# Aufgabe 4: Krocket

Team-ID: 00433

Team-Name: Was?

Bearbeiter dieser Aufgabe:
Mathieu de Borman

18. Oktober 2024

## Inhaltsverzeichnis



## 1 Lösungsidee

Es wird für jedes Tor getestet, welches die knappsten Schüsse sind, die gemacht werden können. Dadurch wird der Schussbereich solange weiter eingegrenzt, bis klar ist, ob durch alle Tore geschossen werden kann. Danach werden Ballgröße und Reihenfolge der Tore getestet.

### 1.1 $r = 0$

Als Erstes wird ein Viereck aus den Pfosten des ersten und letzten Tores konstruiert. Das Viereck soll den gültigen Schussbereich für alle bisher berücksichtigten Tore darstellen. Ausgehend davon werden die Seiten des Vierecks immer weiter in die Mitte, in Richtung von dem Pfosten innerhalb des Vierecks, verschoben. Das wird solange getan bis alle Pfosten außerhalb oder an den Seiten des Vierecks sind.

### 1.2 $r$ = Ballradius

Anschließend wird ausgehend von dem Viereck gestetet, ob ein Ball durchpassen würde und ob alle Tore in der richtigen Reihenfolge durchschießbar sind.

### 1.3 Laufzeitanalyse

Die Laufzeit hängt von einem Parameter ab, nämlich der Anzahl der Tore $n$. Das Programm hat eine Laufzeitkomplexität von $O(n)$ bei einer Speicherkomplexität von $O(n)$.

## 2 Umsetzung

Die Lösungsidee ist in JavaScript implementiert.

### 2.1 Einlesen

Die Eingabe wird über ein Eingabefenster (prompt) ausgelesen. Anschließend werden alle ungültigen Zeichen rausgefiltert (für den Fall das keine saubere Eingabe gemacht wurde) und die gültigen Zeichen in Zahlen umgewandelt. Danach wird überprüft, ob es ein gültiges Krocket Feld ist. Daraufhin werden die Daten als Linienobjekte gespeichert.

### 2.2 $r = 0$

Als Erstes wird ein Viereck aus den Pfosten des ersten und letzten Tores konstruiert. Das Viereck soll den gültigen Schussbereich für alle bisher berücksichtigten Tore darstellen. Anschließend wird durch jedes Tor durchiteriert (Funktion moveSide()). Immer dann wenn mindestens ein Pfosten innerhalb des Vierecks ist, wird die Seite, welche nicht von dem Tor gekreuzt wird (Seite A), bis zu dem Pfosten bewegt (Funktion crop()). Dabei wird immer darauf geachtet, dass die Schnittmenge des Vierecks mit dem letzten Tor möglichst groß ist. Das sorgt dafür, dass alle späteren Tore (die ja in Richtung des letzten Tores sind) wahrscheinlich mindestens teilweise im Viereck liegen. Das wird erreicht, indem zuerst getestet wird ob, eine Linie vom Eck (Seite A und letztes Tor) durch den Pfosten eine mögliche neue Seite wäre (die die andere Seite (Seite B) nicht kreuzt). Ist das nicht machbar, dann geht die verschobene Seite durch die Ecke (Seite B und erstes Tor) und Pfosten. Der Test wird einmal vom ersten zum letzten Tor ausgeführt und einmal vom letzten zum ersten, um möglichst extreme Seiten zu haben.

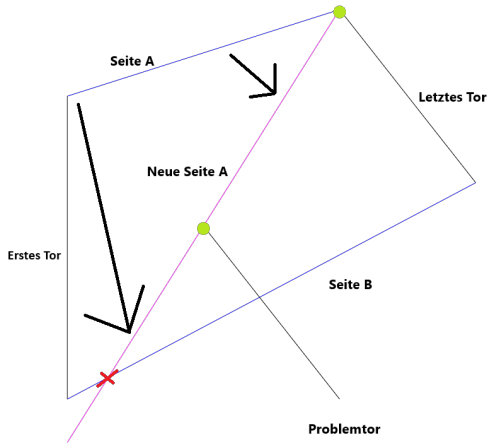



Abbildung 1: Seite A, letztes Tor als Eckpunkt

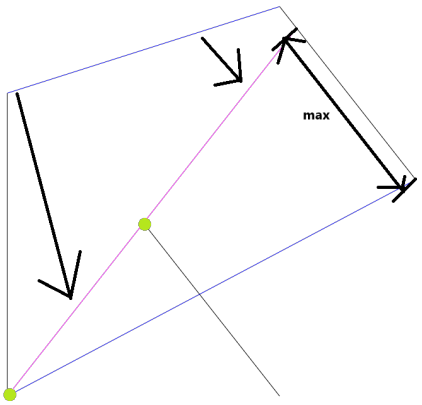



Abbildung 2: Seite B, erstest Tor als Eckpunkt

## 2.3 $r$ = Ballradius

### 2.3.1 Ballradius

Um zu wissen, ob der Ball durch die Tore geschossen werden kann, werden für jede Seite der finalen Vierecke, Parallelen im Abstand $r$ erschaffen. Danch wird für jede Seite und ihre Parallelen getestet, ob diese jedes Tor mindestens zweimal kreuzen. Ist das der Fall für mindestens ein Paar (an Parallelen), dann wissen wir, dass der Schuss auch mit einem Ball der Größe r geht.

### 2.3.2 Reihenfolge

Es wird vom Tor 0 bis Tor n eine Teillinie von einer Parallelen zur Schusslinie gespeichert. Wenn das Tor n+1 diese Linie kreuzt, wissen wir, dass die Reihenfolge der Tore nicht stimmt.

## 2.4 Ausgeben

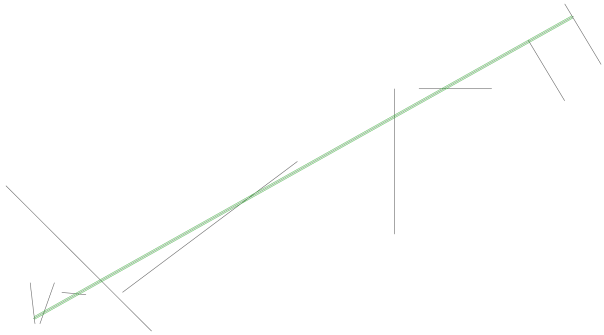
Die Abschussposition und Winkel werden in die Konsole ausgegeben. Der Winkel gibt die Abweichung von der Horizontalen an: 0° bedeutet nach rechts, 90° nach oben, 180° nach links und -90° nach unten.

# 3 Beispiele

Wir rufen nun das JavaScript-Programm mit den verschiedenen BWINF-Eingabedateien auf. Diese Dateien sind in demselben Ordner wie die Programmdatei. Das Programm wird mit Hilfe des Browsers ausgeführt. In dem Dialogfeld kann man dann den Plan des Krocket Felds eingeben. Die Ausgabe kann man in der Konsole auslesen. Das Feld wird als Graphik auf dem Canvas angezeigt und ist etwas verzerrt worden, um gut auf den Bildschirm zu passen. Das Programm terminiert für alle getesteten Eingaben in weniger als 20ms auf einem gewöhnlichen PC.

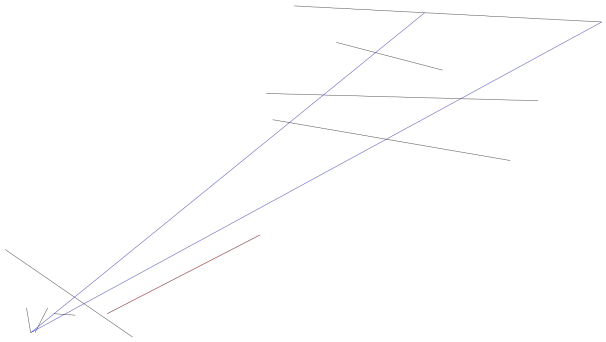
## 3.1 Lösungen

### 3.1.1 krocket1.txt



```
Abschusspunkt: (11.24|5.17) Winkel: 29.28°
```

### 3.1.2 krocket2.txt

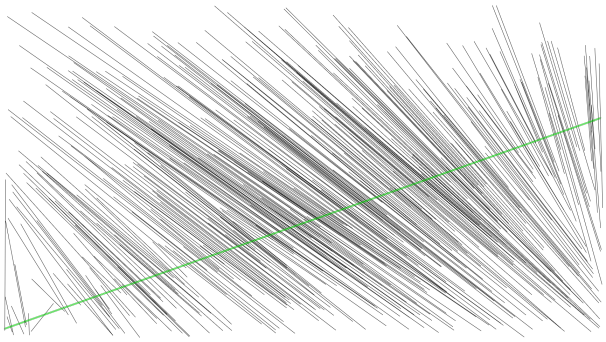


```
Nicht machbar wegen anordnung der Tore
```

### 3.1.3 krocket3.txt

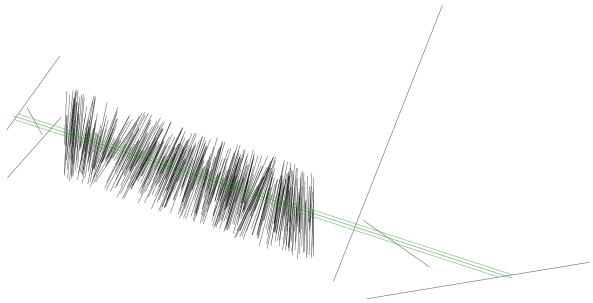
```
Nicht machbar wegen anordnung der Tore
```

### 3.1.4 krocket4.txt



```
Abschusspunkt: (-6.36|181.33) Winkel: 26.38°
```

#### 3.1.5 krocket5.txt



```
Abschusspunkt: (979.2|14681.12) Winkel: -13.43°
```

## 4 Quellcode

```
let i,ball = 0
//definiere was ein punkt ist
class point{
 constructor(x,y){
  this.x = x
  this.y = y
  //funktion zum messen der distanz zwischen zwei punkten
  this.dis = point => Math.sqrt(Math.pow(this.x-point.x,2)+Math.pow(this.y-point.y,2))
 }
}
//definiere was eine linie ist
class line{
 constructor(a,b){
  this.a = a.x<b.x?a:b
  this.b = a.x<b.x?b:a
  this.length = this.a.dis(this.b)
  this.dif = new point(this.b.x-this.a.x,this.b.y-this.a.y)
  this.gradient = this.dif.y/this.dif.x
  this.yIntercept = (-this.a.x)*this.gradient+this.a.y
  this.touch = point => ~~((this.length-(point.dis(this.a)+point.dis(this.b)))*Math.pow(10,10))==0
  this.middle = new point((this.a.x+this.b.x)/2,(this.a.y+this.b.y)/2)
  if(this.a.x==this.b.x&&this.a.y==this.b.y){
   return this.a
  }
 }
 //funktion zum errschaffen einer Parallelen
 parallel(dis){
  let prop = dis/this.length
  let xdif = this.dif.x*prop
  let ydif = this.dif.y*prop
  let a = new point(this.a.x+ydif,this.a.y-xdif)
  let b = new point(this.b.x+ydif,this.b.y-xdif)
  return new line(a,b)
 }
 //funktion um die linie um 90° zu drehen
 flip(){
  let xdif = this.dif.x/2
  let ydif = this.dif.y/2
  let a = new point(this.middle.x+ydif,this.middle.y-xdif)
```

```
  let b = new point(this.middle.x-ydif,this.middle.y+xdif)
  return new line(a,b)
 }
 //funktion um den Schnittpunkt mit einer anderen linie zu finden
 crossPoint(line){
  if(this.gradient == line.gradient){
   return false
  }else if(Math.abs(this.gradient)+Math.abs(line.gradient)==Number.POSITIVE_INFINITY){
   let toggle = Math.abs(line.gradient)==Number.POSITIVE_INFINITY
   let x = toggle?line.a.x:this.a.x
   let y = toggle?x*this.gradient+this.yIntercept:x*line.gradient+line.yIntercept
   return new point(x,y)
  }else{
   let x = (line.yIntercept-this.yIntercept)/(this.gradient-line.gradient)
   let y = this.gradient*x+this.yIntercept
   return new point(x,y)
  }
 }
 //funktion zum überprüfen, ob sich zwei linien kreuzen
 cross(line){
  let point = this.crossPoint(line)
  return point === false?false:this.touch(point)&&line.touch(point)
 }
}
//Lese Eingabe aus
let input = (prompt("Krocket")|| "0").split(/[\s\n]+/)
let vals = []
//Wandle String in Zahlen um und filtere ungültige Zeichen raus
for(let i = 0;i<input.length;i++){
 let maybe = Number(input[i])|| 0
 if(maybe.toString() == input[i]){
  vals.push(maybe)
 }
}
//Wenn die Eingabe kein Krocket Feld war, dann stoppe das Programm
if(!(vals.length>2&&(vals.length-2)%4 == 0)){
 throw new Error("Ungultige Eingabe")
}
const size = (vals.length-2)>>2
let hoops = Array(size)
ball = vals[1]
//Formatiere die Werte als Linien
for(let i = size-1;i>-1;i--){
 let cut = vals.splice(-4,4)
 hoops[i] = new line(new point(cut[0],cut[1]),new point(cut[2],cut[3]))
}
//Merke dir das Erste und Letzte Tor
let enter = hoops[0]
let out = hoops[size-1]
//Merke dir Linien,
//die zusammen mit dem Ersten und Letzten Tor ein Viereck bilden würden
let allSides = Array(2)
let swap = enter.a.dis(out.a)+enter.b.dis(out.b)<enter.a.dis(out.b)+enter.b.dis(out.a)
allSides[0] = swap?new line(enter.a,out.a):new line(enter.a,out.b)
allSides[1] = swap?new line(enter.b,out.b):new line(enter.b,out.a)
//Wenn die Seiten Parallel zum Ersten Tor sind, ist es nicht in einem Schlag lösbar
if(allSides[0].gradient == enter.gradient){
 throw new Error("Nicht machbar wegen position des Tores 1")
```

```
}
//Funktion um eine Seite so zu bewegen, dass die Seite durch den Punkt geht
let connected = false
function crop(end,side,other){
 let outPoint = out.crossPoint(side)
 let inverse = enter.crossPoint(other)
 let potential = new line(new line(outPoint,end).crossPoint(enter),outPoint)
 if(potential.cross(other)){
  potential = new line(new line(inverse,end).crossPoint(out),inverse)
  connected = true
 }
 return potential
}
//funktion um zu Testen, ob es möglich ist,
//mit einem 0 Großen ball in einem Schlag durch alle Tore zu Schießen
function moveSide(hoop, sides){
    //Zähle wie häufig das Tor die Seiten kreuzt
    let crosses = 2
    if(sides[0] instanceof line){
    crosses -= 1-sides[0].cross(hoop)
    }
    if(sides[1] instanceof line){
    crosses -= 1-sides[1].cross(hoop)
    }
    //Wenn nur eine Seite Gekreuzt wird,
    //wird die andere Seite zu dem Pfosten zwischen den Beiden Seiten bewegt
    if(crosses == 1){
    let change = +sides[0].cross(hoop)
    let crossPoint = sides[change].crossPoint(hoop)
    let end = hoop.a.dis(crossPoint)<hoop.b.dis(crossPoint)?hoop.a:hoop.b
    return [crop(end,sides[change],sides[1-change]),sides[1-change]]
    //Wenn keine Seite Gekreuzt wird, wird überprüft ob das Tor sich zwischen den Seiten befindet
    }else if(crosses == 0){
    let a = sides[0].crossPoint(hoop)
    let b = sides[1].crossPoint(hoop)
    //Wenn das Tor sich innerhalb befindet, werden die Seiten zu den Pfosten bewegt
    if(!(new line(hoop.a,a).touch(b)||new line(hoop.a,b).touch(a))){
        let c = hoop.a.dis(a)<hoop.b.dis(a)?hoop.a:hoop.b
        let d = hoop.a.dis(a)<hoop.b.dis(a)?hoop.b:hoop.a
        let out = [crop(c,sides[0],sides[1])]
        out.push(crop(d,sides[1],out[0]))
        if(!connected){
            out.push(crop(d,sides[1],sides[0]))
            out.push(crop(c,sides[0],out[2]))
        }
        return out
    //Wenn das Tor sich außerhalb befindet, bedeutet es, dass es für solche seiten unmöglich ist
    }else{
        return []
    }
 }
 return sides
}
//teste alle Tore vom Ersten zum Letzten und merke dir, wo sich die Seiten zum Schluss befanden
let sides = [...allSides]
for(let i = 1;i<hoops.length-1;i++){
 for(let j = 0;j<sides.length;j+=2){
  sides.splice(j,2,...moveSide(hoops[i],sides.slice(j,2)))
```

```
 }
}
let sidePot = [...sides]
//teste alle Tore vom Letzten zum Ersten und merke dir, wo sich die Seiten zum Schluss befanden
let save = enter
enter = out
out = save
sides = [...allSides]
connected = false
for(let i = hoops.length-2;i>0;i--){
    sides = moveSide(hoops[i],sides)
    for(let j = 0;j<sides.length;j+=2){
        sides.splice(j,2,...moveSide(hoops[i],sides.slice(j,2)))
    }
}
sidePot.push(...sides)
//Überprüfe ob es überhaupt möglich ist mit einem 0 Großen ball durchzuschießen
if(sidePot.length == 0){
 throw new Error("Nicht machbar wegen anordnung der Tore")
}
//Erschaffe linien, Parallel zum ersten und letzten Tor
let para = Array(2)
let border = Array(2)
let enterPara = enter.parallel(ball/2)
if(!(new line(enterPara.middle,out.middle).cross(enter))){
 enterPara = enter.parallel(-ball/2)
}
let outPara = out.parallel(ball/2)
let testLine = new line(outPara.middle,enterPara.middle)
if(testLine instanceof point||!testLine.cross(out)){
 outPara = out.parallel(-ball/2)
}
sidePot.push(enter.flip().parallel(-ball/2))
//function um zu überprüfen, ob der Ball zwischen den Parallelen durchgeschossen werden könnte
function check(){
 border[0] = new line(para[0].crossPoint(enterPara),para[0].crossPoint(outPara))
 border[1] = new line(para[1].crossPoint(enterPara),para[1].crossPoint(outPara))
 for(let i = 0;i<size;i++){
  if(!hoops[i].cross(border[0])||!hoops[i].cross(border[1])){
   return false
  }
 }
 return true
}
//Teste für jede seite, ob der Ball durchgeschossen werden könnte
let ballPass = false
for(let i = 0;i<sidePot.length;i++){
 if(sidePot[i] instanceof line){
  para[0] = sidePot[i]
  para[1] = sidePot[i].parallel(ball)
  if(check()){
   ballPass = true
   break
  }
  para[1] = sidePot[i].parallel(-ball)
  if(check()){
   ballPass = true
   break
```

```
  }
 }
}
//Wenn der Ball nirgends durchpasst, dann melde das
if(!ballPass){
 throw new Error(`Nicht machbar wegen breite des Balls`)
}
//Teste ob die Tore in der richtigen reihenfolge passiert werden
let a = hoops[0].crossPoint(border[0])
let b = (hoops[1]||hoops[0]).crossPoint(border[0])
for(let i = 2;i<size;i++){
 if(hoops[i].cross(new line(a,b))){
  throw new Error(`Nicht machbar wegen reihenfolge des Tores ${i+1}`)
 }
 b = border[0].crossPoint(hoops[i])
}
//Berechne die Schusslinie
let center = border[0].parallel(ball/2)
if(new line(center.middle,border[1].middle).cross(border[0])){
 center = border[0].parallel(-ball/2)
}
//Berechne den Abschusswinkel
let winkel = Math.atan(center.gradient)*(180/Math.PI)
if(center.crossPoint(enter).x<center.crossPoint(out).x){
 winkel = winkel<0?winkel+180:winkel-180
}
save = enter
enter = new line(border[0].crossPoint(out),border[1].crossPoint(out))
out = new line(border[0].crossPoint(save),border[1].crossPoint(save))
//Berechne Abschusspunkt
let punkt = outPara.crossPoint(center)
//Gebe alle werte auf zwei Nachkommastellen gerundet aus
let round = val => Math.round(val*100)/100
console.log(`Abschusspunkt: (${round(punkt.x)}|${round(punkt.y)}) Winkel: ${round(winkel)}\u00B0`)
```