WETTBEWERT KROCKETSPIEL

TEAM ID : 00743
VICTOR USWA
17.11.2024

1.1 KROCKET SPIEL:

Überprüfung, ob alle Tore mit einem Schlag durchquert werden können.

1.2 PROJECT BESCHREIBUNG

Dieses Programm prüft, ob es möglich ist, in einem Krocketspiel alle Tore in der richtigen Reihenfolge mit nur einem einzigen Schlag zu durchqueren. Die Tore werden im zweidimensionalen Raum durch Liniensegmente dargestellt, die durch ihre zwei Endpunkte definiert sind. Der Ball wird als Punkt betrachtet (Ballradius = 0), und das Programm sucht eine gerade Linie, die alle Tore durchläuft.

Anwendungsfall

In einem Krocketspiel werden mehrere Tore in einer bestimmten Reihenfolge aufgestellt. Das Ziel ist, den Ball durch alle Tore mit einem einzigen Schlag zu spielen. Dieses Programm überprüft, ob eine solche Schlagrichtung existiert, die den Ball durch alle Tore führt, ohne dass der Spieler den Ball mehrfach schlagen muss.

Anforderungen

1. Die Tore sind als Liniensegmente im 2D-Raum definiert, wobei jedes Tor zwei Endpunkte hat.
2. Der Ball wird als Punkt betrachtet, dessen Position durch die Koordinaten (x, y) dargestellt wird.
3. Das Programm überprüft, ob alle Tore in der richtigen Reihenfolge von einer einzigen geraden Linie durchquert werden können.

Klassendokumentation

1.3.1 Klasse `Point`

Beschreibung: Die Klasse `Point` repräsentiert einen Punkt im zweidimensionalen Raum.

- Attribute:
- `double x`: Die x-Koordinate des Punktes.
- `double y`: Die y-Koordinate des Punktes.

- Konstruktor:

public Point(double x, double y)

```
public class Point {
    double x, y;

    Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

Erstellt einen Punkt mit den angegebenen x- und y-Koordinaten.

- Beispiel:

Point p1 = new Point(1.0, 2.0);

1.3.2 Klasse `Segment`

Beschreibung : Die Klasse `Segment` repräsentiert ein Tor im Krocketspiel, das als ein Liniensegment im 2D-Raum dargestellt wird.

- Attribute:
- `Point p1`: Der erste Endpunkt des Segments (Tor).
- `Point p2`: Der zweite Endpunkt des Segments (Tor).

- Konstruktor:

public Segment(Point p1, Point p2)

Erstellt ein Segment mit den angegebenen Endpunkten `p1` und `p2`.

```
public class Segment {

    Point p1, p2;

    Segment(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
    }
    Point direction() {
        return new Point(p2.x - p1.x, p2.y - p1.y);
    }
}
```

- Methoden:

- public Point direction():
- Beschreibung: Berechnet den Richtungsvektor des Segments, indem die Differenz der Koordinaten der Endpunkte zurückgegeben wird.
- Rückgabewert: Ein `Point`-Objekt, das die Richtung des Segments repräsentiert.

1.3.3 Klasse ‘ Krocketspiel’

Methodenbeschreibung

`static boolean areCollinear(Point p1, Point p2, Point p3)`
Beschreibung: Überprüft, ob drei Punkte auf einer Linie liegen (kollinear sind).

- Parameter:

- `Point p1`: Der erste Punkt.
- `Point p2`: Der zweite Punkt.
- `Point p3`: Der dritte Punkt.

- Rückgabewert: `true`, wenn die drei Punkte kollinear sind, sonst `false`.

- Logik: Die Punkte sind kollinear, wenn der Flächeninhalt des Dreiecks, das durch diese Punkte gebildet wird, gleich Null ist.

- Beispiel:

```
Point p1 = new Point(0, 0);
Point p2 = new Point(2, 2);
Point p3 = new Point(4, 4);

boolean result = areCollinear(p1, p2, p3);  // true, da alle Punkte auf einer Linie liegen
```

`static boolean areSameDirection(Point dir1, Point dir2)`
Beschreibung: Überprüft, ob zwei Richtungsvektoren parallel sind.

- Parameter:
- `Point dir1`: Der erste Richtungsvektor.
- `Point dir2`: Der zweite Richtungsvektor.

- Rückgabewert: `true`, wenn die beiden Richtungsvektoren parallel sind, sonst `false`.

- Beispiel:

```
Point dir1 = new Point(1, 1);
Point dir2 = new Point(2, 2);

boolean result = areSameDirection(dir1, dir2);  // true, da beide Richtungsvektoren parallel sind
```

```
6
7   static boolean areCollinear(Point p1, Point p2, Point p3) {
8
9       return (p2.y - p1.y) * (p3.x - p2.x) == (p2.x - p1.x) * (p3.y - p2.y);
10  }
11
12
13  static boolean areSameDirection(Point dir1, Point dir2) {
14
15      return (dir1.x * dir2.y - dir1.y * dir2.x) == 0;
16  }
17
```

public static void main(String[] args)
Beschreibung: Die Hauptmethode, die das Programm ausführt. Sie liest die Eingabekoordinaten der Tore vom Benutzer ein, überprüft, ob alle Tore mit einem Schlag durchquert werden können, und gibt das Ergebnis aus.

```
boolean canPassAllGates = true;

for (int i = 1; i < n; i++) {
    Segment currentTor = tore[i];
    if (!areCollinear(start, currentTor.p1, currentTor.p2) ||
        !areSameDirection(direction, currentTor.direction())) {
        canPassAllGates = false;
        break;
    }
}

if (canPassAllGates) {
    System.out.println("Ja, alle Tore können mit einem Schlag durchquert werden!");
    System.out.println("Startpunkt: (" + start.x + ", " + start.y + ")");
    System.out.println("Schlagrichtung: (" + direction.x + ", " + direction.y + ")");
} else {
    System.out.println("Nein, es ist nicht möglich, alle Tore mit einem Schlag zu durchqueren.");
}
```

1.4 Anleitung zur Benutzung

1. Schritt 1: Programm starten
   - Führe das Programm aus. Es wird dich auffordern, die Anzahl der Tore einzugeben.

2. Schritt 2: Tore eingeben
   - Gib die Koordinaten der Tore in der Reihenfolge ein, in der sie durchquert werden sollen. Jedes Tor besteht aus zwei Punkten, die durch ihre x- und y-Koordinaten angegeben werden.

   Beispiel für die Eingabe:

   Wie viele Tore gibt es? 3
   Geben Sie die Koordinaten für Tor 1 ein:
   0 0
   1 1
   Geben Sie die Koordinaten für Tor 2 ein:
   2 2
   3 3
   Geben Sie die Koordinaten für Tor 3 ein:
   4 4
   5 5

3. Schritt 3: Ergebnis
   - Das Programm gibt aus, ob es möglich ist, alle Tore mit einem Schlag zu durchqueren. Wenn es möglich ist, wird auch der Startpunkt und die Schlagrichtung ausgegeben.

   Beispiel für die Ausgabe:

   Ja, alle Tore können mit einem Schlag durchquert werden!
   Startpunkt: (0.0, 0.0)
   Schlagrichtung: (1.0, 1.0)

```
2
2
2
Geben Sie die Koordinaten für Tor 4 ein:
5
5
5
5
Ja, alle Tore können mit einem Schlag durchquert werden!
Startpunkt: (0.0, 0.0)
Schlagrichtung: (0.0, 0.0)
```