

Aufgabe 3: Die Siedler

Teilnahme-ID: *****

Bearbeiter dieser Aufgabe:

14. April 2024

Inhaltsverzeichnis

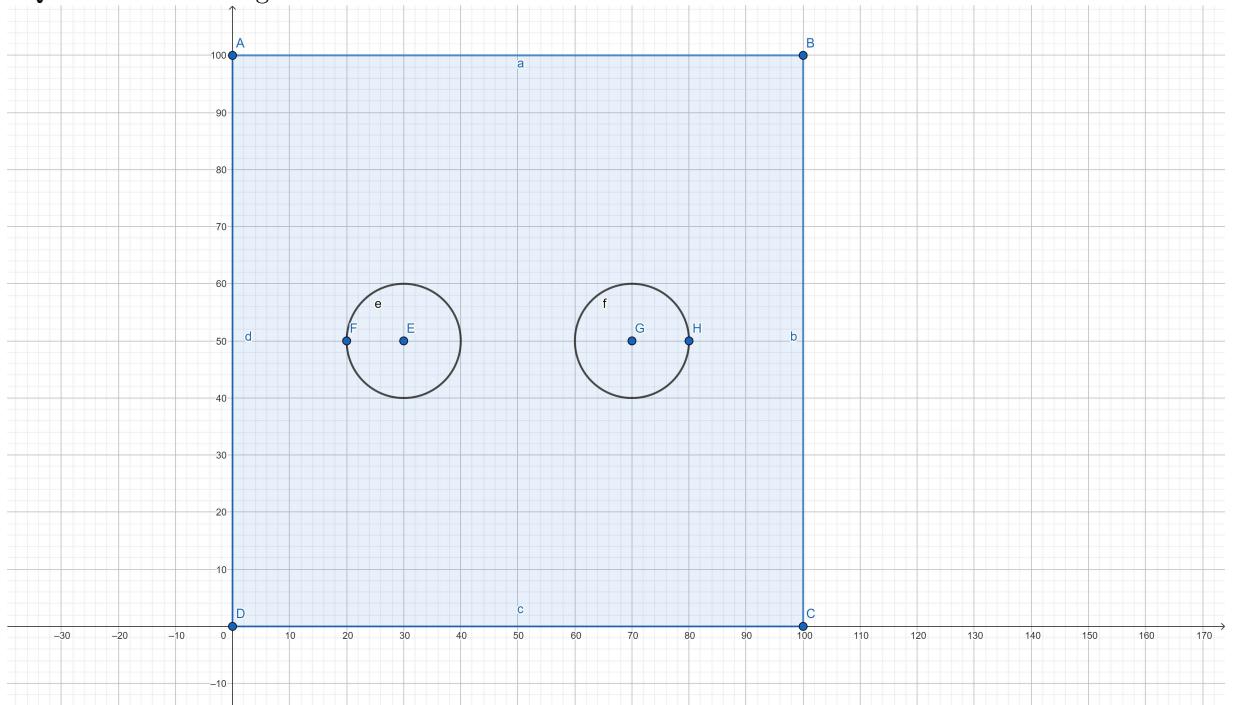
1 Das Problem visualisiert	2
1.1 Polygone schließen	2
1.2 Hinweise	3
2 Lösungsidee	3
2.1 Das Gesundheitszentrum	4
2.1.1 Ortschaften in das Gesundheitszentrum planen - Möglichkeit 1	4
2.1.2 Überschneidende Kreise oder nicht?	5
2.1.3 Das Gesundheitszentrum in das Gebiet planen	6
2.1.4 Punkt im Polygon oder nicht? - Der Raycasting Algorithmus	9
2.2 Ortschaften am Rand des Gebiets planen	13
2.3 Ortschaften in das Gebiet planen - Möglichkeit 1	16
3 Optimierungen	17
3.1 Ortschaften in das Gesundheitszentrum planen - Möglichkeit 2	17
3.2 Ortschaften in das Gebiet planen - Möglichkeit 2	18
3.3 Ortschaften in das Gebiet planen - Noch mehr Möglichkeiten	19
4 Laufzeitanalyse	19
5 Speicheranalyse	20
6 Rundungsfehler	20
7 Geometrie App	21
8 Beispiele	21
8.1 siedler1.txt	21
8.2 siedler2.txt	22
8.3 siedler3.txt	22
8.4 siedler4.txt	22
8.5 siedler5.txt	22
8.6 siedler6.txt	22
8.7 siedler7.txt	23
9 Quellcode	24
10 Quellen	26

Aufgabe 3: Die Siedler

1 Das Problem visualisiert

Gegeben ist ein Polygon mit n Ecken. Innerhalb des Gebiet sollen so viele Ortschaften wie möglich gesetzt werden. Grundsätzlich muss eine Ortschaft einen Abstand von 20km zu jeder anderen Ortschaft haben. Doch alle Ortschaften, die höchstens 85km vom Gesundheitszentrum entfernt sind, brauchen nur einen Abstand von 10km zu jeder anderen Ortschaft. Das Ziel: So viele Ortschaften wie möglich in das Gebiet planen → Maximierungsproblem.

Die Ortschaften werden als Kreise dargestellt. Ein Kreis bestimmt die Fläche, in der keine andere Ortschaft existieren darf, da sonst sich die Krankheit ausbreitet. Zur Vereinfachung der Abbildung wird ein Quadrat als Gebiet genutzt.



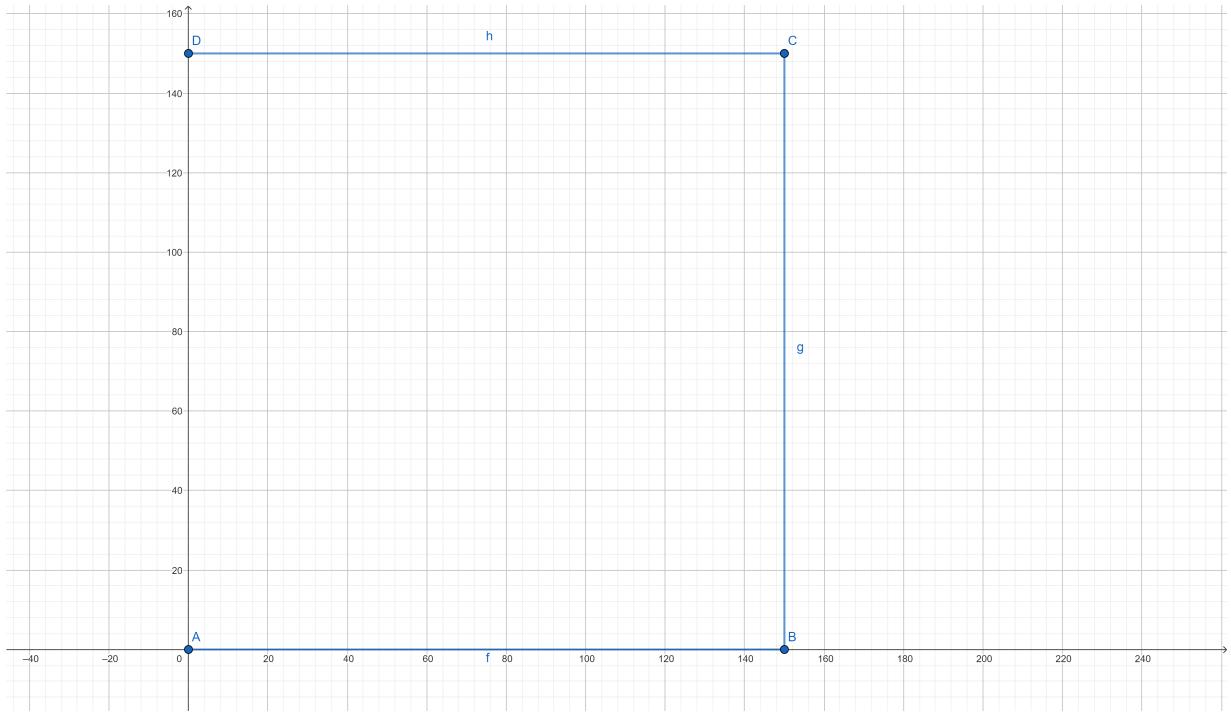
Als Beispiel wurden hier zwei Ortschaften als Kreise gezeichnet. Beide Ortschaften betragen einen Radius von 10 (und somit einen Durchmesser von 20). E und G beschreiben jeweils die Mittelpunkte von e und f. Der euklidische Abstand von $|\vec{EG}|$ beträgt 40.

$$\begin{aligned}\vec{EG} &= \begin{pmatrix} 30 \\ 50 \end{pmatrix} - \begin{pmatrix} 70 \\ 50 \end{pmatrix} = \begin{pmatrix} -40 \\ 0 \end{pmatrix} \\ |\vec{EG}| &= \sqrt{(-40)^2 + 0^2} = 40\end{aligned}$$

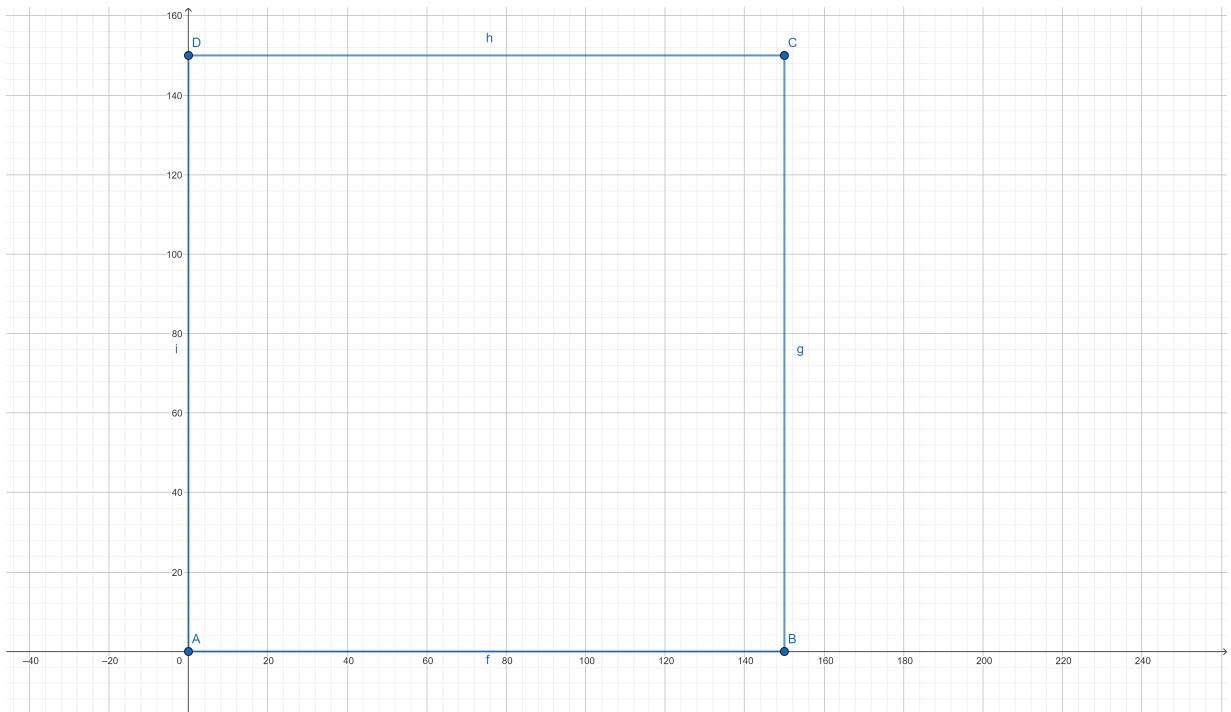
1.1 Polygone schließen

Beim Zeichnen der gegebenen Polygone (siedler1.txt - siedler5.txt) fällt auf, dass die Polygone 'offen' bleiben, sprich es fehlt eine letzte Kante. Das Polygon lässt sich schließen, wenn der letzte Punkt in der Datei mit dem ersten Punkt verbunden wird.

Aufgabe 3: Die Siedler



Wird zu:



Aus diesem Grund wird zu jeder Liste von Punkten der erste Punkt hinten angehängt (Dies erfolgt automatisiert durch Code und nicht manuell!)

1.2 Hinweise

Im weiteren wird eine Ortschaft als Kreis und das Gebiet als Polygon bezeichnet.

2 Lösungsidee

- Den Kreis des GZ (Radius 85) mit möglichst vielen kleineren Kreisen (Radius 5) packen

Aufgabe 3: Die Siedler

2. Den Kreis des GZ (Radius 85) in das Polygon so setzen, dass es möglichst viel Fläche einnimmt
3. Das Polygon mit möglichst vielen Kreisen packen (Radius 10)

Dabei dürfen die Kreise am Rand des Polygon liegen, aber sich niemals mit anderen Kreisen (GZ ausgeschlossen) überschneiden!

Das Problem gehört zu geometrischen Packproblemen. Es wurde bewiesen, dass diese Klasse von Problemen in NP-schwer liegt.

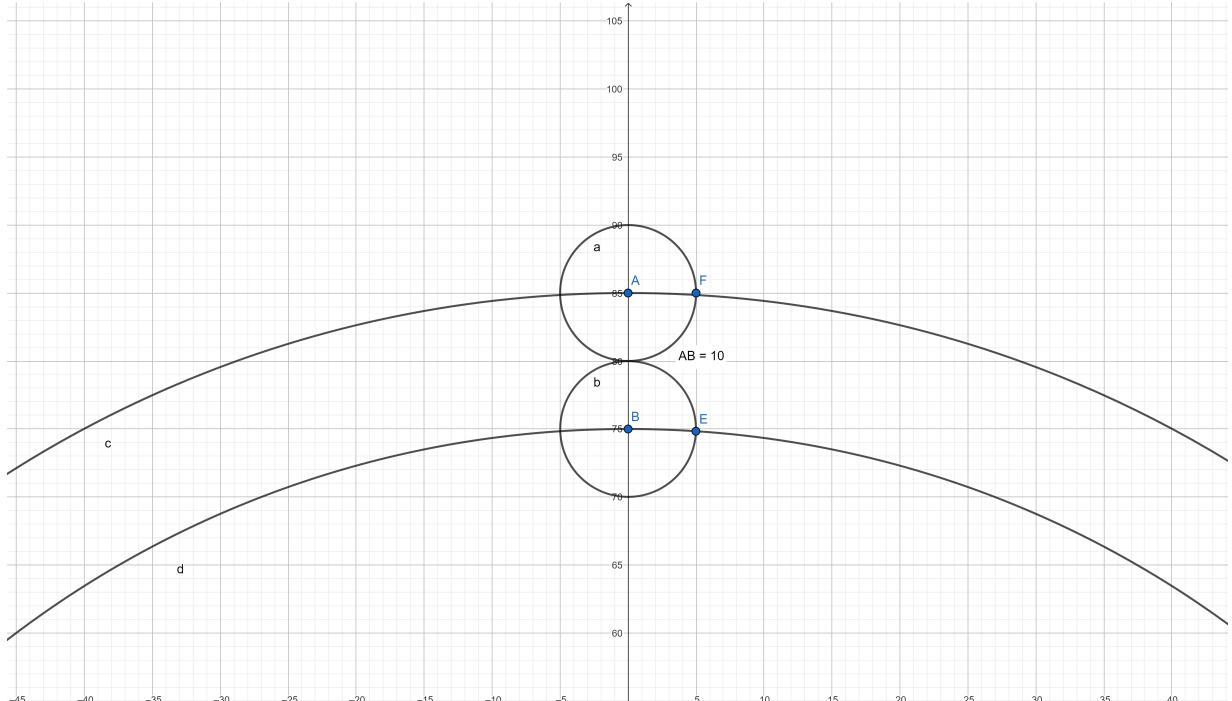
2.1 Das Gesundheitszentrum

Glücklicherweise hat die Regierung genug Geld um genau 1 GZ (Gesundheitszentrum) zu bauen. Es ermöglicht mehr Ortschaften insgesamt zu planen, weil die Ortschaften innerhalb des Kreises des GZ näher aneinander liegen dürfen (min. 10km Abstand). Das Problem: Wie kann man die Anzahl an kleineren Kreisen in einem größeren Kreis maximieren ohne, dass sich die kleinen Kreise überschneiden?

2.1.1 Ortschaften in das Gesundheitszentrum planen - Möglichkeit 1

Hinweis Es wird davon ausgegangen, dass die Ortschaften keinen Mindestabstand zum GZ haben.

Um so viel Platz wie möglich zu sparen, sollten so viele Kreise wie möglich am Bogen des GZ liegen. Deswegen erstmal mehrere kleine Kreise an den Bogen gesetzt. Dann kann der Radius (r_{GZ}) des großen Kreises um genau 10km verkleinert werden, um weitere kleine Kreise am Bogen zu setzen. Es muss um mindestens 10km vekleinert werden, damit der Abstand von 10km zwischen den Mittelpunkten der kleinen Kreisen besteht.



Es gilt:

$$r_1 + r_2 \leq |M_1 M_2|$$

Zuerst müssen die Polarkoordinaten des großen Kreises berechnet. Diese werden danach in kartesische Koordinaten umgewandelt.

Aufgabe 3: Die Siedler

$$\begin{aligned}\alpha &\in [0; 359] \\ x &= r_{GZ} \cdot \cos(\alpha) \\ y &= r_{GZ} \cdot \sin(\alpha)\end{aligned}$$

An der Position (x, y) wird ein kleiner Kreis gesetzt. Von Winkel 0° bis 359° im 1er Schritt des großen Kreis wird der Prozess wiederholt. Dabei wird jedes mal überprüft ob sich der Kreis mit einem anderen kleinen Kreis überschneiden würde. Das sorgt dafür, dass die Krankheit zwischen Ortschaften nicht ausbricht und sich verbreitet.

Algorithm 1 GZ_planen()

```

1: ortschaftenGZ  $\leftarrow$  Liste aller Ortschaften im GZ (leer)
2: for radius = 85; radius > 0; radius -= 10 do
3:   winkel  $\leftarrow$  0
4:   while winkel < 360 do
5:     naechsteOrtschaft  $\leftarrow$  neuer Kreis(Mittelpunkt( $\cos winkel$ ,  $\sin winkel$ ), Radius(10))
6:     if ueberschneidet(naechsteOrtschaft, ortschaftenGZ) == false then
7:       ortschaftenGZ.add(naechsteOrtschaft)
8:     end if
9:     winkel++
10:   end while
11: end for
```

Überschneidungen zweier Kreise bedeutet, dass nicht genug Abstand zwischen zwei Ortschaften ist. Die Methode *ueberschneidet(...)* wird im Folgenden erklärt.

2.1.2 Überschneidende Kreise oder nicht?

Es ist bekannt, dass wenn

$$r_1 + r_2 > |\overrightarrow{M_1 M_2}|$$

dann zwei Kreise sich überschneiden. Besonders wichtig ist dies bei engen Ecken des Polygon.

Brute Force Eine Möglichkeit wäre, den nächsten Kreis, der gesetzt werden soll, mit jedem anderen Kreis zu vergleichen. Dementsprechend wäre das ein Brute Force.

Mindestabstände:

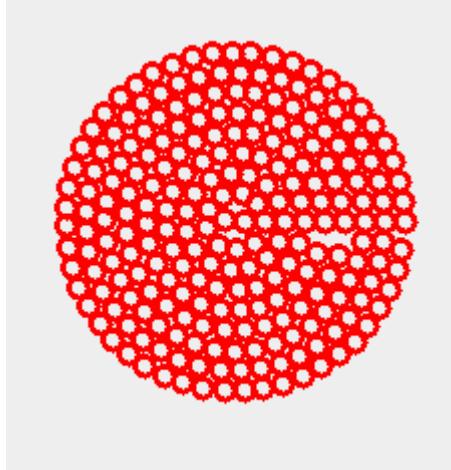
	Ortschaft ₁ im GZ	Ortschaft ₁ nicht im GZ
Ortschaft ₂ im GZ	10km	15km
Ortschaft ₂ nicht im GZ	15km	20km

Aufgabe 3: Die Siedler

Algorithm 2 überschneidet(Kreis aktuelleOrtschaft, List<Kreis> ortsschaften, List<Kreis> ortsschaftenGZ) returns boolean

```
1: for Kreis ortsschaft in ortsschaften do
2:   if aktuelleOrtschaft ∈ ortsschaftenGZ ∧ ortsschaft ∈ ortsschaftenGZ then
3:     if euklidischer_abstand(aktuelleOrtschaft, ortsschaft) < 10 then
4:       return false
5:     end if
6:   end if
7:   if aktuelleOrtschaft ∉ ortsschaftenGZ ∧ ortsschaft ∈ ortsschaftenGZ then
8:     if euklidischer_abstand(aktuelleOrtschaft, ortsschaft) < 15 then
9:       return false
10:    end if
11:  end if
12:  if aktuelleOrtschaft ∉ ortsschaftenGZ ∧ ortsschaft ∉ ortsschaftenGZ then
13:    if euklidischer_abstand(aktuelleOrtschaft, ortsschaft) < 20 then
14:      return false
15:    end if
16:  end if
17: end for
```

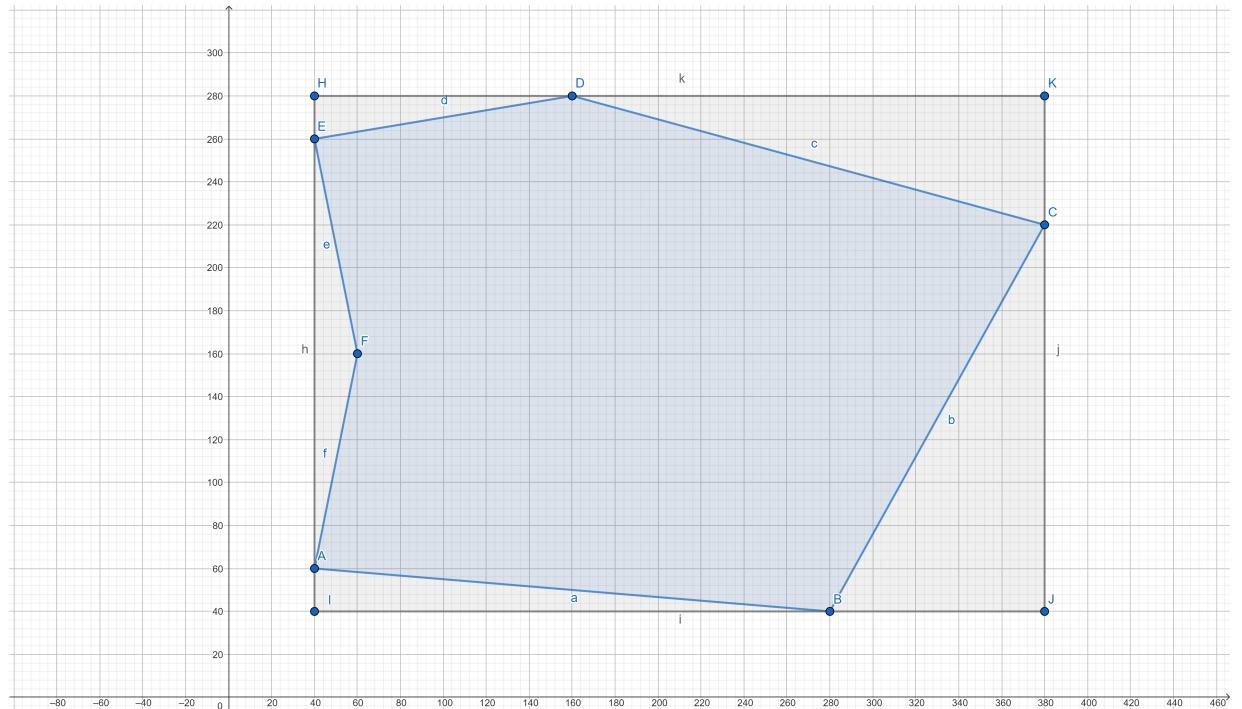
Um das Gesundheitszentrum zu planen ist vorerst nur der erste Fall wichtig
Mögliches Ergebnis (242 Ortschaften):



2.1.3 Das Gesundheitszentrum in das Gebiet planen

Es ist optimal wenn das GZ eine möglichst große Fläche einnimmt, denn innerhalb des GZ dürfen wie erwähnt mehr Ortschaften als sonst sein. Eine mögliche Lösung ist, das GZ auf mehrere Flächen des Polygons zu legen und dann rauszufiltern, wo die meisten kleinen Kreise aus dem GZ im Polygon liegen. Um auch mindestens einmal das komplette Polygon mit einem möglichen GZ an einer Fläche durchzuscanen, sollte das Polygon in ein Rechteck gesteckt werden.

Aufgabe 3: Die Siedler



Somit kann von links nach rechts und von oben nach unten iteriert werden um Überprüfungen zu starten.

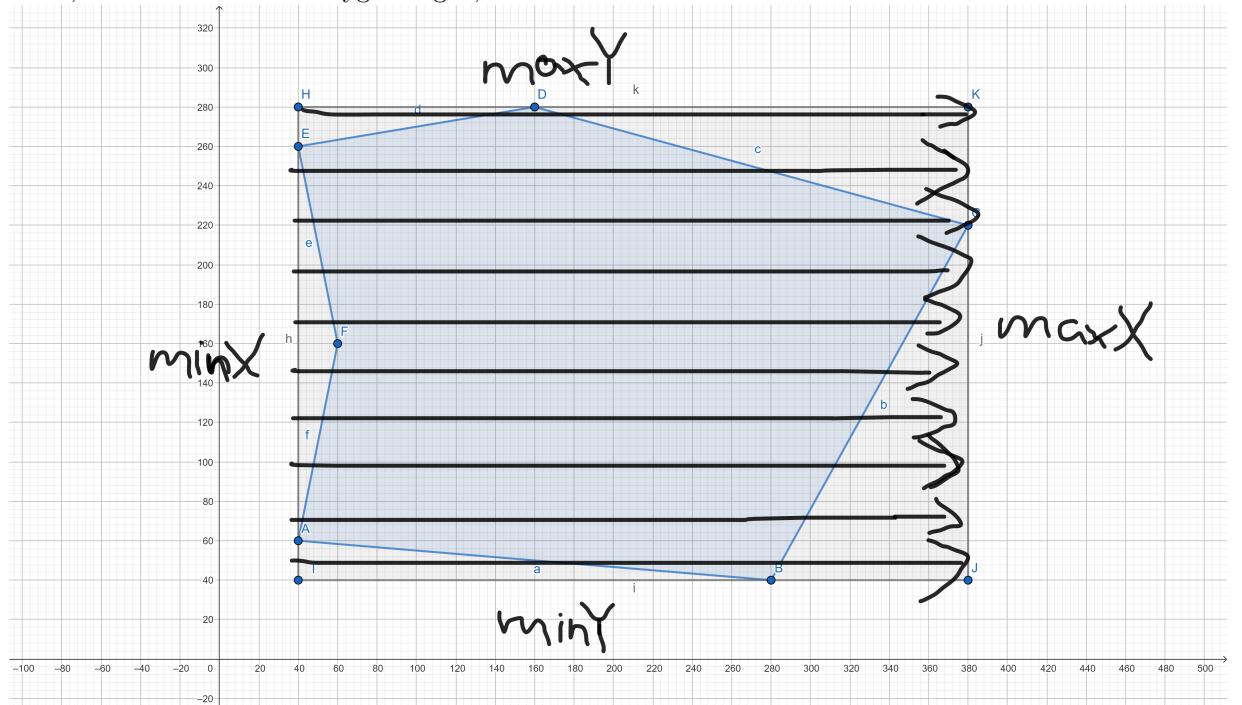
Um so ein Recht zu bilden, das auch eng am Polygon liegt, müssen diese Punkte gefunden werden:

Der kleinste und größte Punkt auf der x-Achse ($\min X$, $\max X$)

Der kleinste und größte Punkt auf der y-Achse ($\min Y$, $\max Y$)

Das geschieht durch einfaches Filtern durch die Punkte des Polygons.

Als nächstes wird ein Punkt im Polygon gesucht, wo das GZ gesetzt wird wobei die Anzahl an kleinen Kreisen, die im GZ und im Polygon liegen, maximal ist.



Aufgabe 3: Die Siedler

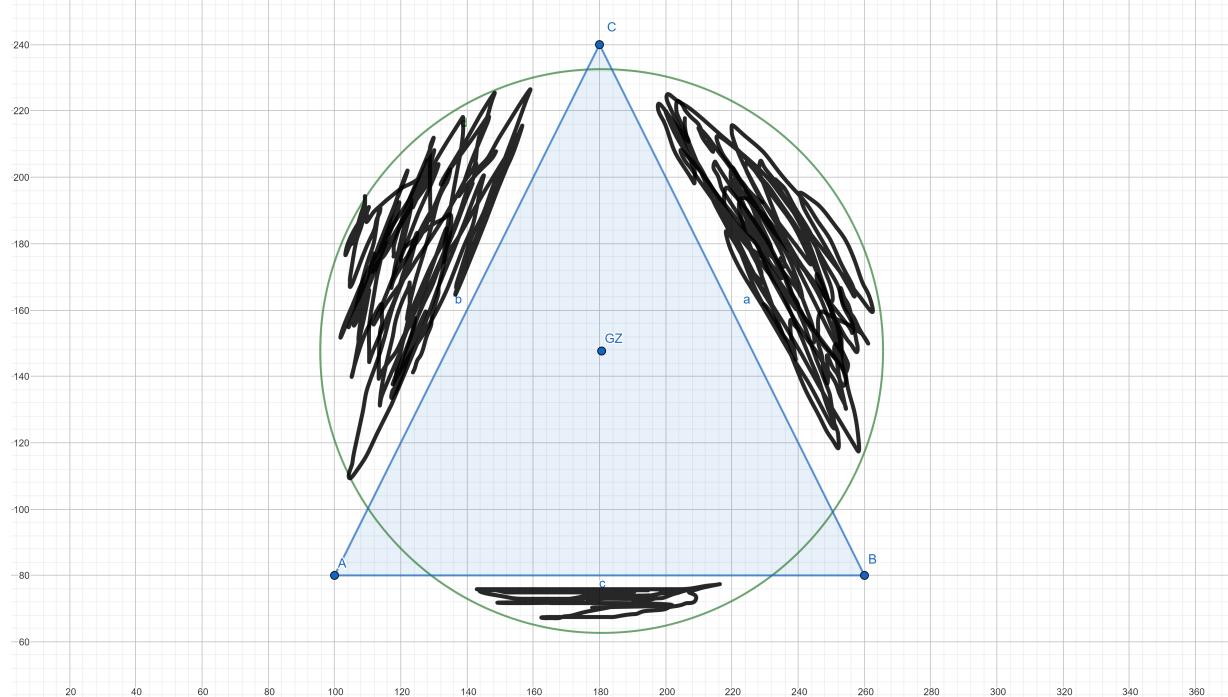
Algorithm 3 GZ_planen(List<Punkt> polygon)

```

1: //.....Vorher erarbeitete Logik.....
2: minX ← Kleinster x-Wert in polygon
3: maxX ← Größter x-Wert in polygon
4: minY ← Kleinster y-Wert in polygon
5: maxY ← Größter y-Wert in polygon
6: besteOrtschaftenGZimPolygon
7: aktuellerPunkt ← Punkt(minX, minY) //Startet von unten links des Rechtecks
8: while aktuellerPunkt.y < maxY do
9:   while aktuellerPunkt.x < maxX do
10:    ortschaftenGZimPolygon //Liste aller Ortschaften die im GZ UND im Polygon sind
11:    if imPolygon(aktuellerPunkt, polygon) then
12:      for ortschaft in ortschaftenGZ do
13:        naechsteOrtschaft ← ortschaft mit Versatz von aktuellerPunkt
14:        if imPolygon(Punkt(naechsteOrtschaft.mPunkt.x, naechsteOrtschaft.mPunkt.y)) then
15:          ortschaftenGZimPolygon.add(naechsteOrtschaft);
16:        end if
17:      end for
18:      //Mehr Kreise als vorher?
19:      if ortschaftenGZimPolygon.size() > besteOrtschaftenGZimPolygon.size() then
20:        besteOrtschaftenGZimPolygon = ortschaftenGZimPolygon
21:      end if
22:      aktuellerPunkt.x++
23:    end while
24:    aktuellerPunkt.x = minX
25:    aktuellerPunkt.y++
26:  end while

```

Beim Planen des GZ in das Polygon werden einige Ortschaften nicht miteingeplant werden können solange der Kreis des GZ (mit Ortschaften am Bogen) vom Polygon nicht komplett eingeschlossen ist.

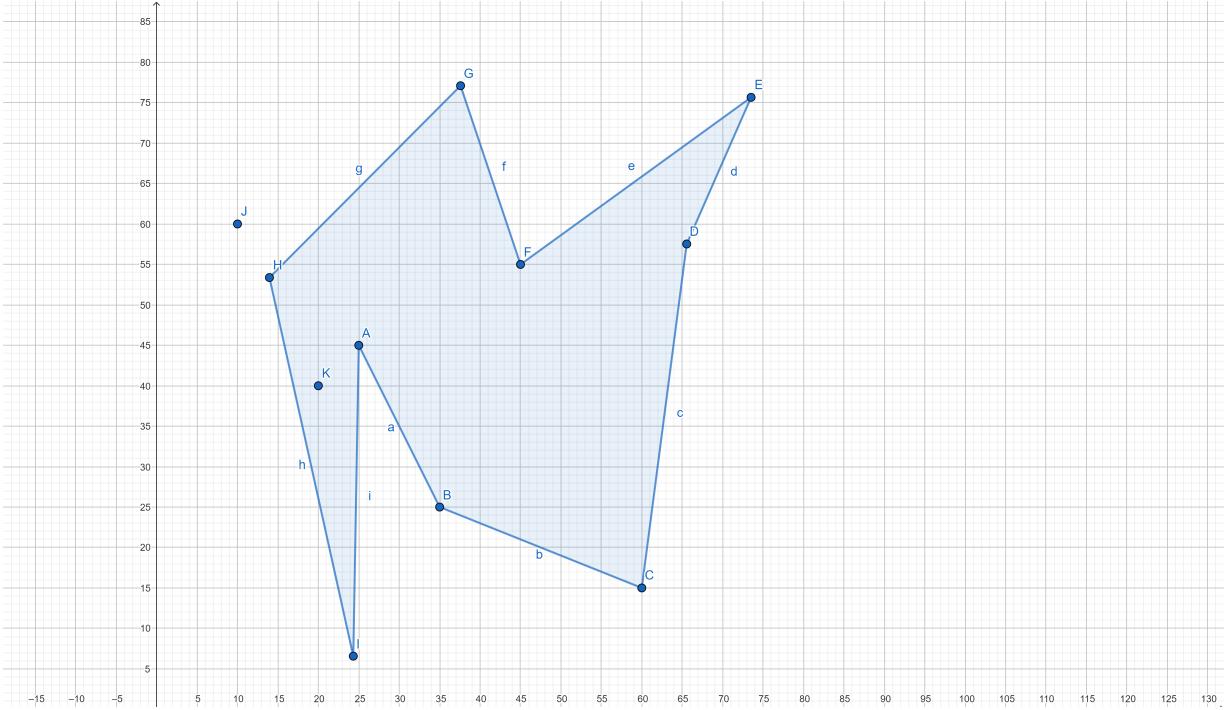


Alle Ortschaften, die im schwarz markierten Bereich sind, fallen raus
Die Methode *imPolygon(...)* wird im Folgenden erklärt.

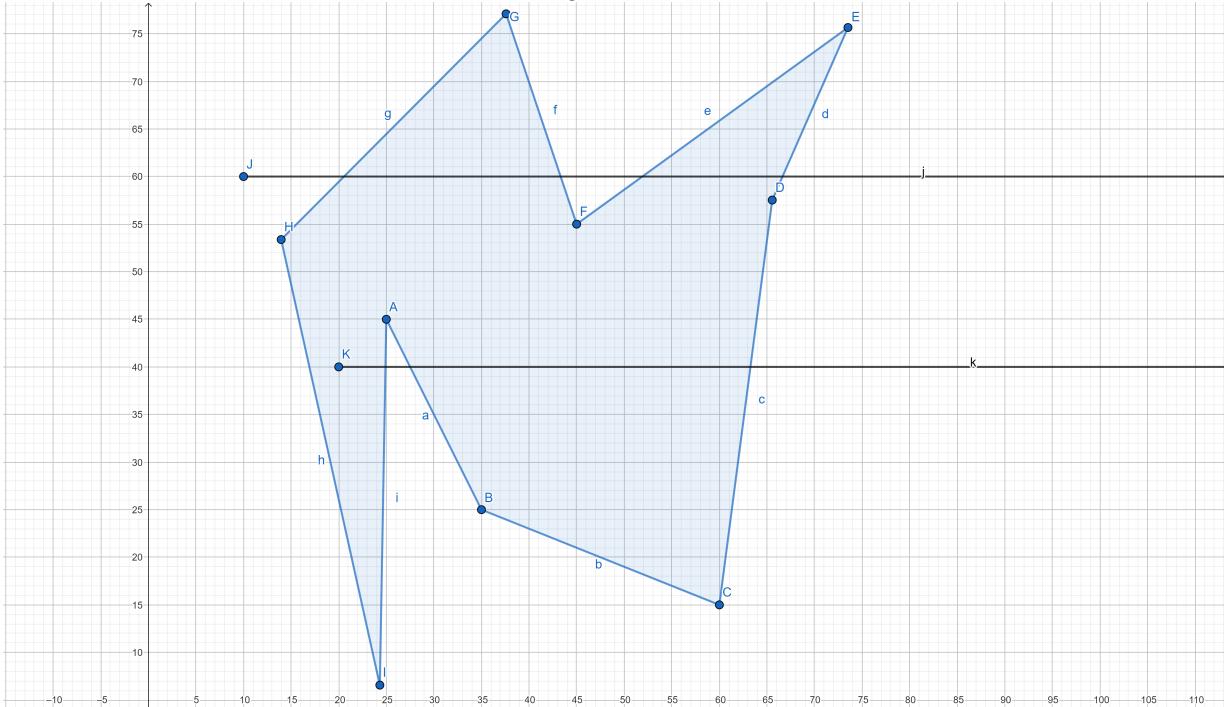
Aufgabe 3: Die Siedler

2.1.4 Punkt im Polygon oder nicht? - Der Raycasting Algorithmus

Visuell kann ganz einfach geprüft werden ob ein Punkt in einem Polygon ist oder nicht.



Punkt J ist außerhalb und Punkt K innerhalb - ganz einfach.



Um nochmal ganz sicher zu sein, kann ein Strahl an den Punkten nach rechts gezeichnet werden. Jetzt wird gezählt, wie oft der Strahl durch die Kanten des Polygon geht. Falls die Zahl gerade ist, dann ist der Punkt nicht im Polygon. Falls die Zahl ungerade ist, dann ist der Punkt im Polygon. Der Strahl von Punkt J geht durch 4 Kanten → nicht im Polygon. Der Strahl von Punkt K geht durch 3 Kanten → im Polygon. Genau diese Logik wird im Folgenden umgesetzt.

Falls eines der Fälle für die Kante AB ($y_A < y_B$)¹ mit dem Strahl von Punkt P nach $x \rightarrow \infty$ zutrifft, dann geht der Strahl nicht durch die Kante.

¹Falls $y_A > y_B$, dann wird der Algorithmus nochmals mit vertauschten Punkten aufgerufen

Aufgabe 3: Die Siedler

$$x_P \geq \max(x_A, x_B)$$

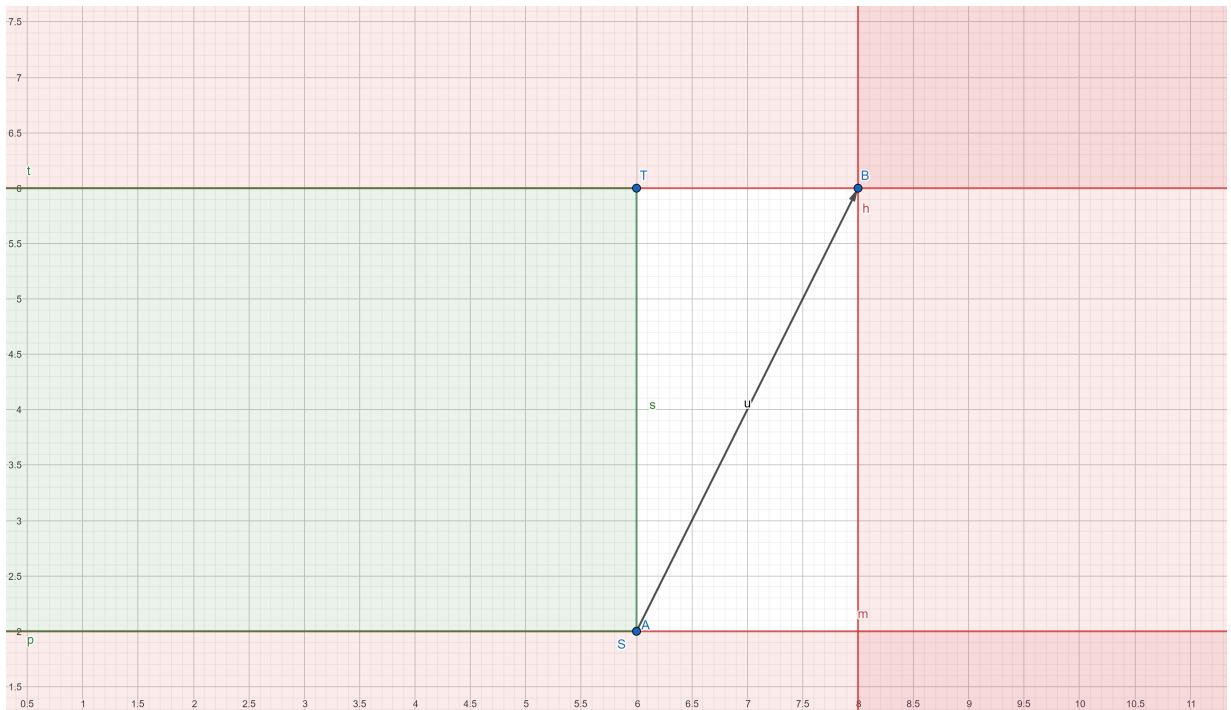
$$y_P > y_B$$

$$y_P < y_A$$



Für diesen Fall geht der Strahl definitiv durch die Kante, wenn die vorher gelisteten Fälle nicht zutreffen:

$$x_P < \min(x_A, x_B)$$



Für den weißen Bereich müssen die Steigungen AB und AP verglichen werden.

Aufgabe 3: Die Siedler

$$m_{AB} = \frac{y_B - y_A}{x_B - x_A}$$

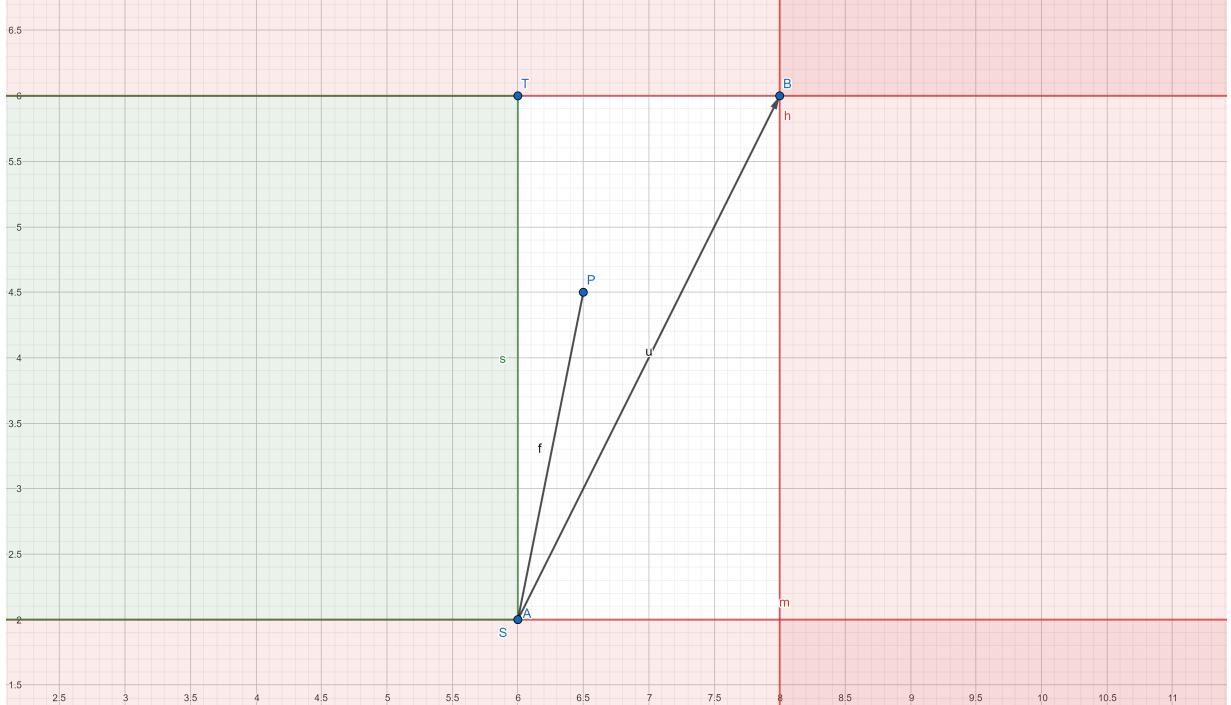
$$m_{AP} = \frac{y_P - y_A}{x_P - x_A}$$

Strahl geht durch AB:

$$m_{AB} < m_{AP}$$

Strahl geht nicht durch AB:

$$m_{AB} \geq m_{AP}$$



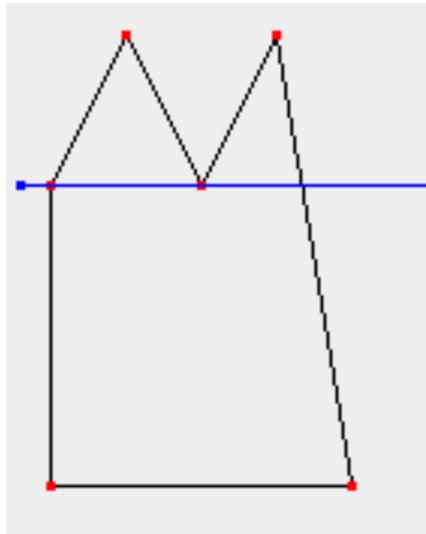
$$m_{AB} = \frac{6 - 2}{8 - 6} = 2$$

$$m_{AP} = \frac{4.5 - 2}{6.5 - 6} = 5$$

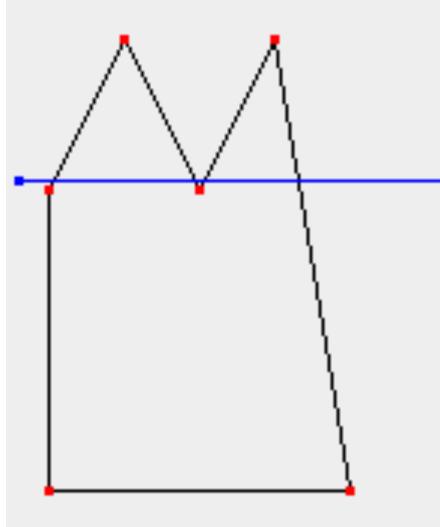
Der Strahl geht hierbei durch AB, da $m_{AB} < m_{AP}$

Edge Case Falls $y_P = y_A$ oder $y_P = y_B$ ist, dann muss y_P minimal erhöht werden.

Aufgabe 3: Die Siedler



Der blaue Strahl geht hierbei durch zwei Ecken und einer Kante. Eine Ecke zählt zweimal als Kante. Somit wird der Zähler auf 5 stehen ($2 \cdot 1 + 2 \cdot 1 + 1$). 5 ist ungerade, also ist der blaue Punkt im Polygon. Das stimmt natürlich nicht. Deswegen muss minimal erhöht werden.



Jetzt geht der Strahl nur noch durch 4 Kanten und somit ist der blaue Punkt außerhalb².

Jedes mal wenn eine Kante berührt wird, soll ein Zähler sich um 1 erhöhen. Der Endwert vom Zähler entscheidet, ob der Punkt sich im Polygon befindet oder nicht.

Algorithm 4 imPolygon(Punkt P, List<Punkt> polygon) returns boolean

```

1: zähler ← 0
2: for int i = 0; i < polygon.size()-1; i++ do
3:   if schneidetKante(polygon.get(i), polygon.get(i+1), P) == true then
4:     zähler++
5:   end if
6: end for
7: if zähler % 2 == 0 then //Zähler ist gerade → nicht im Polygon
8:   return false
9: end if
10: return true //Ansonsten ist Zähler ungerade → im Polygon

```

²Zur Verständlichkeit des Problems wurde y_P hier um eine relativ große Zahl erhöht. In der Praxis ist die Erhöhung relativ klein (0.000001), also für das menschliche Auge nicht sichtbar.

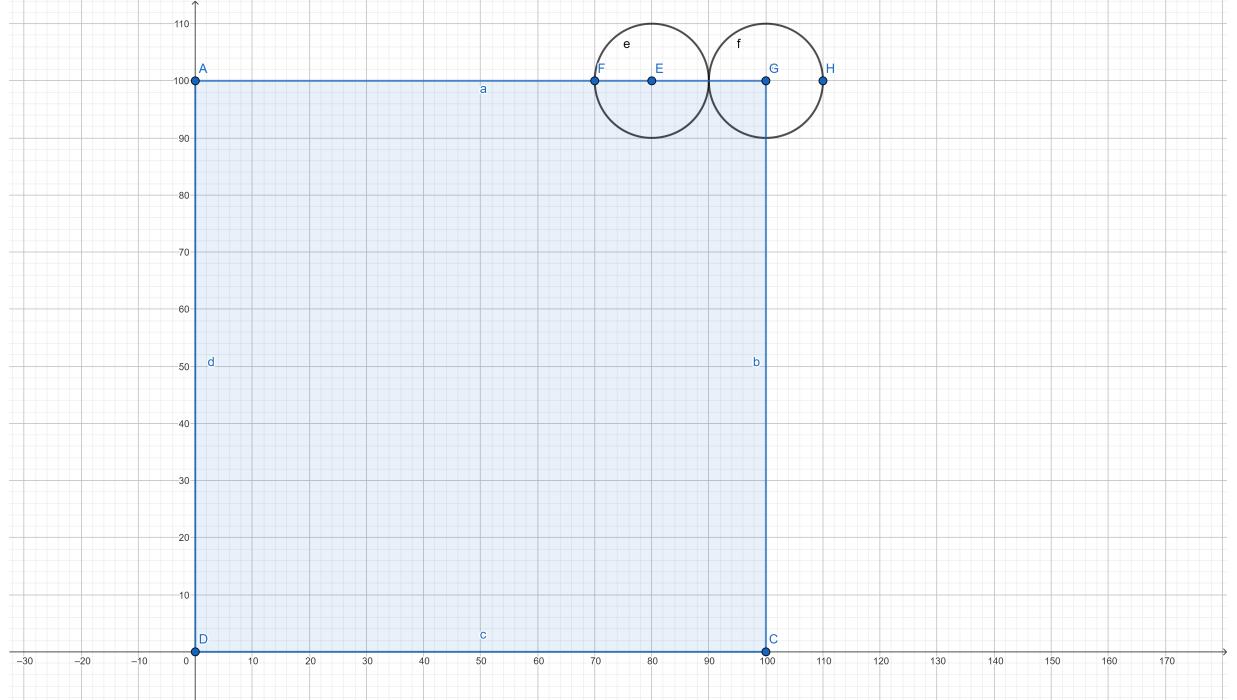
Aufgabe 3: Die Siedler

Ob sich der Kreis mit anderen Kreisen überschneiden würde, wird mit dem vorher erklärten Algorithmus überprüft. Jedoch wird hierbei der Radius nicht von den zwei großen Kreisen vervielfacht.

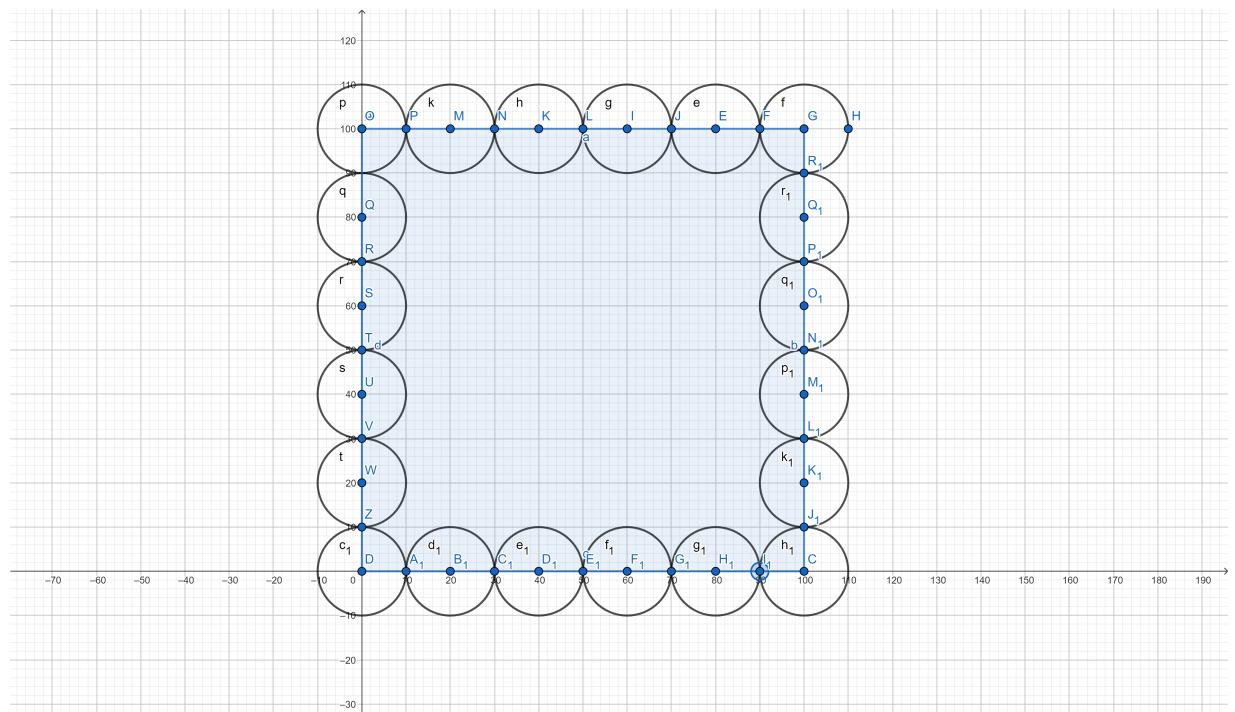
Dieser Prozess wird schließlich von Kreis zu Kreis wiederholt.

2.2 Ortschaften am Rand des Gebiets planen

Um Platz im Gebiet zu sparen, sollten so viele Kreise wie möglich am Rand liegen. Denn dabei wird weniger Fläche als sonst eingenommen (abhängig vom Winkel der Seiten des Polygon).



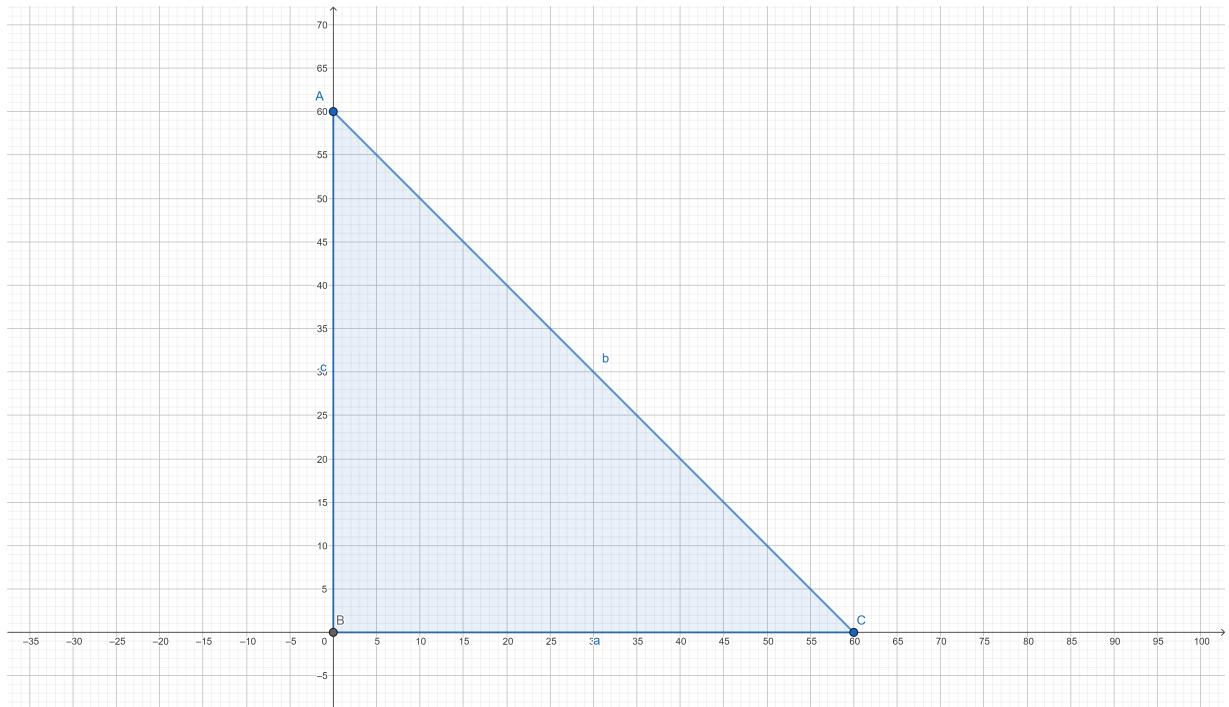
Um dann alle Seiten des Polygons mit Kreisen zu füllen, wird der Prozess von Position zu Position (am Rand) wiederholt.



Doch wie wird erkannt wo jeweils die nächsten Kreise gesetzt werden müssen? Dafür müssen die Po-
13/26

Aufgabe 3: Die Siedler

Kartkoordinaten berechnet werden. Für die Erklärung wird ein anderes Polygon verwendet:



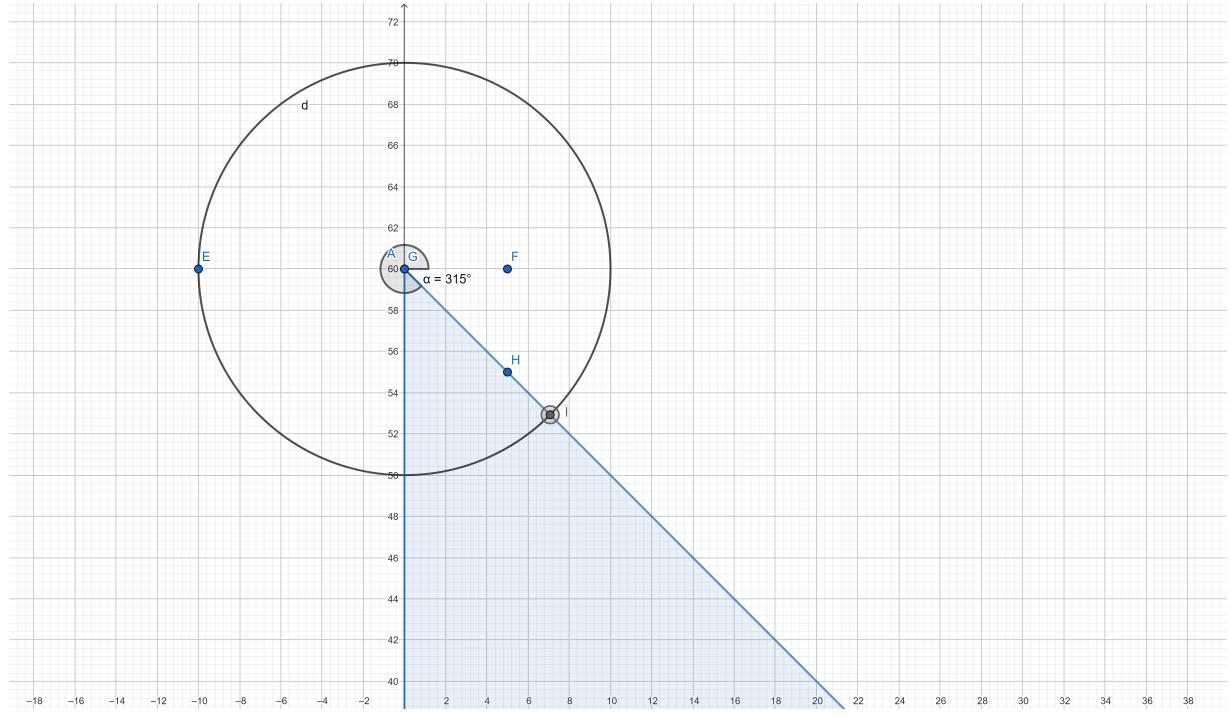
Zunächst wird der Steigungswinkel der Geraden \overrightarrow{AC} des Polygons benötigt. Dieser wird hier als α beschrieben

$$\alpha = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$$

Mit dem Winkel α können jetzt die kartesischen Koordinaten berechnet werden. $posX$ und $posY$ dienen hier zum Versatz.

$$x = 10 \cdot \cos(\alpha) + posX$$
$$y = 10 \cdot \sin(\alpha) + posY$$

Aufgabe 3: Die Siedler



(Erster Kreis bereits dort)

GFI bilden hier das Steigungsdreieck, das den Steigungswinkel -45° bzw. 315° liefert. I ist dabei der ausgerechnete Punkt. Es fällt auf, dass $|AI|$ genau so groß wie der Radius r ist. Um die vorher erwähnte Bedingung zu erfüllen muss der Radius verdoppelt werden. So ergibt sich:

$$x = 2 \cdot 10 \cdot \cos(\alpha) + posX$$

$$y = 2 \cdot 10 \cdot \sin(\alpha) + posY$$

Für das Beispiel eingesetzt:

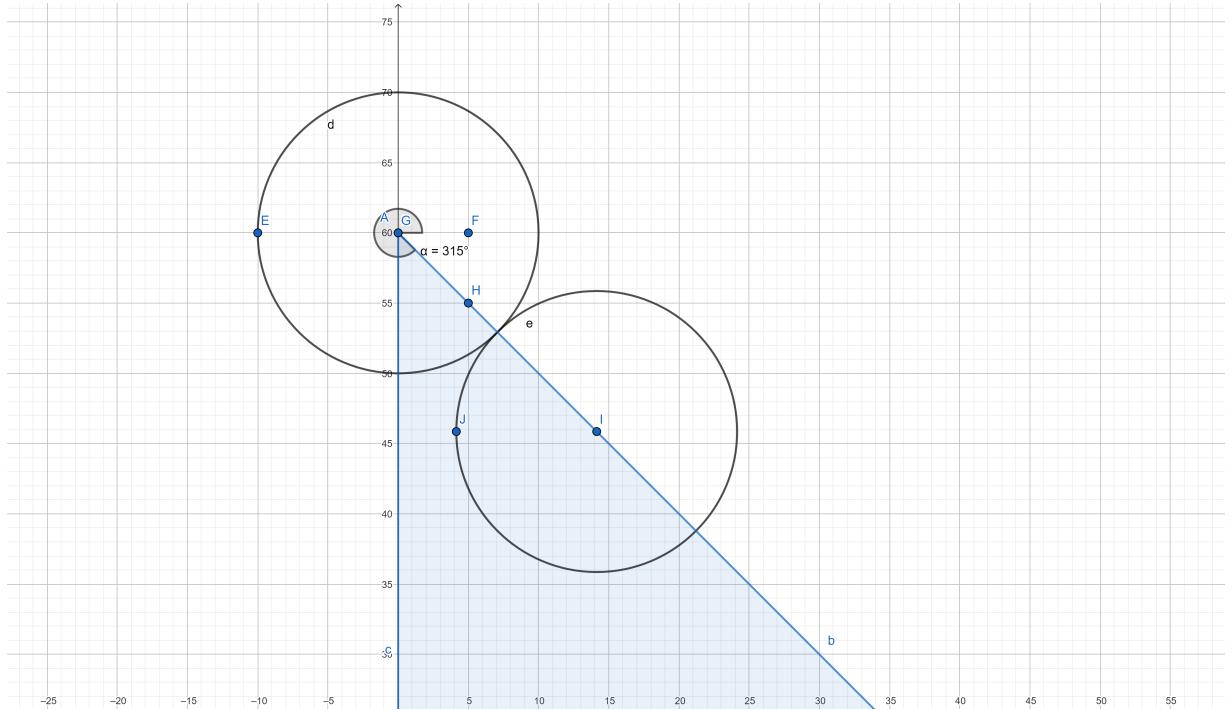
$$\begin{aligned} \alpha &= \arctan\left(\frac{0 - 60}{60 - 0}\right) \\ &= \arctan(-1) \\ &= -45 \end{aligned}$$

$$\begin{aligned} x &= 20 \cdot \cos(-45) + 0 \\ &\approx 14.1421 \end{aligned}$$

$$\begin{aligned} y &= 20 \cdot \sin(-45) + 60 \\ &\approx 45.8579 \end{aligned}$$

Bevor der Kreis gesetzt wird, wird natürlich auch überprüft ob eine Überschneidung stattfindet.

Aufgabe 3: Die Siedler



Falls sich der Kreis überschneiden würde, dann wird der Mittelpunkt des Kreises immer weiter verschoben, bis es einen geeigneten Punkt findet, wo es keinen anderen Kreis überschneidet oder nicht mehr auf \overrightarrow{AB} liegt.

$$x = i \cdot 2 \cdot 10 \cdot \cos(\alpha) + posX$$

$$y = i \cdot 2 \cdot 10 \cdot \sin(\alpha) + posY$$

Genau deswegen wird das Ergebnis nicht immer optimal sein. i kann jedes mal wenn ein Kreis einen anderen überschneidet, um 2, 1, 0.5, 0.1, 0.01, 0.001 erhöht werden. Je kleiner die Erhöhung, desto dichter die Kreise, desto mehr Platz und somit mehr Kreise insgesamt.

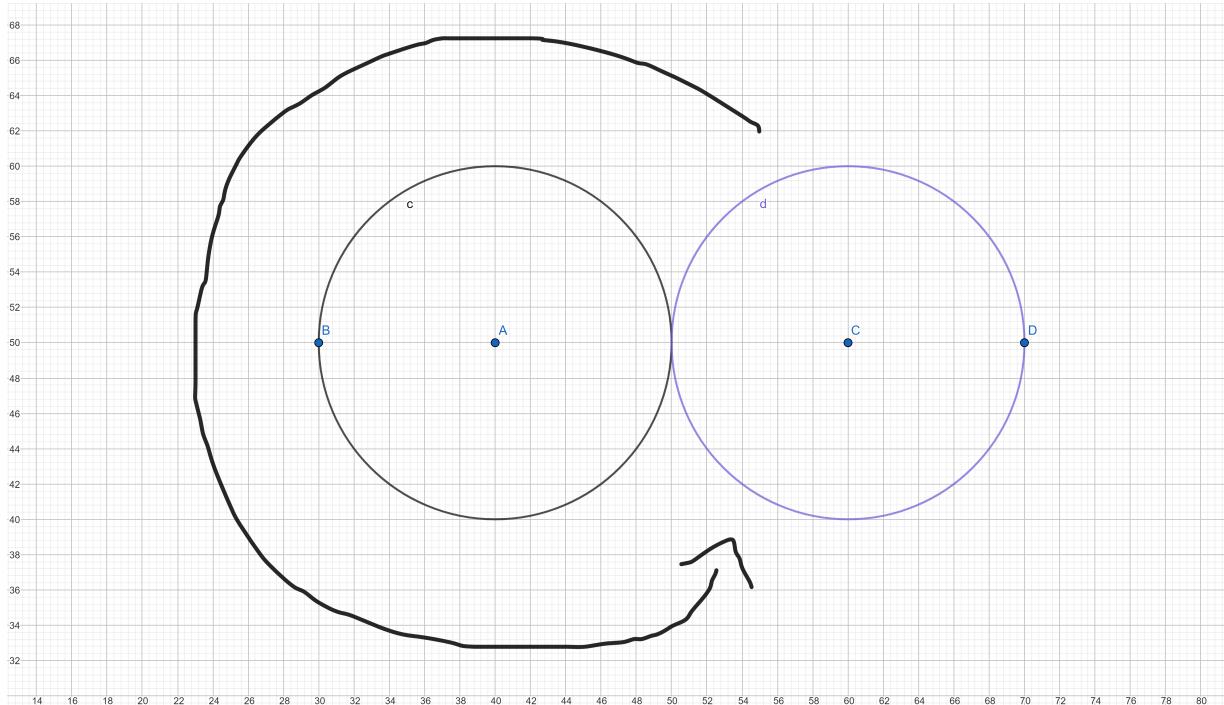
Es passen auf \overrightarrow{AB} maximal $\frac{|\overrightarrow{AB}|}{20} + 1$ Kreise. Der zusätzliche Kreis kommt davon, dass auch genau auf den Punkten A, B ein Kreis gesetzt werden darf.

Beim Verschieben muss darauf geachtet werden, dass die genutzte Länge, also alle r der Kreise auf einer Seite addiert nicht größer als $|\overrightarrow{AB}|$ ist.

2.3 Ortschaften in das Gebiet planen - Möglichkeit 1

Wenn der Rand mit Kreisen befüllt ist, kann nun auch das Innere des Gebiets mit Kreisen befüllt werden. Dafür wird durch jeden einzelnen Kreis am Rand iteriert. Für jeden Kreis soll geprüft werden ob umherum ein verfügbarer geeigneter Platz ist für einen weiteren Kreis. Im Unterschied zu Kreisen am Rand, gibt es keinen 'Pfad' an dem Kreise platziert werden müssen

Aufgabe 3: Die Siedler



Das wird für jede neue 'Schicht' von Kreisen wiederholt. Die erste Schicht sind die Kreise am Rand. Die letzte Schicht wird erkannt, wenn kein Kreis mehr gesetzt werden kann.

Algorithm 5 innen_planen(ortschaftenRand, ortschaftenGZ, polygon)

```
1: ortschaftenInnen //Liste der inneren Ortschaften
2: ortschaften ← ortschaftenRand + ortschaftenGZ //Liste aller Ortschaften insgesamt
3: gefunden ← true //Es ist genug Platz fuer andere Kreise da um die Schleife zu starten
4: while gefunden do
5:   gefunden ← false //Zu Beginn ist noch kein Platz gefunden
6:   for i = 0; i < aktuelleSchicht.size(); i++ do
7:     winkel ← 0
8:     while winkel < 360 do
9:       x = 20 * cos(winkel) + aktuelleSchicht.get(i).mittelPunkt.x
10:      y = 20 * sin(winkel) + aktuelleSchicht.get(i).mittelPunkt.y
11:      kreis = new Kreis(Mittelpunkt(x, y), Radius(10))
12:      if imPolygon(kreis, polygon) == true && ueberschneidet(kreis, ortschaften) == false then
13:        ortschaften.add(kreis)
14:        naechsteSchicht.add(kreis)
15:        ortschaftenInnen.add(kreis)
16:        gefunden = true //Mindestens ein geeigneter Platz wurde gefunden
17:      end if
18:      winkel++
19:    end while
20:  end for
21:  aktuelleSchicht = naechsteSchicht
22: end while
23: return ortschaftenInnen
```

3 Optimierungen

3.1 Ortschaften in das Gesundheitszentrum planen - Möglichkeit 2

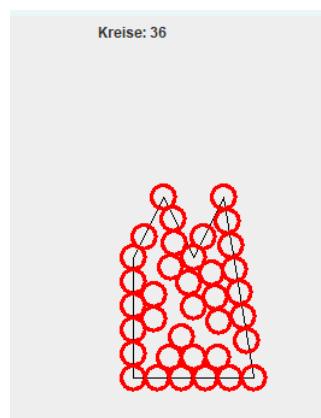
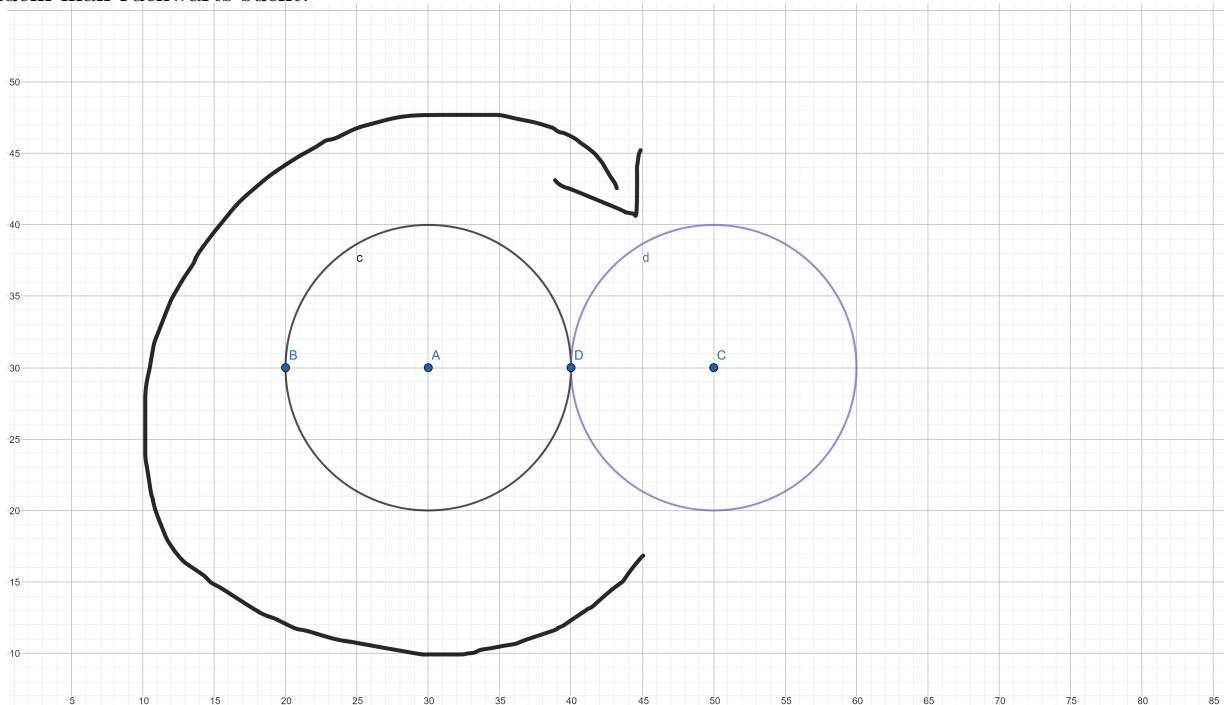
Es gibt einen anderen Algorithmus, um noch mehr als 242 Ortschaften in das GZ zu planen. Leider konnte ich ihn nicht herausarbeiten. Auf dieser Website können die Werte der Kreise eingegeben werden: <http://www.packomania.com/>

Aufgabe 3: Die Siedler

Durchmesser großer Kreis: $(2 \cdot (85 + 5))$
5 Einheiten größer, weil kleine Kreise innherhalb sein müssen
Durchmesser kleiner Kreis: $2 \cdot 5$
Ergebnis: 272

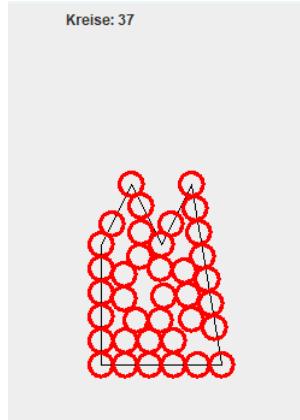
3.2 Ortschaften in das Gebiet planen - Möglichkeit 2

Manchmal ist es besser die Suche von verfügbaren geeigneten Punkten umherum eines Kreises zu finden indem man rückwärts sucht.



Möglichkeit 1: 36 Ortschaften

Aufgabe 3: Die Siedler



Grund liegt hier beim Ansetzpunkt. Es ist wahrscheinlicher, dass die Kreise dichter gesetzt werden

3.3 Ortschaften in das Gebiet planen - Noch mehr Möglichkeiten

Es gibt noch mehr Möglichkeiten, da der Ansetzpunkt 360 verschiedene Punkte besitzen kann. Deswegen wird zur Methode *innen_planen(...)* die Variable *ansetzPunkt* als Parameter hinzugefügt.

Algorithm 6 innen_planen(..., ansetzPunkt)

```
1: ...
2: while ... do
3:   ...
4:   for ... do
5:     winkel ← ansetzPunkt
6:     while winkel < ansetzPunkt+360 do
7:       ...
8:       end while
9:     end for
10:    ...
11: end while
12: ...
```

Aus diesem Grund werden die Algorithmen in mehreren Möglichkeiten ausgeführt und es wird dann der Besiedlungsplan mit den meisten Ortschaften ausgegeben.

4 Laufzeitanalyse

Gesundheitszentrum Das Planen von Ortschaften in das GZ hat eine Laufzeit von $\mathcal{O}(1)$, denn es ist unabhängig vom Input. Der Kreis des GZ bleibt immer bei einem Radius von 85 und wird immer 249 Ortschaften beinhalten (Bei einer Dichte von 0.01). Das Planen des GZ in das Polygon hat einen höheren Zeitaufwand. Da zuerst durch die Liste der n Eckpunkten des Polygon iteriert wird, gibt es einen Zeitaufwand von $\mathcal{O}(n)$. Als nächstes wird durch das komplette Viereck iteriert, was grundsätzlich einen Zeitaufwand von $\mathcal{O}(a \cdot b)$, wobei $a = |\overrightarrow{\min X \max X}|$ und $b = |\overrightarrow{\min Y \max Y}|$ bedeutet. Dazu wird noch für jede Ortschaft im GZ (249 Ortschaften - nicht abhängig vom Input) geprüft, ob sie im Polygon liegt. Diese Überprüfung dauert $\mathcal{O}(n + 1)$ und wird 249mal wiederholt, da es $n + 1$ Kanten bei n Ecken gibt und es wird durch jede Kante iteriert. Für das GZ ergibt sich somit eine Laufzeit von $\mathcal{O}(1 + n + a \cdot b \cdot 249 \cdot (n + 1))$. Das wird gekürzt auf $\mathcal{O}(a \cdot b \cdot n)$.

Rand Hierbei wird wieder durch jede Kante iteriert: $\mathcal{O}(n + 1)$. Wie in einem früheren Abschnitt erklärt, kann auf einer Seite des Polygon höchstens $\frac{|\overrightarrow{AB}|}{20} + 1$ Kreise liegen. Dabei wird für jeden Kreis überprüft ob er sich mit irgendeinem anderen Kreis überschneidet. Somit entsteht eine Laufzeit von $\mathcal{O}((n + 1) \cdot (k \cdot m))$,

Aufgabe 3: Die Siedler

wobei $k = \frac{|\vec{AB}|}{20} + 1$ und $m =$ die Anzahl der bereits existierenden Ortschaften ist. Das wird gekürzt auf $\mathcal{O}(n \cdot (k \cdot m))$

Innen Hierbei wird durch jede Ortschaft der aktuellen Schicht iteriert: $\mathcal{O}(s)$, wobei $s =$ die Anzahl der Schichten ist. Danach wird umherum für jede Ortschaft nach 360 möglichen Punkten gesucht. Dabei wird jedes Mal geprüft ob der Punkt im Polygon ist und ob er sich überschneidet: $\mathcal{O}(360 \cdot (n + 1 + m))$. Somit ergibt sich eine Laufzeit von $\mathcal{O}(s \cdot (360 \cdot (n + 1 + m)))$. Das wird gekürzt auf $\mathcal{O}(s \cdot n \cdot m)$

5 Speicheranalyse

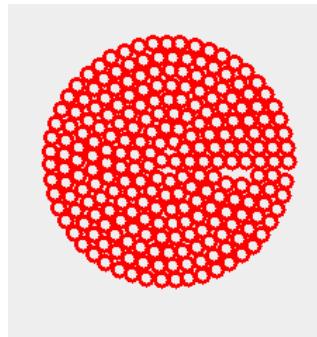
Gesundheitszentrum Auch hier ist das Planen von Ortschaften in das GZ eine Speicherkomplexität von $\mathcal{O}(1)$, denn es ist unabhängig vom Output. Wenn durch das Viereck gelaufen wird, dann wird für jede Position eine Liste erstellt, die dann alle Ortschaften hinzufügt, die innerhalb des Polygons und GZ sind. Jede Ortschaft wird dabei versetzt und neu gespeichert: $\mathcal{O}(a \cdot b \cdot g)$, wobei $g =$ Anzahl der Ortschaften im GZ ist.

Rand Für jede Kante wird maximal k -mal die Variable *naechsterPunkt* gespeichert: $\mathcal{O}((n + 1) \cdot k)$ wird gekürzt zu $\mathcal{O}(n \cdot k)$

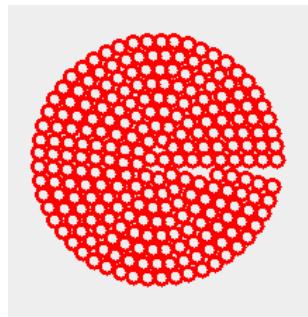
Innen Für jede Schicht wird der Winkel gespeichert, für jeden Winkel die Koordinate. An der Position mit den berechneten Koordinaten werden die Überschneidungen mit Kanten gezählt, wofür die Variable *zähler* gebraucht wird: $\mathcal{O}(s \cdot 360 \cdot n + 1)$ zu $\mathcal{O}(s \cdot n)$

6 Rundungsfehler

Bei weiterem Ausprobieren von unterschiedlichen Werten fällt auf, dass noch mehr Ortschaften in das GZ passen, wenn die Polarkoordinaten präziser berechnet werden. Sprich, anstatt von Winkel 0° bis 359° in 1er Schritten zu iterieren, kann in 0.1er Schritten iteriert werden. 0.01er sogar noch mehr. Bei weiterem Verkleinern der Schritte bleibt die Anzahl der kleinen Kreise gleich.

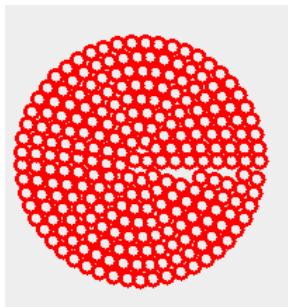


1: 242 Ortschaften



0.1: 247 Ortschaften

Aufgabe 3: Die Siedler



0.01: 249 Ortschaften

Grund dafür sind 'Rundungsfehler' im Code. Im Code wird der Datentyp *double* verwendet. Werte mit Nachkommastellen können von einem *double* nicht genau repräsentiert werden. $0.3 - 0.1 = 0.2$ Ein *double* würde jedoch 0.199999999999998 einspeichern. Das ist ungenau. Mit diesen Werten wird auch weitergerechnet, weswegen es zu kleinen Abweichungen kommt. Aus diesem Grund werden bei der Überprüfung von Überschneidungen auch kleinere Abstände wie 19.999999, 14.999999 und 9.999999 akzeptiert.

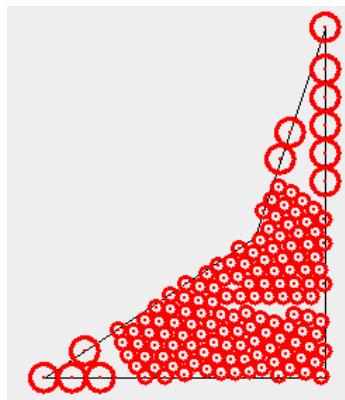
7 Geometrie App

Anstatt stumpf den Besiedlungsplan durch die Koordinaten der Punkte auszugeben, habe ich via Java Swing das Gebiet und die Ortschaften in ein Fenster eingezeichnet. Es half mir stark um eine Lösung zu entwickeln, da ich direkt sehen kann, wo die Ortschaften im Gebiet liegen. In den dokumentierten Beispielen wird auch die Geometrie-App zur Visualisierung des Besiedlungsplan genutzt.

8 Beispiele

Beim Ausführen wird die Anzahl der insgesamten Ortschaften und deren Positionen in der Konsole ausgegeben.

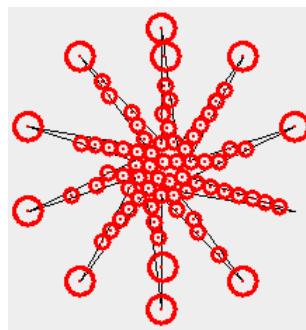
8.1 siedler1.txt



Insgesamt 145 Ortschaften

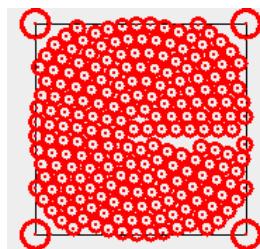
Aufgabe 3: Die Siedler

8.2 siedler2.txt



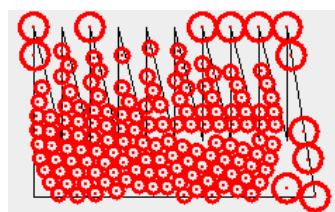
Insgesamt 85 Ortschaften

8.3 siedler3.txt



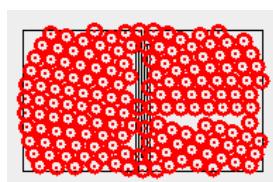
Insgesamt 224 Ortschaften

8.4 siedler4.txt



Insgesamt 165 Ortschaften

8.5 siedler5.txt

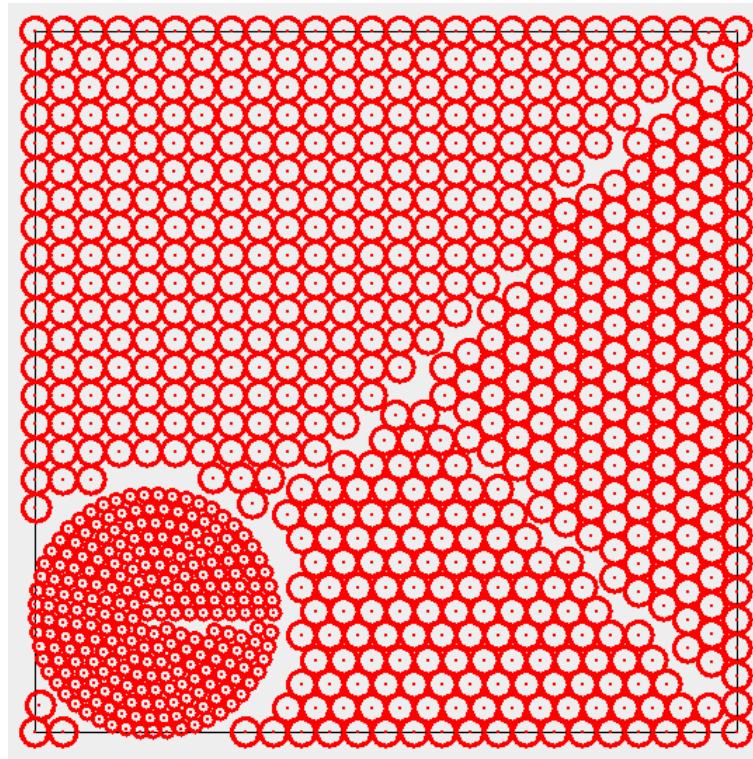


Insgesamt 161 Ortschaften

8.6 siedler6.txt

Mal ein etwas größeres Gebiet, um auch das Planen von innen zu zeigen

Aufgabe 3: Die Siedler



Insgesamt 867 Ortschaften

8.7 siedler7.txt

Überschneidendes Gebiet



Insgesamt 99 Ortschaften

Aufgabe 3: Die Siedler

9 Quellcode

Da Java mit der Maßeinheit Bogenmaß bei Winkeln arbeitet, müssen vor anderen Berechnungen die Winkel in Grad zu Winkel in Bogenmaß umgewandelt werden:

$$\text{bogenmaszWinkel} = \frac{\text{gradWinkel} \cdot \pi}{180}$$

Für eine bessere Lesbarkeit, wurde der Rest des Quellcode separat als .java Dateien im Ordner '/Quellcode' angehängt.

Wichtigste Teile:

```
1 List<Kreis> GZ_planen(List<Punkt> polygon) {
2     List<Kreis> ortsschafoten_GZ = new ArrayList<>();
3     for (double radius = 85; radius > 0; radius -= 10) {
4         double winkel_grad = 0;
5         while (winkel_grad < 360) {
6             double winkel_bogenmasz = (winkel_grad * Math.PI) / 180;
7             double x = radius * Math.cos(winkel_bogenmasz);
8             double y = radius * Math.sin(winkel_bogenmasz);
9             Kreis naechsterKreis = new Kreis(new Punkt(x, y), 5);
10            naechsterKreis.imGZ = true;
11            if(!ueberschneidet(naechsterKreis, ortsschafoten_GZ)) {
12                ortsschafoten_GZ.add(naechsterKreis);
13            }
14            winkel_grad += 0.01;
15        }
16    }
17    //GZ in das Gebiet planen:
18    double minX = Double.POSITIVE_INFINITY;
19    double maxX = Double.NEGATIVE_INFINITY;
20    double minY = Double.POSITIVE_INFINITY;
21    double maxY = Double.NEGATIVE_INFINITY;
22    for (Punkt punkt : polygon) {
23        if(punkt.x < minX)
24            minX = punkt.x;
25        if(punkt.y < minY)
26            minY = punkt.y;
27        if(punkt.x > maxX)
28            maxX = punkt.x;
29        if(punkt.y > maxY)
30            maxY = punkt.y;
31    }
32    Punkt aktuellerPunkt = new Punkt(minX, minY);
33    List<Kreis> beste_ortschafoten_GZ_im_polygon = new ArrayList<>();
34    while (aktuellerPunkt.y < maxY) {
35        while (aktuellerPunkt.x < maxX) {
36            List<Kreis> ortsschafoten_GZ_im_polygon = new ArrayList<>();
37            if(imPolygon(aktuellerPunkt, polygon)) {
38                for (Kreis ortschaft : ortsschafoten_GZ) {
39                    Kreis naechsteOrtschaft =
40                        new Kreis(
41                            new Punkt(
42                                ortschaft.mPunkt.x+aktuellerPunkt.x, ortschaft.mPunkt.y+ aktuellerPunkt.y), 5);
43
44                    if(imPolygon(new Punkt
45                                (naechsteOrtschaft.mPunkt.x, naechsteOrtschaft.mPunkt.y), polygon)) {
46
47                        ortsschafoten_GZ_im_polygon.add(naechsteOrtschaft);
48                    }
49                }
50                if(ortschafoten_GZ_im_polygon.size() > beste_ortschafoten_GZ_im_polygon.size()) {
51                    beste_ortschafoten_GZ_im_polygon = new ArrayList<>(ortschafoten_GZ_im_polygon);
52                }
53            }
54            aktuellerPunkt.x++;
55        }
56        aktuellerPunkt.x = minX;
57        aktuellerPunkt.y+=1;
58    }
59    return beste_ortschafoten_GZ_im_polygon;
60 }
```

Aufgabe 3: Die Siedler

```
61     List<Kreis> innen_planen_vorwaerts(List<Kreis> ortsschaften_rand,
63                                         List<Kreis> ortsschaften_GZ, List<Punkt> polygon, int ansetzPunkt) {
65
65     List<Kreis> aktuelleSchicht = new ArrayList<>(ortsschaften_rand);
67     List<Kreis> naechsteSchicht = new ArrayList<>();
68     List<Kreis> ortsschaften_innen = new ArrayList<>();
69     List<Kreis> ortsschaften = new ArrayList<>();
70     ortsschaften.addAll(ortsschaften_rand);
71     ortsschaften.addAll(ortsschaften_GZ);
72     boolean gefunden = true;
73     while (gefunden){
74         gefunden = false;
75         for (int i = 0; i < aktuelleSchicht.size(); i++) {
76             int winkel_grad = ansetzPunkt;
77             while (winkel_grad < ansetzPunkt + 360) {
78                 double winkel_bogenmasz = (winkel_grad * Math.PI) / 180;
79                 double x = 20 * Math.cos(winkel_bogenmasz) + aktuelleSchicht.get(i).mPunkt.x;
80                 double y = 20 * Math.sin(winkel_bogenmasz) + aktuelleSchicht.get(i).mPunkt.y;
81                 Kreis kreis = new Kreis(new Punkt(x, y), 10);
82                 if(imPolygon(kreis.mPunkt, polygon) && !ueberschneidet(kreis, ortsschaften)) {
83                     ortsschaften.add(kreis);
84                     naechsteSchicht.add(kreis);
85                     ortsschaften_innen.add(kreis);
86                     gefunden = true;
87                 }
88                 winkel_grad += 1;
89             }
90         }
91         aktuelleSchicht = new ArrayList<>(naechsteSchicht);
92     }
93     return ortsschaften_innen;
}
```

Aufgabe 3: Die Siedler

```
1  List<Kreis> rand_planen(List<Kreis> ortsschaften_GZ, List<Punkt> polygon) {
2      List<Kreis> ortsschaften_rand = new ArrayList<>();
3      List<Kreis> ortsschaften = new ArrayList<>(ortsschaften_GZ);
4      for (int i = 0; i < polygon.size() - 1; i++) {
5          Punkt A = polygon.get(i);
6          Punkt B = polygon.get(i + 1);
7          if (A.x > B.x) {
8              Punkt temp = B;
9              B = A;
10             A = temp;
11         }
12         double winkel_bogenmasz = Math.atan((B.y - A.y) / (B.x - A.x));
13         double cos_wert = Math.cos(winkel_bogenmasz);
14         double sin_wert = Math.sin(winkel_bogenmasz);
15         Kreis ortsschaftA = new Kreis(A, 10);
16         if (!ueberschneidet(ortsschaftA, ortsschaften)) {
17             ortsschaften_rand.add(ortsschaftA);
18         }
19         Kreis ortsschaftB = new Kreis(B, 10);
20         if (!ueberschneidet(ortsschaftB, ortsschaften)) {
21             ortsschaften_rand.add(ortsschaftB);
22         }
23         double vielfach = 0.0; //In der Dokumentation als i beschrieben
24         Punkt aktuellerPunkt = A;
25         double verbleibendeLaenge = getAbstand(A, B) - 2 * 10;
26         while (verbleibendeLaenge > 0) {
27             Punkt naechsterPunkt =
28                 new Punkt
29                     (vielfach*2*10*cos_wert+aktuellerPunkt.x, vielfach*2*10*sin_wert+aktuellerPunkt.y);
30             if (ueberschneidet(new Kreis(naechsterPunkt, 10), ortsschaften)) {
31                 vielfach += 0.25;
32                 continue;
33             }
34             aktuellerPunkt = naechsterPunkt;
35             verbleibendeLaenge -= vielfach * 2 * 10;
36             if (verbleibendeLaenge > 0) {
37                 Kreis ortsschaft = new Kreis(aktuellerPunkt, 10);
38                 ortsschaften_rand.add(ortsschaft);
39                 ortsschaften.add(ortsschaft);
40             }
41             vielfach = 1;
42         }
43     }
44     return ortsschaften_rand;
45 }
46 }
```

10 Quellen

Ortschaften am Rand:

2 <https://studyflix.de/mathematik/polarkoordinaten-1476>
3 <https://studyflix.de/mathematik/steigungswinkel-1905>

4 Circle Packing Problem:

6 https://en.m.wikipedia.org/wiki/Circle_packing
7 https://de.wikipedia.org/wiki/Kreispackung_in_einem_Kreis

8 Raycasting Algorithmus:

10 https://rosettacode.org/wiki/Ray-casting_algorithm#

12 Erkenntnis von NP-Schwer:

13 <https://math.mit.edu/research/highschool/rsi/documents/2015Chou.pdf>

14 Geometrie-App:

16 <https://www.youtube.com/watch?v=Kmgo00avvEw>