Schnittstellen

Raum-Zeit-Scheduler

1 Allgemein

Einige allgemeine Eigenschaften aller Schnittstellen.

1.1 Kein `null`

Im Allgemeinen werden nirgendwo `null`-Argumente akzeptiert. Ebenso werden auch keine `null`-Objekte zurückgegeben. Stattdessen wird werden leere Objekte verwendet. Beispielsweise ein Pfad ohne Knoten oder eine leere Liste. Dies sorgt für weniger Fallunterscheidungen und eine leichter verständliche Logik.

1.2 Immutable

An vielen Stellen werden unveränderliche Objekte genutzt. Beispiele dafür sind `LocalDateTime`, `ImmutableList` oder `ImmutablePolygon`. Dies gewährleistet zum einen die Sicherheit, dass sich Objekte nicht unerwartet ändern und zudem ohne Gefahr direkt zurückgegeben werden können.

Dies ist von großem Vorteil, da sich an den meisten Stellen Objekte, einmal erstellt, nicht ändern. Das Klonen von Objekten beim Initialisieren oder setzen (Setter-Methoden) entfällt somit ebenso wie das Verschachteln in unmodifizierbaren Wrappern beim Zurückgeben (Getter-Methoden).

Die meisten Klassen verwenden Immutable-Varianten um ihre Objekte zu speichern. Beispielsweise speichert ein `DynamicObstacle` ein `ImmutablePolygon`. Bei der Konstruktion akzeptiert es allerdings jedes `Polygon` und wandelt es gegebenenfalls in ein `ImmutablePolygon`. Ein `ImmutablePolygon` ist allerdings vorzuziehen, da dann eine Umwandlung entfällt.[1]

1.3 Zweckgebundene Objekte

Stationäre Hindernisse (`StaticObstacle`) oder Pfade (`Path`) sind im Grunde nichts anderes als Polygone bzw. Listen von Punkten. Dennoch werden diese in spezielle Objekte gekapselt. Neben nützlichen Methoden, die Polygone und Listen nicht bieten, gewährleistet die Kapselung Garantien. Pfade sollten beispielsweise nicht nur aus einem Punkt bestehen und die Punkte selbst sollten keine ungültigen Ordinaten wie `NaN` enthalten. Eine immer wiederkehrende Überprüfung von Hindernissen und Pfaden entfällt somit und ist nur einmalig bei Initialisierungen nötig.

[1] Momentan sieht man vielen Klassen ihre Verwendung von Immutables nicht sofort an. Ich habe vor, für alle betroffenen Konstruktoren und Setter nur eine Variante mit Immutables anzubieten, sodass man gezwungen ist, Immutables zu übergeben.

2 Initialisierung

Bevor der Scheduler neue Aufgaben einplanen kann, muss er mit Informationen über die Welt und die Arbeitseinheiten initialisiert werden. Nach der Initialisierung können keine Änderungen mehr an der Welt oder den Arbeitseinheiten vorgenommen werden.[2]

```
public Scheduler(World world, Collection<WorkerUnitSpecification>
      workerSpecs)
```

2.1 Welt

Die Welt wird mit einer Liste aus stationären Hindernissen und einer Liste aus dynamischen Hindernissen initialisiert.

```
public World(Collection<StaticObstacle> staticObstacles,
      Collection<DynamicObstacle> dynamicObstacles)
```

2.2 Hindernisse

Hindernisse besitzen eine Form, die durch ein Polygon beschrieben wird.

```
public StaticObstacle(Polygon shape)
public DynamicObstacle(Polygon shape, Trajectory trajectory)
```

Zur Erstellung eines Polygons kann der `StaticGeometryBuilder` verwendet werden. Für eine rechteckige Form könnte dies so aussehen:

```
new StaticObstacle(immutableBox(-1, -1, 1, 1))
```

Dynamische Hindernisse besitzen überdies noch eine Trajektorie. Diese besteht wiederum aus einem räumlichen Pfad (`SpatialPath`) und einer Liste von Zeitpunkten (`LocalDateTime`). Beide Listen müssen dieselbe Größe besitzen, da sie nur kombiniert einen dreidimensionalen Pfad ergeben.

```
public SpatialPath(List<Point> vertices)
public SimpleTrajectory(SpatialPath spatialPath, List<LocalDateTime> times)
```

2.3 Arbeitseinheitspezifikation

Die Arbeitseinheitspezifikation legt die Eigenschaften einer Arbeitseinheit fest. Diese umfasst die Form, die Höchstgeschwindigkeit, die Anfangsposition und die Anfangszeit.

```
public WorkerUnitSpecification(Polygon shape, double maxSpeed, Point
      initialLocation, LocalDateTime initialTime)
```

[2] Später soll es allerdings schon möglich sein, neue Arbeiter einzufügen oder bestehende zu entfernen. Ähnliches könnte auch für die Welt möglich sein. Der Einfachheit halber stehen diese Funktionen allerdings bisher nicht zur Verfügung.

3 Einplanung

Zur Einplanung einer neuen Aufgabe muss eine Aufgabenspezifikation (`TaskSpecification`) angelegt werden. Diese gibt den räumlichen und zeitlichen Rahmen sowie die Dauer der Aufgabe fest.

```
public TaskSpecification(Polygon locationSpace, LocalDateTime
      earliestStartTime, LocalDateTime latestStartTime, Duration duration)
```

Diese Spezifikation wird dann der `schedule`-Funktion des Schedulers übergeben:

```
public boolean schedule(TaskSpecification specification)
```

Der Rückgabewert gibt an, ob die Spezifikation erfüllt werden konnte.[3] Wurde sie nicht erfüllt, wurde auch keine Aufgabe eingeplant.

[3] Die Rückmeldung des Schedulers ist bisher noch recht dürftig. Hier müssen wir noch besprechen, welche Rückmeldungen nötig sind und wie diese am besten umgesetzt werden sollen.

4 Arbeitseinheitreferenz

Der Scheduler bietet Referenzen (`WorkerUnitReference`) auf seine Arbeitseinheiten an. Diese ermöglichen Auskünfte über die Eigenschaften der Arbeitseinheiten. Außerdem lässt sich die Trajektorie berechnen:

```
public Trajectory calcTrajectory()
```