Hochschule der Medien Stuttgart
Bibliotheks- und Informationsmanagement (Bachelor)
Sommersemester 2016

A mazing Library

Schriftliche Ausarbeitung im Seminar
Einführung in das Programmieren
Prof. Pfeffer

Vorgelegt von:
Annabel Welsch
aw119@hdm-stuttgart.de
und
Carolyn Marschall
cm093@hdm-stuttgart.de

Abgabedatum: 28.06.2016

Idee

Unsere Idee war ein Bibliothekslabyrinth, in dem der Spieler ein Tor erreichen muss um das Level zu beenden.

Das erste Level soll ein einfaches Labyrinth ohne Hindernisse sein.

Für das zweite Level haben wir uns überlegt einen Bücherwagen als bewegliches Hindernis über eine Kreuzung fahren zu lassen.

Im dritten Level muss bevor das Tor erreicht werden kann, ein bestimmter Punkt passiert werden.

Level 4 soll eine Kombination aus Level 2 und Level 3 sein.

Umsetzung

1. Labyrinth

Damit wir schnell etwas sehen, wollten wir gleich ein Labyrinth zeichnen. Da uns erste Ideen gefehlt haben, haben wir gegoogelt und ein JSFiddle gefunden das vielversprechend aussah.

> JSFiddle (o.J.): Canvas Labyrinth, von hhanuman. URL: http://jsfiddle.net/hhanuman/hW4uV/4/

Zunächst hatten wir das Problem, dass wir kein jQuery eingebunden hatten und daher der Code bei uns nicht funktioniert hat. Nach und nach haben wir den Code dann an unsere Bedürfnisse angepasst.

Unser Labyrinth wird über ein Array gezeichnet. Das Array besteht aus 15 Arrays mit 15 Ziffern (0,1 oder2). Wobei 0 für den Weg, 1 für eine Wand und 2 für das Ziel steht.

Die Feldgröße wird durch die Variable *size* festgelegt. *Size* wird durch die Breite des Canvas und einem Delimiter berechnet. Der Delimiter ist bei uns 15, er legt fest in wie viele Felder der Canvas geteilt wird.

2. Figuren zeichnen

Alle unsere Figuren, Player, Wand, Tor, Hindernisse, werden nach demselben Prinzip gezeichnet. Anfangs waren der Einfachheit halber alle geometrische Formen.

Nach erfolgloser Recherche wie wir unsere Bilder laden können, haben wir im Code von Elena… und Sarah… geschaut und mit *context.drawImage* eine gute Möglichkeit gefunden. Damit unsere Bilder in der richtigen Größe und an der richtigen Position gezeichnet werden, haben wir *context.drawImage* in eine Funktion geschrieben mit den Variablen x, y, w, h. Dadurch dass wir die x- und y-Werte des zu zeichnenden Bildes mit *size* multiplizieren und die Breite und Höhe auf *size* setzen, bekommt das Bild die Größe eines Feldes. Dies ermöglicht auch, dass wir die Figuren über Felder bewegen können, anstatt nur über Pixel.

```
//Zeichnet den Spieler
    function Player() {
        function draw(x,y,w,h,color){
                context.drawImage(document.getElementById('imagePlayer'),x,y,w,h);
        };
    draw(player.x*size,player.y*size,size,size);
    };
```

Die innere Funktion „draw“ bestimmt wie der Player aussieht und die äußere Funktion „Player“ zeichnet den Player nach spezifischen Maßgaben.

3. Steuerung und Kollision

Die Grundidee für den Code der Steuerung über den Keycode haben wir aus einem YouTube Tutorial von Caleb Prenger

> Youtube (2014): JavaScript Canvas Tutorial - Move a sprite/character on screen using the keyboard, von Caleb Prenger. URL: https://www.youtube.com/watch?v=OFzs4unxVtU (28.06.16)

Dieser hat uns jedoch keine Kollisionserkennung geboten. Daher haben wir den Eventhandler von Elena Mahal und Sarah Ernst übernommen.

> JSFiddle (o.J.): Simple Maze, von Nate. URL: http://jsfiddle.net/n8j1s/4y22135r/ (28.06.16)

```
// Figur bewegen
    function move(e){
        $(document).keyup(function(e){
            if((e.which == 38) && canMove(player.x, player.y-1))//Hoch
                player.y--;
            else if((e.which == 40) && canMove(player.x, player.y+1)) //Runter
                player.y++;
            else if((e.which == 37) && canMove(player.x-1, player.y)) //Links
                player.x--;
            else if((e.which == 39) && canMove(player.x+1, player.y)) //Rechts
                player.x++;

            Maze();
            Player();
            e.preventDefault();
        });
    };
```

Bei der Kollisrionserkennung hatten wir einige Probleme sie so hinzubekommen wie wir sie wollten, besonders als wir anfingen mehr als nur ein Level zu haben.

Anfangs hatten wir ein stark komprimiertes Stück Code vor uns, das wir nur ansatzweise verstanden haben, daher haben wir es aufgedröselt und die einzelnen Events abgefragt die wir benötigen.

Der Code muss unterscheiden können welches Labyrinth gerade abgerufen wird. Dies wird über die Variable „currentMaze“ geregelt.

Das Erkennen des Randes ist im Vergleich damit einfach gewesen.

```
//Kollisions erkennung mit Wand, Rand und Tor
    function canMove(x,y){
        if (x<0){return false;} //erkennt den Rand
        else if (y<0){return false;}
        else if (x>=delimiter){return false;}
        else if (y>=delimiter){return false;}

        else if (currentMaze[y][x] ==1){return false;} // erkennt die Wand
```

4. Tor

Gezeichnet wird das Tor in der Funktion „Maze“, da wir uns dazu entschieden haben es im Array zu verankern und es nicht als selbstständige Funktion an eine bestimmte Position zu zeichnen.

```
//Zeichnet Labyrinth
    function Maze() {
    currentMaze = Levels [currentLevel];
        function BlackOrWhite(something){
            if (something == 0){return "white";};
        };

        for(var i = 0;i<15;i++){
            for(var j = 0;j<15;j++){
                if (currentMaze[i][j] == 1) {
                    context.drawImage(document.getElementById('imageWand'),size*j,size*i,size,size) ;} //Zeichnet Wand
                else if (currentMaze[i][j] == 2)    {
                    context.drawImage(document.getElementById('imageTor'),size*j,size*i,size,size)  ;} //Zeichnet Tor
                else {
                rect(size*j,size*i,size,size,BlackOrWhite((currentMaze[i][j])));                      //Zeichnet Weg
                };
            };
        };
```

Die Kollisionserkennung findet wieder in der *canMove* Funktion des Players statt. Alles Wichtige für das nächste Level wird bei berühren des Tors geladen. Zu Beginn hatten wir überlegt die Eigenschaften eines Levels am Anfang zu laden, es hat sich aber herausgestellt, dass diese Variante einfacher ist.

```
else if (currentMaze[y][x] ==2){alert('Gewonnen!');currentLevel++; player.x=0; player.y=0; PosWagen.x=0; PosWagen.y=0; Zusa
```

5. Zusatzladen

Um die Hindernisse nicht in allen Leveln sichtbar zu haben, werden ihre Positionen in globalen Variablen gespeichert. Die Position ist bei allen am Anfang dieselbe.

```
//setzt den Knopf auf Position 0
        var PosKnopf = {
            x: 0,
            x: 0
        };

//setzt den Blockade auf Position 0
        var PosBlockade = {
            x: 0,
            x: 0
        };
```

Erst für bestimmte Level werden die Hindernisse an ihre gewünschte Position gebracht. Dies geschieht über die Funktion *ZusatzLaden*.

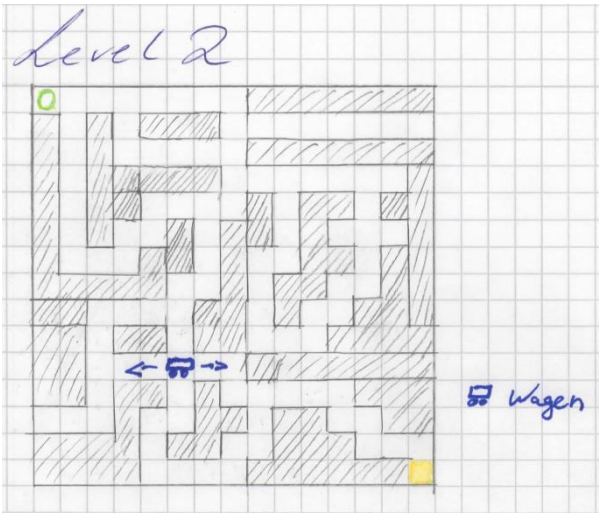
```
//Setzt Zusatz auf Position
    function ZusatzLaden (){

        if (currentLevel == 1) {PosWagen = {
                                    x: 5,
                                    y: 10
                                    };
        };
        if (currentLevel == 2) {PosKnopf = {
                                    x: 6,
                                    y: 14
                                    };
                                PosBlockade = {
                                    x: 12,
                                    y: 10
                                    };
        };
        if (currentLevel == 3) {PosWagen = {
                                    x: 8,
                                    y: 7
                                };
                                PosKnopf = {
                                    x: 0,
                                    y: 2
                                    };
                                PosBlockade = {
                                    x: 5,
                                    y: 11
                                    };
        };
```

Geladen werden diese Hindernisse jedoch in der Funktion *Maze*. Durch eine if-Schleife werden sie nur dann gezeichnet, wenn sie gebraucht werden.

```
  //Lädt den Knopf und Blockade im 3. Level
  if (currentLevel == 2) {Knopf(); Blockade()};

  //Lädt den Zusatz im 4. Level
  if (currentLevel == 3) {Bücherwagen(); Knopf(); Blockade()};
```

5.1 Bücherwagen

Als erstes Hindernis ist ein Bücherwagen geplant, der sich relativ zum Spieler bewegt.



Der Wagen soll sich nur bewegen, wenn der Spieler sich bewegt, daher reagiert er auf Tastendruck. Hier haben wir den Code des Spielers als Vorlage genommen und so umgeändert, dass der Wagen nach rechts fährt wen die Pfeiltasten Rechts und Links gedrückt werden und nach links wenn Pfeil Hoch und Runter gedrückt werden.

```
//Bücherwagen bewegen und Kollisionserkennung
    function moveWagen(e){
        $(document).keyup(function(e){

                if((e.which == 38 || e.which == 40 ) && canMoveWagen(PosWagen.x-1, PosWagen.y)) //Hoch und Runter: Links
                    PosWagen.x--;
                if((e.which == 37 || e.which == 39 ) && canMoveWagen(PosWagen.x+1, PosWagen.y)) //Links und Rechts: Rechts
                    PosWagen.x++;
            });

        Maze();
        Player();

    };
```

Der Bücherwagen reagiert auf die Kollision mit der Wand. Zu Nächst hat nur der Spieler auf eine Kollision mit dem Wagen reagiert, was dazu geführt hat, dass Spieler und Wagen die Positionen tauschen konnten. Als wir dann auch den Wagen auf den Spieler reagieren ließen, hat sich dieses Problem behoben. Im zweiten Level ist es dennoch recht einfach am Wagen vorbei zukommen, was an der Position des Wagens liegen könnte.

```
    function canMoveWagen (x,y){
        if (currentMaze[y][x] ==1){return false;}       // erkennt die Wand

        else if ((x == player.x)&& (y==player.y)) {return false;} // erkennt den Player

        else {return true;}
    };

};//Game over
```

5.2 Knopf und Blockade

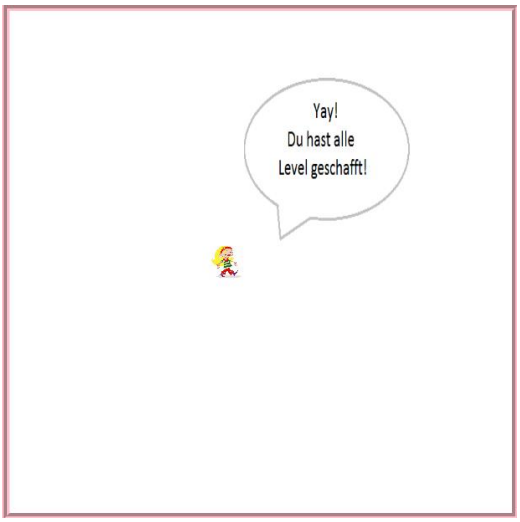
Die Kollisionserkennung für die Blockade und den Knopf ist dieselbe wie für den Bücherwagen. Der einzige Unterschied ist, dass beim erkennen des Knopfes die Position der Blockade außerhalb des Spielfeldes gesetzt wird.

Damit man über den Knopf auch laufen kann, im Gegensatz zu der Wand oder den anderen Hindernissen, haben wir am Ende noch ein „return true“ angehängt.

```
//Kollisions erkennung mit Wand, Rand und Tor
    function canMove(x,y){
        if (x<0){return false;} //erkennt den Rand
        else if (y<0){return false;}
        else if (x>=delimiter){return false;}
        else if (y>=delimiter){return false;}

        else if (currentMaze[y][x] ==1){return false;} // erkennt die Wand

        else if ((x == PosWagen.x)&& (y==PosWagen.y)) {return false;} // erkennt den Wagen

        else if ((x == PosBlockade.x)&& (y==PosBlockade.y)) {return false;} // erkennt die Blockade

        else if ((x == PosKnopf.x)&& (y==PosKnopf.y)) {PosBlockade.x=15; PosBlockade.y=15; return true;}
```

5.3 Benachrichtigung am Ende

Am Ende des Spiels wollten wir eine Benachrichtigung, dass nun alle Level geschafft wurden. Für uns die einfachste Lösung war, diese Benachrichtigung wie ein eigenes Level zu behandeln. Dazu haben wir ein leeres Array ohne Wände als fünftes Level gezeichnet. Den Player haben wir in die Mitte gesetzt und eine Sprechblase darüber. Die Sprechblase wird über die Funktion *ZusatzLaden* aufgerufen. Das Bild zur Sprechblase haben wir selbst mit Paint gezeichnet.



6. Grafiken

Unsere Bilder haben wir von folgender Website, die Bilder unter freier Lizenz anbietet:

> Bobek Ltd. (2016): Public Domain Pictures. URL: http://www.publicdomainpictures.net/

Links zu den Bildern:

> Player: http://www.publicdomainpictures.net/view-image.php?image=143497&picture=einkaufs-frau-clipart
>
> Wand: http://www.publicdomainpictures.net/view-image.php?image=162457&picture=bucherregal
>
> Tor: http://www.publicdomainpictures.net/view-image.php?image=76911&picture=grune-tor
>
> Knopf: http://www.publicdomainpictures.net/view-image.php?image=102595&picture=grune-taste

Nur den Wagen haben wir von einer anderen Seite:

> Patientenbücherei an der Universität Mainz (o.J.): der Bücherwagen kommt in der Regel auf folgende Stationen. URL: http://www.patientenbuecherei.de/buecherwagen.php

Ausblick

Zu nächst könnte man noch die Bewegung des Wagens verfeinern, so dass es nicht ganz so einfach wird an ihm vorbei zukommen. Andererseits sollte es auch nicht unmöglich werden.

Außerdem ist der Anzahl der möglichen Level keine Grenze gesetzt, so dass man noch weitere Level schreiben könnte mit neuen Hindernissen, z.B. dass ein Passwort eingegeben werden muss oder eine gegnerische Figur die den Spieler verfolgt. Auch die Übertragung in 3D wäre möglich. Zu dem könnte man die Spielzüge begrenzen, so dass das Ziel nur über den kürzesten Weg erreicht werden kann.