1 Funktionsweise

Die grundlegende Idee hinter dieser Lösung der Aufgabe liegt darin sich die Tore als Fenster, die Geraden, welche auf den Toren liegen, als Wände und den Startpunkt des Balles als Glühbirne (Das Licht wird nicht reflektiert und bewegt sich ausschließlich geradlinig) vorzustellen. Falls die Glühbirne das letzte Fenster erleuchtet kann, gibt es gleichzeitig eine gerade Linie, welche durch alle Fenster (Tore) geht. Das Program teilt das erste Tor in endlich viele Segmente der Länge ϵ und schaut dann für jeden Anfangspunkt des Segmentes, ob man von dort aus das letzte Tor erreichen kann. Dabei wird wie folgt vorgegangen. Der Anfangspunkt hier ist eine Datenstruktur, welche die $x-$ und y-Koordinaten des Punktes enthält. Ein Tor ist eine Datenstruktur welche eine Liste von zwei Punkten $p[2]$, welche die Anfangs- und Endpunkte sind.

Algorithm 1 Teilschattenlösungsalgorithmus

```
1:  function SOLVE(Punkt s, Tor t[], int l)
2:      w ← atan2(l[1].p[0].y − s.y, l[1].p[0].x − s.x)
3:      v ← atan2(l[1].p[1].y − s.y, l[1].p[1].x − s.x)
4:      max ← max (w, v)
5:      min ← min (w, v)
6:      if l < 2 then return min
7:      end if
8:      for i ← 2; i < l; i ← i + 1 do
9:          w ← atan2(l[i].p[0].y − s.y, l[i].p[0].x − s.x)
10:         v ← atan2(l[i].p[1].y − s.y, l[i].p[1].x − s.x)
11:         cmax ← max (w, v)
12:         cmin ← min (w, v)
13:         if cmin > max then return −10
14:         end if
15:         if cmax < min then return −10
16:         end if
17:         if cmin < min then cmin ← min
18:         end if
19:         if cmax > max then cmax ← max
20:         end if
21:         max ← cmax
22:         min ← cmin
23:     end for
24: end function
```

Man stelle sich einen Kreis vor, dessen Mittelpunkt bei dem Punkt `s` liegt und einen Punkt `t` schneidet. Der Winkel der Zentrale welche `t` schneidet wird in den Zeilen 2, 3, 9, **??**, wobei hier `t` der Start- und Endpunkt eines Tores und `s` der gegebene Startpunkt ist. Bei der n-ten Iteration wird der maximale und minimale Winkel einer Zentrale errechnet, welche alle vorherigen $n-1$ und das jetzige n-te Tor schneiden. Beim durchlaufen aller Zentralen von 1 bis n findet sich ein geeigneter Winkel für alle Tore beim gegebenen Startpunkt. Die Funktion gibt den Winkel zurück.

Die Neuanpassung der Tore um die Größe des Balles zu berücksichtigen funktioniert auf Basis des Strahlensatzes. Man stelle sich das Tor als Hypothenuse eines rechtwinkligen Dreieckes vor, wobei die x-Achse die Ankathete und die y-Achse die Gegenkathete ist. Da der Ball durch das Tor schreiten kann, wenn die Entfernung des Schnittpunkt des Tores und des Balles größer oder gleich seinem Radius sind, muss errechnet werden, welche Veränderung der x-Achse jeden möglichen Schnittpunkt so weit von den tatsächlichen Pfosten entfernt, dass er hindurchpasst. Dadurch wird bei dem Dreieck auf der Hypothenuse von dem Schnittpunkt dieser mit der Ankathete nach $\frac{d}{2}$ Schritten ein Punkt A eingezeichnet. Dieser wird mit dem direkt darunter liegenden Punkt B auf der Ankathete verbunden, sodass eine V-Figur entsteht. Der gesuchte Wert x' ist nun die Entfernung der Ankathete zu dem auf ihr eingezeichneten Punkt B. Die aus dem Strahlensatz folgende Formel $x' = \frac{r * \Delta x}{\sqrt{\Delta x^2 + \Delta y^2}}$, wobei Δx die Länge und Δy die Höhe des Dreiecks, r den Radius des Balls.
Mithilfe der oben genannten Formel wird jedes Tor vorneherein bei der Einlese angepasst. Falls ein Tor zu schmal ist, beendet das Programm vorzeitig.

2 Umsetzung

2.1 Datenstrukturen

`point`	Speichert die x- und y-Koordinaten.
`gate`	Speichert genau zwei Punkte, stellt das Tor dar.

2.2 Funktionen

`solve(struct point start, const struct gate *gates, size_t n)`	Setzt den oben beschriebenen Algorithmus und errechnet den notwendigen Winkel um das letzte Tor sowie alle vorherigen von einem gegebenen Startpunkt zu durchschreiten.
`open_file(char *path)`	Liest die bei dem Kommandozeilenparameter gegebenen Pfad angegebene Datei ein und erstellt so die Liste von Toren und passt diese gleichzeitig an den Radius des Balles an.
`prepare_gate(struct gate *gate)`	Passt die Tore so an, dass der Ball hindurchpasst. Funktioniert nach dem oben beschriebenen Ansatz mithilfe des Strahlensatzes.
`main(int argc, char **argv)`	Herzstück des Programmes.

Die Mainfunktion ruft zuerst die Funktion `open_file` auf, wobei dieser als Parameter der erste String der Kommandozeile gegeben wird. Diese liest nun die Variablen `n` und `r` aus und liest alle Koordinaten in eine Liste von `n` Toren ein. Danach rechnet wird in der Mainfunktion ausgerechnet, wieviel man sich in x-Richtung bewegen muss, damit man bei der Traversierung von dem ersten Tor genau ϵ geht. Nun geht es in eine Schleife hinein, in der zuerst der Startpunkt berechnet wird, je nach dem ob das erste Tor senkrecht steht oder nicht. Bei einem nicht senkrecht stehendem Tor wird mithilfe des vorher ausgerechneten Werten jeweils ein Punkt errechnet, welcher von allen anderen zukünftig und schon berechneten Punkten einen Abstand von ϵ hat. Die Funktion `solve` wird nun mit dem Punkt, der Liste von Toren und der Anzahl Tore `n` aufgerufen. Es wird dann geprüft, ob der Rückgabewert größer als $-\pi$ ist, da es dann eine Lösung gibt. Zusammen mit der x und y Koordinate wird der errechnete Winkel in Grad umgewandelt und auf die Konsole ausgegeben. Die Berechnung wird abgebrochen. Falls die Rechnung nicht erfolgreich war, wird der nächste Startpunkt ausgerechnet bis entweder eine Lösung gefunden wurde oder alle Segmente durchquert wurden. Eine mögliche Verbesserung wäre hier eine Granularisierung, wo zuerst ein relativ hohes ϵ gesetzt wird, was nach einem anderen Parameter schrittweise verkleinert wird.

3 Beispiele

Dateiname	Ausgabe des Programms	Laufzeit
krocket1.txt	(x: 11.181292 — y: 9.755150), Winkel: 25.823171 Grad	$0.004s$
krocket2.txt	—	$0.002s$
krocket3.txt	(x: 21.730476 — y: 110.456099), Winkel: 0.073520 Grad	$0.010s$
krocket4.txt	(x: 6.124292 — y: 153.499956), Winkel: 27.037373 Grad	$0.006s$
krocket5.txt	(x: 867.102632 — y: 13907.777711), Winkel: -11.531236 Grad	$0.041s$

4 Quellcode

Epsilon kann als Compilerflag `-DEPSILON=` gesetzt werden, wobei nach dem `=` der Wert kommen soll.

4.1 `main`

```
int main(int argc, char **argv)
{
 if (argc < 2) {
  fprintf(stderr, "Verwendung: fünf [Pfad zur Datei]\n");
```

```
 return 1;
}

struct gate *gates = open_file(argv[1]);
if (gates == NULL) {
 return 2;
}
if (n < 1) {
 free(gates);
 return 0;
}
double cx = MIN(gates[0].point[0].x, gates[0].point[1].x);
double mx = MAX(gates[0].point[0].x, gates[0].point[1].x);

double cy = MAX(gates[0].point[0].y, gates[0].point[1].y);
double my = MIN(gates[0].point[0].y, gates[0].point[1].y);

double m = slope(gates[0].point[0], gates[0].point[1]);
double t = elevation(gates[0].point[0], m);

double plus = EPSILON / (sqrt(1 + m * m));

char loop = 1;

while(loop) {
 struct point point;
 if (cx == mx) {
  loop = cy > my ? 1 : 0;
  point = (struct point) {cx, cy};
  cy -= EPSILON;
 } else {
  loop = cx < mx ? 1 : 0;
  point = (struct point) {cx, cx * m + t};
  cx += plus;
 }
 double solution = 0;
 if ((solution = solve(point, gates, n )) > -  M_PI)  {
  printf("(x: %lf | y: %lf), Winkel: %lf Grad\n ", point.x, point.y, solution *57.2958);
  break;
 }
}

free(gates);


return 0;
}
```

4.2 solve

```
/*  Funktion solve:
 *
 *  Parameter:
 *   struct point start   der Punkt, für welchen ausgerechnet wird,
 *       ob von ihm aus ein Ziel erreicht werden kann.
 *   const struct gate *gates   ein Feld, welches die Koordinaten der Anfangs-
 *       und Endpunkte der jeweiligen Ziele enthält.
 *      Es muss mindestens ein Eintrag am Pointer enthalten sein.
 *
 *
 *   size_t  n    anzahl Elemente in dem Feld ,,gates''
```

```
 *  Rückgabewert:
 *   Falls erfolgreich: Winkel der Gerade, welche das Ziel erreicht (Mittelwert des größtmöglichen und
 *       kleinstmöglichen Winkels)
 *   Falls nicht erfolgreich / oder n < 2: -1
 *
 */

double solve(struct point start, const struct gate *gates, size_t n)
{

 double angles[2];
 for (int j = 0; j < 2; j++) {
  angles[j] = atan2(gates[1].point[j].y - start.y, gates[1].point[j].x - start.x);
 }
 double lower = MIN(angles[0], angles[1]);
 double upper = MAX(angles[0], angles[1]);



 for (size_t i = 2; i < n; i++) {
  for (int j = 0; j < 2; j++) {
   angles[j] = atan2(gates[i].point[j].y - start.y, gates[i].point[j].x - start.x);
  }


  double clower = MIN(angles[0], angles[1]);
  double cupper = MAX(angles[0], angles[1]);



  if (clower > upper) {
   return -5;
  }

  if (cupper < lower) {
   return -5;
  }

  if (clower < lower) {
   clower = lower;
  }


  if (upper < cupper) {
   cupper = upper;
  }

  upper = cupper;
  lower = clower;
 }
 // Der Winkel der in der Mitte liegt.

 return  (lower + upper) / 2;
}
```

4.3 prepare_gate

```
/* Funktion prepare_gate:
 *  Verändert die x-Koordinate so, dass der Ball durch das Tor hindurchpasst.
 *  Parameter:
 *   struct gate *gate   Das zu verändende Tor
 *  Rückgabewert:
```

```
 *   int     Erfolg
 *
 *
 */


char prepare_gate(struct gate *gate)
{

 // Falls ein Tor komplett senkrecht steht.
 if (gate->point[0].x == gate->point[1].x) {
  gate->point[0].y -= r;
  gate->point[1].y += r;
  return 1;
 }

 double x = fabs(gate->point[0].x - gate->point[1].x);
 double y = fabs(gate->point[0].y - gate->point[1].y);
 if (sqrt(x * x + y * y) < 2 * r) {
  printf("Es gibt mindestens ein Tor, durch welches der Ball nicht hindurchpasst.\n");
  return 0;
 }
 double g = (r * x) / sqrt(x * x + y * y);
 if (gate->point[0].x > gate->point[1].x) {
  gate->point[0].x -= g;
  gate->point[1].x += g;
 } else {
  gate->point[1].x -= g;
  gate->point[0].x += g;
 }
 return 1;
}
```

4.4 Datenstrukturen

```
struct point {
 double x;
 double y;
};

struct gate {
 struct point point[2];
};
```