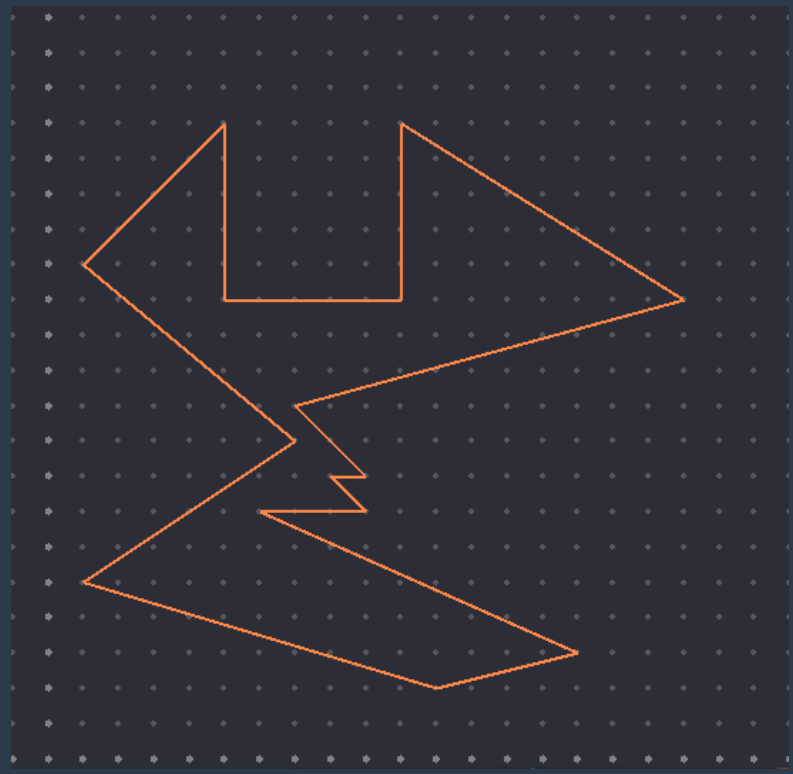
Problem N

Around the Track

Sebastian Dürr, Adrian Samoticha

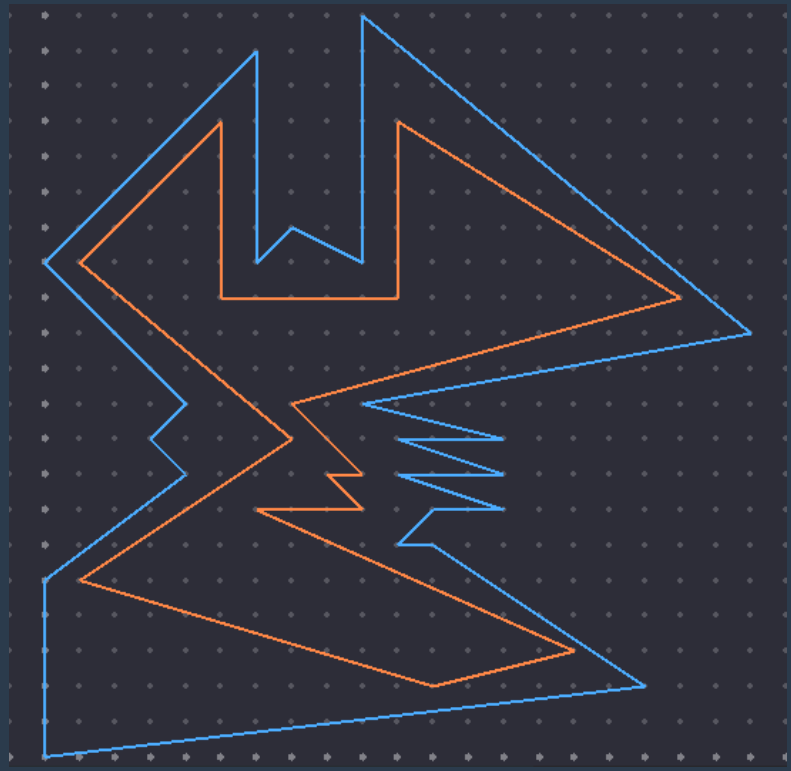
Problemstellung

- Rennstrecke definiert als Raum zwischen einem inneren und einem äußeren Polygon
- Das Äußere enthält das Innere und schneidet dieses nicht



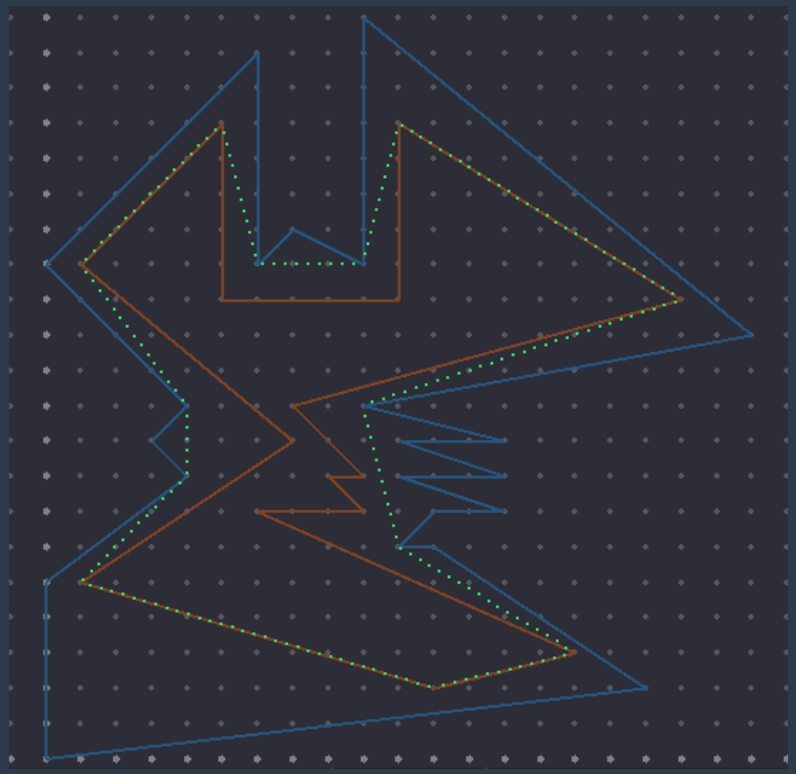
Problemstellung

- Rennstrecke definiert als Raum zwischen einem inneren und einem äußeren Polygon
- Das Äußere enthält das Innere und schneidet dieses nicht



Problemstellung

- Rennstrecke definiert als Raum zwischen einem inneren und einem äußeren Polygon
- Das Äußere enthält das Innere und schneidet dieses nicht
- Gesucht ist der kürzeste Weg durch die Rennstrecke



Eingabe und Ausgabe

```
5
1 1
5 1
5 5
3 3
1 5
5
0 0
6 0
6 6
3 4
0 6
```

Eingabe

Zwei Polygone (erst innen, dann außen):

- Anzahl der Punkte ($3 \leq n, m \leq 50$)
- n bzw. m Zeilen mit Koordinaten x_i und y_i ($-5000 \leq x_i, y_i \leq 5000$)
- Punkte sind gegen den Uhrzeigersinn sortiert

Ausgabe

Die Länge des kürzesten Weges

- Genauigkeit von 10^{-6}

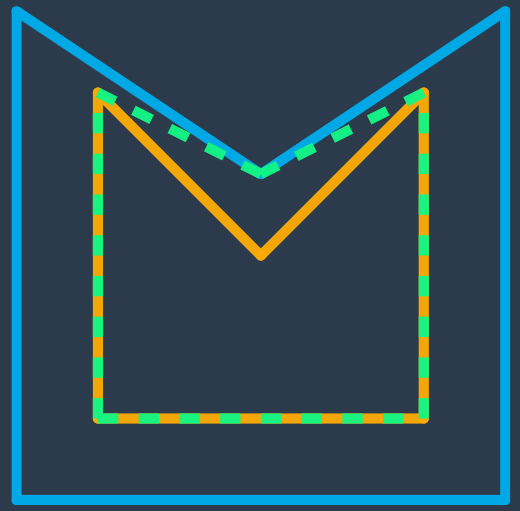
Lösungsvorschlag I
Beobachtung

Der kürzeste Weg um ein Polygon ist dessen konvexe Hülle

→ Wenn das äußere Polygon nicht die konvexe Hülle des Inneren schneidet, ist dies die Lösung

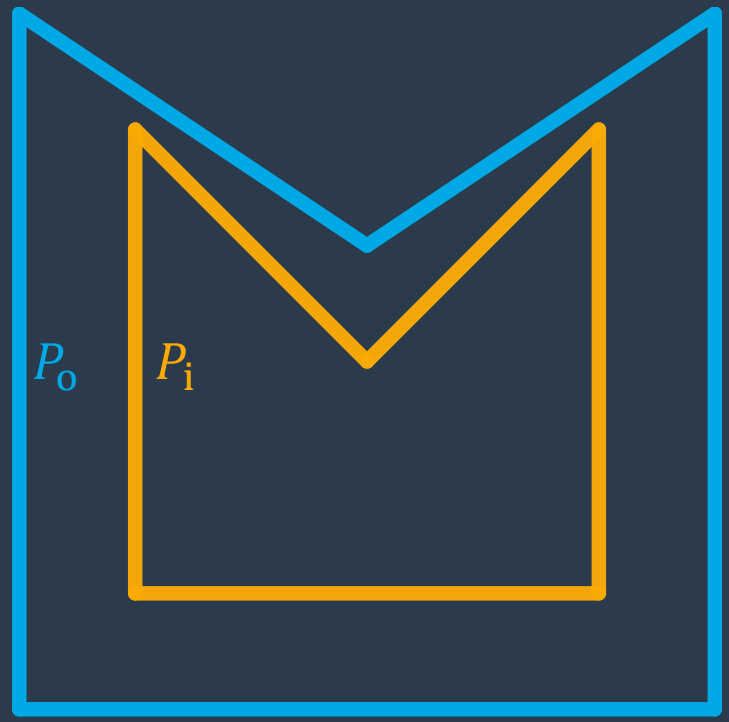
Lösungsvorschlag I
Beobachtung

Lösung für den Fall, dass das äußere Polygon die konvexe Hülle des Inneren schneidet



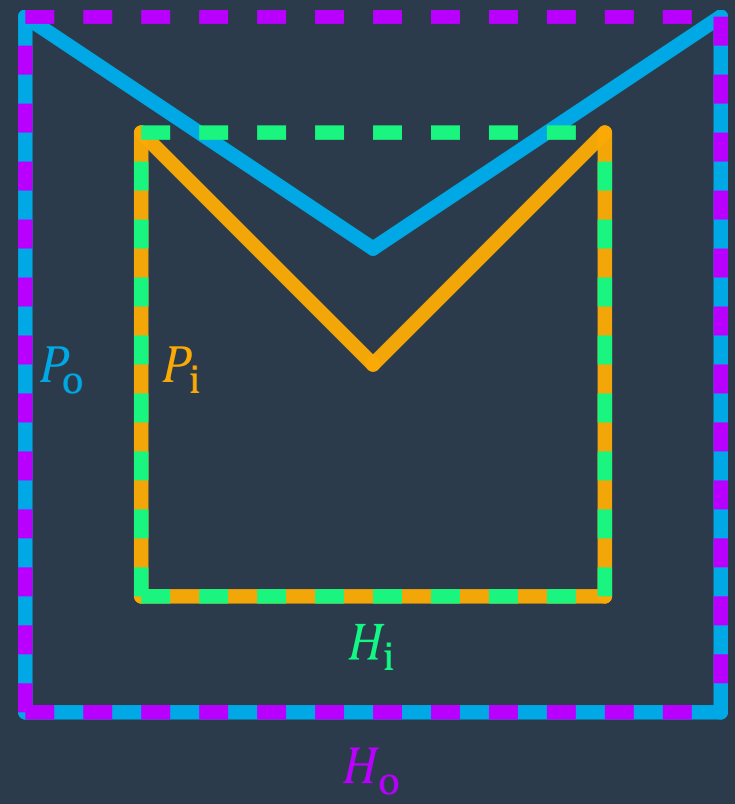
→ Gesucht ist die die konvexe Hülle des inneren Polygons relativ zum äußeren Polygon

Lösungsvorschlag I
Idee



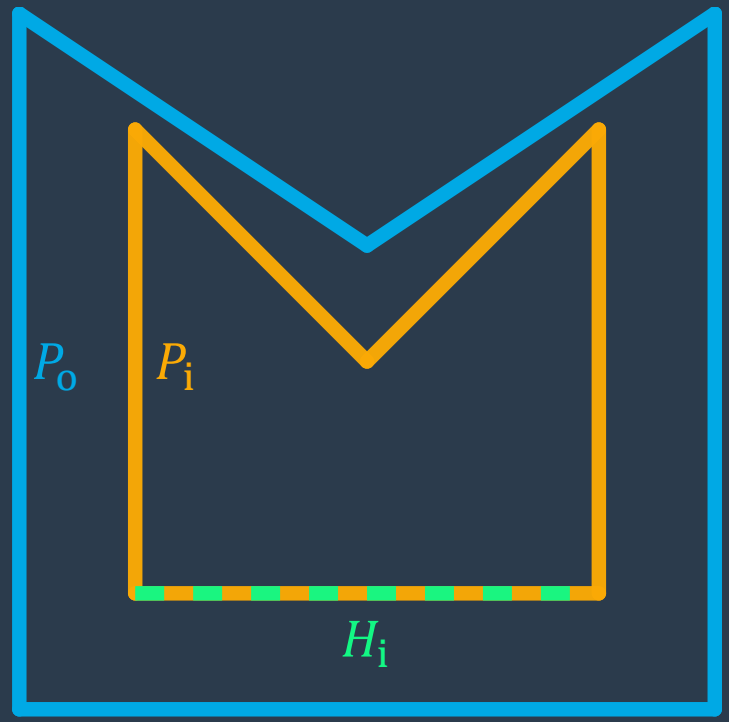
Lösungsvorschlag I
Idee

- Berechne die konvexen Hüllen H_i von P_i und H_o von P_o



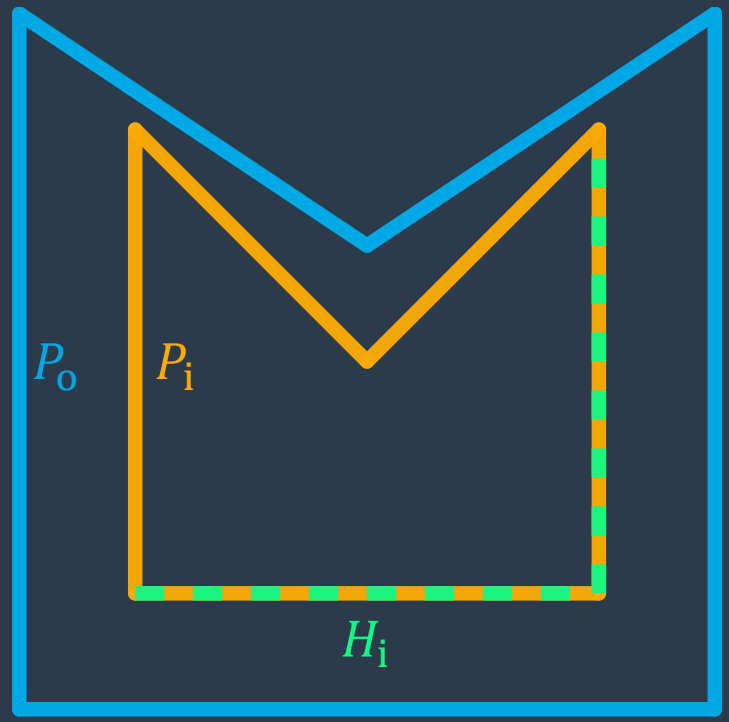
Lösungsvorschlag I
Idee

- Berechne die konvexen Hüllen H_i von P_i und H_o von P_o
- Gehe auf H_i entlang bis eine Einbuchtung C_i gefunden wird



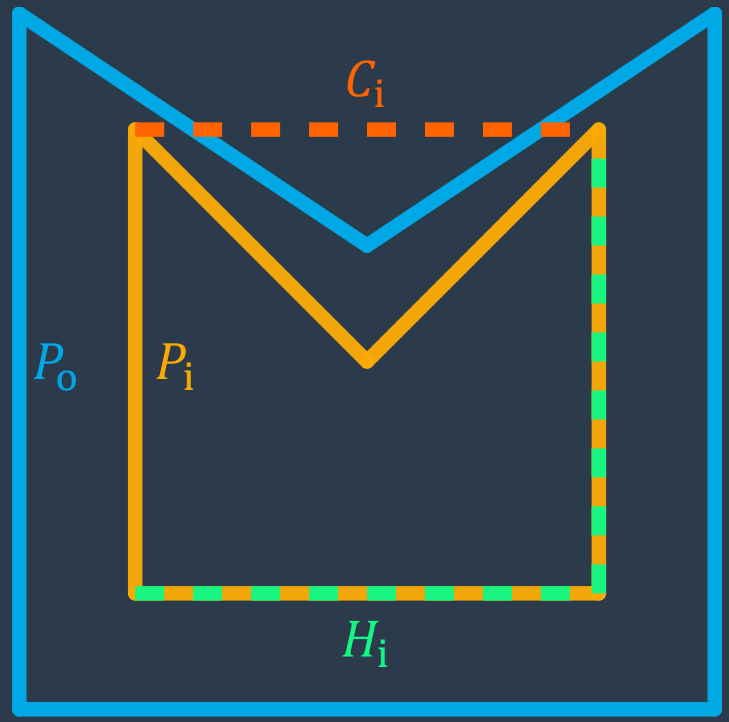
Lösungsvorschlag I
Idee

- Berechne die konvexen Hüllen H_i von P_i und H_o von P_o
- Gehe auf H_i entlang bis eine Einbuchtung C_i gefunden wird



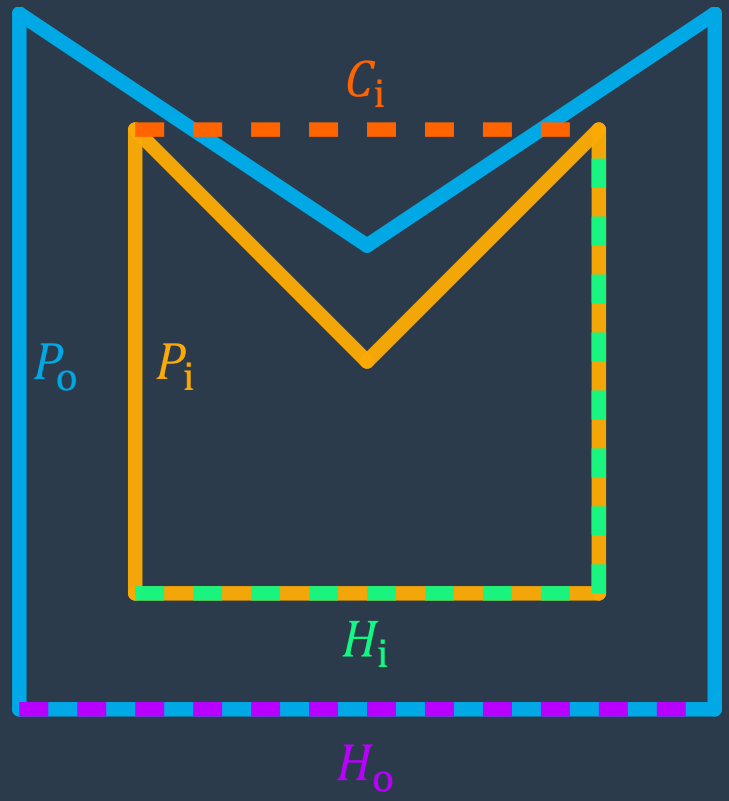
Lösungsvorschlag I
Idee

- Berechne die konvexen Hüllen H_i von P_i und H_o von P_o
- Gehe auf H_i entlang bis eine Einbuchtung C_i gefunden wird



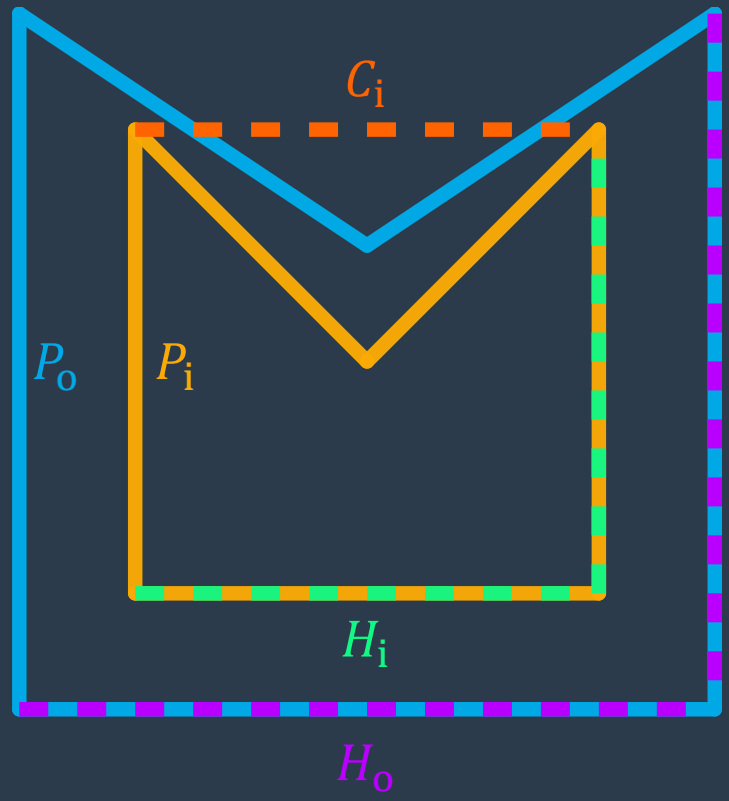
Lösungsvorschlag I
Idee

- Berechne die konvexen Hüllen H_i von P_i und H_o von P_o
- Gehe auf H_i entlang bis eine Einbuchtung C_i gefunden wird
- Gehe auf H_o entlang bis eine Einbuchtung C_o gefunden wird



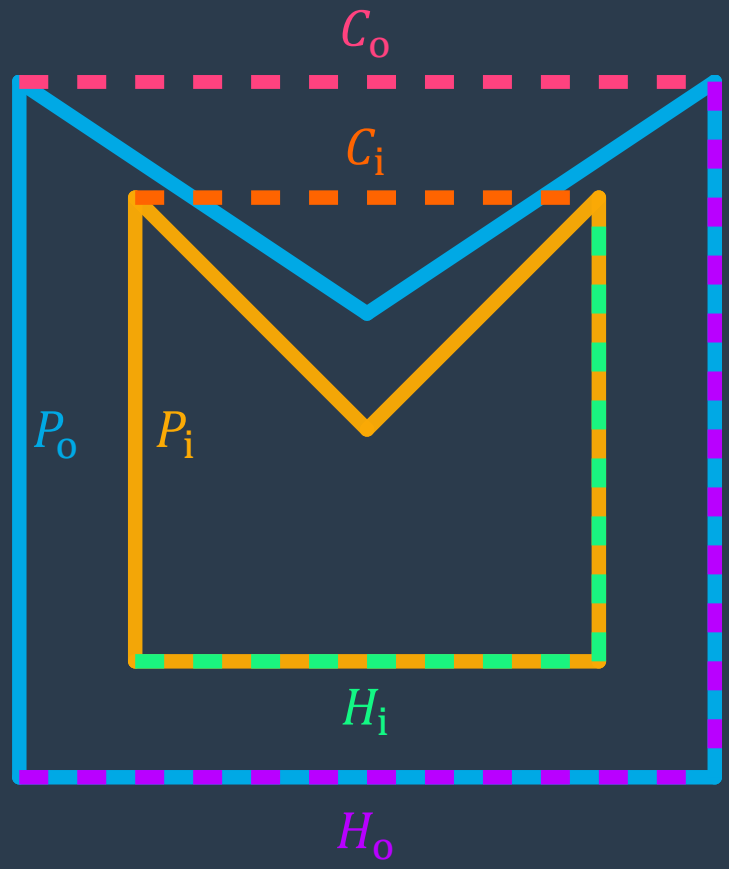
Lösungsvorschlag I
Idee

- Berechne die konvexen Hüllen H_i von P_i und H_o von P_o
- Gehe auf H_i entlang bis eine Einbuchtung C_i gefunden wird
- Gehe auf H_o entlang bis eine Einbuchtung C_o gefunden wird



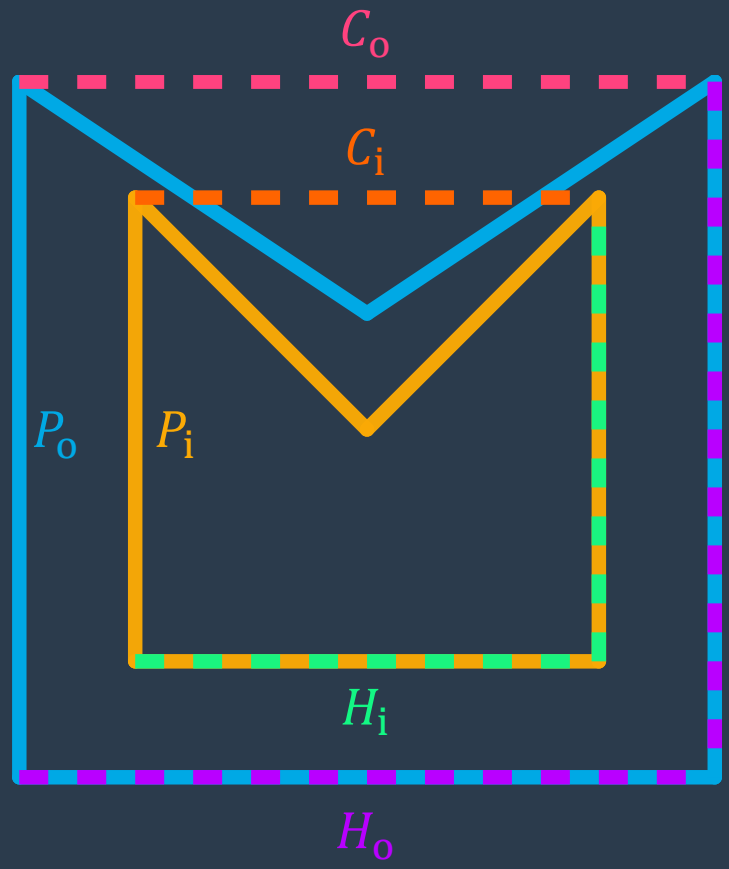
Lösungsvorschlag I
Idee

- Berechne die konvexen Hüllen H_i von P_i und H_o von P_o
- Gehe auf H_i entlang bis eine Einbuchtung C_i gefunden wird
- Gehe auf H_o entlang bis eine Einbuchtung C_o gefunden wird



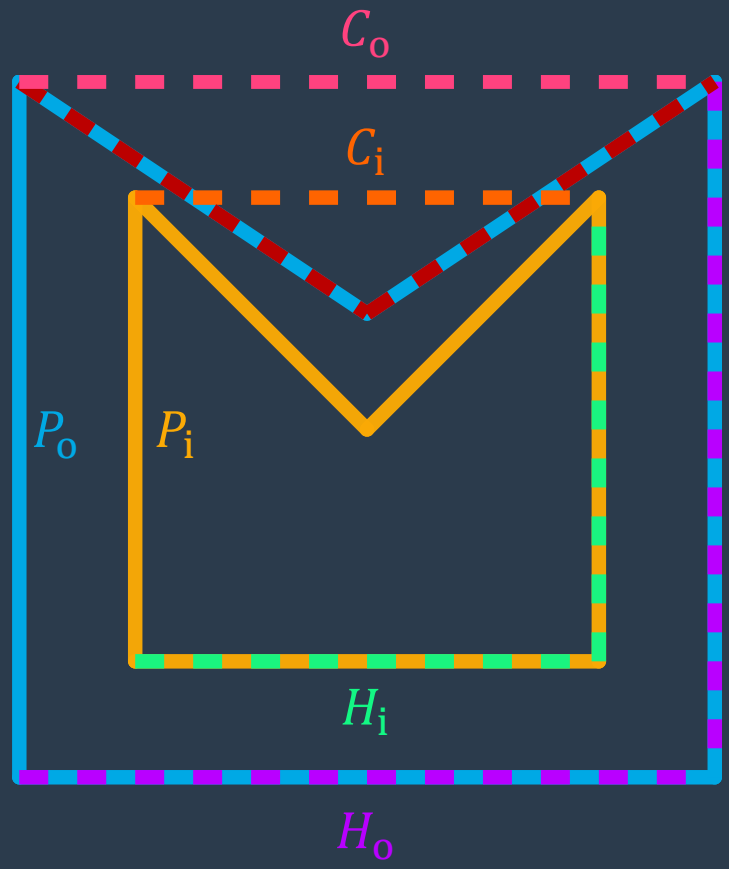
Lösungsvorschlag I
Idee

- Berechne die konvexen Hüllen H_i von P_i und H_o von P_o
- Gehe auf H_i entlang bis eine Einbuchtung C_i gefunden wird
- Gehe auf H_o entlang bis eine Einbuchtung C_o gefunden wird
- Wenn C_i von keiner Kante von P_o geschnitten wird, ist H_i die Lösung für diese Einbuchtung



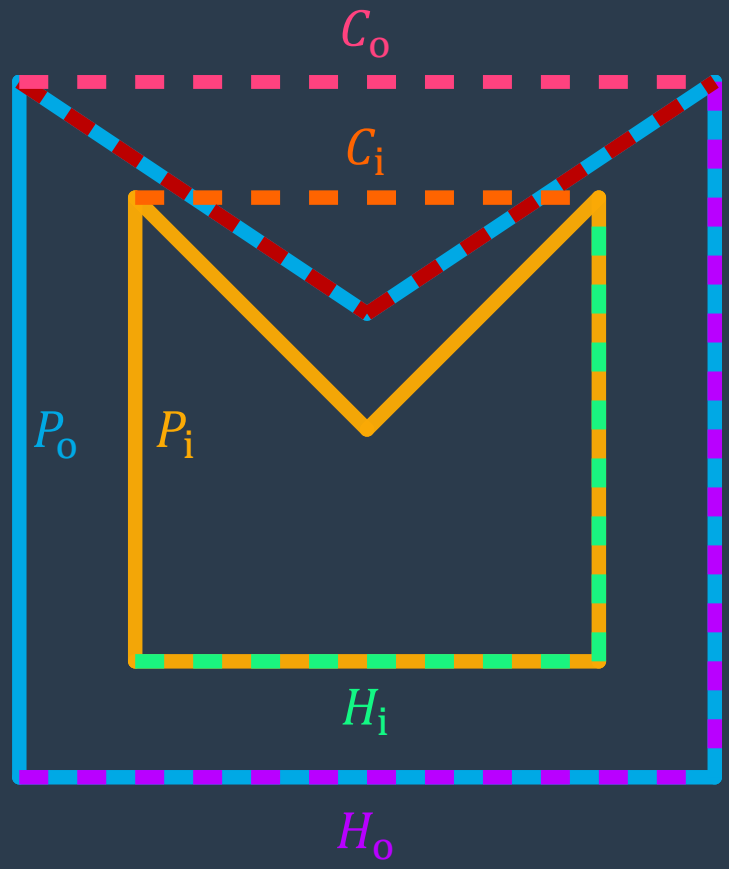
Lösungsvorschlag I
Idee

- Wenn C_i von mindestens einer Kante aus P_o geschnitten wird, erstelle ein Teilproblem und löse dieses



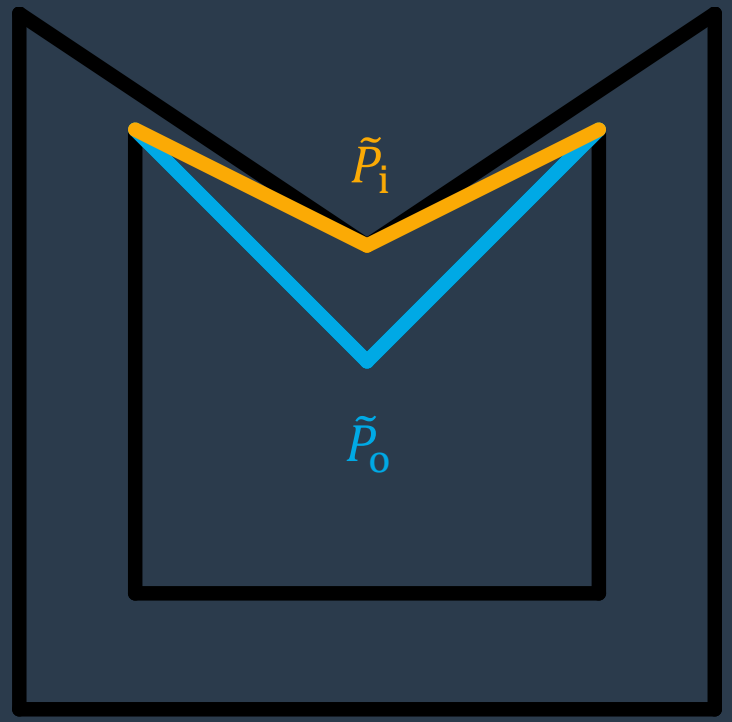
Lösungsvorschlag I
Idee

- Wenn C_i von mindestens einer Kante aus P_o geschnitten wird, erstelle ein Teilproblem und löse dieses
- Das neue innere Polygon $\tilde{P}_\mathrm{i}$ enthält den Start- und Endpunkt von C_i sowie alle Punkte von P_o, die zwischen C_o und innerhalb C_i liegen
- Das neue äußere Polygon $\tilde{P}_\mathrm{o}$ enthält den Start- und Endpunkt von C_i sowie alle Punkte von P_i, die innerhalb von C_i liegen



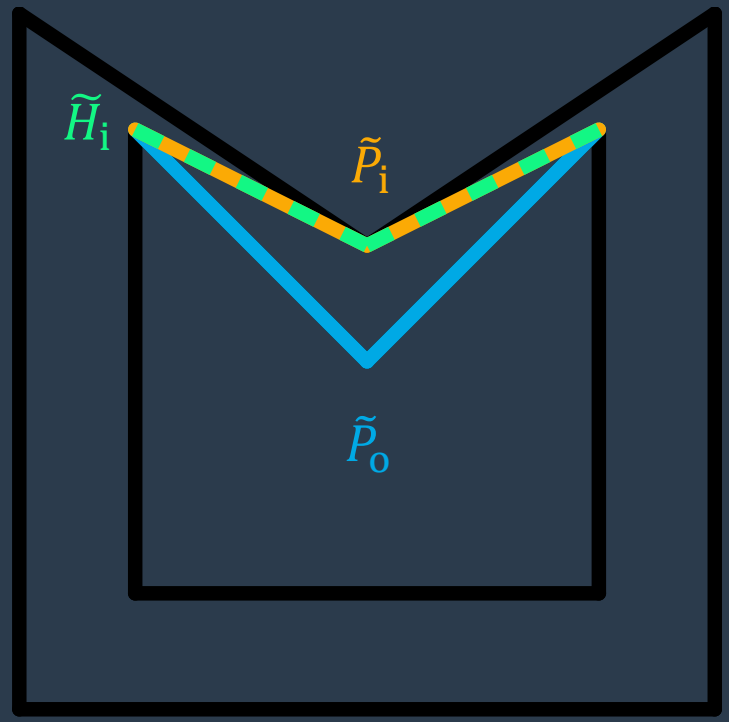
Lösungsvorschlag I
Idee

- Das neue innere Polygon $\tilde{P}_\mathrm{i}$ enthält den Start- und Endpunkt von C_i sowie alle Punkte von P_o, die zwischen C_o und innerhalb C_i liegen
- Das neue äußere Polygon $\tilde{P}_\mathrm{o}$ enthält den Start- und Endpunkt von C_i sowie alle Punkte von P_i, die innerhalb von C_i liegen



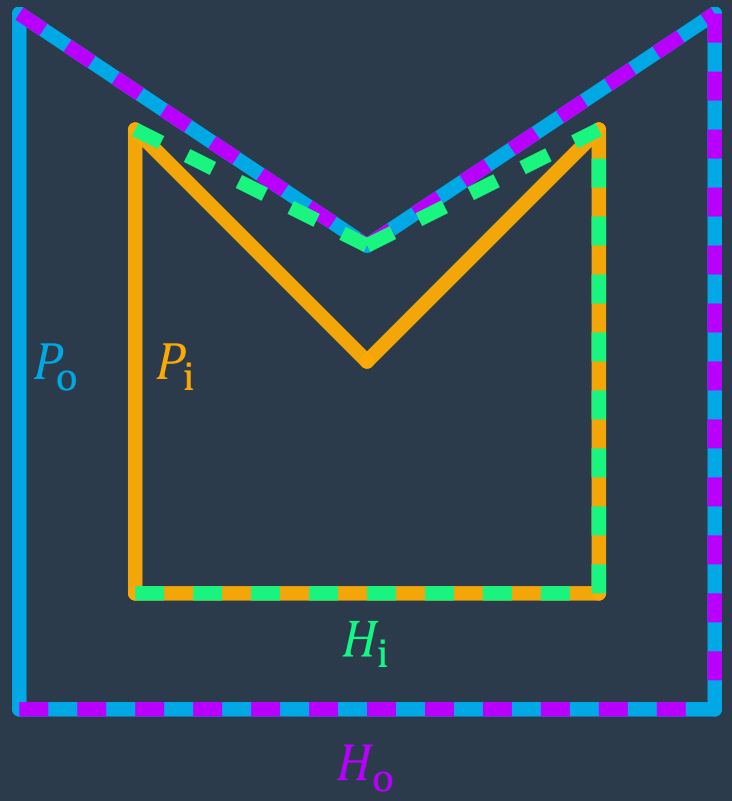
Lösungsvorschlag I
Idee

- Das neue innere Polygon $\tilde{P}_\mathrm{i}$ enthält den Start- und Endpunkt von C_i sowie alle Punkte von P_o, die zwischen C_o und innerhalb C_i liegen
- Das neue äußere Polygon $\tilde{P}_\mathrm{o}$ enthält den Start- und Endpunkt von C_i sowie alle Punkte von P_i, die innerhalb von C_i liegen
- $\tilde{H}_\mathrm{i}$ ist die Lösung dieses Teilproblems



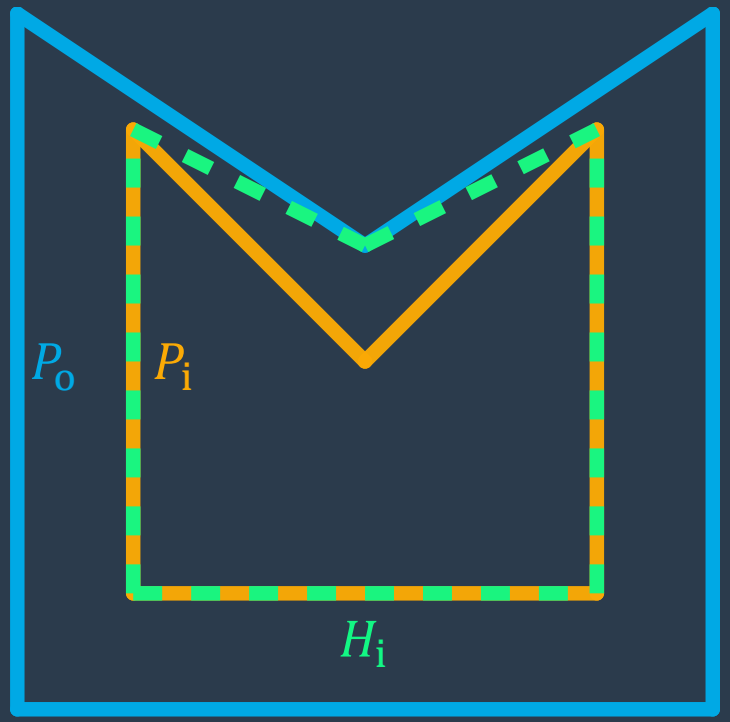
Lösungsvorschlag I
Idee

- $\widetilde{H}_{\mathrm{i}}$ ist die Lösung dieses Teilproblems
- Füge die Lösung des Teilproblems zum ursprünglichen Problem hinzu

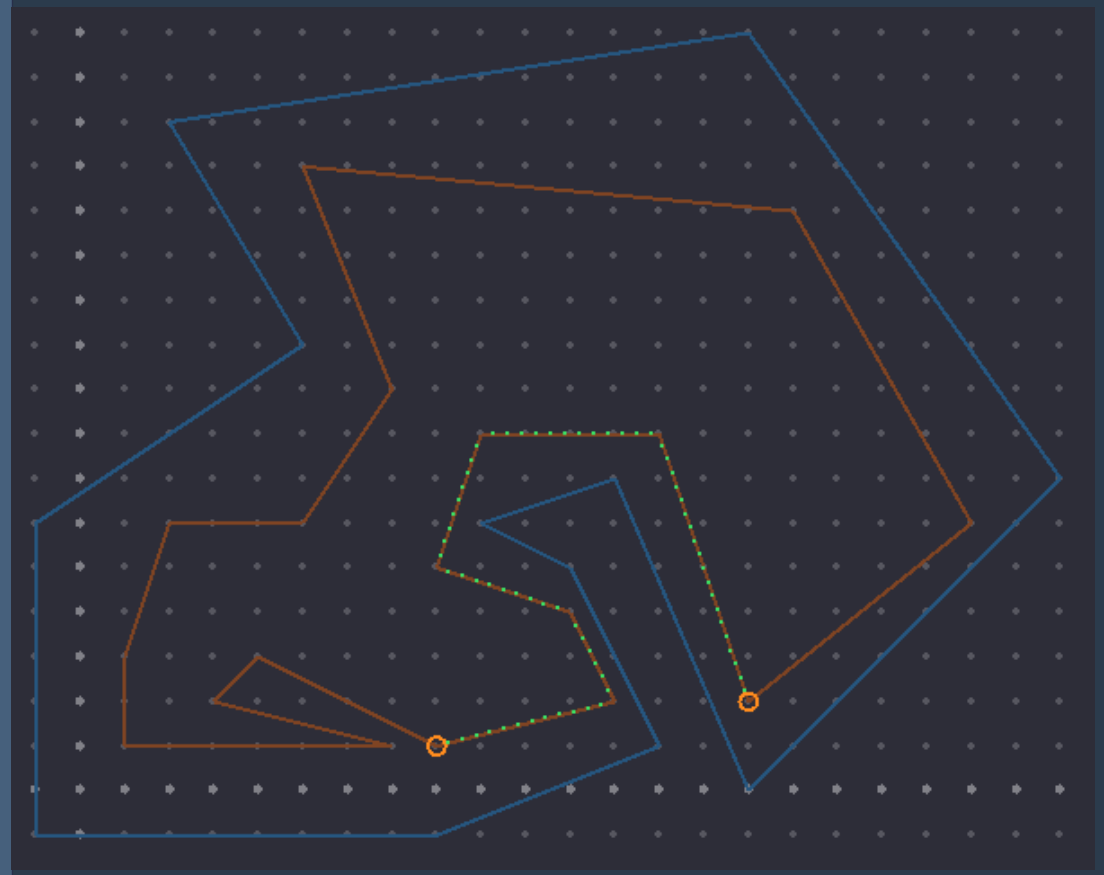


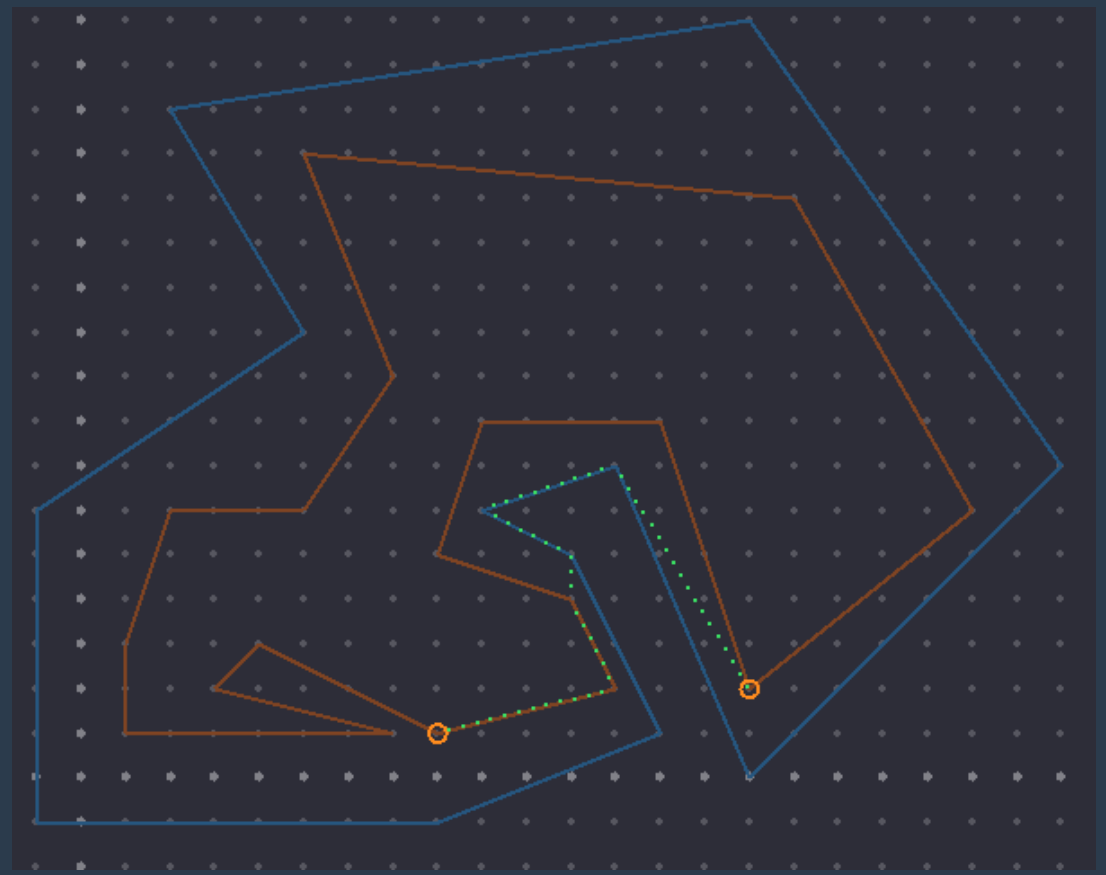
Lösungsvorschlag I
Idee

- Die konvexe Hülle $\widetilde{H}_\mathrm{i}$ des inneren Polygons $\tilde{P}_\mathrm{i}$ ist die Lösung dieses Teilproblems
- Füge die Lösung des Teilproblems zum ursprünglichen Problem hinzu
- Führe die vorherigen Schritte durch, bis alle Punkte abgegangen wurden

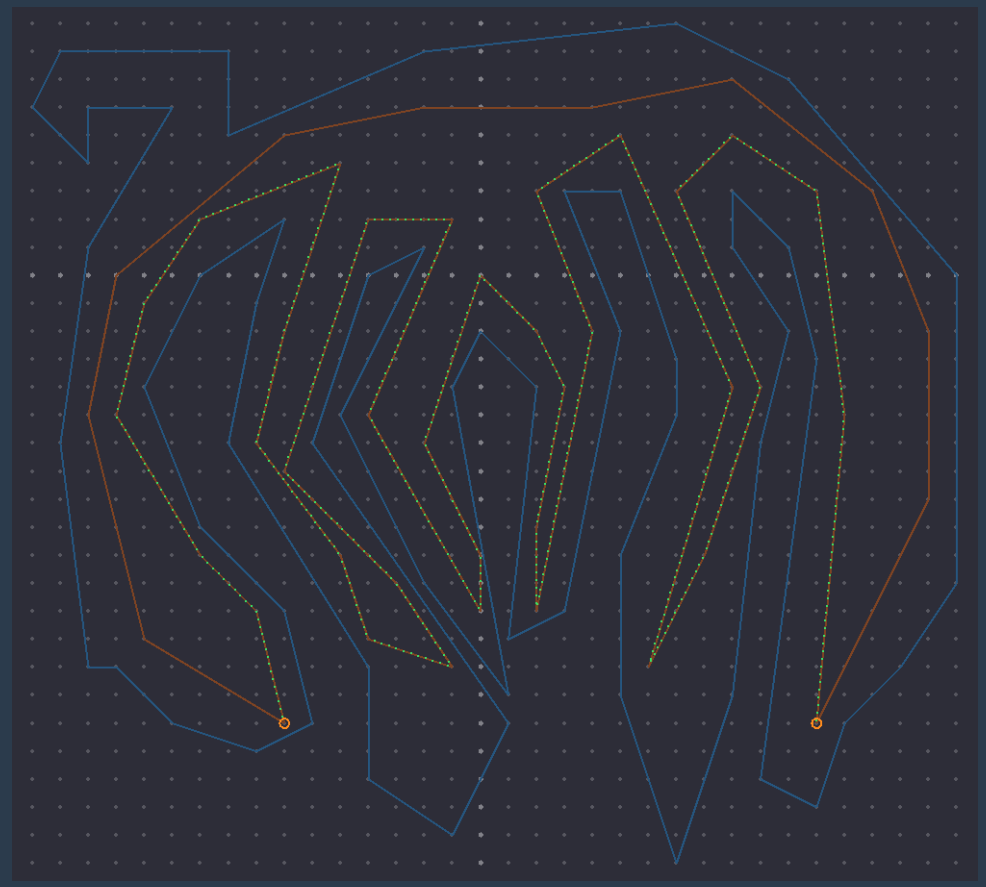


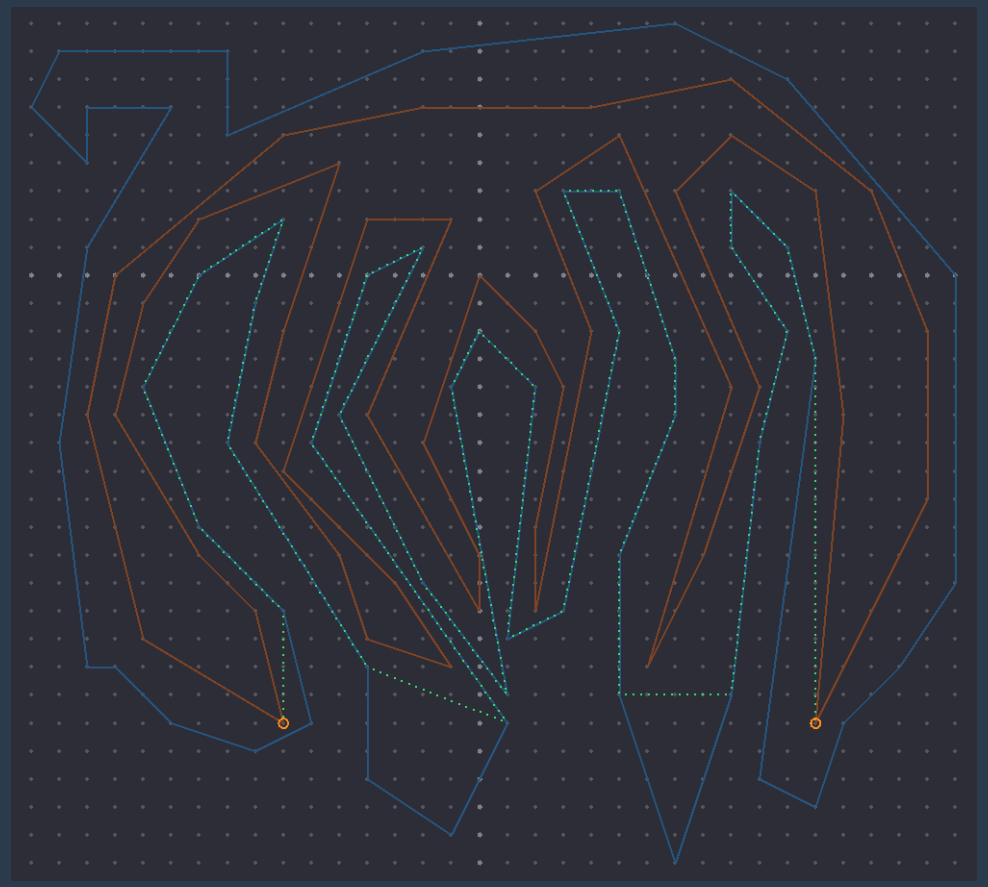
Lösungsvorschlag I
Besonderheiten





Lösungsvorschlag I
Besonderheiten





Lösungsvorschlag I
Pseudocode

Function *relativeConvexHull*$(P_\mathrm{i}, P_\mathrm{o})$
  input : Polygons $P_\mathrm{i} = \{p_1, p_2, ..., p_n\}$ and $P_\mathrm{o} = \{q_1, q_2, ..., q_m\}$, $P_\mathrm{i} \subset P_\mathrm{o}$
  output: Relative convex hull of P_i

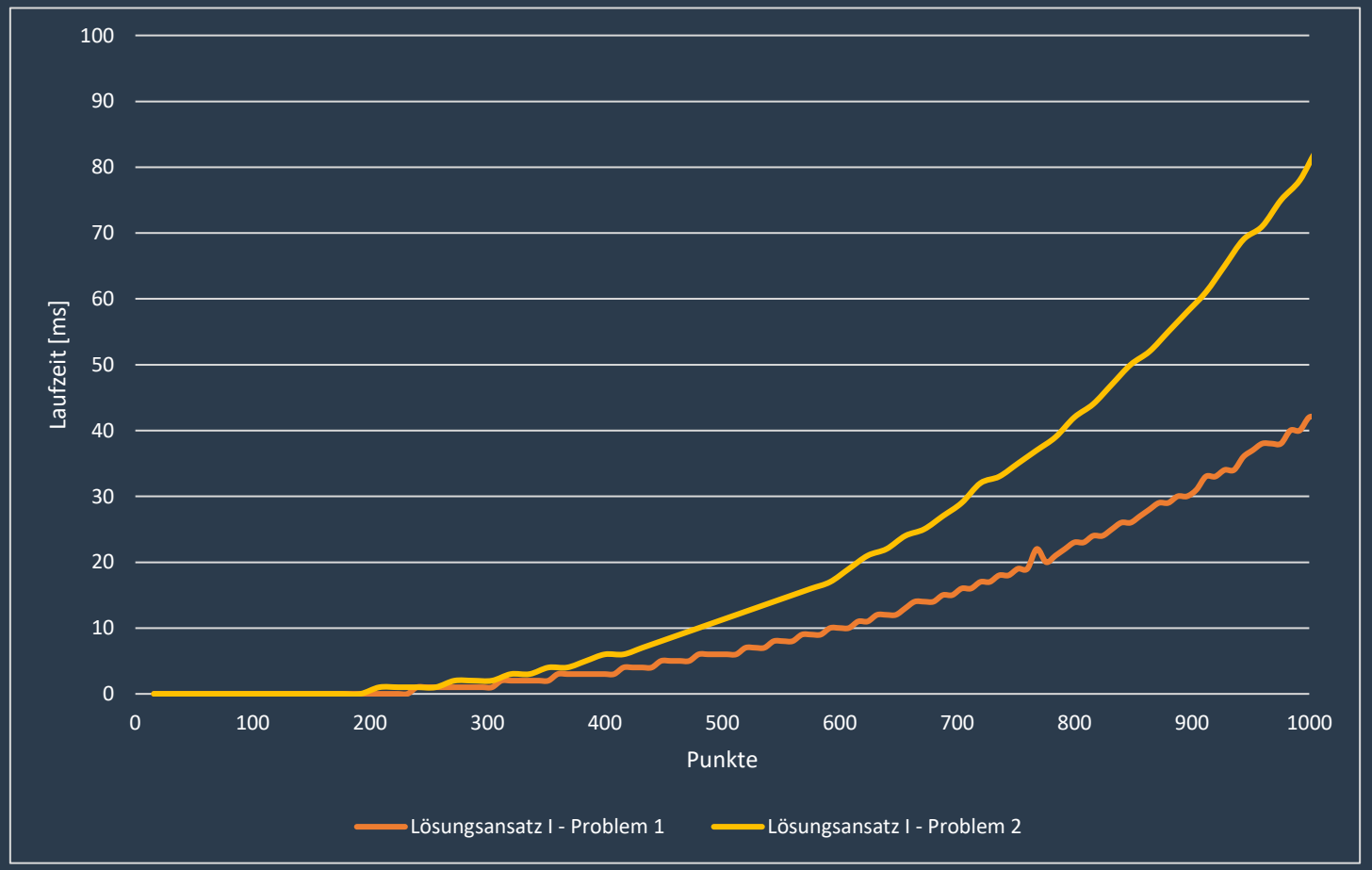
1 compute convex hulls H_i and H_o of P_i and P_o, respectively
2 **while** *cavity* C_i *between two consecutive vertices in* H_i **do**
3 **if** *overlapping cavity* C_o *between two consecutive vertices in* H_o **then**
4 $p_s \leftarrow$ start of cavity C_i
5 $p_e \leftarrow$ end of cavity C_i
6 $\tilde{P}_\mathrm{o} \leftarrow \{ p_s, p_e \} \cup \{ p \in P_\mathrm{i} \mid p \text{ is inside the cavity } C_\mathrm{i} \}$
7 $\tilde{P}_\mathrm{i} \leftarrow \{ p_s, p_e \} \cup \{ p \in P_\mathrm{o} \mid p \text{ is inside the cavity } C_\mathrm{i} \text{ and } \tilde{P}_\mathrm{o} \}$
8 **if** $\#\tilde{P}_\mathrm{i} \leq 3$ **then**
9 insert middle point of $\tilde{P}_\mathrm{i}$ into H_i between p_s and p_e
10 **else**
11 $\tilde{H}_\mathrm{i} \leftarrow relativeConvexHull(\tilde{P}_\mathrm{i}, \tilde{P}_\mathrm{o})$
12 insert points of $\tilde{H}_\mathrm{i}$ into H_i between p_s and p_e
13 **end**
14 **end**
15 mark C_i as resolved
16 **end**
17 **return** H_i
end

Lösungsvorschlag I
Laufzeit

Problem 1

n = 6, m = 10
→ N = 16

n = 10, m = 14
→ N = 24



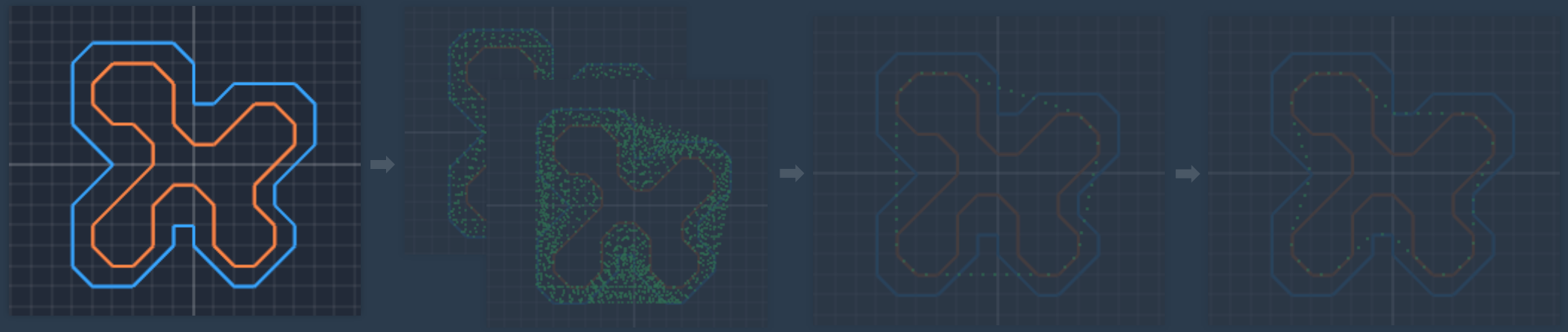
Problem 2

n = 8, m = 8
→ N = 16

n = 16, m = 16
→ N = 32

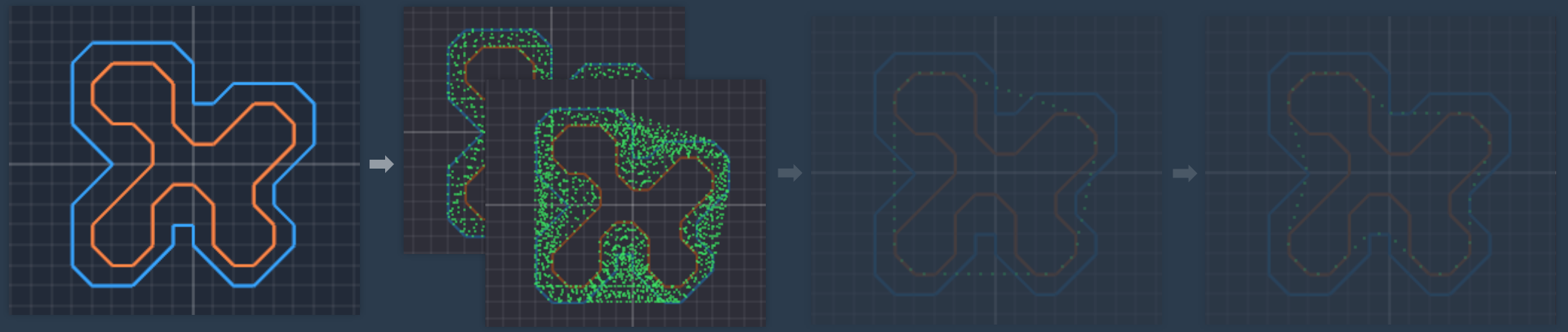
Problem N

Lösungsvorschlag II



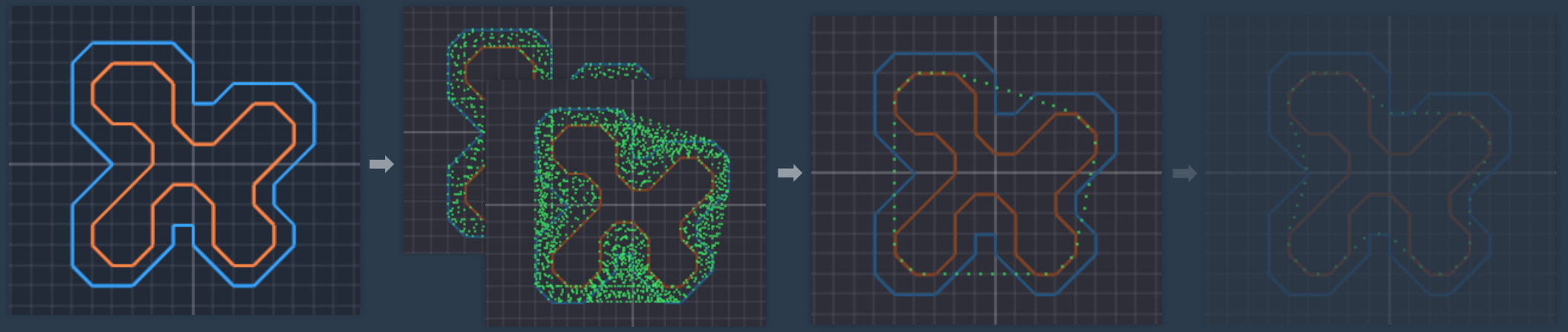
```
solve() {
    G = createGraph()
    C = createConvexHullOfInnerPolygon(G)
    return findShortestPathBetweenCornersOfConvexHull(G, C)
}
```

Lösungsvorschlag II



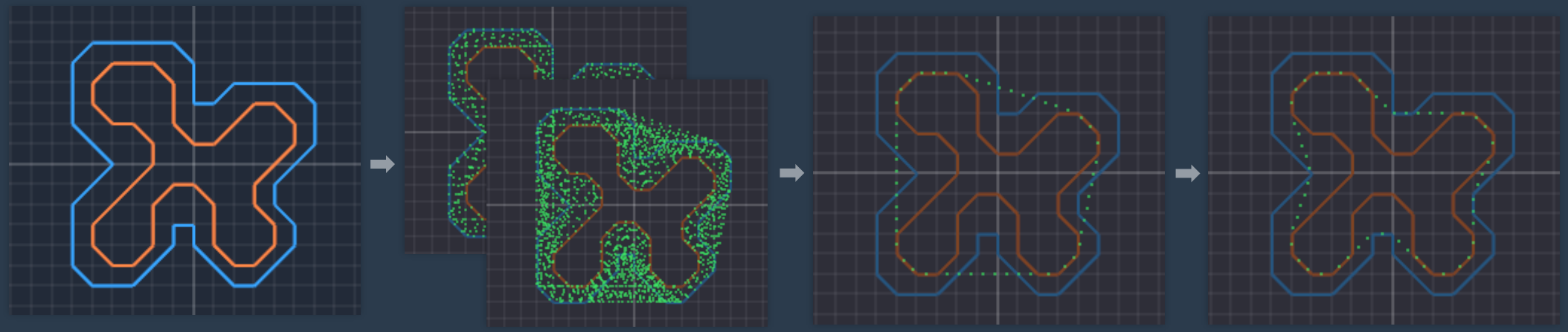
```
solve() {
    G = createGraph()
    C = createConvexHullOfInnerPolygon(G)
    return findShortestPathBetweenCornersOfConvexHull(G, C)
}
```

Lösungsvorschlag II



```
solve() {
    G = createGraph()
    C = createConvexHullOfInnerPolygon(G)
    return findShortestPathBetweenCornersOfConvexHull(G, C)
}
```

Lösungsvorschlag II



```
solve() {
    G = createGraph()
    C = createConvexHullOfInnerPolygon(G)
    return findShortestPathBetweenCornersOfConvexHull(G, C)
}
```

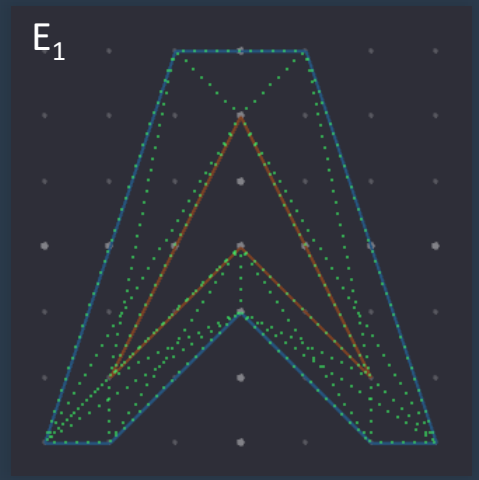
Lösungsvorschlag II
Erzeugung des Graphen

- Der Graph besteht aus einer Knotenmenge und **zwei** Kantenmengen:

createGraph

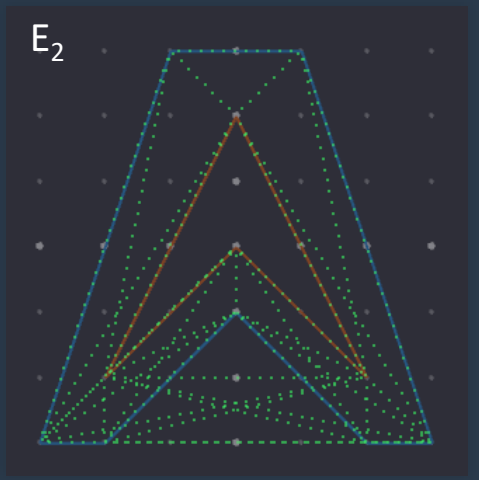
createConvexHull

findShortestPath



Respektiert beide Polygone.

Dient der Suche nach kürzesten Wegen.



Respektiert nur das innere Polygon.

Erleichtert das Generieren der konvexen Hülle des inneren Polygons.

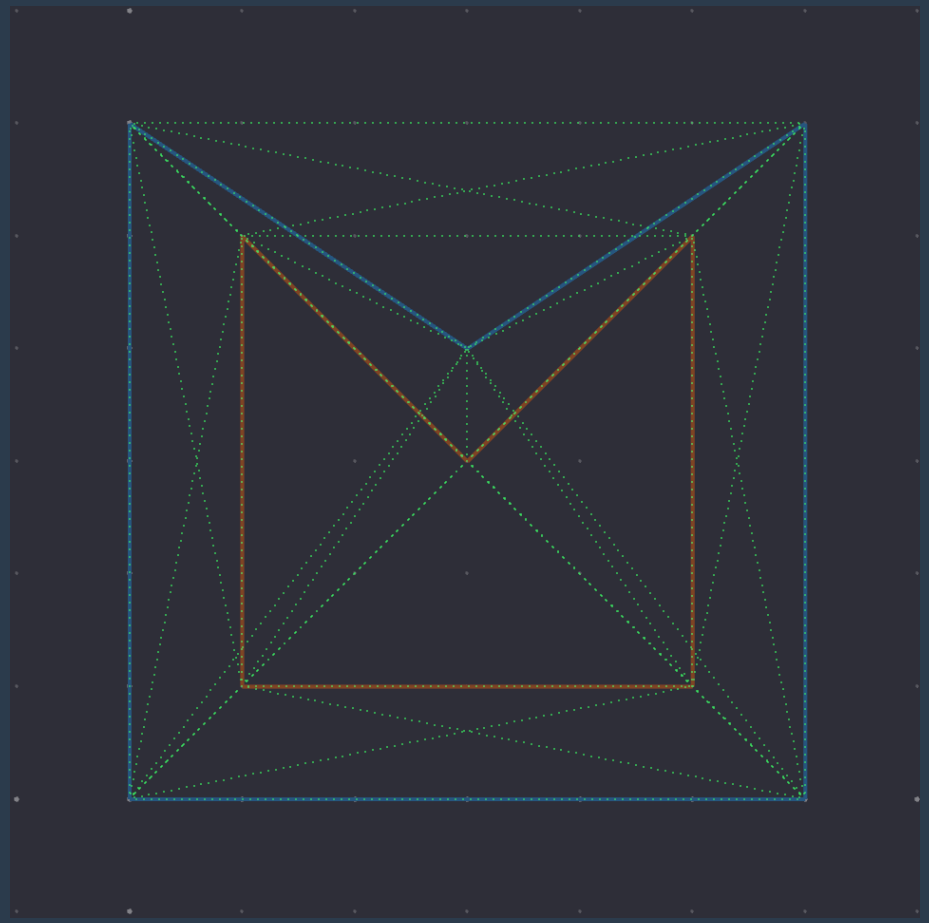
Nur notwendig, wenn ihr den Bonusalgorithmus implementiert.

Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath



← Alle potentiellen Kanten

Welche Kanten sollen entfernt werden?

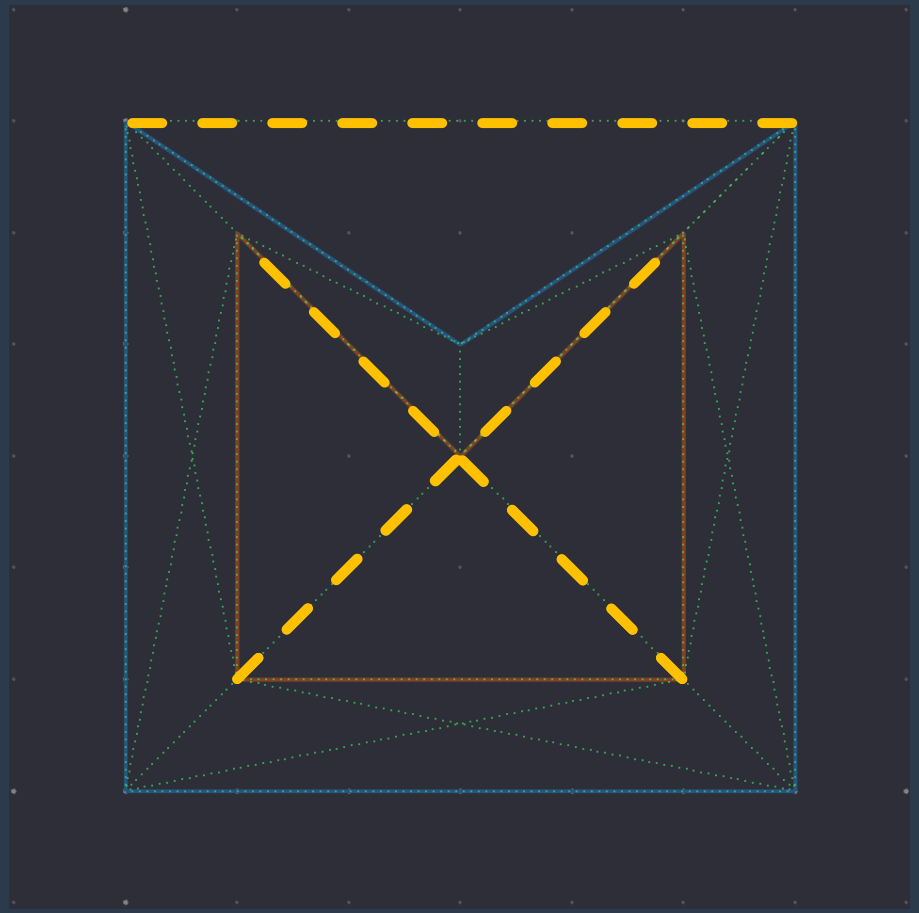
Idee: Entferne Kanten, die die Polygone schneiden.

Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath



Verläuft eine Kante innerhalb oder außerhalb eines Polygons?

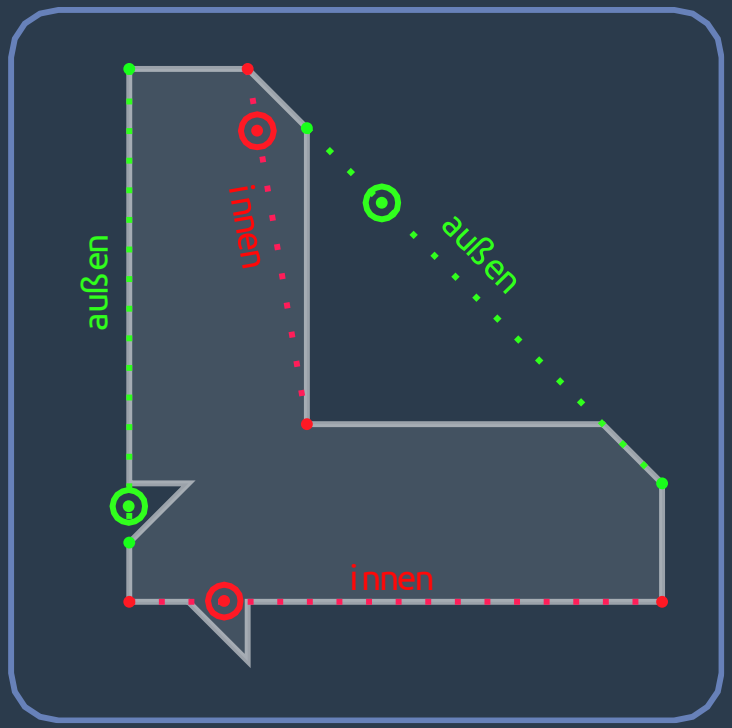
createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Erzeugung des Graphen

Beobachtung:



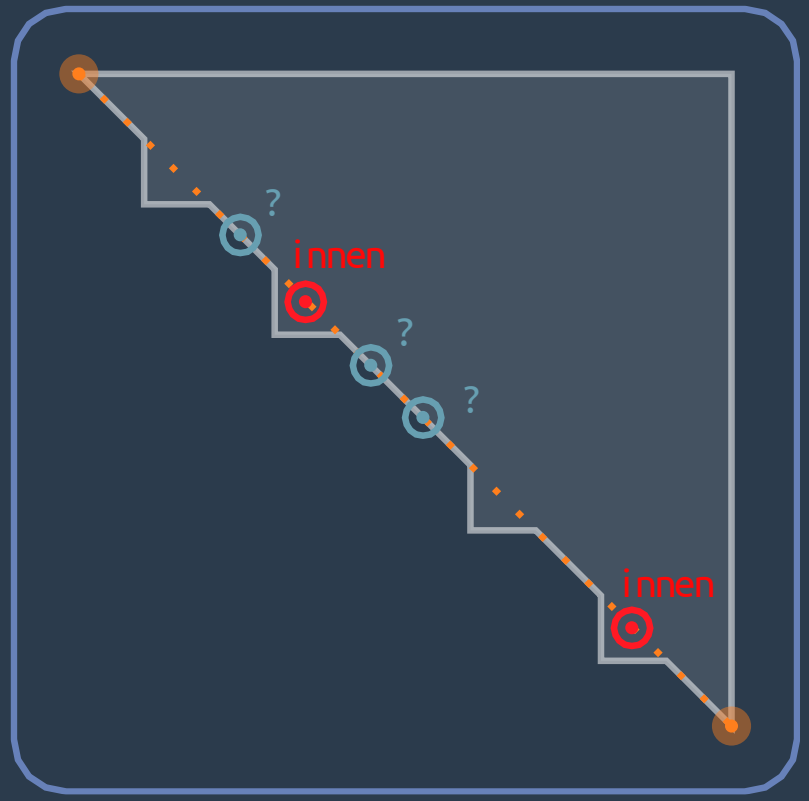
Eine Kante verläuft innerhalb/außerhalb eines Polygons, wenn es einen oder mehr Punkte auf der Kante gibt, die sich innerhalb/außerhalb des Polygons befinden.

Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath



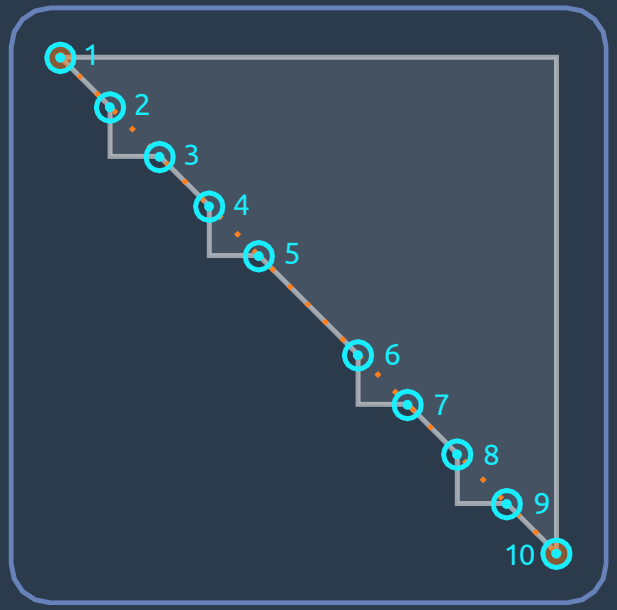
Welchen Punkt wählen wir?

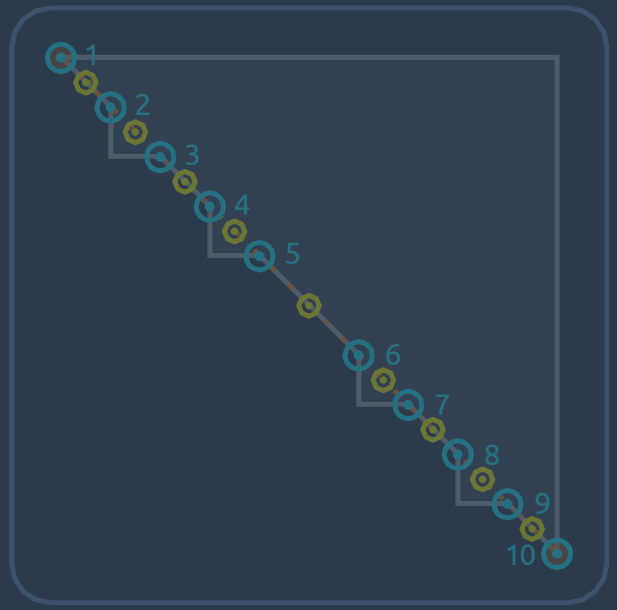
Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath





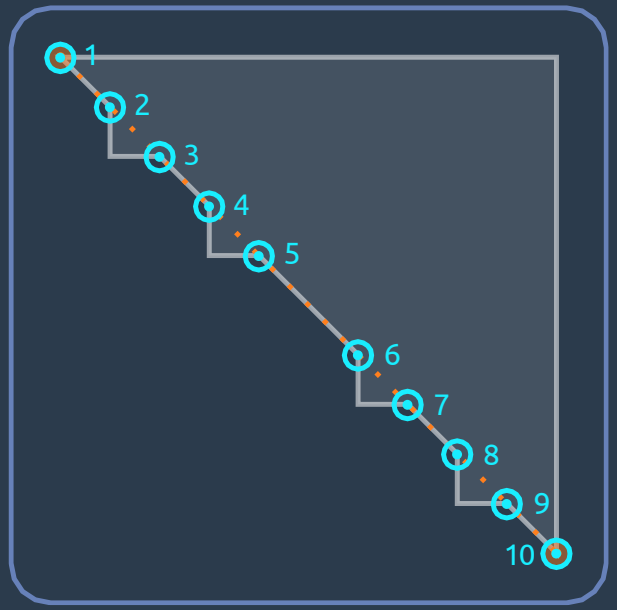
1. Finde alle Punkte, die auf der Kante liegen.

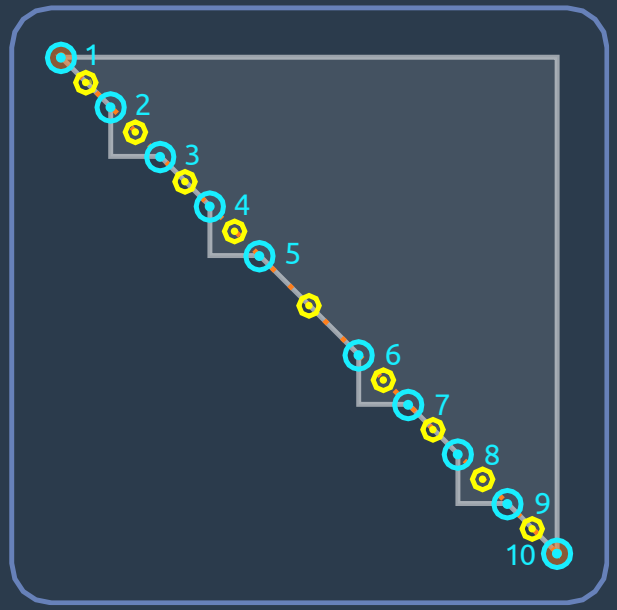
Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath





2. Überprüfe die entstandenen Liniensegmente.

createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Erzeugung des Graphen

```
Polygon::isViolatedByLineSegment(LineSegment s, bool doProhibidOutside) {
    if Polygon intersects with s return true

     // little optimization
     p = any point on s that is not s.p1 or s.p2
     if p is not on perimeter of Polygon then
         return p is within perimeter of Polygon == doProhibidInside

    pointsOnLineSegment = list of all points on s including s.p1 and s.p2

    sort pointsOnLineSegment by distance to s.p1

    for i = 1 to #pointsOnLineSegment (exclusive) do
        seg = new LineSegment(pointsOnLineSegment[i - 1], pointsOnLineSegment[i])
        midpoint = middle point of seg

        if midpoint lies on perimeter of Polygon then continue
        if midpoint is within perimeter of Polygon == doProhibidOutside then return true

    return false
}
```

Warnung: Array-Indizes fangen mit 0 an. ;)

Lösungsvorschlag II
Erzeugung des Graphen

- **createGraph**
- createConvexHull
- findShortestPath

```
Polygon::isViolatedByLineSegment(LineSegment s, bool doProhibidOutside) {
    if Polygon intersects with s return true

    // little optimization
    p = any point on s that is not s.p1 or s.p2
    if p is not on perimeter of Polygon then
        return p is within perimeter of Polygon == doProhibidInside

    pointsOnLineSegment = list of all points on s including s.p1 and s.p2

    sort pointsOnLineSegment by distance to s.p1

    for i = 1 to #pointsOnLineSegment (exclusive) do
        seg = new LineSegment(pointsOnLineSegment[i - 1], pointsOnLineSegment[i])
        midpoint = middle point of seg

        if midpoint lies on perimeter of Polygon then continue
        if midpoint is within perimeter of Polygon == doProhibidOutside then return true

    return false
}
```

Kanten, die ein Polygon schneiden, werden direkt aussortiert.

Warnung: Array-Indizes fangen mit 0 an. ;)

Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath

```
Polygon::isViolatedByLineSegment(LineSegment s, bool doProhibidOutside) {
    if Polygon intersects with s return true

    // little optimization
    p = any point on s that is not s.p1 or s.p2
    if p is not on perimeter of Polygon then
        return p is within perimeter of Polygon == doProhibidInside

    pointsOnLineSegment = list of all points on s including s.p1 and s.p2

    sort pointsOnLineSegment by distance to s.p1

    for i = 1 to #pointsOnLineSegment (exclusive) do
        seg = new LineSegment(pointsOnLineSegment[i - 1], pointsOnLineSegment[i])
        midpoint = middle point of seg

        if midpoint lies on perimeter of Polygon then continue
        if midpoint is within perimeter of Polygon == doProhibidOutside then return true

    return false
}
```

Wir wählen einen beliebigen Punkt auf `s` und überprüfen, ob wir direkt rausfinden können, ob sich die Kante innerhalb/außerhalb des `Polygons` befindet.

Warnung: Array-Indizes fangen mit 0 an. ;)

Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath

```
Polygon::isViolatedByLineSegment(LineSegment s, bool doProhibidOutside) {
    if Polygon intersects with s return true

     // little optimization
     p = any point on s that is not s.p1 or s.p2
     if p is not on perimeter of Polygon then
         return p is within perimeter of Polygon == doProhibidInside

    pointsOnLineSegment = list of all points on s including s.p1 and s.p2

    sort pointsOnLineSegment by distance to s.p1

    for i = 1 to #pointsOnLineSegment (exclusive) do
        seg = new LineSegment(pointsOnLineSegment[i - 1], pointsOnLineSegment[i])
        midpoint = middle point of seg

        if midpoint lies on perimeter of Polygon then continue
        if midpoint is within perimeter of Polygon == doProhibidOutside then return true

    return false
}
```

Falls nicht, erstellen wir eine sortierte Liste mit allen Knoten, die sich auf der Kante `s` befinden.

Warnung: Array-Indizes fangen mit 0 an. ;)

Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath

```
Polygon::isViolatedByLineSegment(LineSegment s, bool doProhibidOutside) {
    if Polygon intersects with s return true

     // little optimization
     p = any point on s that is not s.p1 or s.p2
     if p is not on perimeter of Polygon then
         return p is within perimeter of Polygon == doProhibidInside

    pointsOnLineSegment = list of all points on s including s.p1 and s.p2

    sort pointsOnLineSegment by distance to s.p1

    for i = 1 to #pointsOnLineSegment (exclusive) do
        seg = new LineSegment(pointsOnLineSegment[i - 1], pointsOnLineSegment[i])
        midpoint = middle point of seg

        if midpoint lies on perimeter of Polygon then continue
        if midpoint is within perimeter of Polygon == doProhibidOutside then return true

    return false
}
```

Wir generieren alle Untersegmente und deren Mittelpunkte.

Warnung: Array-Indizes fangen mit 0 an. ;)

Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath

```
Polygon::isViolatedByLineSegment(LineSegment s, bool doProhibidOutside) {
    if Polygon intersects with s return true

     // little optimization
     p = any point on s that is not s.p1 or s.p2
     if p is not on perimeter of Polygon then
         return p is within perimeter of Polygon == doProhibidInside

    pointsOnLineSegment = list of all points on s including s.p1 and s.p2

    sort pointsOnLineSegment by distance to s.p1

    for i = 1 to #pointsOnLineSegment (exclusive) do
        seg = new LineSegment(pointsOnLineSegment[i - 1], pointsOnLineSegment[i])
        midpoint = middle point of seg

        if midpoint lies on perimeter of Polygon then continue
        if midpoint is within perimeter of Polygon == doProhibidOutside then return true

    return false
}
```

Falls wir einen Mittelpunkt finden, der nicht auf dem Umfang des `Polygons` liegt, so haben wir rausgefunden, ob sich die Kante `s` innerhalb oder außerhalb des `Polygons` befindet.

Warnung: Array-Indizes fangen mit 0 an. ;)

Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath

```
Polygon::isViolatedByLineSegment(LineSegment s, bool doProhibidOutside) {
    if Polygon intersects with s return true

     // little optimization
     p = any point on s that is not s.p1 or s.p2
     if p is not on perimeter of Polygon then
         return p is within perimeter of Polygon == doProhibidInside

    pointsOnLineSegment = list of all points on s including s.p1 and s.p2

    sort pointsOnLineSegment by distance to s.p1

    for i = 1 to #pointsOnLineSegment (exclusive) do
        seg = new LineSegment(pointsOnLineSegment[i - 1], pointsOnLineSegment[i])
        midpoint = middle point of seg

        if midpoint lies on perimeter of Polygon then continue
        if midpoint is within perimeter of Polygon == doProhibidOutside then return true

    return false
}
```

Kanten, die vollständig auf dem Umfang des `Polygons` liegen, werden grundsätzlich erlaubt.

Warnung: Array-Indizes fangen mit 0 an. ;)

Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath

```
addEdgeIfItIsLegal(Graph G = (V, E₁, E₂), Node source ∈ V, Node destination ∈ V) {
    s = source.position
    d = destination.position
    segment = new LineSegment(s, d)

    if innerPolygon.isViolatedByLineSegment(segment, true) then
        return

    edgeLength = length of segment
    E₂ += (source, destination, edgeLength)

    if not outerPolygon.isViolatedByLineSegment(segment, false) then
        E₁ += (source, destination, edgeLength)
}
```

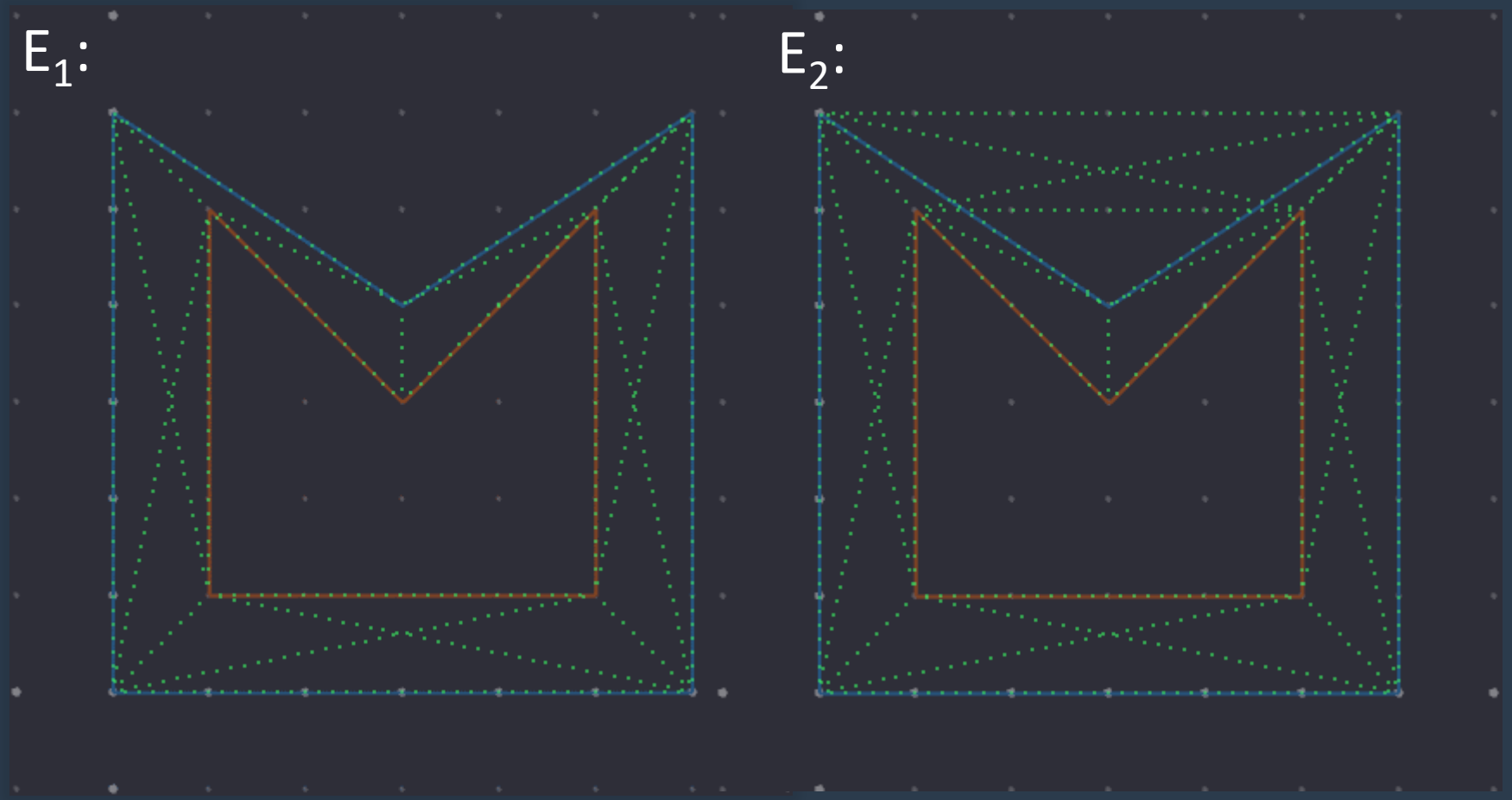
Info: G ist ungerichtet.

Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath



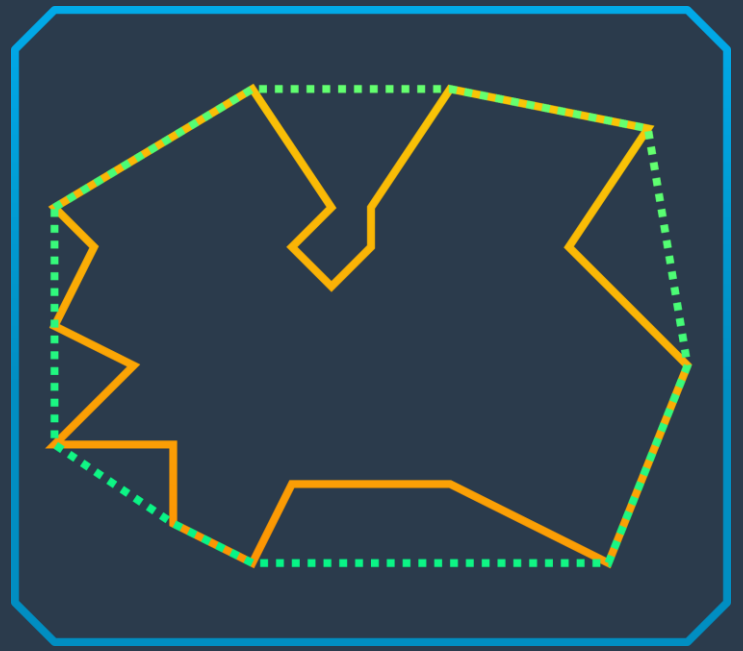
createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle

- Gesucht ist die konvexe Hülle des inneren Polygons.



- Melkman $(O(n))$
- Graham Scan $(O(n \log n))$
- Bonusalgorithmus $(O(n)^*)$

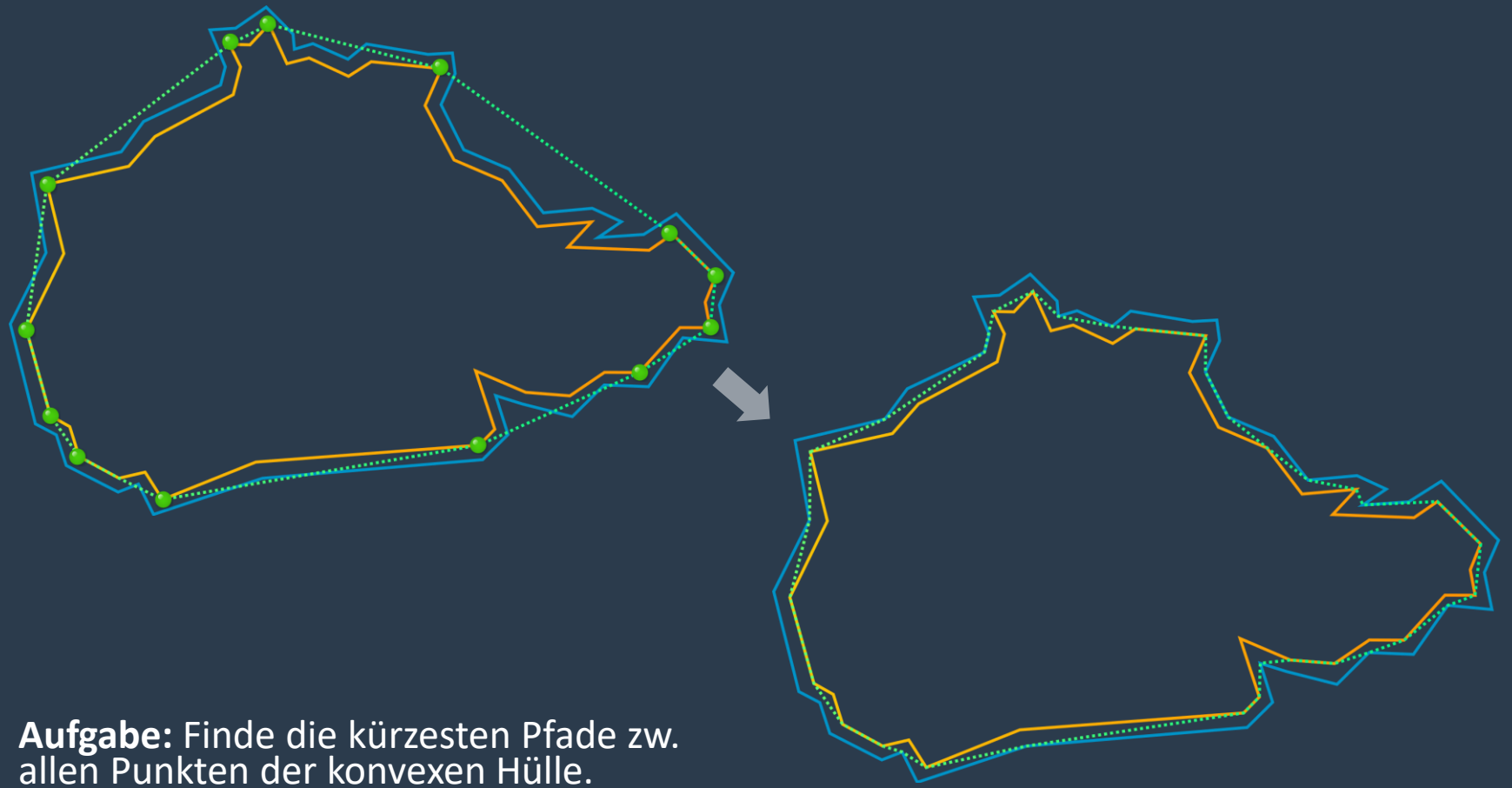
* Sofern ihr den Graphen mit einer Adjazenzmatrix implementiert.

createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Finden des kürzesten Pfades



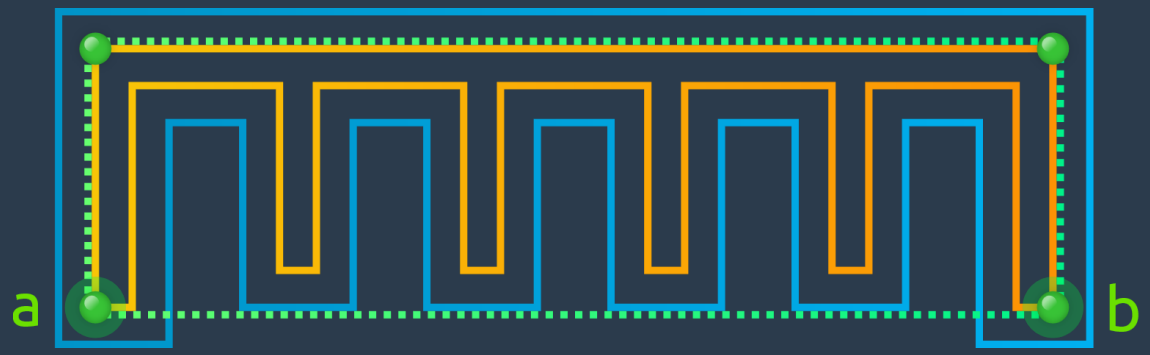
Aufgabe: Finde die kürzesten Pfade zw. allen Punkten der konvexen Hülle.

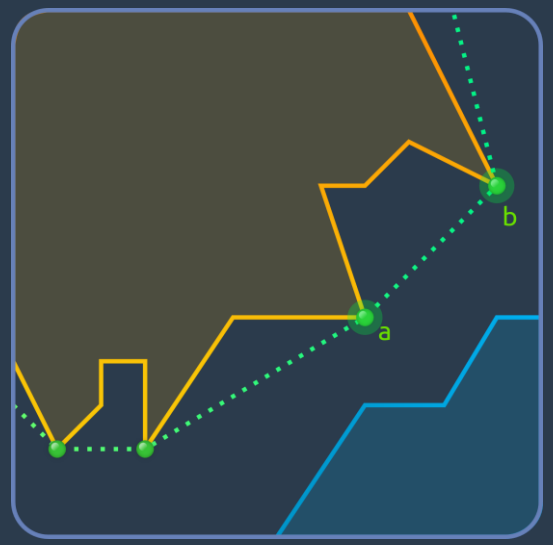
createGraph

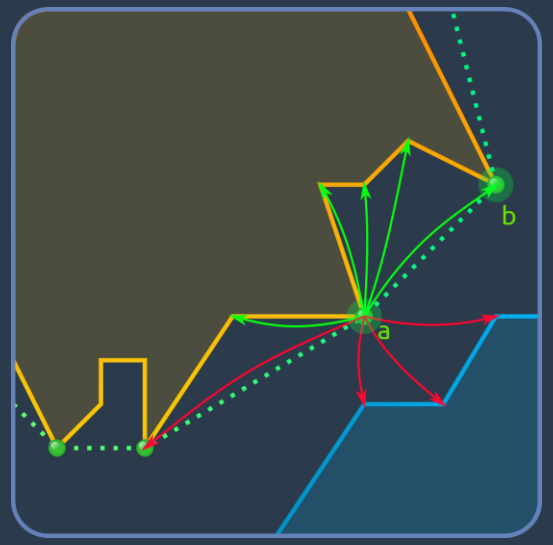
createConvexHull

findShortestPath

Lösungsvorschlag II
Finden des kürzesten Pfades







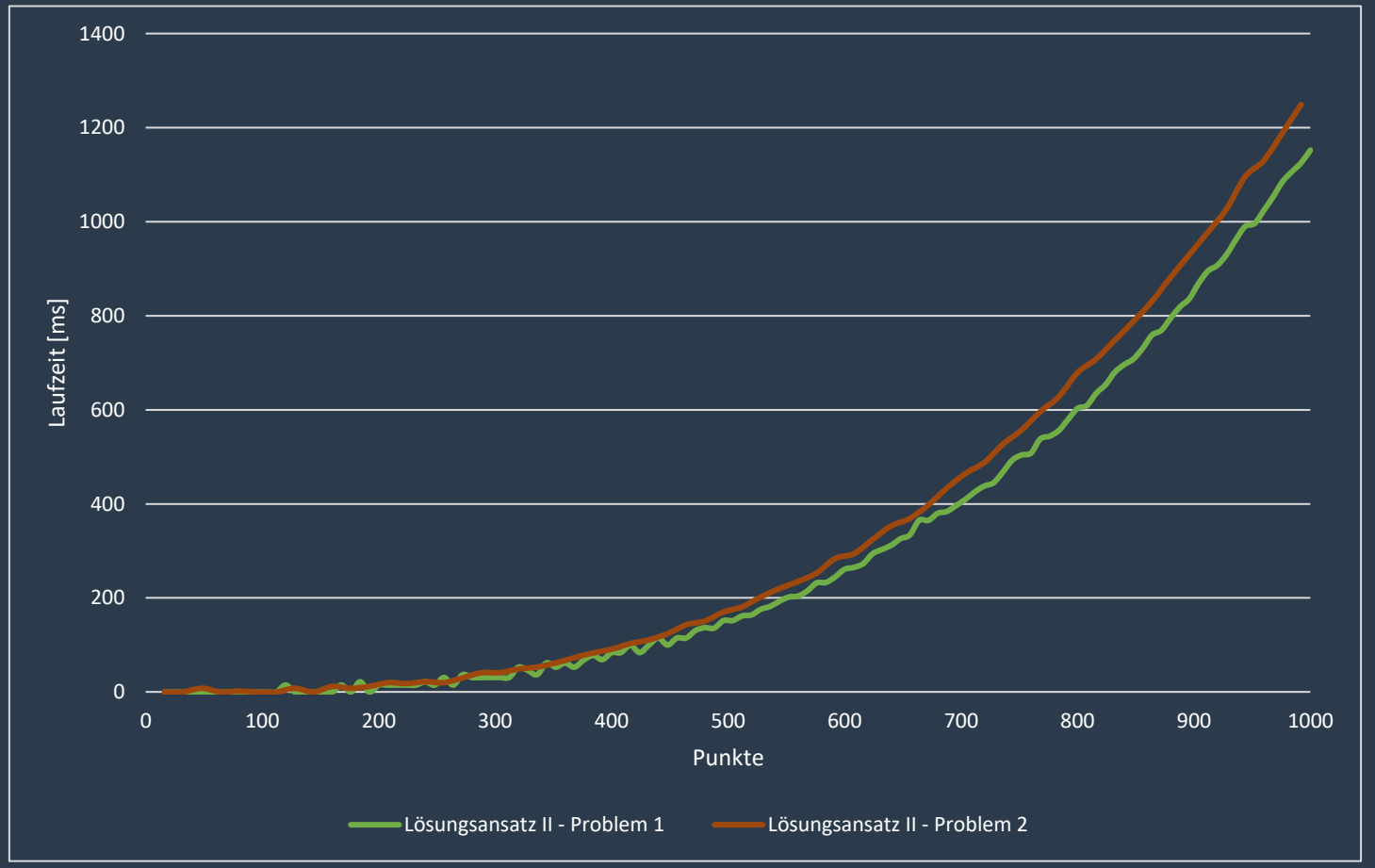
- Pfadsuche nur innerhalb der konvexen Hülle.
- Ecken der konvexen Hülle (außer b) dürfen nicht besucht werden.

Lösungsvorschlag II
Laufzeit

Problem 1

n = 6, m = 10
→ N = 16

n = 10, m = 14
→ N = 24



Problem 2

n = 8, m = 8
→ N = 16

n = 16, m = 16
→ N = 32

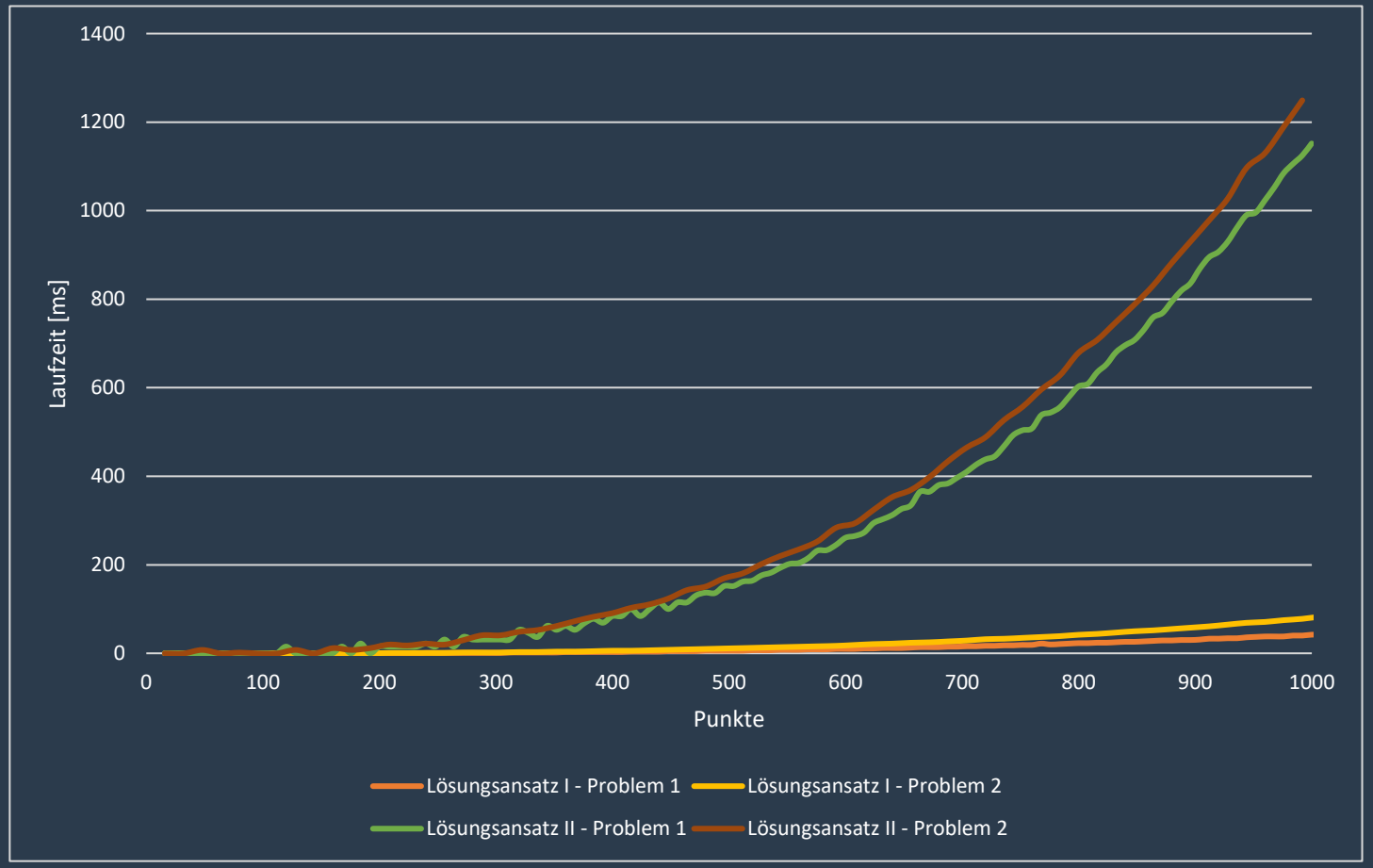
Problem N

Vergleich der Laufzeiten

Problem 1

n = 6, m = 10
→ N = 16

n = 10, m = 14
→ N = 24



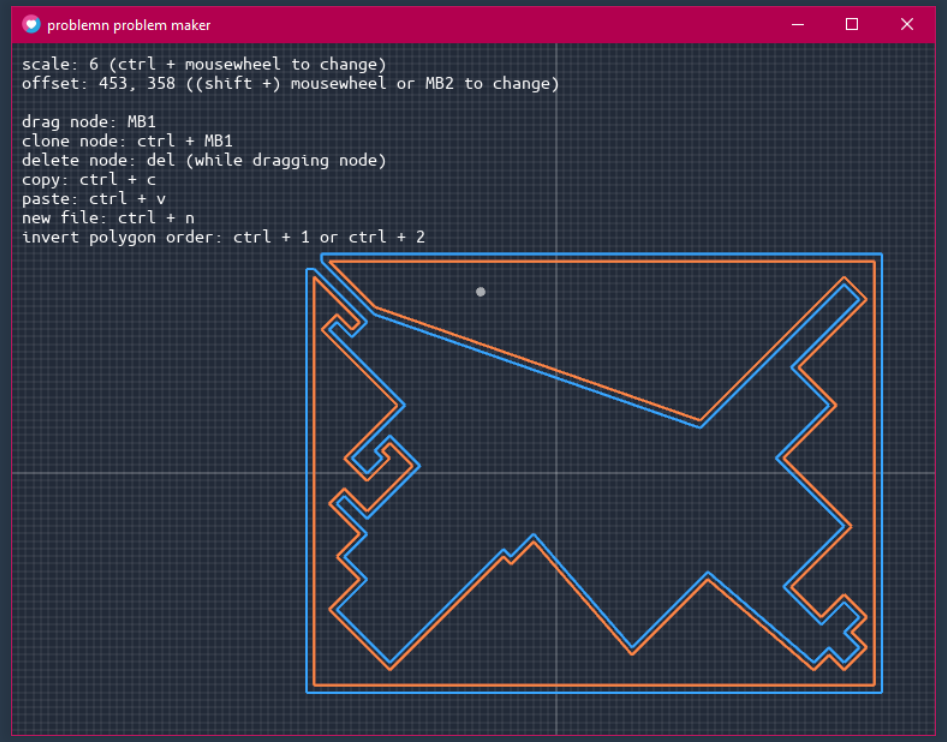
Problem 2

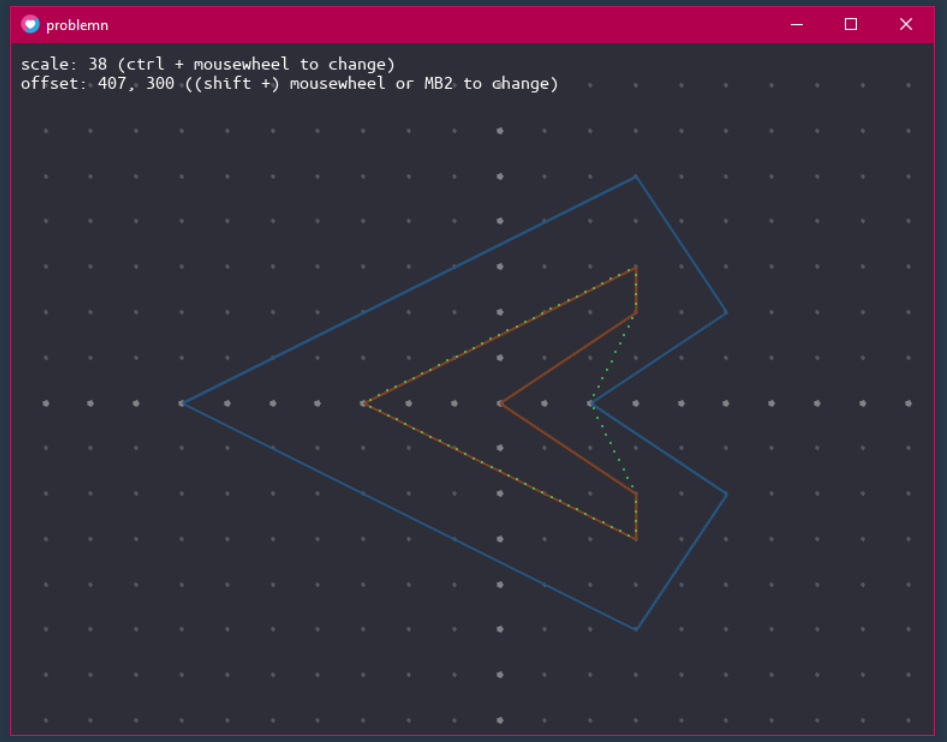
n = 8, m = 8
→ N = 16

n = 16, m = 16
→ N = 32

Problem N

Hilfstools


problemn problem maker
scale: 6 (ctrl + mousewheel to change)
offset: 453, 358 ((shift +) mousewheel or MB2 to change)
drag node: MB1
clone node: ctrl + MB1
delete node: del (while dragging node)
copy: ctrl + c
paste: ctrl + v
new file: ctrl + n
invert polygon order: ctrl + 1 or ctrl + 2


problemn
scale: 38 (ctrl + mousewheel to change)
offset: 407, 300 ((shift +) mousewheel or MB2 to change)

Danke für’s Zuhören
und viel Spaß beim Lösen!

Und vergesst die Bonusinhalte nicht. ;)

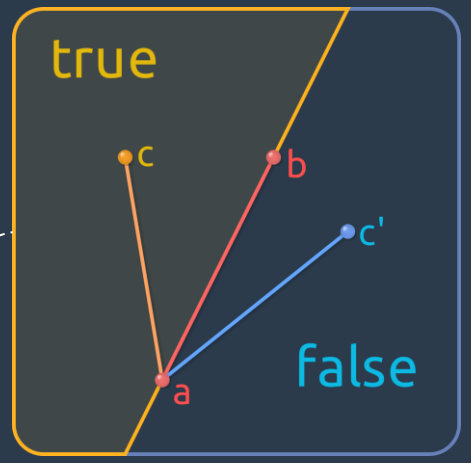
Utilities

```java
public static Orientation getOrientation(Point a, Point b, Point c) {
    final long crossProduct = (((long) b.x - a.x) * ((long) c.y - a.y)) -
        (((long) b.y - a.y) * ((long) c.x - a.x));

    if (crossProduct < 0) {
        return Orientation.CLOCKWISE;
    } else if (crossProduct > 0) {
        return Orientation.COUTER_CLOCKWISE;
    } else {
        return Orientation.COLLINEAR;
    }
}
```

```java
public boolean isLeft(Point a, Point b) {
    return getOrientation(a, b, this) == Orientation.COUNTER_CLOCKWISE;
}

public boolean isRight(Point a, Point b) {
    return getOrientation(a, b, this) == Orientation.CLOCKWISE;
}
```

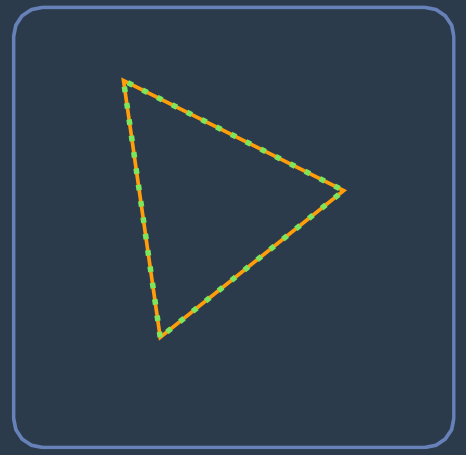


Utilities

```
public static boolean doIntersect(Point p1, Point p2, Point q1, Point q2) {
    final Line2D line1 = new Line2D.Double(p1, p2);
    final Line2D line2 = new Line2D.Double(q1, q2);

    return line1.intersectsLine(line2);
}
```

Utilities



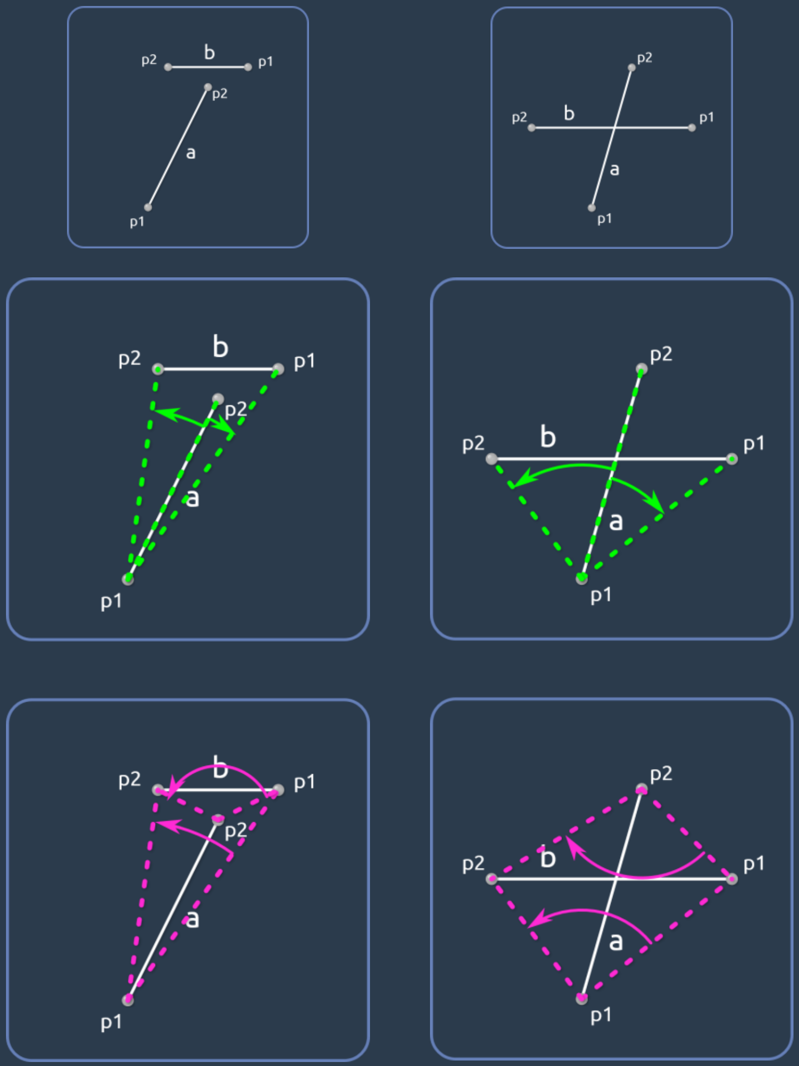
Wenn zwei Liniensegmente einen oder beide Endpunkte miteinander teilen, so schneiden sie sich nicht. Wir wollen uns ja entlang der Polygone bewegen können.

Wichtig für Lösungsvorschlag II, auch wenn ihr Java verwendet.

```
doLineSegmentsIntersect(LineSegment a, LineSegment b) {
    if a.p1 == b.p1 or
       a.p1 == b.p2 or
       a.p2 == b.p1 or
       a.p2 == b.p2 then return false

    return b.p2.isLeft(a.p1, b.p1) ≠ b.p2.isLeft(a.p2, b.p1) and
           b.p1.isLeft(a.p1, a.p2) ≠ b.p2.isLeft(a.p1, a.p2)
}
```

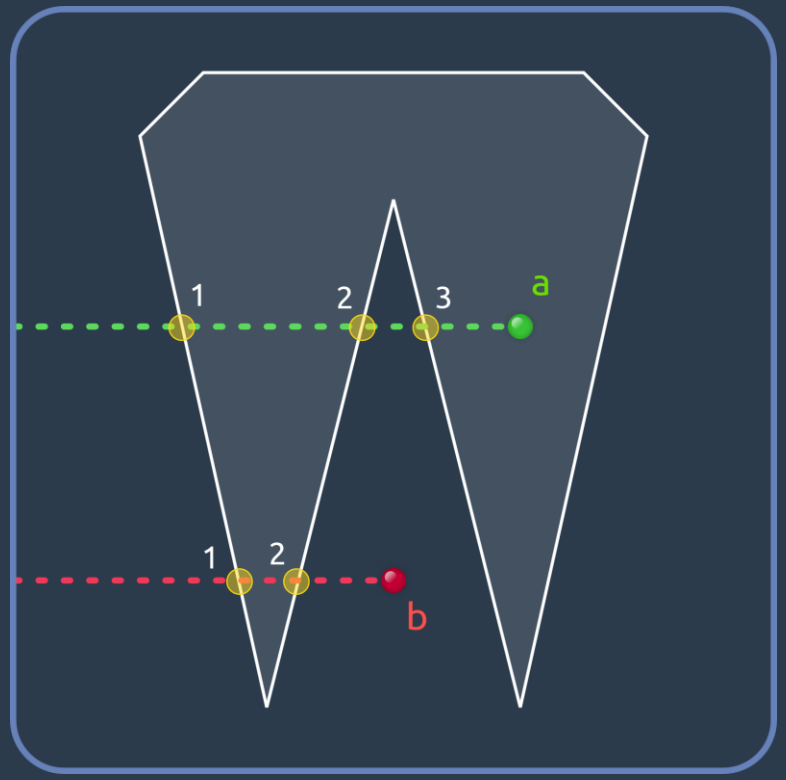
Utilities




```
doLineSegmentsIntersect(LineSegment a, LineSegment b) {
    if a.p1 == b.p1 or
       a.p1 == b.p2 or
       a.p2 == b.p1 or
       a.p2 == b.p2 then return false

    return b.p2.isLeft(a.p1, b.p1) ≠ b.p2.isLeft(a.p2, b.p1) and
           b.p1.isLeft(a.p1, a.p2) ≠ b.p2.isLeft(a.p1, a.p2)
}
```

Utilities



Ungerade Anzahl Schnitte

⇕

Punkt befindet sich im Polygon

Utilities

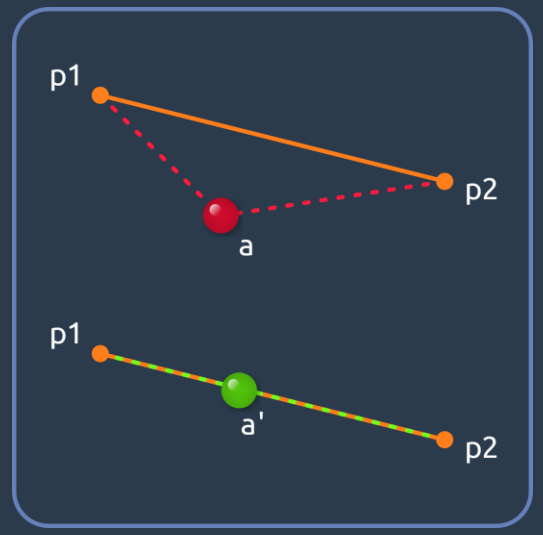
```
bool Point::isInside(std::vector<Point> &polygon) {
    auto nvert = polygon.size();
    size_t i, j;
    bool c = 0;
    for (i = 0, j = nvert - 1; i < nvert; j = i++) {
        if (polygon[i] == *this) {
            return true;
        }
        if (((polygon[i].y > this->y) != (polygon[j].y > this->y)) &&
            (this->x < (polygon[j].x - polygon[i].x) *
            (this->y - polygon[i].y) / (double) (polygon[j].y - polygon[i].y) + polygon[i].x))
            c = !c;
    }
    return c;
}
```

Lösungsvorschlag II
Erzeugung des Graphen

createGraph

createConvexHull

findShortestPath



Distanz zwischen a und b:

$$\sqrt{(b.y - a.y)^2 + (b.x - a.x)^2}$$

Der Punkt a liegt genau dann auf dem Liniensegment, wenn die Summe der Abstände zwischen a und den beiden Endpunkten gleich der Länge des Liniensegmentes ist.

createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle

Bonusalgorithmus:

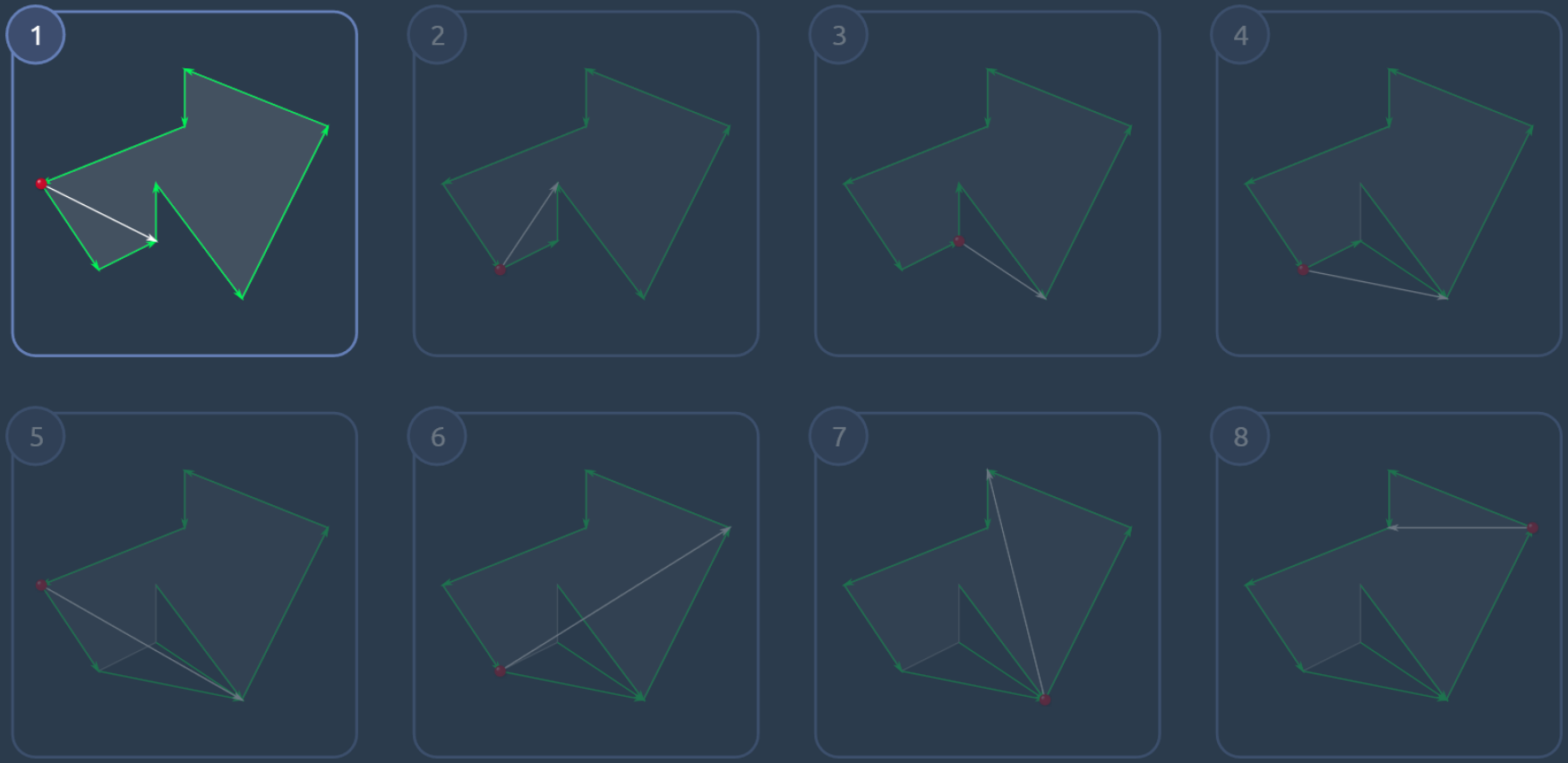
1. Gebe jedem Knoten auf dem Polygon einen Pointer, der auf den jeweils nächsten Knoten zeigt.
2. Fange auf dem linksten Knoten an.
3. Wenn es vom momentanen Knoten c eine nach rechts abbiegende Abkürzung zum übernächsten Knoten d gibt, lege den Pointer des Knotens c auf d und gehe einen Knoten zurück. Ansonsten, folge dem Pointer von c.
4. Wiederhole 3. so lange, bis du wieder am linksten Knoten angekommen bist.

Lösungsvorschlag II
Ermitteln der konvexen Hülle

createGraph

createConvexHull

findShortestPath



Lösungsvorschlag II
Ermitteln der konvexen Hülle

createGraph

createConvexHull

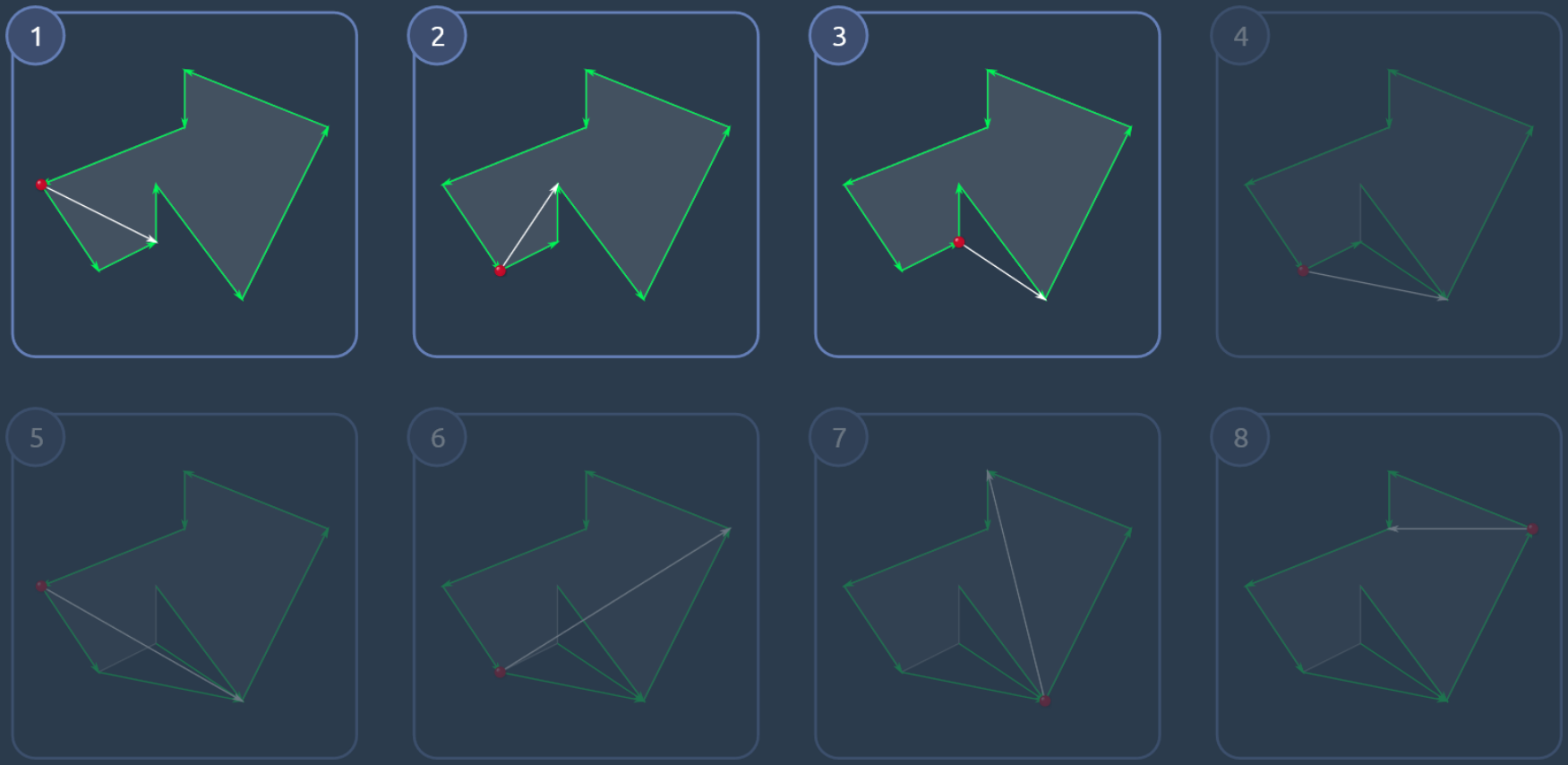
findShortestPath

createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle

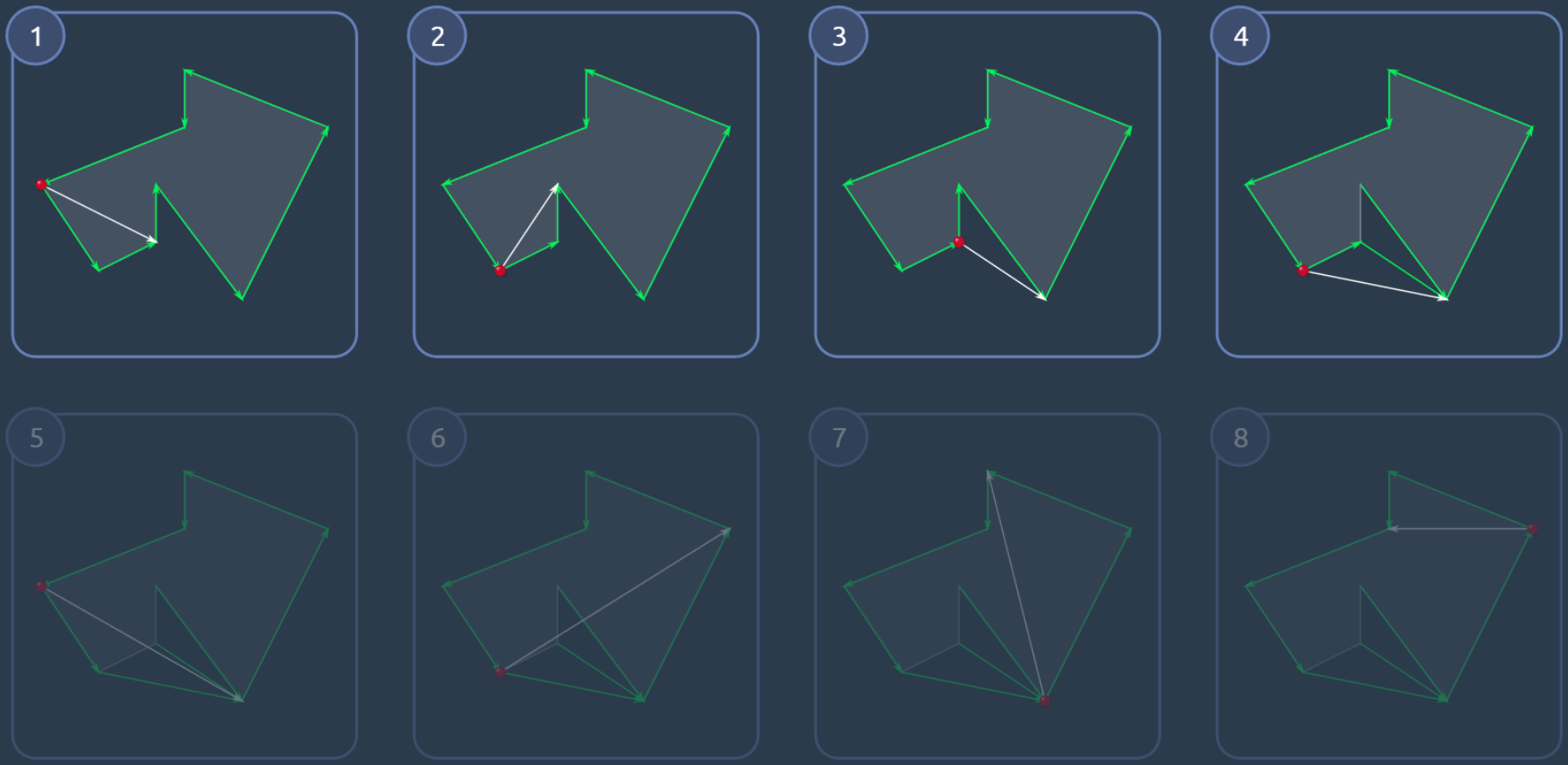


createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle

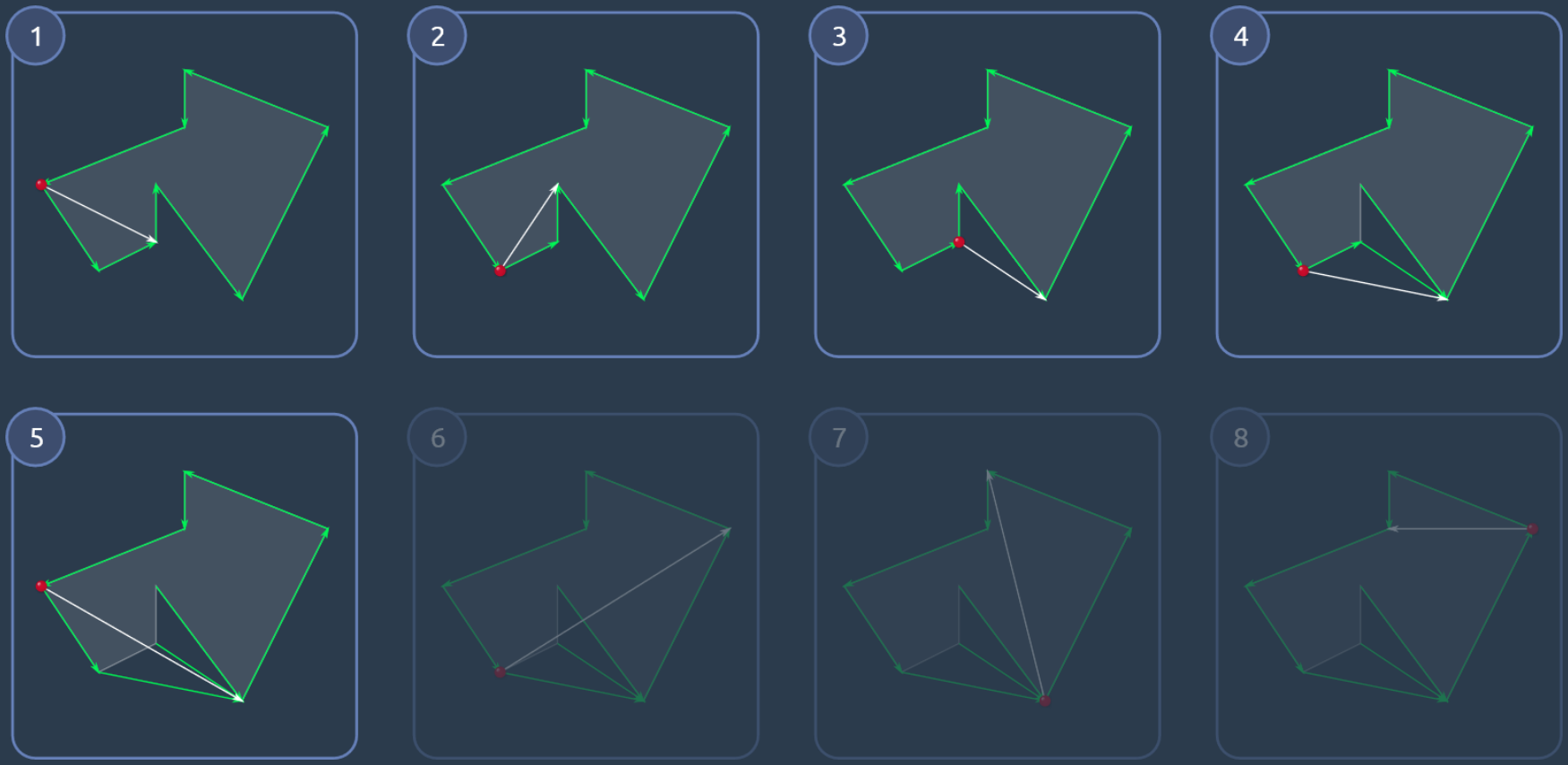


createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle



createGraph

createConvexHull

findShortestPath

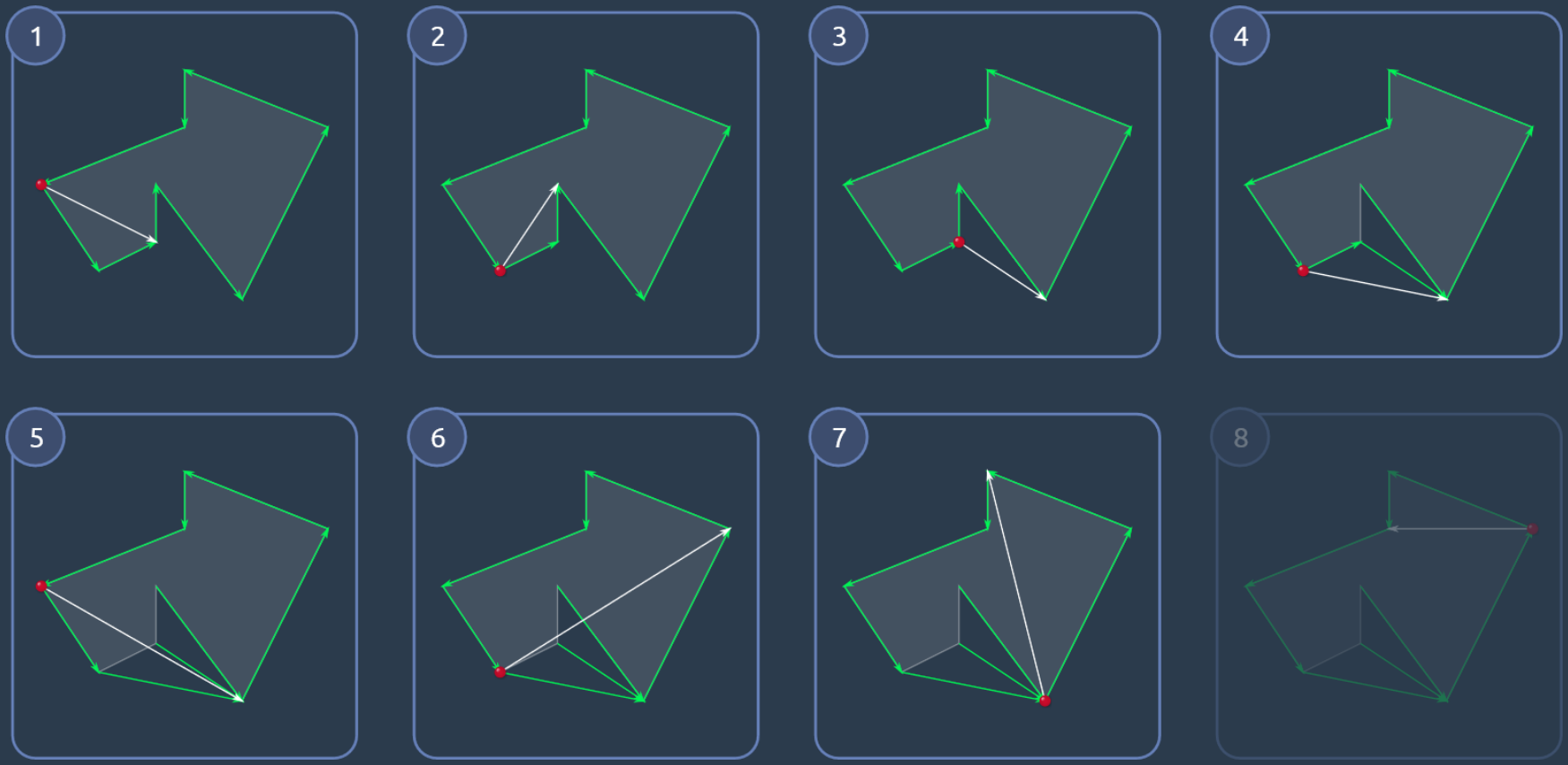
Lösungsvorschlag II
Ermitteln der konvexen Hülle

createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle

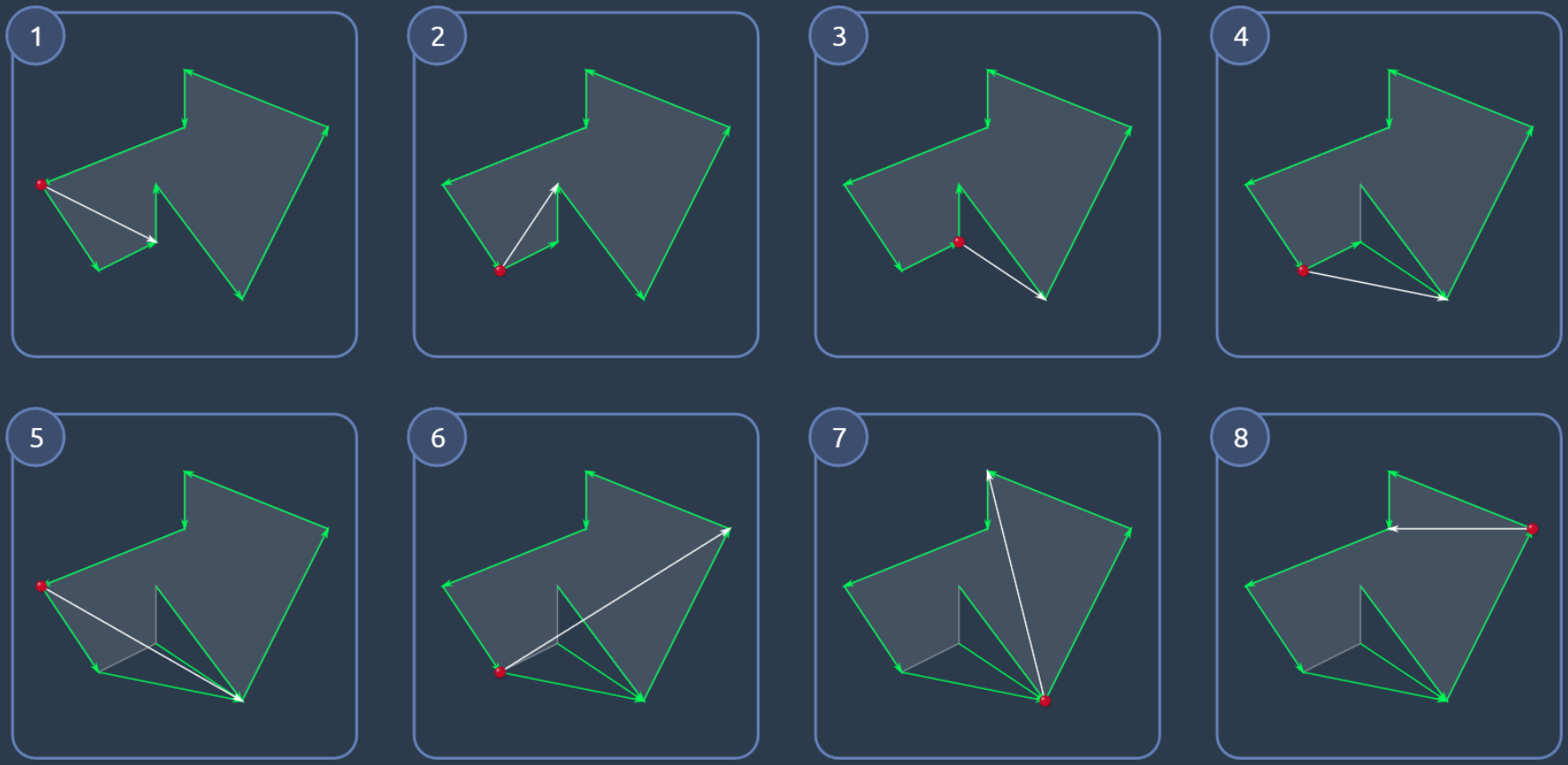


createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle

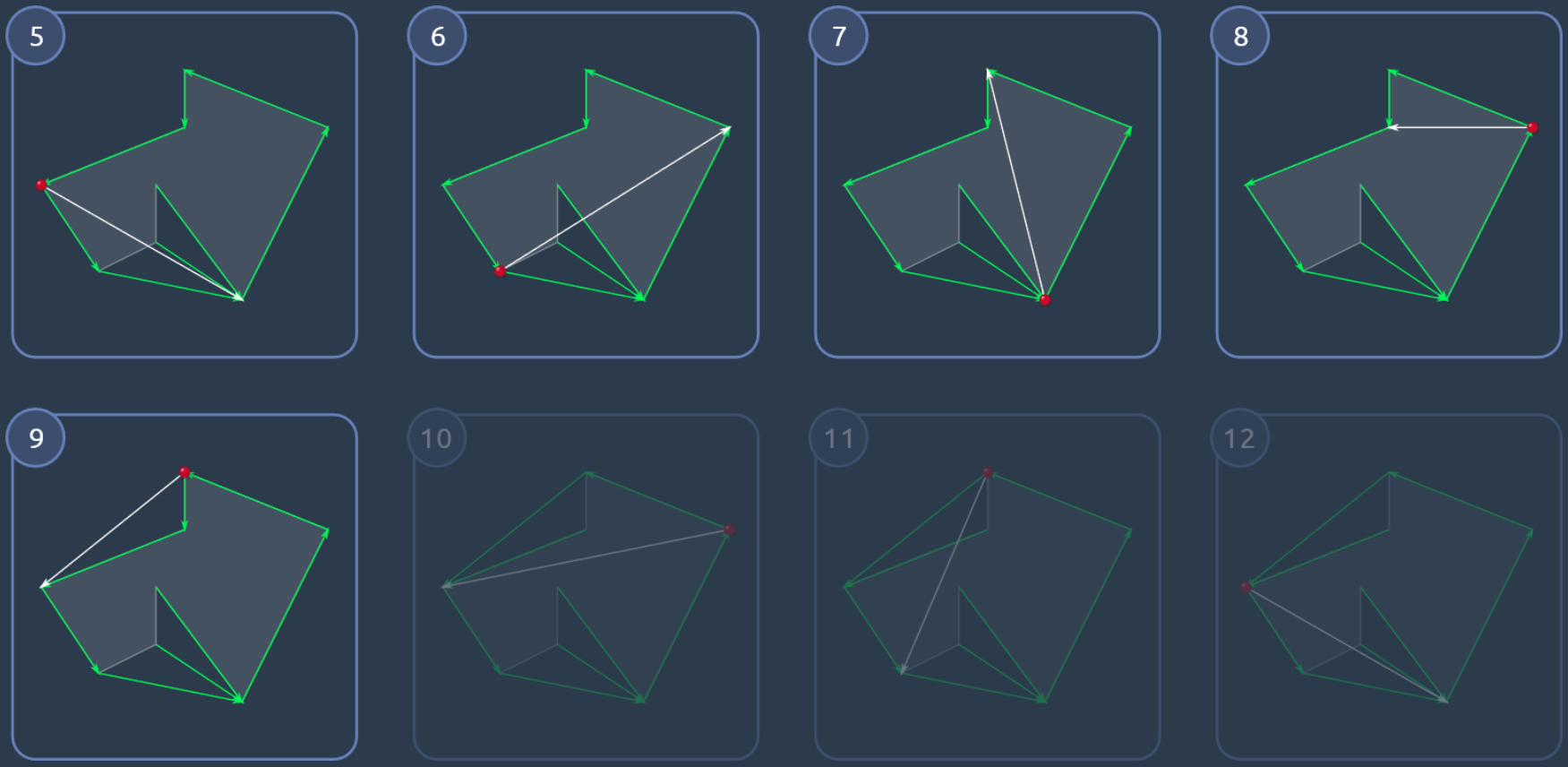


createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle

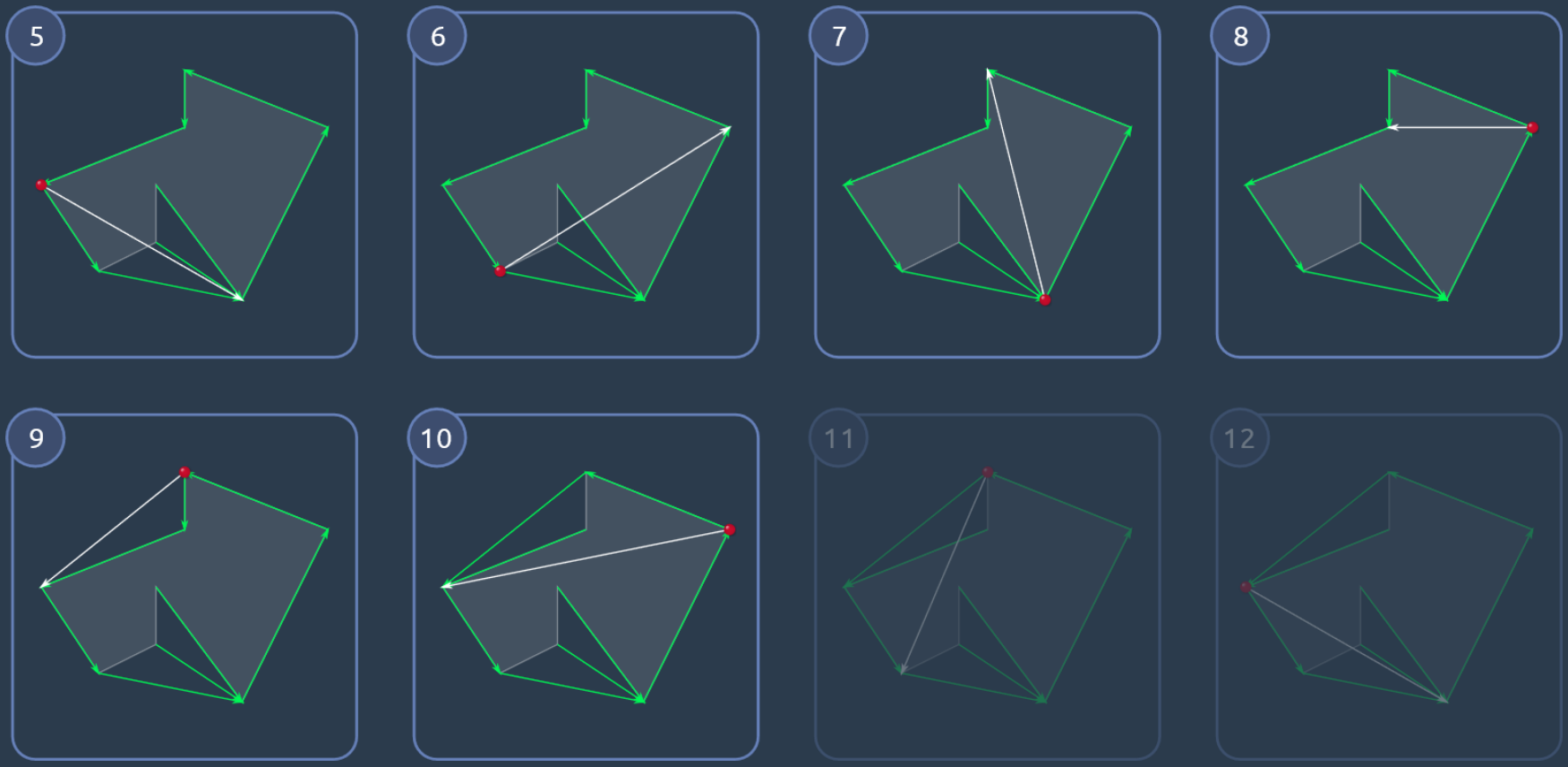


createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle

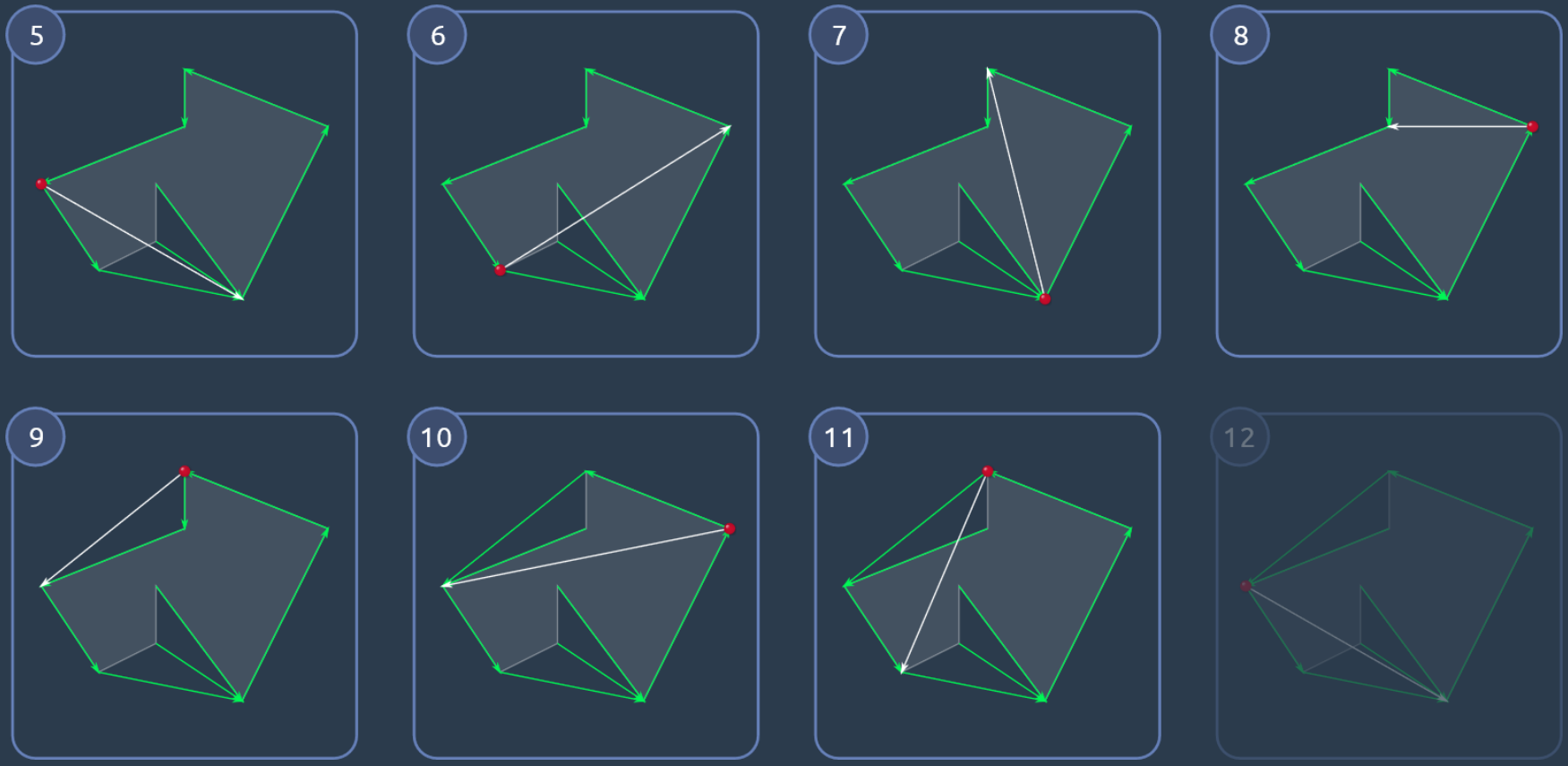


createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle

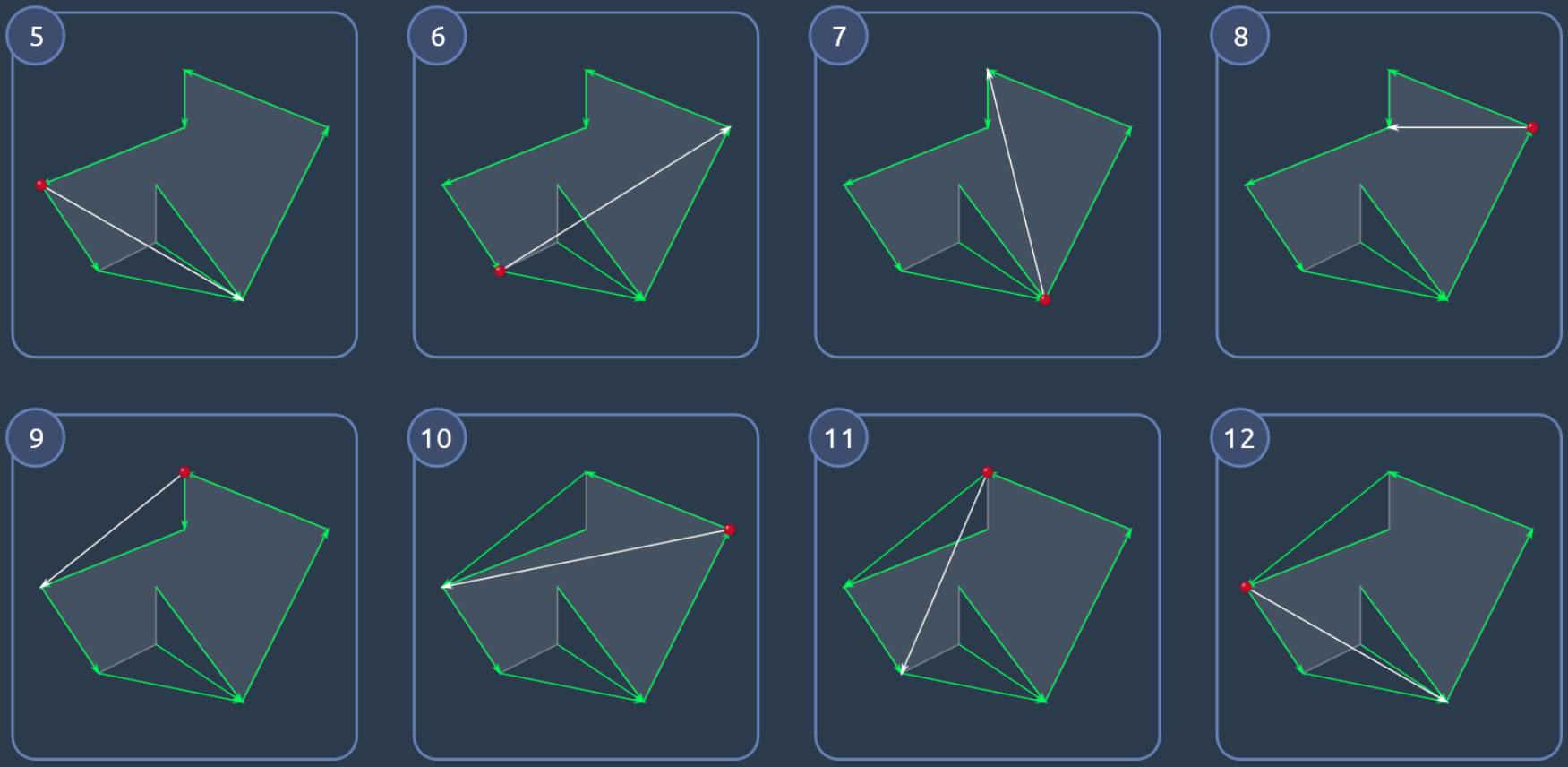


createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle



createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle

```
createConvexHullOfInnerPolygon(Graph G = (V, E1, E2)) {
    nodeToNextNode[#innerPolygon] = array, such that nodeToNextNode[i] = (i + 1) % #innerPolygon

    startingNode = leftmost node of inner polygon
    currentNode = startingNode
    stack = new Stack()
    do
        nextNode = nodeToNextNode[currentNode]
        desiredNode = nodeToNextNode[nextNode]

        isShortcutPossible = ∃(currentNode, desiredNode) ∈ E2 and
                             desiredNode.position is right of line segment from currentNode to nextNode

        if isShortcutPossible then
            nodeToNextNode[currentNode] = desiredNode
            if stack is not empty then
                currentNode = stack.pop()
        else
            stack.push(currentNode)
            currentNode = nodeToNextNode[currentNode]
    until currentNode == startingNode and stack is not empty

    generate polygon from nodeToNextNode and return it
}
```

createGraph

createConvexHull

findShortestPath

Lösungsvorschlag II
Ermitteln der konvexen Hülle

```
createConvexHullOfInnerPolygon(Graph G = (V, E1, E2)) {
    nodeToNextNode[#innerPolygon] = array, such that nodeToNextNode[i] = (i + 1) % #innerPolygon

    startingNode = leftmost node of inner polygon
    currentNode = startingNode
    stack = new Stack()
    do
        nextNode = nodeToNextNode[currentNode]
        desiredNode = nodeToNextNode[nextNode]

        isShortcutPossible = ∃(currentNode, desiredNode) ∈ E2 and
                             desiredNode.position is right of line segment from currentNode to nextNode

        if isShortcutPossible then
            nodeToNextNode[currentNode] =
            if stack is not empty then
                currentNode = stack.pop()
        else
            stack.push(currentNode)
            currentNode = nodeToNextNode[
    until currentNode == startingNode and

    generate polygon from nodeToNextNode and return it
}
```

Warum ist es wichtig, dass nur nach rechts abbiegende Abkürzungen genommen werden? Probiere mal dieses Beispiel. Was passiert, wenn man diesen Check weglässt?