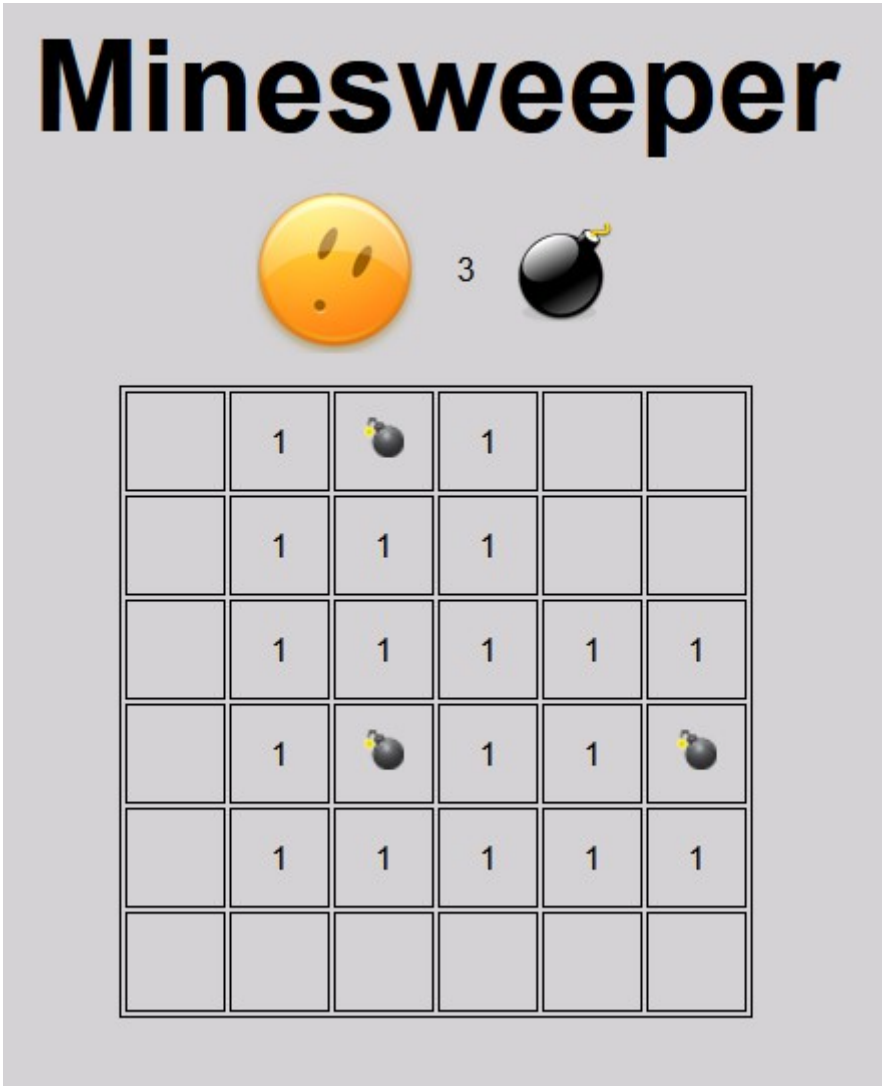
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

Entsprechen einfacher ist also die Formel.

Für folgende Zellen existiert eine Zelle „oben rechts":

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

Das im Browser eingeblendete Feld sollt nun so aussehen:



Wechselt man in der Methode `initGame()` die Zeile

```
field.push('c');
```

wieder in

```
field.push('x');
```

um, so wird nicht mehr der Inhalt, sondern die Schaltflächen angezeigt.

Damit man auf die Schaltfläche klicken kann, muss im Code welcher das Feld anzeigt, des entsprechende onClick-Event eingefügt werden:

```
if(field[i]=='x')
      code+='<td><button onclick="discover('+i+')"></button></td>';
```

Die Methode **discover** sieht dann so aus:

```
function discover(i)
{
	// unhide field
	field[i]='';
	// if bombe, unhide all fields
	if(contents[i]=='b')
	{
		for(var i=0; i<field.length; i++)
			field[i]='';
		document.getElementById('smiley').src='ms-sad-face.png';
	}
	// if the number of bombs to find equals the number of unhidden fields,
	// the user clicked each possible button, so he won the game!
	else if(countHiddenFields()==3)
	{
		document.getElementById('smiley').src='ms-smiling-face.png';
	}

	displayField();
}
```

Erklärungen:

- Als erstes wird das Feld aufgedeckt, indem in der Liste **field** das entsprechende Feld leert.
- Trifft man auch eine Bombe, also enthält das aktuell angeklickt Feld ein „b", so werden alle Felder aufgedeckt und eine trauriges Smiley angezeigt. Der Spieler hat nun verloren.
- Ist die Anzahl der noch auf zu deckenden Felder identisch mit der Anzahl der Bomben, so hat der Spieler gewonnen, da er alle möglichen Felder aufgedeckt hat.
- Am Ende muss das Spielfeld erneut angezeigt werden.
- Die Funktion zum zählen der noch noch aufgedeckten Felder sieht so aus:

```
// count fields that are still hidden
function countHiddenFields()
{
	var count = 0;
	for(var i=0; i<field.length; i++)
		if(field[i]=='x')
			count++;
	return count;
}
```

Als nächstes müsste man noch eine „Reset" Schaltfläche einbauen.

Wäre das nicht eine gute Prüfungsfrage?

CSS Code

```
<style type="text/css">
	* {
		font-family:Arial;
	}

	body, html {
		background-color:#FFFDFF;
		margin:0px;
		padding:0px;
		text-align:center;
	}

	#area {
		margin-top:32px;
		display:inline-block;
		background-color:#D4D2D4;
		box-shadow:5px 5px 5px #808080;
		border-radius:50px;
		width:600px;
		height:800px;
	}

	p.title {
		font-size:4em;
		text-align:center;
		font-weight:bold;
		margin-bottom:0px;
	}

	div#field {
		display:inline-block;
	}

	table#buttons {
		border:1px #000 solid;
	}

	table#buttons td {
		border:1px #000 solid;
		text-align:center;
		padding:0px;
		width:48px;
		height:48px;
	}

	table#buttons td button {
		width:48px;
		height:48px;
	}

	#images img {
		vertical-align:middle;
		margin:16px;
	}
</style>
```