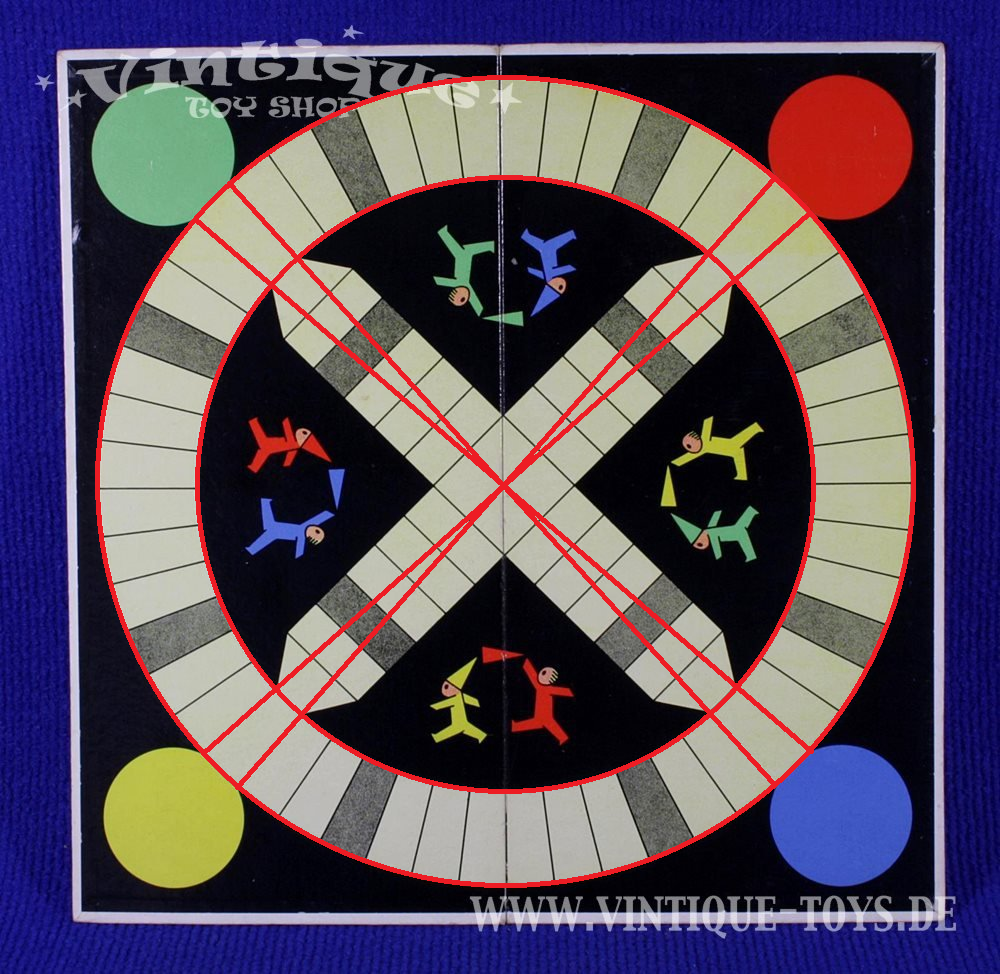
Fang den Hut

Implementierung in Java und JavaScript

# Überlegungen zur Darstellung des Spielfeldes



Methode 3

Methode 2

Methode 1

1. Lässt man die Rundung des Spielfeldes außer Acht, so ergeben sich viele Rechtecke in einem Radius um den Mittelpunkt. Jedoch entstehen dadurch Lücken.
2. Die Lücken könnten mit einem Trapez geschlossen werden.
3. Eine Lösung, bei der zwei Kreisbogen mit Linien verbunden werden.

## Mögliche Methoden bei p5.js

Zu 2) Der rechte Quelltext erzeugt ein Trapez. Die Eckpunkte ließen sich mit den Kreisen berechnen.

noFill();

beginShape();

vertex(30, 20);

vertex(65, 20);

vertex(85, 75);

vertex(10, 75);

endShape(CLOSE);

Zu 3) Mit den Eckpunkten des Kreisbogens ließe sich mit curve(…) ein Bogen zeichnen, der dem Objekt entspricht.

Benötigt werden jeweils die vorherigen und nachfolgenden Eckpunkte (Kontrollpunkte), mit denen die Kurve aufgespannt wird.

## Mathematische Erläuterung zur Berechnung der Eckpunkte

Der Winkel wird im Bogenmaß angegeben: , wobei der gesamte Kreis ist.

Die x-Stelle lässt sich mit , der y-Wert mit bestimmen.

Berechnung der Mittelpunkte der Felder:

13\*4 = 52 Felder => wir erhalten für

Möchte man die Randpunkte erhalten, so muss der Winkel um die Hälfte ergänzt oder abgezogen werden: und

# Bestimmung, ob die Maus sich über einem Feld befindet:

Innerhalb des äußeren Rings:

Wenn der Abstand der Maus größer als der Radius des inneren Ringes und kleiner als der Abstand des äußeren Ringes ist, dann befindet sich die Maus im Ring.

Erkennung des einzelnen Feldes:

Entscheidend ist nun, ob sich die Maus zwischen zwei Geraden des Trapezes befindet.

Methoden: Liegt der Punkt zwischen den Geraden?

Die Geraden der Kanten werden mit den Eckpunkten erzeugt und dann wird verglichen, wo sich der Punkt befindet.

Daraus resultiert die Methode:

/\*\*

\*Liefert einen positiven Wert, wenn sich der Punkt unterhalb der Geraden befindet, einen negativen

\*Wert, wenn sich der Punkt oberhalb der Geraden befindet, und 0 falls sich der Punkt auf der

\*Geraden befindet.

\*/

function lageGeradePunkt(gX1,gY1,gX2,gY2,pX,pY){

m = (gY2-gY1)/(gX2-gX1);

yGerade = m\*pX+gY2-m\*gX2;

return pY-yGerade;

}

## Methoden: Anzahl der Schnittpunkte mit dem Rand des Objektes.

Eine Gerade wird durch zwei Eckpunkte aufgestellt. Nun werden für alle Kanten die entsprechenden Geraden bestimmt. Außerdem wird eine Gerade vom Ursprung zum Mauszeiger erzeugt. Der Schnittpunkt der Geraden kann nun daraufhin untersucht werden, ob er auf der Strecke liegt.

/\*\*

\* Zunächst wird der Schnittpunkt der Geraden vom Ursprung zum Punkt und zwei Eckpunkten bestimmt.

Liegt der Schnittpunkt im x-Bereich zwischen den beiden x-Stellen der Punkte, so ergibt das Produkt der Differenzen immer einen negativen Wert. Äquivalent kann so berechnet werden ob der Punkt zwischen den y-Werten liegt. Ist dies der Fall, so schneidet die Ursprungsgerade die Strecke.

\*//

function schnittMitDerStrecke(g1X1,g1Y1,g1X2,g1Y2, g2X1,g2Y1,g2X2,g2Y2){

m1 = (g1Y2-g1Y1)/(g1X2-g1X1);

m2 =(g2Y2-g2Y1)/(g2X2-g2X1);

b1 = g1Y1-m1\*g1X1;

b2 = g2Y1-m2\*g2X1;

xKoordinate =(b2-b1)/(m1-m2);

yKoordinate = m\*pX+gY2-m\*gX2;

return (xKoordinate-x1)\*(xKoordinate-x2)<0 && (yKoordinate-y1)\*(yKoordinate-y2) < 0

}

function punktLiegtInForm(pX, pY, form){

schnittpunkte = 0;

for (let i = 0; i < form.punkte.length; i++){

if( schnittMitDerStrecke(

form.punkte[i].x,form.punkte[i].y,

form.punkte[(i+1)%form.punkte.length].x, form.punkte[(i+1)%form.punkte.length].y,

0,0,pX,pY)){

Schnittpunkte++;

}

}

return schnittpunkte %2 == 0;

}